



Softverski definisane mreže SDN

Živko Bojović

Beograd, 2025.

UNIVERZITET SINGIDUNUM

Živko Bojović

SOFTVERSKI DEFINISANE MREŽE

Prvo izdanje

Beograd, 2025.

SOFTVERSKI DEFINISANE MREŽE

Autor:

prof. dr Živko Bojović

Recenzenti:

prof. dr Mladen Veinović

Redovni profesor Univerziteta Singidunum

prof. dr Ivan Luković

Redovni profesor Fakulteta organizacionih nauka, Univerziteta u Beogradu

Izdavač:

Univerzitet Singidunum

Danijelova 32

www.singidunum.ac.rs

Za izdavača:

prof. dr Milovan Stanišić

Priprema za štampu:

Miloš Višnjic

Dizajn korica:

Aleksandar Mihajlović, MA

Fotografija korica:

Cover designed by Freepik / rawpixel.com

Godina izdanja:

2025.

Tiraž:

400

Štampa:

Caligraph, Beograd

ISBN: 978-86-7912-840-9

Copyright © 2025 Univerzitet Singidunum

Izdavač zadržava sva prava.

Reprodukcija pojedinih delova ili celine ove publikacije nije dozvoljena.

Sadržaj

Predgovor

VIII

POGLAVLJE 1 RAČUNARSKE MREŽE SA TRADICIONALNOM ARHITEKTUROM

1. Računarske mreže sa tradicionalnom arhitekturom	5
1.1. Istorijski razvoj mrežne arhitekture	5
1.2. Troslojna arhitektura tradicionalnih računarskih mreža	7
1.3. Karakteristike troslojne arhitekture	11
1.4. Prednosti i izazovi	13
1.4.1. Ključne prednosti	13
1.4.2. Glavni izazovi	14
1.5. Troslojna mrežna arhitektura u realnim okruženjima	16
Zaključak	17

POGLAVLJE 2 KONCEPT SOFVERSKI DEFINISANOG UMREŽAVANJA

2. Koncept sofverski definisanog umrežavanja	21
2.1. Uvod u softverski definisano umrežavanje	21
2.2. Arhitektura SDN mreža	23
2.3. Centralizacija mrežne logike	27
2.4. Programabilnost u SDN mrežama	29
2.5. Automatizacija u SDN mrežama	31
2.6. Modeli implementacije SDN tehnologije	32
2.6.1. Open SDN	33
2.6.2. Hibridni SDN	34
2.6.3. SDN preko API-a	37
2.6.4. SDN overlay mreže	38
Zaključak	39

POGLAVLJE 3 VIRTUELIZACIJA U INFRASTRUKTURI SDN MREŽA

3. Virtuelizacija u infrastrukturi SDN mreža	43
3.1. Istorijat virtuelizacije	43
3.2. Koncept virtuelizacije	44
3.2.1. Softver za virtuelizaciju	45
3.2.2. Virtuelne mašine	46
3.2.3. Alokacija resursa	48
3.2.4. Prednosti i izazovi primene virtuelnih mašina	50
3.3. Razlike između virtuelizacije i emulacije	51
3.4. Virtuelizacija mreže	52
3.4.1. Infrastruktura virtuelnih mreža	54
3.4.2. Ključne komponente virtuelne mreže	54
Zaključak	65

POGLAVLJE 4 KONTEJNERIZACIJA U SDN MREŽAMA

4. Kontejnerizacija u SDN mrežama	71
4.1. Uvod u kontejnerizaciju	71
4.2. Koncept kontejnerizacije	72
4.3. Vrste virtuelnih kontejnera	76
4.4. <i>Docker</i> platforma	81
4.5. Orkestracija	85
4.5.1. <i>Kubernetes</i> platforma	86
4.5.2. Arhitektura <i>Kubernetes</i> platforme	87
4.5.3. Objekti u <i>Kubernetes</i> platformi	91
4.5.4. Vrste objekata u <i>Kubernetes</i> platformi	93
4.6. Kontejnerizacija u SDN okruženju	101
4.7. Primena kontejnerizacije u realnim SDN okruženjima	102
Zaključak	103

POGLAVLJE 5 NFV KONCEPT U SDN MREŽAMA

5. NFV koncept u SDN mrežama	109
5.1. Uvod	109
5.2. Osnove NFV koncepta	110
5.3. ETSI NFV programski okvir (eng. <i>framework</i>)	111
5.3.1. Komponente NFV arhitekture	113
5.4. Referentne tačke u NFV arhitekturi	123
Zaključak	123

POGLAVLJE 6 KONTROLNA RAVAN U SDN MREŽAMA

6. Kontrolna ravan u SDN mrežama	129
6.1. Uvod	129
6.2. Arhitektura kontrolne ravni	130
6.3. Komponente kontrolne ravni	132
6.3.1. SDN kontroler	133
6.3.2. SDN orkestrator	136
6.3.3. Interfejsi u SDN arhitekturi	138
6.4. Funkcionalnosti kontrolne ravni	140
6.5. Bezbednost na kontrolnoj ravni	142
Zaključak	143

POGLAVLJE 7 PROTOKOLI ZA KOMUNIKACIJU NA KONTROLNOJ RAVNI

7. Protokoli za komunikaciju na kontrolnoj ravni	147
7.1. Uvod	147
7.2. Protokoli za kontrolu ravni podataka	148
7.2.1. <i>OpenFlow</i> protokol	149
7.2.2. Primena <i>OpenFlow</i> protokola	154
7.3. Protokoli za upravljanje konfiguracijom mrežnih uređaja	156
7.3.1. <i>Open vSwitch Database Management</i> protokol	157
7.4. Protokoli za razmenu informacija o topologiji mreže	160
7.4.1. <i>NetFlow</i> protokol	160
7.4.2. <i>OF-Config</i> protokol	163
7.5. Protokoli za prenos politika koje se definišu na kontroleru i implementiraju na mrežnim uređajima	164
7.5.1. <i>NETCONF</i> protokol	165
Zaključak	167

POGLAVLJE 8 SD-WAN TEHNOLOGIJA

8. SD-WAN tehnologija	173
8.1. Uvod u SD-WAN	173
8.2. SD-WAN arhitektura	175
8.3. <i>On-Premises</i> SD-WAN implementacija	177
8.4. <i>Cloud-Hosted</i> SD-WAN	180
8.5. Hibridni model SD-WAN arhitekture	182
Zaključak	184

POGLAVLJE 9
SDN I MREŽNI SLICING

9. SDN i mrežni <i>slicing</i>	189
9.1. Moderni trendovi u komunikacijama	189
9.2. Definicija mrežnog <i>slicing</i> -a u 5G/6G mrežama	192
9.3. SDN i dinamičko kreiranje i upravljanje <i>slice</i> -vima	194
9.4. Različiti tipovi <i>slice</i> -va i uloga SDN-a u njihovom optimizovanju	196
9.5. SDN upravljanje resursima između različitih <i>slice</i> -va	199
9.6. Izazovi implementacije mrežnog <i>slicing</i> -a u 5G i 6G mrežama	200
<i>Zaključak</i>	201
Literatura	203
Lista akronima	210
Lista slika	215

Predgovor

Svedoci smo brzog razvoja informacionih i komunikacionih tehnologija i njihove sveobuhvatne primene u skoro svim segmentima društva. Jedno od ključnih područja ovog razvoja jeste evolucija računarskih mreža, jer je praksa pokazala da mreže sa tradicionalnom arhitekturom, sve više pokazuju ograničenja u pogledu fleksibilnosti, skalabilnosti i jednostavnosti upravljanja. U odgovoru na ove izazove, koncept softverski definisanih mreža (SDN) je nastao kao revolucija u pristupu dizajnu i upravljanju mrežnim sistemima. Ova knjiga ima za cilj da pruži sveobuhvatan pregled ovog inovativnog koncepta, njegovih tehnoloških osnova i praktične primene.

U uvodnom poglavlju (Poglavlje 1) je dat iscrpan pregled tradicionalnih mrežnih arhitektura, sa željom da se studentima, ali i drugim čitaocima, predoči istorijski kontekst i suštinski objasni koncept troslojnog modela arhitekture računarskih mreža. On predstavlja osnovu koja treba da omogući, da se jasno sagledaju prednosti, ali i ograničenja tradicionalnog pristupa. Upravo iz ovih ograničenja, treba da se izvuču zaključci i da se pravilno identifikuju motivi za promenom paradigme umrežavanja, koju koju upravo donosi SDN tehnologija. U drugom poglavlju predstavljen je koncept softverski definisanog umrežavanja. Obrađuju se ključni principi, uključujući arhitekturu SDN mreža, centralizaciju mrežne logike i programabilnost. Poglavlje se posebno bavi automatizacijom kao jednim od najvažnijih aspekata SDN-a, kao i različitim modelima implementacije, poput *Open* SDN-a, SDN-a preko API-a, hibridnih SDN modela i *overlay* mreža. Virtuelizacija, kao neizbežan koncept u transformaciji mrežne infrastrukture, detaljno je obrađena u trećem poglavlju. Ovde se istražuju istorijat i koncept virtuelizacije, uključujući infrastrukturu virtuelnih mreža i ključne komponente, poput virtuelnih mašina i procesa alokacije resursa. Ova tema se dalje nadovezuje na kontejnerizaciju, koja je fokus četvrtog poglavlja. Pored objašnjenja koncepta različitih platformi (*Docker* i *Kubernetes*), poglavlje opisuje primenu kontejnerizacije u SDN okruženjima i njen značaj za orkestraciju resursa. Peto poglavlje se bavi NFV (eng. *Network Function Virtualization*) i opisuje kako njegova integracija u SDN okruženje, dodatno obogaćuje arhitekturu mreže. Poseban osvrt je dat na fleksibilno i skalabilno upravljanje mrežnim funkcijama. Obrađuje se ETSI NFV programski okvir i njegove ključne komponente, pružajući praktičan uvid u mogućnosti ove tehnologije. Ključna uloga kontrolne ravni SDN mreža je objašnjena u Poglavlju 6. U njemu se analiziraju arhitektura, funkcionalnosti i bezbednosni aspekti, kao i uloga SDN kontrolera i orkestratora. Sedmo poglavlje je posvećeno protokolima za komunikaciju na kontrolnoj ravni, uključujući *OpenFlow*, *OVSD*, *OF-CONFIG*, *NetFlow* i *NETCONF* protokole, koji omogućavaju efikasnu razmenu informacija između slojeva mreže. U osmom poglavlju, knjiga se bavi SD-WAN tehnologijom, koja predstavlja ključnu primenu SDN-a u širokopojasnim mrežama. Analiziraju se različiti modeli implementacije, uključujući *on-premises*, *cloud-hosted* i hibridne arhitekture, sa naglaskom na njihovu ulogu u modernim mrežnim sistemima. Poslednje poglavlje objašnjava značaj integracije SDN mreža sa mrežnim *slicing*-om, posebno u kontekstu 5G i 6G mreža. Dinamičko kreiranje i upravljanje *slice*-vima, optimizacija resursa i izazovi implementacije predstavljaju neke od ključnih tema. Ovaj deo knjige pruža uvid u budućnost komunikacionih tehnologija i ulogu SDN-a u njihovom razvoju.

Knjiga je nastala kao rezultat višegodišnjeg rada autora na predmetima računarske mreže, softverski definisano umrežavanje i distribuirani računarski servisi – *cloud computing*. Prvenstveno je namenjena, kao osnovna udžbenička literatura, studentima osnovnih akademskih studija na Singidunum univerzitetu. Činjenica da na našem jeziku ne postoji knjiga koja na ovakav način obrađuje ovu tematsku oblast, predstavljala je dodatni motiv za autora, da uloži napor u objedinjavanju materije koja povezuje ključne oblasti softverski definisanog umrežavanja. Pored teorijskih osnova, knjiga je obogaćena praktičnim primerima i referencama na aktuelne tehnologije. Namenjena je studentima, ali i istraživačima, mrežnim administratorima i svima koji žele da razumeju moderni koncept upravljanja mrežama. Nadam se da će ovaj udžbenik doprineti vašem razumevanju i inspirisati vas za dalja istraživanja u oblasti softverski definisanih mreža.

Autor



POGLAVLJE 1



RAČUNARSKE MREŽE SA TRADICIONALNOM ARHITEKTUROM

1. RAČUNARSKE MREŽE SA TRADICIONALNOM ARHITEKTUROM

1.1. ISTORIJSKI RAZVOJ MREŽNE ARHITEKTURE

Mnoge računarske mreže i dalje imaju tradicionalni dizajn mreže, koji se zasniva na troslojnoj arhitekturi. Ova arhitektura se razvijala paralelno sa evolucijom računarskih mreža i potrebama velikih organizacija za skalabilnim, pouzdanim i efikasnim načinom povezivanja. Postala je standard za dizajniranje mreža, zbog svoje sposobnosti, da odgovori na sve složenije zahteve u pogledu kapaciteta i upravljanja mrežama. Međutim, krajem 1980-ih i početkom 1990-ih infrastrukture došlo je do rasta kompleksnosti u računarskim mrežama. Mnoge organizacije, počele su da povezuju svoje računare i servere preko jednostavnih lokalnih mreža (LAN). Na samom početku, ove mreže su bile ravnog (eng. *flat*) dizajna i nisu posedovale rešenja, koja bi se zasnivala na logičkoj segmentaciji. Kako su potrebe za umrežavanjem rasle, posebno sa uvođenjem Interneta i povećanjem broja korisnika i aplikacija, one su se pokazale neefikasnim. Povećan broj uređaja i zahteva, dovodio je do zagušenja saobraćaja i problema sa skalabilnošću. Da bi se rešili ovi problemi, počelo se sa primenom segmentacije mreže na različite slojeve, sa specifičnim funkcijama. Tako je nastala troslojna arhitektura sa pristupnim, distribucionim i slojem jezgra.

Ključan napredak u razvoju i primeni troslojne arhitekture nastupio je krajem 1990-ih. Sa širenjem mreža, rasla je njihova složenost i potreba za boljom kontrolom i efikasnijim upravljanjem saobraćajem. Troslojna arhitektura je omogućila jednostavnije upravljanje i skaliranje, a velika preduzeća počela sa širokom popularizacijom ovog modela, pa je ova organizacija mreže ubrzo postala industrijski standard. Od presudnog značaja je bila činjenica, da su mreže zasnovane na troslojnom modelu omogućavale jednostavnije proširenje dodavanjem dodatnih uređaja na pristupnom sloju (eng. *access layer*) i jačanjem kapaciteta distribucionog sloja (eng. *distribution layer*), što je bilo veoma korisno za okruženja sa stalnim rastom broja korisnika. Njihova skalabilnost je bila od velikog značaja. Razdvajanjem funkcija u svakom sloju mreže, dodavanjem novih pristupnih *switch*-eva i bez velikih promena u jezgru (eng. *core*) ili distribucionom sloju mreže, omogućen je brz rast mreže. Troslojna arhitektura je omogućila lakšu segmentaciju mreže, sa manjim opterećenjima na pojedinačnim uređajima i efikasnijim prenosom podataka. Definisanjem slojeva sa specifičnim ulogama (pristupni, distributivni i sloj jezgra), stvoreni su uslovi, da se izvrši optimizacija resursa. Preduzeća su počela da dodaju nove uređaje, proširuju opseg i kapacitete mreže, ali i da nadograđuju postojeću infrastrukturu na modularan način.

Pouzdanost i otpornost na kvarove, bili su i ostali ključni zahtevi u velikim računarskim mrežama, naročito u poslovnim okruženjima i *data centrima*. Zato se mrežama sa troslojnom arhitekturom, kao odgovor na zahteve da se obezbedi dostupnost servisa, čak i u slučaju kvara pojedinih uređaja, obezbeđuje redundansa kroz višestruke veze između distributivnog i sloja jezgra. Međutim, u nekim slučajevima, obezbeđivanje redundanse je stvaralo probleme. Oni su se manifestovali pojavom petlji u sloju mrežnog interfejsa. Primenom protokola kao što su *Multiple-Instance Spanning Tree Protocol* (MSTP) i *Spanning Tree*

Protocol (STP), omogućeno je izbegavanje petlji u mreži, pa su redundansa i *failover* mehanizmi značajno doprineli kontinuitetu u radu mreže.

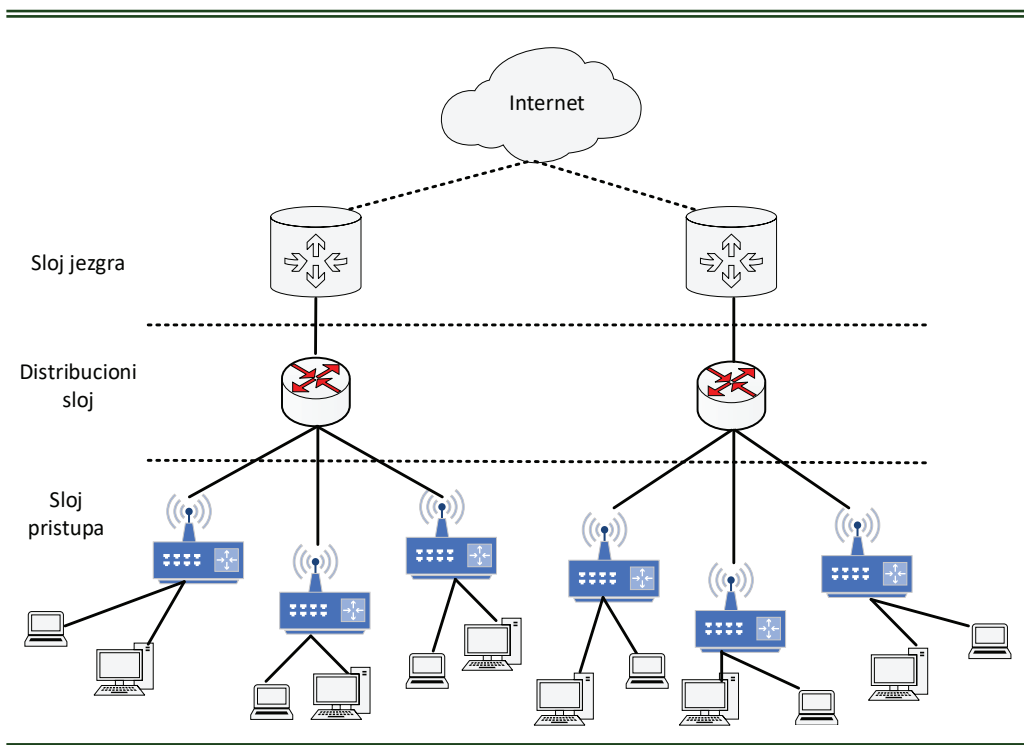
Troslojna arhitektura je doprinela, da se u značajnoj meri pojednostavi upravljanje i kontrola saobraćaja. Jasno su definisane funkcije svakog sloja, gde je pristupni sloj odgovoran za direktno povezivanje korisnika, distributcioni sloj za agregaciju i primenu različitih mrežnih politika, a sloj jezgra za brz prenos podataka. Time je omogućeno administratorima, da efikasno upravljaju mrežom, koristeći mehanizme na odgovarajućim slojevima kao što su VLAN (eng. *Virtual Local Area Network*) i ACL (eng. *Access Control List*) pravila. Stvorena je struktura, u kojoj je bilo moguće, uraditi dijagnostiku problema na jednostavniji način i lakše rešavati kvarove jednostavnijom identifikacijom problema u mreži (npr. u pristupnom ili distribucionom sloju).

Početkom 2000-ih godina, sa pojavom naprednih tehnologija, resursno zahtevnijih -inteligentnih servisa i novih vidova komunikacija (npr. *Machine-to Machine* - M2M), broj korisnika je povećan eksponencijalno, a njihove potrebe su se drastično promenile. Tradicionalna troslojna arhitektura počela je da pokazuje određena ograničenja u fleksibilnosti i skalabilnosti, što se pre svega ogledalo u nemogućnosti, da se prati dinamika promena u realnim okruženjima. Kao jedino rešenje, nametnula se softverizacija mreže. Ona podrazumeva unošenje znatno većeg nivoa programabilnosti u mrežu i primenu različitih tehnika virtuelizacije mrežne infrastrukture.

1.2. TROSLOJNA ARHITEKTURA TRADICIONALNIH RAČUNARSKIH MREŽA

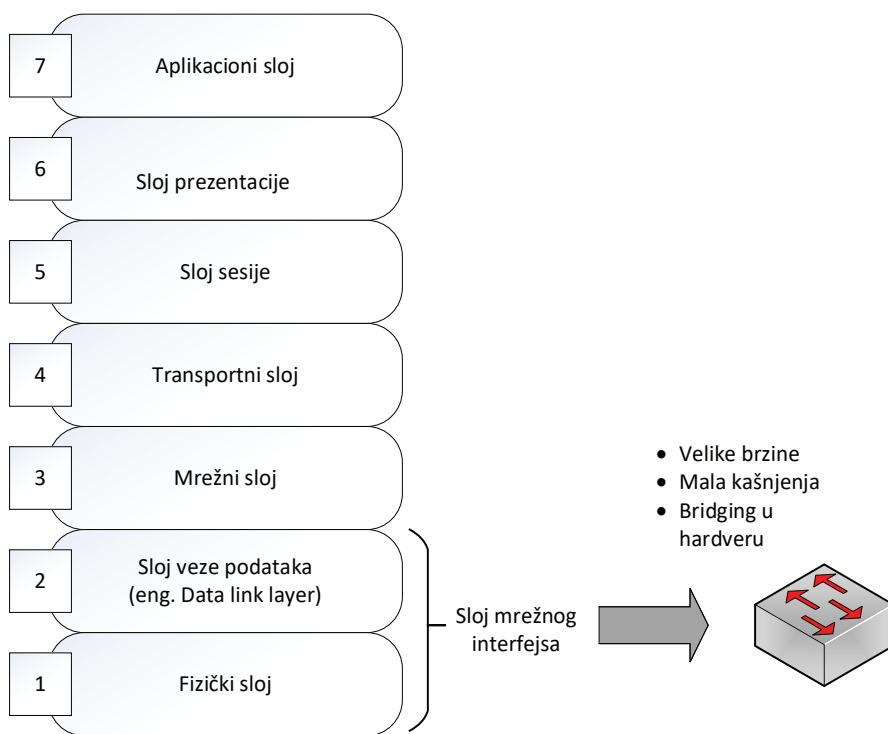
Konceptualni model tradicionalne arhitekture računarskih mreža, danas se uglavnom koristi u preduzećima, ali i u velikim računarskim mrežama, kao što su mreže provajdera (eng. *Internet Service Provider* - ISP). Ona podrazumeva implementaciju troslojne organizacije mreže: pristupni sloj (eng. *Access Layer*), sloj agregacije ili distribucije (eng. *Distribution Layer*) i sloj jezgra (eng. *Core Layer*) (Slika 1.). Svaki sloj ima specifične zadatke i funkcije, čija realizacija treba da obezbedi bolje performanse, pouzdanost i fleksibilnost mrežne infrastrukture. Osnovni cilj je da se omogući jednostavnije upravljanje, lakše održavanje i proširenje postojeće infrastrukture (zadovoljavajuća skalabilnost) i visoka pouzdanost u radu. To u praksi znači:

- jednostavnost upravljanja – svaki sloj ima svoju ulogu, pa administratori mreže mogu lakše identifikovati i rešavati probleme bez uticaja na celu mrežu,
- široku skalabilnost – mreža se može jednostavno proširiti dodavanjem novih uređaja u pristupnom sloju, bez potrebe za modifikacijom *core* sloja.
- potreban nivo bezbednosti – distribucionni sloj može da primenjuje različite bezbednosne politike, omogućavajući veću kontrolu nad protokom podataka u mreži.



Slika 1. Tradicionalne računarske mreže sa troslojnom organizacijom

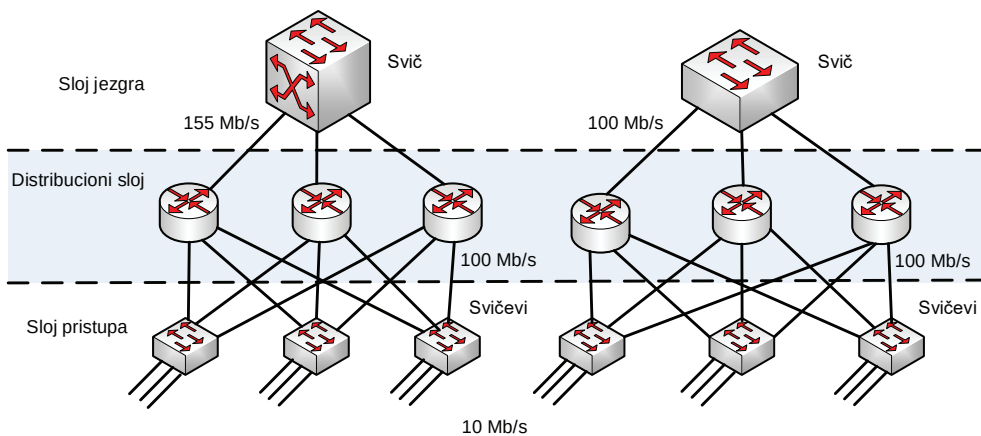
Ključnu ulogu u troslojnoj arhitekturi mreža ima pristupni sloj, jer on omogućava direktno povezivanje računara, servera, štampača, IP telefona, bežičnih pristupnih tačaka (ang. *Access Point* - AP) i drugih krajnjih uređaja sa mrežom. Presudan značaj u povezivanju imaju *switch*-evi, uređaji koji su zaduženi za prebacivanje podataka između krajnjih uređaja i ostatka mreže. Još jedna važna funkcija ovog sloja je upravljanje pristupom. Na ovom sloju se implementiraju osnovne bezbednosne kontrole (poput autentifikacije korisnika ili uređaja, primenom protokola kao što su IEEE 802.1X protokoli) i omogućava upravljanje VLAN-ovima (eng. *Virtual Local Area Networks*), što pomaže u segmentaciji mreže i kontroli saobraćaja na osnovu potreba različitih grupa korisnika (Slika 2.).



Slika 2. Sloj pristupa u tradicionalnoj, troslojnoj arhitekturi

Druga važna funkcija na ovom sloju je kontrola propusnosti i kvalitet usluge (eng. *Quality of service* - QoS). U tom smislu, ovde se implementiraju različiti koncepti, kojima se ostvaruje željeni nivo kvaliteta servisa, poput postavljanja prioriteta za različite tipove saobraćaja (npr. VoIP saobraćaj može imati prioritet u odnosu na prenos podataka i sl). Ovo je posebno važno za optimizaciju performansi aplikacija i usluga koje koriste mrežu. Lako se može zaključiti, da je funkcionalnost pristupnog sloja osnova i za skalabilnost mreže, jer ona treba da omogući jednostavno dodavanje i povezivanje novih korisnika, uređaja i aplikacija. Omogućavajući segmentaciju mreže i lokalizaciju saobraćaja, pristupni sloj pomaže u izbegavanju zagušenja na višim slojevima mreže. Time se obezbeđuje fleksibilnost, koja je ključna za prilagođavanje mreže rastućim potrebama preduzeća.

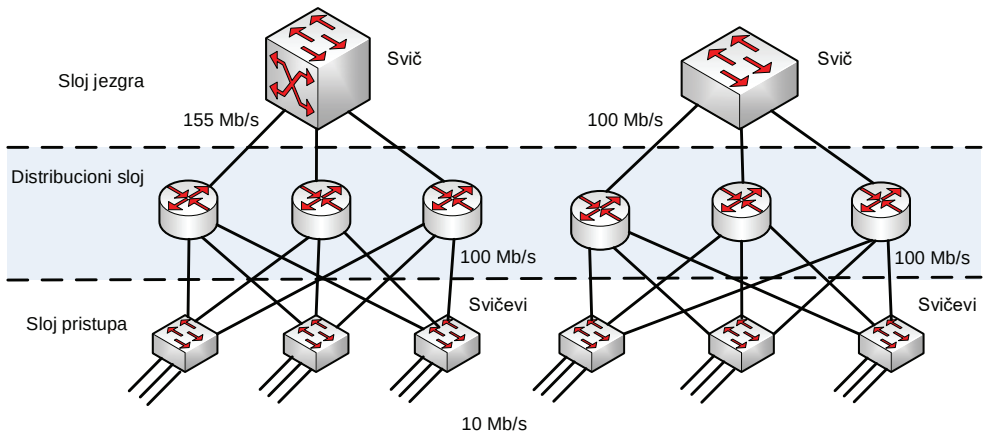
Iako svi slojevi u troslojnoj arhitekturi doprinose celokupnom upravljanju mrežom, distributivni sloj ima glavnu ulogu u centralizovanoj kontroli i implementaciji saobraćajnih pravila, što direktno doprinosi pojednostavljivanju upravljanja u troslojnoj arhitekturi mreža (Slika 3.). Ovaj sloj je odgovoran za primenu mrežnih politika i agregaciju saobraćaja, čime olakšava kontrolu i upravljanje protokom podataka unutar mreže. U praksi, primena mrežnih politika podrazumeva postavljanje pravila, koja se odnose na bezbednost, kvalitet servisa i filtriranje saobraćaja. Njihovom primenom, administratori mreže kontrolišu i upravljaju saobraćajem unutar mrežnih segmenata. S druge strane, agregacijom i konsolidacijom saobraćaja sa pristupnog sloja, distributivni sloj smanjuje opterećenje u jezgru mreže i omogućava lakše upravljanje resursima. Važna funkcija ovog sloja je i povezivanje različitih mrežnih segmenata, kao što su VLAN-ovi. Sve ovo ukazuje na značaj distributivnog sloja za organizaciju i kontrolu saobraćaja između različitih delova mreže.



Slika 3. Distribucionni sloj u tradicionalnoj, troslojnoj arhitekturi

Kritična komponenta mreže koja obezbeđuje visoku brzinu prenosa i pouzdanost za celu mrežnu arhitekturu je sloj jezgra (Slika 4.). Njegova uloga je da brzo i efikasno prenese velike količine podataka između različitih delova mreže. Time ovaj sloj omogućava, da se distributivni i pristupni sloj fokusiraju na lokalne operacije i upravljanje. Zbog svojih visokih zahteva za performansama, sloj jezgra se obično sastoji od resursno najmoćnijih mrežnih uređaja u infrastrukturi. Njegova osnovna funkcija je da obezbedi kičmu (eng. *backbone*) mreže, kroz koju prolaze svi paketi na putu ka svojim odredištima. Zbog toga, uređaji u ovom sloju moraju da budu visokih performansi, sposobni da obrade velike količine saobraćaja uz minimalnu latenciju. Da bi se izbegli prekidi u radu mreže, u sloju jezgra se često implementira više uređaja i kreiraju redundantne veze. U slučaju kvara jednog uređaja ili veze, saobraćaj se preusmerava preko drugih (alternativnih) veza i time se obezbeđuje visoka dostupnost mreže. Ovo je od ogromnog značaja, posebno u velikim poslovnim okruženjima, gde su prekidi neprihvatljivi. Za razliku od distributivnog sloja, sloj jezgra ne uključuje složene operacije poput filtriranja

saobraćaja, primene politika ili usmeravanja po pojedinačnim kriterijumima. U njemu je fokus na brzini i efikasnosti, što znači da je pakete potrebno preneti između različitih delova mreže, što je brže moguće i bez dodatne obrade. Time se smanjuju kašnjenja (latencija), što je posebno važno imajući u vidu da sloj jezgra povezuje različite lokacije ili mrežne segmente (jezgro mreže omogućava komunikaciju između geografski razdvojenih lokacija ili npr. *data* centara).



Slika 4. Sloj jezgra u tradicionalnoj, troslojnoj arhitekturi

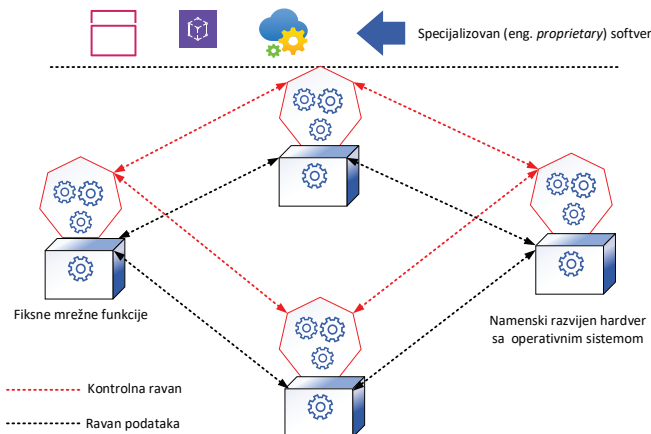
Dakle, lako se može zaključiti, da je sloj jezgra u računarskim mrežama sa tradicionalnom arhitekturom bitan za obezbeđivanje:

- potrebnog nivoa performansi – posebno je važno, da se u velikim mrežama sa mnogo korisnika i uređaja minimizuje kašnjenja u prenosu podataka i time direktno utiče na kvalitet mrežnih usluga,
- skalabilnosti i proširivosti – mreža treba da raste bez pojave uskih grla i bez potrebe za značajnim promenama u strukturi jezgra,
- visoke dostupnosti i otpornosti na greške – redundantne veze i uređaji u ovom sloju, treba da omoguće automatski preusmeravanje saobraćaja u slučaju problema, što znači da mreža mora nastaviti sa radom i u slučaju kvara na jednom delu infrastrukture.

1.3. KARAKTERISTIKE TROSLOJNE ARHITEKTURE

Pojedine karakteristike računarskih mreža sa tradicionalnom arhitekturom, danas predstavljaju ozbiljnu kočnicu, u pokušaju, da se na nove zahteve korisnika odgovori većom fleksibilnošću, efikasnošću i sposobnošću za brzim promenama. Jedan od razloga je činjenica, da je mrežna topologija u tradicionalnim mrežama fiksna ili unapred definisana. Svaka promena u mreži, kao što je dodavanje novih uređaja ili promene u aktuelnim konekcijama, zahteva statičku (ručnu) konfiguraciju uređaja, kao što su ruteri, svičevi i pristupne tačke i često izaziva prekid u radu. Tradicionalni uređaji poseduju lokalizovanu inteligenciju, što znači da nezavisno donose odluke, kako će usmeravati saobraćaj i to na osnovu dostupnih informacija i tabela rutiranja. Ovakva spora i sklona greškama konfiguracija, pogotovo u velikim mrežama komplikuje upravljanje saobraćajem. Dodatno treba naglasiti, da je upravljanje saobraćajem ionako veoma komplikovano, jer se koristi veliki broj različitih protokola za rutiranje saobraćaja (OSPF, BGP, IS-IS, RIP i dr.), od kojih su neki vlasničke (eng. *proprietary*) prirode, što otvara problem interoperabilnosti opreme.

Model prikazan na slici 5. jasno ukazuje na problem interoperabilnosti, imajući u vidu da su brojne funkcionalnosti u računarskim mrežama sa tradicionalnom arhitekturom, definisane u specijalizovanim tj. vlasničkim softverima. Ovi softveri se često pokreću na posebno razvijenom hardveru. Ovako integrisani hardver i softver, kao jedinstvena celina, predstavljaju svojstvo određenog proizvođača [1]. Proizvođači dizajniraju i prave svoju opremu, rukovodeći se generičkim skupom zahteva i nudeći funkcionalnost koja kombinuje specifičan hardver i softver. Oni žele da zaštite svoju intelektualnu svojinu, odnosno svoje vlasništvo, što znači da ga klijenti mogu koristiti samo pod unapred definisanim uslovima. Međutim, ovakav pristup ograničava prava korisnika da modifikuju, distribuiraju ili pristupe izvornom kodu softvera. Time se umanjuje fleksibilnost i mogućnost racionalnog korišćenja raspoloživih resursa. Korisnici postaju zavisni od proizvođača, što proizvodi negativne efekte sa finansijske i tehničke strane (opremu moraju da kupuju samo od određenog proizvođača) [2].



Slika 5. Karakteristike računarskih mreža sa tradicionalnom arhitekturom

U realnim mrežnim okruženjima sa tradicionalnom organizacijom i brojnim vlasničkim rešenjima, problem interoperabilnosti je samo jedan od problema koji se mora rešavati. Drugi važan problem je nemogućnost, da se na zahtev obezbedi veći propusni opseg ili neki drugi mrežni resurs. Sa eksponencijalnim rastom broja uređaja u pravom eksploziji novih i inteligentnijih usluga (npr. video i mobilne aplikacije), potrebno je obezbediti značajno veću količinu resursa i agilnost u radu mreže. Mreže sa tradicionalnom arhitekturom nisu u mogućnosti da odgovore na sve zahteve iz mreže, pa se provajderima servisa smanjuje operativna efikasnost. Oni nisu u stanju da prevaziđu probleme, koji nastaju kao posledica krutosti arhitekture, koju je teško modifikovati i prilagoditi sve heterogenijim zahtevima korisnika. Nizak nivo fleksibilnosti i zavisnost mrežnih funkcija od fizičkog hardvera ograničavaju skalabilnost i otežavaju saradnju između mrežnih uređaja sa fiksnom funkcijom. U nekim slučajevima, kvar na nekom od tih mrežnih uređaja, zbog rigidne konfiguracije i centralizacije upravljanja, može da prouzrokuje kaskadni efekat, tj. lančane probleme u drugim delovima mreže, što može dovesti do širenja problema i potencijalnog kolapsa celog sistema. Ograničena fleksibilnost umanjuje sposobnost da se efikasno i dinamički odgovori ne samo na promene u opterećenju, već i na bilo kakve promene u mreži.

U praksi, da bi se instalirao novi hardver potrebno je obezbediti određenu količinu električne energije i prostor za njegov smeštaj, što može biti problem i dodatni trošak. Ograničenja postoje čak i u pogledu softvera, jer tradicionalni mrežni uređaji ne prate svaku promenu (npr. u smislu broja ruta koje treba kreirati). Sve ovo ukazuje da provajderi servisa imaju ograničene mogućnosti za nadogradnju i proširenje prilično heterogene mrežne infrastrukture. Nameće se pitanje: kako da se upravlja infrastrukturom kada ne postoji jedinstven interfejs za upravljanje? Na primer, za potrebe merenja saobraćaja i opterećenja računarskih resursa [3] i dalje se koriste alati sa standardizovanim protokolima, kao što su SNMP (eng. *Simple Network Management Protocol*), *NetFlow* i *syslog*). To otvara novo pitanje: šta raditi u situacijama, kada je potrebno treba pratiti parametre, koji su specifični za opremu određenih proizvođača? Generalno, problem nedostatka jedinstvenog interfejsa za upravljanje je čest u tradicionalnim mrežama, jer svaki mrežni uređaj (npt. ruter, svič ili *firewall*) ima sopstveni interfejs za konfiguraciju i upravljanje, što otežava centralizovano upravljanje i koordinaciju u mreži. Administratori mreže moraju poznavati različite alate i interfejse kako bi upravljali uređajima različitih proizvođača (na primer, CLI komande za uređaje određenog proizvođača, *web* interfejse i sl.). Sve ovo ukazuje na složenost problema, jer je za obuku osoblja potrebno vreme, a u nekim situacijama se zahteva zapošljavanje visoko obučenog osoblja sposobnog da održava opremu različitih proizvođača. Jasno je da dodatna obuka ili zapošljavanje obučenog osoblja predstavlja dodatni trošak i usporava donošenje odluka o nabavci i ugradnji opreme drugog proizvođača. Sve ovo potencijalno vodi ka *vendor lock-in* problemu, tj. „zaključavanju“ provajdera sa opremom jednog proizvođača, što je potencijalno veoma opasno. Da bi se odgovorilo na zahteve korisnika, potrebno je izvršiti promene u kapacitetima, ali one u ovakvim okolnostima zahtevaju dodatna finansijska sredstva.

Činjenica je, da u mrežama sa tradicionalnom arhitekturom, administratori moraju ručno da upravljaju (konfigurišu) većim brojem uređaja kroz različite interfejse, čime se povećava verovatnoća ljudskih grešaka. Greška u konfiguraciji jednog uređaja može izazvati probleme u mrežnom saobraćaju ili dovesti do propusta u pogledu bezbednosti.

Štaviše, neusklađenost u konfiguraciji mrežnih uređaja (uređaji u mreži često imaju različite konfiguracije ili mrežne politike), može dovesti do problema sa kompatibilnošću i bezbednosnim propustima. Ovde se mora imati u vidu, da raznovrsnost interfejsa komplikuje automatizaciju mrežnih zadataka (različite uređaje nije lako integrisati u jedinstveni sistem za upravljanje). Zato u slučaju problema u mreži, mrežni administratori moraju pojedinačno da pristupaju svakom uređaju, da bi identifikovali i rešili problem. Ovo može produžiti vreme reakcije i smanjiti efikasnost upravljanja mrežom.

1.4. PREDNOSTI I IZAZOVI

Troslojna mrežna arhitektura poseduje brojne prednosti, ali sa sobom nosi i ozbiljne izazove u današnjem kontekstu računarskih komunikacija. Ti izazovi su vezani za brzinu prilagođavanja i fleksibilnost u realizaciji novih zahteva u mreži, što je trenutno dominantan trend, koji u najvećoj meri utiče na mrežne arhitekture da se opredele za određeni dizajn mreže. U dinamičnim i modernim okruženjima gde su potrebne brze promene i visoka prilagodljivost u mrežnoj infrastrukturi, problemi kao što su kompleksnost upravljanja, troškovi izgradnje mrežne infrastrukture i ograničena fleksibilnost troslojne arhitekture, postaju sve izraženiji. To je jedan od razloga, zbog kojih se sve više istražuju mogućnosti primene drugih rešenja kao što je tehnologija softverski definisanog umrežavanja (eng. *Software-Defined Networks* – SDN). Međutim, uprkos činjenici, da ta nova rešenja omogućavaju veću automatizaciju i prilagodljivost, ona često zadržavaju i neke elemente troslojne arhitekture kao osnovu.

1.4.1. Ključne prednosti

Uprkos prethodnopomenutim izazovima, prednosti koje troslojna organizacija mreže poseduje u pogledu skalabilnosti, bezbednosti i pouzdanosti, čine je još uvek relevantnom za neka okruženja (pre svega korporativne mreže, kampuse, a jednim delom i data centre) [4]. Te prednosti se pre svega odnose na:

- Skalabilnost,
- mogućnost segmentacije i primene adekvatnih bezbednosnih rešenja,
- redundansu i visoku dostupnost,
- jednostavnost upravljanja i održavanja u nekim mrežama i
- efikasno upravljanje saobraćajem u određenim okruženjima.

Skalabilnost je jedna od najvećih prednosti troslojne arhitekture, jer omogućava dodavanje novih uređaja, pristupnih tačaka i proširenje mreže bez potrebe za velikim promenama u već postojećoj infrastrukturi. Na primer, korporativna mreža može lako da dodaje nove *switch*-eve u pristupnoj mreži i povezuje nove sajtove ili radne stanice, jer distributivni i sloj jezgra ostaju nepromenjeni. S druge strane, mogućnost logičke segmentacije mreže korišćenjem VLAN-ova i drugih mehanizama, omogućava odvajanje

saobraćaja različitih poslovnih funkcija, aplikacija ili tipova korisnika, što doprinosi boljoj kontroli saobraćaja i bezbednosti. Ovo je jako korisno za kampus mreže univerziteta, gde se segmentacija mreže na različite organizacione celine, vrši u pristupnom sloju, dok se kontrola pristupa i bezbednosne politike primenjuju u distribucionom sloju.

Troslojna organizacija mreže podrazumeva i postojanje višestrukih redundantnih veza, posebno u distribucionom i sloju jezgra. Time se obezbeđuje kontinuitet rada, čak i u slučaju greške i prekida u radu određenih uređaja, jer se saobraćaj preusmerava preko alternativnih ruta. Smanjenje rizika od prekida saobraćaja je jako važno u *data* centrima i kritičnim poslovnim okruženjima. Sa dobro definisanim funkcijama različitih slojeva (pristup korisnicima, agregacija, rutiranje), mrežnim administratorima se olakšava identifikacija i rešavanje problema. Ova prednost se može iskoristiti u korporativnom okruženju, gde pristupni sloj može lako da se nadogradi ili zameni bez značajnijeg uticaja na sloj jezgra, čime se omogućava lakše održavanje i upravljanje. Dakle, u ovakvim okruženjima, gde distributivni sloj preuzima zadatke filtriranja saobraćaja i primene mrežnih politika kojima se garantuje QoS, a sloj jezgro osigurava brz transport podataka, troslojna arhitektura omogućava efikasno upravljanje i kontrolu saobraćaja,

1.4.2. Glavni izazovi

Tradicionalna troslojna arhitektura ima čvrst temelj u mrežnom dizajnu zbog svoje pouzdanosti i organizovanog pristupa mrežnoj segmentaciji. Međutim, nedostatak fleksibilnosti, visoki troškovi održavanja i poteškoće sa brzim prilagođavanjem, čine je manje pogodnom za savremene mrežne potrebe, gde su brzina, automatizacija i dinamičnost ključni faktori [5]. U tom smislu potrebno je ukazati na sledeće izazove sa kojima se susreću mreže sa tradicionalnom arhitekturom:

- *ograničena fleksibilnost i sporost prilagođavanja* – statička konfiguracija i ručno upravljanje mrežnim uređajima predstavljaju dugotrajan proces, koji otežava brze promene i prilagođavanje (npr. kreiranje novih VLAN-ova ili promene u rutiranju zahtevaju intervenciju mrežnih administratora). U okruženjima gde je brza implementacija novih usluga ključna, poput *cloud* okruženja ili *data* centara sa dinamičkim radnim opterećenjima, ova sporost značajno usporava rad i povećava troškove.
- *kompleksnost upravljanja i održavanja* – postojanje velikog broja mrežnih uređaja, kao što su ruteri i *switch*-evi, koji obavljaju specifične funkcije na svakom sloju, zahteva pažljivu koordinaciju i ozbiljno znanje o konfiguracijama. U velikim mrežama sa složenim upravljanjem, posebno kada se primenjuju politike bezbednosti u mreži ili rutiranje, to može dovesti do ljudskih grešaka i povećanja operativnih troškova.
- *troškovi implementacije i održavanja* – povećanje broja mrežnih uređaja i implementacija specijalizovane opreme (npr. ruteri sa visokim performansama u jezgru) povećava složenost infrastrukture. Troškovi inicijalne implementacije, održavanja i potencijalnih proširenja, u manjim preduzećima i organizacijama su ozbiljna prepreka u realizaciji njihovih potreba.

- *veće kašnjenje i manja efikasnost* – sa tri sloja između korisnika i resursa, paketi prolaze kroz više skokova (eng. *hop*). Svaki skok može da unese dodatno kašnjenje i smanji ukupnu efikasnost mreže, što je ozbiljan problem za *real-time* i *data centre*, jer usporava komunikaciju između servera i sistema za skladištenje podataka.
- *ograničena automatizacija i agilnost* – teškoće u integraciji modernih mrežnih koncepata (npr. SDN ili virtuelizacija infrastrukture), ograničavaju mogućnost automatskog upravljanja saobraćajem i resursima i značajno usporavaju implementaciju novih politika ili pristupa mreži.
- *zavisnost od fizičke infrastrukture* – troslojna arhitektura je tradicionalno dizajnirana za rad sa fizičkim uređajima i povezivanjem putem fizičkih veza, što u današnjem kontekstu utiče na manju fleksibilnost u mreži. U modernim *data* centrima, serveri i mrežni resursi se često premeštaju ili virtuelizuju, pa rigidnost mrežne infrastrukture usporava rad i otežava migracije radnih opterećenja između različitih fizičkih lokacija.
- *nedostatak dinamičkog rutiranja i optimizacije* – oslanjanje na klasične protokole za rutiranje (OSPF, BGP i dr.), koji su efikasni ali ne i uvek optimalni u kontekstu dinamičkog upravljanja mrežnim resursima, stvara problem, koji se ogleda u neoptimalnom rutiranju i nepotrebnom opterećivanju mrežnih veza (linkova).
- *problem skalabilnosti u horizontalnom smislu* – troslojna arhitektura je skalabilna u vertikalnom smislu (dodavanje novih slojeva ili uređaja unutar postojećih slojeva), ali u horizontalnom smislu, gde se zahtevaju promene između slojeva, može biti izazovna. U situacijama gde je potrebno brzo proširenje mrežnih kapaciteta ili povezivanje više *data* centara na različitim geografskim lokacijama, tradicionalni troslojni pristup može zahtevati kompleksne rekonfiguracije i dodavanje dodatnih uređaja, što povećava vreme i troškove.

1.5. TROSLOJNA MREŽNA ARHITEKTURA U REALNIM OKRUŽENJIMA

Uprkos napretku u mrežnim tehnologijama kao što su SDN i *cloud computing*, troslojna mrežna arhitektura se i dalje koristi u mnogim tradicionalnim i savremenim okruženjima zbog svoje stabilnosti, skalabilnosti i lakoće upravljanja [4]. Danas se ova arhitektura primenjuje u sledećim scenarijima:

- a) Korporativne mreže velikih preduzeća (npr. banke, telekom provajderi, velike maloprodajne mreže i multinacionalne kompanije) – velika preduzeća sa centralizovanim kancelarijama i više udaljenih sajtova koriste troslojnu arhitekturu, jer im omogućava:
 - stabilno i sigurno povezivanje zaposlenih i IT resursa,
 - segmentaciju mreže putem VLAN-ova, što je ključno za kontrolu pristupa i bezbednost različitih delova mreže,
 - skalabilnost dodavanjem novih uređaja ili proširenjem poslovanja bez potrebe za velikim promenama u mrežnoj infrastrukturi i
 - redundansu i otpornost tj. neprekidan rad u slučaju kvarova, što je od ključne važnosti za poslovni kontinuitet.
- b) Kampus mreže (univerziteti i obrazovne institucije) – ova arhitektura se koristi jer je neophodno povezati studente, fakultetsko osoblje i različite akademske resurse i to kroz:
 - centralizovano upravljanje velikim brojem korisnika i uređaja kroz pristupni sloj,
 - agregaciju saobraćaja i primenu politika za različite akademske zgrade na distribucionom sloju, čime se omogućava efikasno upravljanje saobraćajem i
 - stabilan i pouzdan prenos podataka između zgrada, servera i internet konekcija implementacijom mreže čije jezgro pruža visokopropusne i brze veze između različitih delova kampusa.
- c) *Data* centri sa tradicionalnom infrastrukturuom (npr. *data* centri koji hostuju aplikacije za preduzeća, virtuelne privatne servere (VPS) i neke privatne *cloud* implementacije) – jedan deo ovih entiteta i dalje koristi troslojnu arhitekturu, jer obezbeđuje:
 - jasnu segmentaciju resursa unutar data centra (npr. odvajanje servera različitih klijenata, aplikacija ili servisa),
 - primenu bezbednosni politika i agregaciju saobraćaja iz pristupnog sloja i visoke brzine prenosa između različitih servera,
 - redundansu i visoka dostupnost u radu usluga i aplikacija, što je kritično za data centre.

- d) Telekom i kablovski provajderi – pružaju internet usluge i rešenja za povezivanje velikom broju korisnika i često koriste varijacije troslojne arhitekture u svojim mrežama jer omogućavaju:
 - agregaciju i distribuciju korisničkog saobraćaja sa više lokacija preko pristupnih slojeva ka jezgru mreže,
 - implementaciju QoS politika, što je ključno za prioritizaciju saobraćaja u mrežama,
 - brz prenos podataka između različitih geografskih lokacija, povezivanje sa drugim mrežama i podršku za BGP i druge protokole za dinamičko rutiranje.
- e) Hibridne mreže sa delimičnom SDN integracijom – koriste se zajedno sa SDN rešenjima kako bi se postigao balans između fleksibilnosti i stabilnosti. Na primer, velike kompanije koje žele da uvedu elemente *cloud* i tehnologije virtuelizacije u svoje mreže, vrše postepen prelaz, kako bi se zadržala stabilnost postojećih mreža.

ZAKLJUČAK

Troslojna mrežna arhitektura se danas koristi u okruženjima gde su stabilnost, skalabilnost i upravljivost ključni faktori. Iako savremene tehnologije kao što su SDN, *cloud* i virtuelizacija donose veću fleksibilnost i automatizaciju, mnoge organizacije i dalje zadržavaju troslojni pristup, posebno u situacijama gde je ključna dugoročna pouzdanost u radu. Upotreba ove arhitekture omogućava organizacijama da iskoriste već postojeću infrastrukturu, dok polako prelaze na modernije mrežne modele, što čini troslojnu arhitekturu i dalje relevantnom u brojnim mrežnim okruženjima. Međutim, prelazak na SDN tehnologiju kao novu mrežnu paradigmu, ima za cilj, da se kroz veći nivo programabilnosti, mreža što više približi potrebama savremenog društva i korisnika. Centralizovano upravljanje i kontrola, veća fleksibilnost i agilnost, visok stepen automatizacije i povećana skalabilnost, veći nivo bezbednosti i smanjenje operativnih troškova, samo su neke od prednosti koje donosi SDN. Sve to nameće potrebu da se menja dizajn mreže, ali na dobro osmišljen način.



POGLAVLJE 2



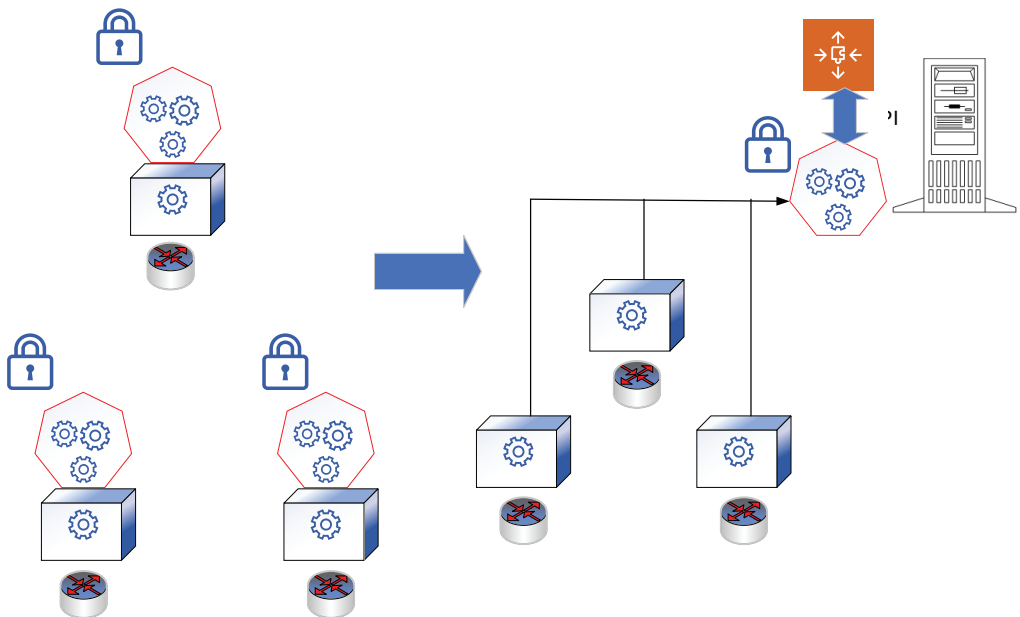
KONCEPT SOFVERSKI DEFINISANOG UMREŽAVANJA

2. KONCEPT SOFVERSKI DEFINISANOG UMREŽAVANJA

2.1. UVOD U SOFVERSKI DEFINISANO UMREŽAVANJE

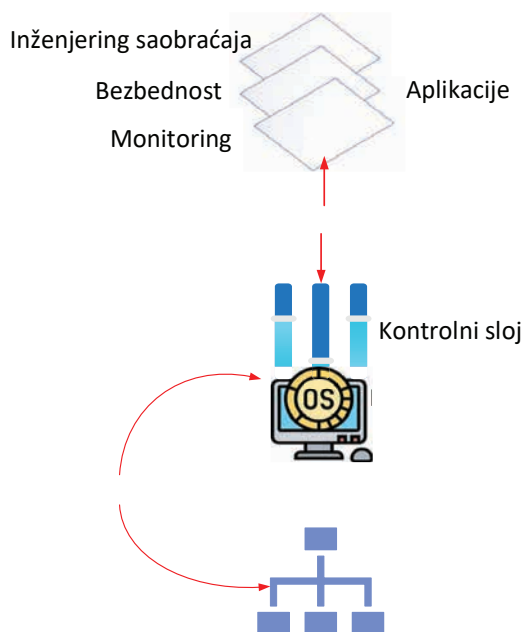
Softverski definisano umrežavanje predstavlja novu paradigmu u oblasti umrežavanja, koja se zasniva na modernom pristupu u izradi dizajna i upravljanju računarskim mrežama. Njen koncept se zasniva na razdvajanju kontrolne ravni (eng. *control plane*) od mrežne infrastrukture tj. ravni podataka (eng. *data plane*), koja ostaje zadužena za komutaciju i prosleđivanje paketa. Time, mrežni administratori dobijaju mogućnost, da umesto tradicionalnih hardverski orijentisanih metoda, koriste softverske aplikacije za centralizovano upravljanje mrežnim resursima i dinamičku realizaciju heterogenih zahteva u mreži.

Razdvajanje kontrolne od ravni podataka, uvodi suštinske razlike u odnosu na tradicionalno umrežavanje, gde je u svakom mrežnom uređaju (ruter), ugrađena i kontrolna i ravan podataka (Slika 6.) [6]. Praksa je pokazala, da koncept u kome mrežni uređaji sami donose odluke o rutiranju i prosleđivanju paketa, nije adekvatno rešenje za upravljanje složenim mrežnim okruženjim. To se pre svega odnosi na velike mreže, kao što su *data* centri ili mreže telekom provajdera, gde nastaju izazovi, vezani za potrebu da se obezbedi fleksibilniji pristup u rešavanju potreba korisnika.



Slika 6. Koncept razdvajanja kontrolne od ravni podataka [6]

Konceptualni model, kojim se kontrolna ravan (odlučuje o putanji paketa) razdvaja od ravni podataka (obavlja prosleđivanje paketa), doprinosi realizaciji nove, SDN arhitekture, koju krasi visok nivo programabilnosti. Kompletne mrežne logike se centralizuje na jednom mestu u mreži, tj. na SDN kontroleru. Ovaj softverski entitet komunicira sa mrežnim uređajima koristeći protokole, kao što je *OpenFlow* protokol. Lokalizacijom funkcija kontrolnog sloja na jednom mestu u mreži, omogućen je globalan pogled na mrežnu infrastrukturu, pa SDN kontroler deluje kao "mozak" mreže. On obezbeđuje programabilni pristup mrežnoj infrastrukturi putem API-ja (npr. REST API) (slika 7.). Drugim rečima, centralizovana kontrolna ravan upravlja mrežnim resursima i donosi odluke o rutiranju saobraćaja, dok fizički ili virtuelni uređaji izvršavaju odluke kontrolnog sloja i prosleđuju mrežni saobraćaj.



Slika 7. Uloga mrežnog softvera u SDN arhitekturi

Suštinski posmatrano, implementacija SDN tehnologije u računarske mreže, donosi brojne prednosti, kao što su [7]:

- *centralizovano upravljanje* – mogućnost da se mrežom upravlja iz jednog mesta korišćenjem kontrolera,
- *programabilnost* – mrežne operacije se mogu automatizovati i time brže implementirati nove politike u mreži,
- *smanjenje kompleksnosti upravljanja* – jednostavnije upravljanje mrežnim resursima, posebno u velikim mrežama, smanjuje potrebu za pojedinačnom konfiguracijom svakog uređaja u mreži.

Potrebno je istaći, da SDN tehnologija donosi potpuno nov način razmišljanja o mrežama. On se suštinski razlikuje od tradicionalnog pristupa, koji se oslanja na specijalizovani hardver. Time se u kontekstu savremenih IT trendova, kao što su implementacija *cloud computing*-a i izgradnja velikih *data* centara, omogućava brže i dinamičnije prilagođavanje mrežnih resursa sve većim i heterogenijim zahtevima korisnika i njihovih aplikacija. Uvođenjem SDN-a, preduzeća dobijaju mogućnost, da optimizuju troškove i resurse, je je obezbeđena veća agilnost i efikasnost u upravljanju mrežama. Time je SDN postao ključni element modernih mrežnih arhitektura, pružajući solidnu osnovu da se učini sledeći korak u evoluciji mrežnih tehnologija.

2.2. ARHITEKTURA SDN MREŽA

Kao što je prethodno pomenuto, SDN arhitektura predstavlja novu paradigmu u umrežavanju, koja nudi fleksibilniju i skalabilniju mrežnu infrastrukturu. Time, administratori mreže i brojne aplikacije počinju da dinamički upravljaju mrežnim resursima, čineći mreže bržim, pouzdanijim i jednostavnijim za upravljanje [8]. Sama implementacija SDN koncepta, u računarske mreže unosi znatno veći nivo apstrakcije (Slika 8.), jer se:

- korisničke aplikacije odvajaju od mrežnih elemenata, a mrežni uređaji se apstrahuju u odnosu na upravljački sloj
- kontrolna ravan razdvaja od ravni za prosleđivanje paketa i time omogućava dinamička alokacija resursa prema potrebama korisnika i
- omogućava fleksibilna apstrakcija tokova paketa, korišćenjem novog softverskog resursa, koji se naziva tabela tokova (eng. *flow table*).



Slika 8. Apstrakcija u SDN mrežama

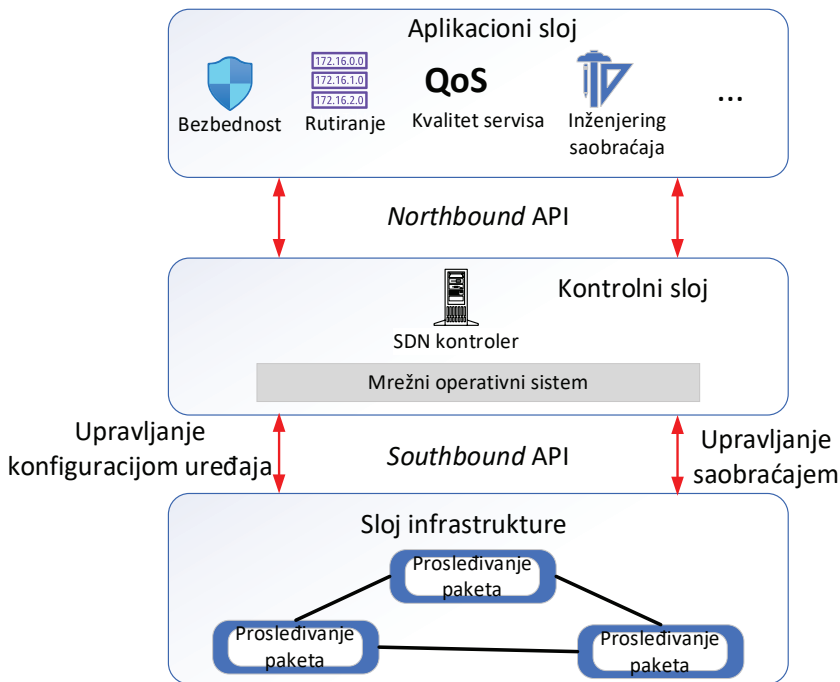
Ono što karakteriše mrežni softver u mrežama gde je primenjena SDN tehnologija, odnosi se na jasno definisanu podelu funkcionalnosti, između četiri različita sloja SDN arhitekture:

- sloja upravljanja (eng. *management plane*),
- sloja kontrole (eng. *control plane*),
- sloja servisa (eng. *services plane*) i
- sloja prosleđivanja (eng. *forwarding plane*).

Ova podela omogućava bolju modularnost, fleksibilnost i mogućnost modifikacije ponašanja mrežnih uređaja. Uprkos odvajanju kontrolne ravni od ravni za prosleđivanje podataka, mrežni softver se i dalje može organizovati u različite slojeve i module. Jasno je, da razdvajanje kontrolne i ravni podataka predstavlja ključan mehanizam. Kontrolna ravan zadržava logiku upravljanja i donošenje odluka u vezi sa rutiranjem i komutiranjem paketa u mreži. Lokalizovana je u SDN kontroleru, koji može posedovati centralizovanu, ali i distribuiranu arhitekturu. Za izvršenje odluka SDN kontrolera su nadležni mrežni uređaji (*switch*-evi i ruteri), koji fizički prosleđuju pakete kroz mrežu. Oni nemaju autonomiju u donošenju odluka već primaju instrukcije iz kontrolnog sloja kako da se ponašaju. Dakle, ovaj način razdvajanja omogućava da se kontrola i donošenje odluka koncentriše na jednom mestu (u kontroleru), dok se mrežni uređaji koji čine infrastrukturu mreže, fokusiraju na efikasno prosleđivanje saobraćaja.

Drugi važan mehanizam u razdvajanju mrežnog sloja je vertikalno razdvajanje. Primenom ovog mehanizma u arhitekturi SDN mreže se kreiraju sledeći slojevi (Slika 9), koji međusobno komuniciraju preko različitih API-a (eng. *Application Programming Interface*) [6]:

- infrastrukturni sloj (eng. *Infrastructure Layer*) – upravlja osnovnom mrežnom infrastrukturom i uređajima koji fizički prenose saobraćaj (obuhvata *switch*-eve, rutere, i druge mrežne uređaje koji funkcionišu kao izvršioci instrukcija sa kontrolnog sloja)
- kontrolni sloj (eng. *Control Layer*) – upravlja mrežnom logikom (SDN kontroler) i donosi odluke o rutiranju saobraćaja i upravlja mrežnim resursima na osnovu podataka iz infrastrukturnog sloja
- aplikacioni sloj (eng. *Application Layer*) – sastoji se od aplikacija koje koriste mrežu i komuniciraju sa kontrolnim slojem preko API-a. Ove aplikacije mogu postavljati specifične politike za upravljanje saobraćajem, bezbednost, balansiranje opterećenja i dr.



Slika 9. Slojevita arhitektura SDN mreža

Upotreba API-a i ima važnu ulogu u razdvajanju različitih funkcionalnosti mrežnog softvera softvera, čime se obezbeđuje modularnost i fleksibilnost u razvoju i integraciji novih aplikacija i mrežnih servisa. Za komunikaciju između kontrolnog i infrastrukturnog sloja je nadležan *southbound* API. Najpoznatiji *southbound* API je *OpenFlow* protokol, koji omogućava kontroleru da šalje uputstva mrežnim uređajima o tome kako treba da prosleđuju saobraćaj. *Northbound* API-ji povezuje kontrolni sloj sa aplikacionim slojem, omogućavajući aplikacijama da komuniciraju sa kontrolerom i da od njega zahtevaju informacije o stanju mreže ili pak da definišu specifične mrežne politike.

Važno je istaći, da se unutar samog kontrolnog sloja, često koristi modularni dizajn, kako bi se dodatno razdvojile različite funkcionalnosti. Upravo SDN kontroleri (npr. *OpenDaylight* ili *ONOS* kontroler), koji omogućavaju modularan pristup, administratorima daju mogućnost, da unutar kontrolnog sloja lako dodaju ili uklanjaju funkcionalnosti, bez potrebe za promenom celokupnog sistema. Za obavljanje specifičnih zadataka, kreiraju se različiti moduli, kao što su:

- modul za rutiranje – upravlja dinamičkim rutiranjem unutar mreže,
- modul za balansiranje opterećenja – distribuira saobraćaj između servera kako bi se smanjila latencija i povećala efikasnost
- modul za primenu politike bezbednosti – implementira pravila koja se odnose na pristup mrežnim resursima, filtriranje saobraćaja i detekciju pretnji.

Još jedan važan mehanizam koji se primenjuje je u SDN mrežama je razdvajanje virtuelnih mreža i fizičke infrastrukture. Naime, SDN omogućava stvaranje virtuelnih mreža koje su logički odvojene od fizičke mrežne infrastrukture. Ove virtuelne mreže koriste softver, kojim se definišu topologije, rutiranje i bezbednosne politike, nezavisno od stvarnih fizičkih uređaja. Time se korsićima, preduzećima i različitim organizacijama, daje mogućnost da koriste istu fizičku mrežnu infrastrukturu. Na deljenoj infrastrukturi, SDN kontroler omogućava da pojedini korisnici imaju sopstvenu virtuelnu mrežu, sa specifičnim pravilima, ali uz međusobnu izolaciju i garancije za bezbednost.

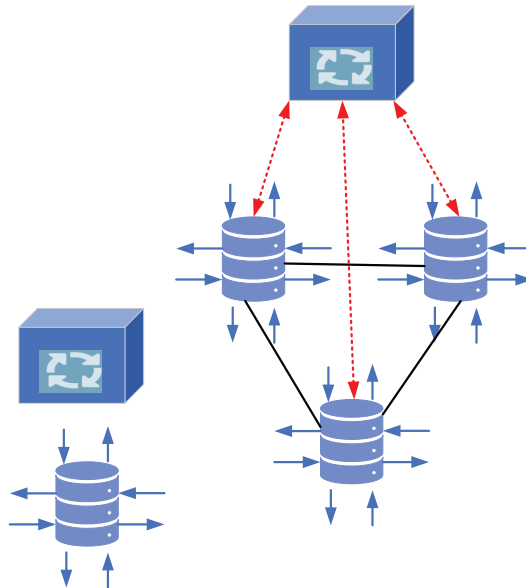
Razdvajanjem mrežnog softvera, SDN omogućava modernim mrežama da budu programabilne, fleksibilne i skalabilne [9]. Podela zadataka između kontrolnog, aplikacionog i infrastrukturnog sloja, kao i upotreba API-ja za komunikaciju između tih slojeva, omogućava bržu adaptaciju na nove i sve zahtevnije potrebe korisnika i aplikacija. Primenom ovakve arhitekture se transformiše način na koji se mreže dizajniraju i upravljaju, čineći ih pogodnijim za dinamična okruženja kao što su *cloud* servisi i *data* centri. Dakle, prednosti razdvajanja mrežnog softvera se ogledaju u:

- *povećanoj fleksibilnosti* – administratori lako dodaju nove aplikacije ili funkcionalnosti bez potrebe za nadogradnjom hardvera ili rekonfigurisanjem celokupne mreže
- *bržoj adaptaciji na zahteve iz mreže* – može se brzo odgovoriti na promene u saobraćaju, pretnje ili aplikacione zahteve kroz pprimenu softverskih pravila na nivou kontrolera ili aplikacija
- *smanjenju troškova* – korišćenjem standardizovanih API-ja i modularnog dizajna, smanjuju se troškovi vezani za korišćenje specijalizovanog mrežnog hardvera i pojednostavljuje upravljanje mrežom
- *efikasnijem upravljanju* – centralizovana kontrola i globalan uvid u stanje mreže, omogućava preciznije upravljanje resursima i bolje praćenje performansi.

2.3. CENTRALIZACIJA MREŽNE LOGIKE

Arhitektura SDN-a funkcioniše kroz jasno definisanu podelu zadataka između infrastrukturnog sloja, kontrolnog sloja, i aplikacionog sloja, uz korišćenje standardizovanih API-ja za međusobnu komunikaciju. Ova podela omogućava centralizovanu kontrolu i programabilnost, što mrežu čini fleksibilnijom, bržom, i lakšom za upravljanje. Ključna karakteristika SDN umrežavanje je centralizacija mrežne logike, koja se postiže odvajanjem kontrolnog od infrastrukturnog sloja. To je u potpunoj suprotnosti od tradicionalne arhitekture mreže, koju karakteriše distribuirana kontrolna ravan, u kojoj svaki mrežni uređaj (ruter ili *switch*) poseduje sopstvenu logiku za donošenje odluka o rutiranju i prosleđivanju paketa.

Centralizacija mrežne logike je novi pristup, u kojem se sva logika prenosi na SDN kontroler, koji ima globalni uvid u mrežu, uključujući informacije o topologiji, stanju linkova, statusima uređaja i trenutnim mrežnim opterećenjima. Kontroler koristi te informacije kako bi optimizovao mrežno ponašanje i primenio mrežne politike u skladu sa potrebama korisnika i aplikacija. S druge strane, mrežni uređaji postaju jednostavniji i "manje inteligentni". Njihov osnovni zadatak je da sprovedu „naredbe“ kontrolera (Slika 10).



Slika 10. Centralizacija mrežne logike

Centralizovani pristup u organizaciji mrežne logike, donosi nekoliko značajnih prednosti:

- *globalni uvid u stanje mreže* – kontroler ima pregled stanja celokupne mreže (uključujući sve uređaje i linkove) i mogućnost, da u realnom vremenu donosi optimalne odluke, što olakšava rešavanje problema, praćenje performansi, i optimizaciju resursa. To je teško postići u mrežama sa distribuiranom kontrolnom ravni (oslanja se na lokalne informacije od svakog uređaja).
- *dinamičko i automatsko upravljanje* – omogućava bržu reakciju na promene u mreži, kao što su fluktuacije saobraćaja ili kvarovi na linkovima, čime smanjuje potrebu za manuelnom konfiguracijom mrežnih uređaja i mogućnost ljudskih grešaka (npr. kontroler automatski preusmerava saobraćaj sa preopterećenog na manje opterećen link).
- *mogućnost implementacije naprednih mrežnih politika* – kontroler omogućava implementaciju složenih mrežnih politika, koje su specifične za određene aplikacije ili servise. Na primer, može da optimizuje QoS za aplikacije sa zahtevima za određenim propusnim opsegom ili da kreira virtualne mreže sa posebnim pravilima za sigurnost.
- *efikasnije upravljanje resursima* – centralizovana kontrola pruža mogućnost boljeg korišćenja mrežnih resursa, jer kontroler može da balansira opterećenje između različitih linkova i uređaja. On je sposoban, da detektuje neiskorišćene resurse i preusmeri saobraćaj, kako bi se mreža koristila na efikasniji način.

Iako centralizacija donosi brojne prednosti, ona takođe donosi određene izazove:

- *centralizovani kontroler može postati potencijalna tačka kvara* (eng. *Single Point of Failure*) – centralizovani kontroler može postati tačka koja može ugroziti celu mrežu u slučaju kvara ili napada, jer mrežni uređaji gube uputstva za prosleđivanje paketa, što može uzrokovati probleme u saobraćaju. Ovaj izazov se često rešava implementacijom redundantnih kontrolera ili upotrebom distribuiranih kontrolera kako bi se obezbedila otpornost sistema.
- *skalabilnost* – sa rastom mreže, povećava se količina informacija koje kontroler treba da obradi i na osnovu njih donese odluke, što postaje izazov, pre svega u velikim mrežama (*data* centri ili mreže telekom provajdera). On se može rešiti, upotrebom hijerarhijskih ili distribuiranih SDN kontrolera, koji mogu da dele teret upravljanja mrežom.
- *latencija* – komunikacija između kontrolera i mrežnih uređaja može povećati latenciju, u slučajevima, gde je kontroler fizički udaljen od mrežnih uređaja. Zato se kontroleri trebaju implementirati bliže uređajima ili se mora primeniti hijerarhijska arhitektura sa više kontrolera, koji su sposobni da brže donose lokalne odluke.

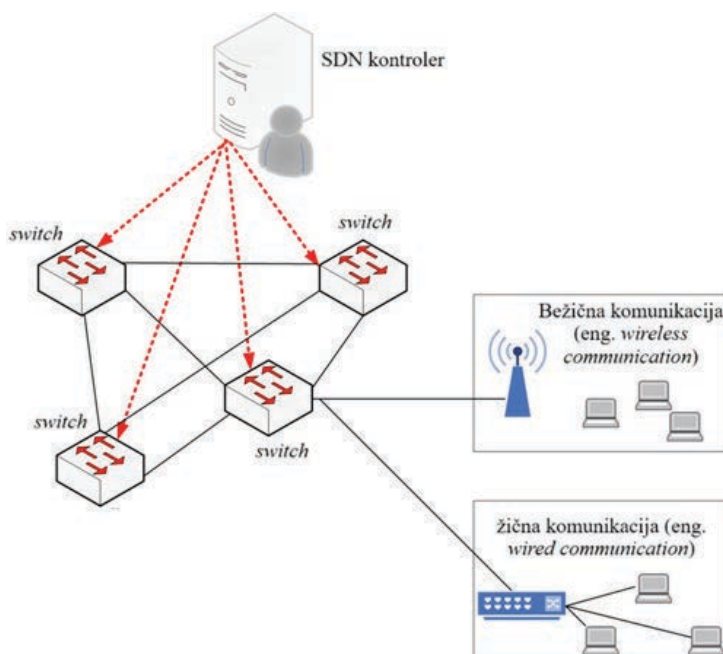
Upravljanje velikim *data* centrima, kao što su oni koje koriste Google ili Facebook, predstavlja praktičan primer centralizacije mrežne logike. Primenom SDN tehnologije, optimizovano je rutiranje između stotina ili hiljada servera u *data* centru. Obezbeđeno je efikasno preusmeravanje saobraćaja, sa ciljem da se izbegne preopterećenje i optimizuje korišćenje resursa u mreži. U mrežama telekom operatora, SDN omogućava dinamičko prilagođavanje kapaciteta mreže na osnovu trenutnih potreba (npr. povećanje kapaciteta za mobilni internet u specifičnom području i u vreme određenih događaja).

2.4. PROGRAMABILNOST U SDN MREŽAMA

Programabilnost mreže je jedan od ključnih koncepta, koji karakteriše mreže u kojima je primenjena SDN tehnologija. Ona omogućava adaptibilnost i fleksibilnost u pogledu realizacije mrežnih zahteva, što je posebno važno u *data* centrima, *cloud* okruženju i mrežama telekom provajdera [10]. Visok nivo programabilnosti omogućava administratorima, da kontrolišu rad mreže i upravljaju saobraćajem korišćenjem softverskih aplikacija i API-a (Slika 11). Ono što karakteriše programabilnost u SDN mrežama su:

- *centralizovana kontrola* – centralizovana logika smeštena u SDN kontroleru i globalni uvid u stanje mreže, omogućavaju dinamičku promenu pravila rutiranja i znatno bržu reakciju na promene u mreži, kao što su preopterećenja ili zahtevi korisnika.
- *razdvajanje kontrolne i ravni podataka* – pravila za rutiranje i komutiranje saobraćaja se menjaju jednostavno kroz softver i u formi instrukcija šalju mrežnim uređajima, bez potrebe za dodatnim podešavanjima.
- *programiranje mreže korišćenjem API-a* – korišćenjem *Southbound* i *Northbound* API-a omogućava se jednostavno programiranje ponašanja u mreži, jer aplikacije mogu definisati pravila i politike bez potrebe za direktnom interakcijom sa svakim pojedinačnim mrežnim uređajem.
- *upotreba API-ja za programiranje mreže* – omogućava komunikaciju između kontrolera i mrežnih uređaja, ali i omogućava aplikacijama da komuniciraju sa kontrolerom i programiraju mrežne resurse prema specifičnim potrebama.
- *automatizacija mrežnih zadataka* – mnoge mrežne operacije, kao što su rutiranje, balansiranje opterećenja, konfiguracija bezbedonosnih politika i otklanjanje problema su automatizovane, čime se smanjuje potreba za manuelnim podešavanjem i omogućava brže i efikasnije reagovanje na promene u mreži.
- *virtuelizacija i orkestracija* – često korišćenje različitih tehnika virtuelizacije omogućava kreiranje virtuelnih mrežnih funkcija koje se lako mogu skalirati i prilagoditi.

Važno je naglasiti, da stvaranje virtuelnih segmenata mreže olakšava izolaciju različitih korisnika ili servisa Kroz softverski definisano podešavanje mrežnih politika, obezbeđuje se postojanje različitih virtuelnih mreža na istoj fizičkoj infrastrukturi, koje su logički odvojene. Posebno je važno istaći, da se koordinacija između virtuelnih mrežnih funkcija (eng. *virtual network functions* - VNFs) i fizičke mrežne infrastrukture vrši primenom različitih modela orkestracije kako bi se osiguralo optimalno korišćenje resursa i pružanje visokog kvaliteta usluga.



Slika 11. Programabilnost u SDN mrežama

Dakle, prednosti koje donosi primena SDN tehnologije u pogledu većeg nivoa programabilnosti u mrežama su:

- *fleksibilnost* – brze promene mrežne topologije, rutiranja i sigurnosnih politika, što je korisno u situacijama kada se pojavljuju novi korisnici ili aplikacije sa specifičnim zahtevima.
- *efikasnije upravljanje* – administratori mogu koristiti centralizovane softverske alate za upravljanje mrežom, što smanjuje složenost upravljanja velikim mrežama.
- *velika inovativnost* – nove mrežne funkcionalnosti se mogu implementirati kao softverske aplikacije, umesto kroz zamenu ili nadogradnju hardvera.
- *smanjenje troškova* – optimizacija resursa, kao i upotreba standardizovanog mrežnog hardvera, što smanjuje troškove održavanja infrastrukture.

U praksi se danas mogu sresti različiti primeri upotrebe SDN rešenja, kojima se utiče na programabilnost mreže, kao što su balansiranje opterećenja u *data* centrima, primena aplikacija za bezbednost, kojima se detektuju upadi i automatski blokira saobraćaj iz sumnjivih izvora ili ograničava pristup određenim delovima mreže ili segmentacija mreže, kreiranjem logički odvojenih segmenata unutar iste fizičke mreže (što je korisno za obezbeđivanje različitih nivoa pristupa). Važno je napomenuti i da SDN omogućava dinamičku primenu pravila za prioritizaciju određenih vrsta saobraćaja (npr. video konferencije ili VoIP), kako bi se obezbedio visok kvalitet usluge. Aplikacije mogu automatski menjati ova pravila na osnovu trenutnih potreba korisnika.

2.5. AUTOMATIZACIJA U SDN MREŽAMA

Automatizacija je jedna od ključnih prednosti primene SDN tehnologije umrežavanja. Ona kroz automatizovane procese omogućava efikasnije upravljanje i kontrolu nad mrežom. U tradicionalnim mrežama, konfiguracije i optimizacije su često zahtevale manuelne intervencije. To je usporavalo implementaciju novih servisa i povećavalo mogućnost ljudskih grešaka. Sa primenom SDN tehnologije, stvaraju se uslovi za automatizaciji mrežnih operacija, što omogućava brže i pouzdanije prilagođavanje mreže potrebama različitih korisnika i same mreže [11].

Osnovni aspekti automatizacije u SDN mrežama, odnose se na:

- automatsku dodelu resursa (eng. *provisioning*)
- automatizaciju rutiranja i upravljanja saobraćajem
- automatizaciju procesa obezbeđivanja kvaliteta servisa
- automatizacija pravila i politika kojima se definiše bezbednost u mreži i
- orkestraciju mrežnih servisa.

SDN omogućava automatskui dodelu mrežnih resursa na osnovu trenutnih potreba. Ovo možemo ilustrovati na primeru nove aplikacije ili virtuelne mašine, kojima treba pristup mreži. Da bi se realizovao ovaj zahtev, SDN kontroler automatski konfiguriše *switch*-eve i rutere, kako bi što pre omogućio omogućio optimalan tok saobraćaja. Time se ubrzava proces pokretanja novih servisa, a sama mreža postaje mnogo fleksibilnija i sposobna da brzo odgovori na nove zahteve za resursima. Ovde je potrebno ukazati i na aspekt automatizacije procesa rutiranja saobraćaja. Ovaj proces se u tradicionalnim mrežama odvijao statično i uglavnom se oslanja na protokole kao što su OSPF ili BGP. U SDN mrežama, primenom SDN kontrolera se može automatizovati rutiranje i dinamički donositi odluke na osnovu trenutnog stanja mreže. Na primer, SDN kontroler može detektovati preopterećenje na određenom linku i automatski preusmeriti saobraćaj na manje opterećene linkove, čime se poboljšava efikasnost i izbegava gubitak paketa.

Još jedna važna prednost, koju donosi SDN tehnologija je mogućnost, da se automatski vrši prilagođavanje pravila kojima se zahteva potreban nivo QoS-a, kako bi se obezbedila optimalna iskustva za korisnike i aplikacije [12]. To je slučaj, kada se detektuje povećan saobraćaj od kritične aplikacije (npr. video konferencije ili VoIP saobraćaj). SDN kontroler može automatski prilagoditi pravila kako bi prioritetizovao taj saobraćaj i obezbedio potrebnu propusnost. Potrebno je napomenuti, da SDN omogućava dinamičku primenu različitih politika bezbednosti, kao što su dinamička implementacija *firewall* pravila, izolacija saobraćaja i otkrivanje pretnji. To podrazumeva, da u slučaju detekcije sumnjivih aktivnosti u određenom delu mreže, kontroler automatski primenjuje pravila, kako bi ograničio ili blokirao saobraćaj iz sumnjivih izvora. Ovime se smanjuje vreme potrebno za reagovanje na potencijalne pretnje i povećava bezbednost mreže.

Važan aspekt automatizacije u SDN mrežama je i orkestracija mrežnih servisa. U praksi to podrazumeva koordinaciju između virtuelnih mrežnih funkcija (npr. virtuelni ruteri ili *firewall*-ovi) i fizičke infrastrukture. Orkestracija omogućava automatsko pokretanje, konfiguraciju, i upravljanje mrežnim servisima, što je od posebnog značaja

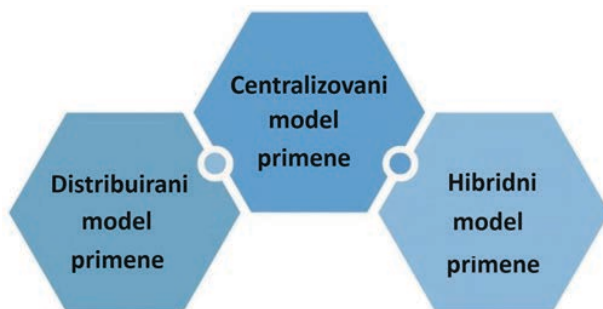
u okruženjima, kao što su *data* centri i *cloud* infrastruktura, gde se često dodaju novi servisi, uklanjaju postojeći ili se vrši njihova modifikacija i prilagođavanje promenjenim zahtevima. Primer je pokretanje nove aplikacije koja zahteva dodatne resurse, gde je uloga orkestracije, da omogući automatsku dodelu potrebnih mrežnih resursa i njihovo povezivanje sa aplikacijom.

Dakle, ključne prednosti koje SDN donosi u pogledu automatizacije su:

- brže reagovanje na promene – omogućava se brže prilagođenje mreže trenutnim promenama, kao što su povećanje saobraćaja ili dodavanje novih uređaja, čime se povećava efikasnost i smanjuje rizik od zagušenja
- manja verovatnoća pojave greške – korišćenjem automatizovanih skripti i alata, smanjuje se potreba za manuelnim podešavanjem, čime se smanjuje i verovatnoća ljudskih grešaka i povećava pouzdanost rada.
- ušteda vremena i resursa – smanjuje se vreme potrebno za konfigurisanje i upravljanje mrežom, što omogućava administratorima da se fokusiraju na strateške zadatke.

2.6. MODELI IMPLEMENTACIJE SDN TEHNOLOGIJE

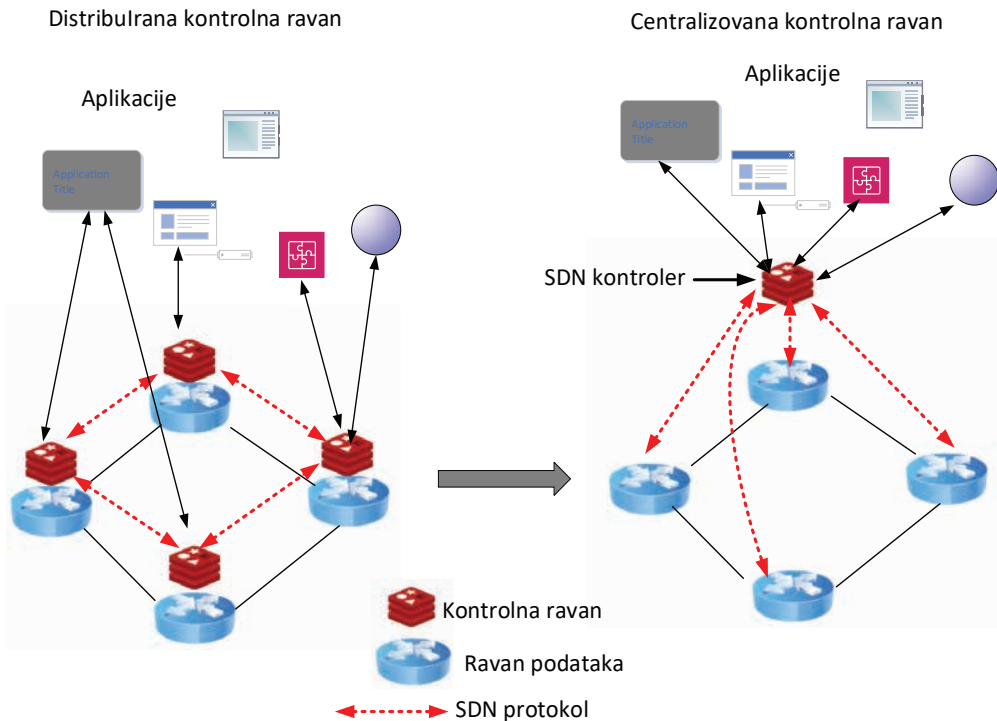
U računarskim mrežama postoji nekoliko modela primene SDN tehnologije. Neki korisnici u potpunosti zamenjuju tradicionalnu (fizičku) infrastrukturu sa virtuelnom. Međutim, visoka cena zamene značajno utiče na većinu korisnika, pa se odlučuju za druge metode. To se prvenstveno odnosi na velike računarske mreže, gde korisnici, pored troškova implementacije, uglavnom razmišljaju o brzini implementacije, kao ključnom kriterijumu za postizanje ciljeva. Da bi se iskoristile prednosti SDN-a, prevazišli složenost i ograničenja postojeće infrastrukture i zadovoljile specifične operativne potrebe, provajderi se obično odlučuju da implementiraju hibridnu mrežnu infrastrukturu ili tzv. *overlay* mreže sa preklapanjem, gde postoji određeno preklapanje funkcionalnosti tradicionalne mreže sa SDN funkcionalnostima. Primena SDN funkcionalnosti preko API-ja se takođe koristi, ali ređe (Slika 12).



Slika 12. Modeli primene SDN tehnologije

2.6.1. Open SDN

Open SDN model predstavlja klasičan pristup u primeni SDN tehnologije, koji se zasniva na razdvajanju kontrolne od ravni podataka (prosleđivanje paketa). Većina tradicionalnih mrežnih uređaja nema ovu mogućnost, pa se korisnici odlučuju da tradicionalne uređaje zamene SDN uređajima, što znači da zamene lokalnu upravljačku ravan sa odgovarajućim SDN kontrolnim slojem [13]. Zadatak ovog sloja je, da uspostavi komunikaciju sa uređajima u infrastrukturnom sloju primenom SDN protokola (npr. *OpenFlow* protokol) i da direktno upravlja njihovim procesima (Slika 13).



Slika 13. Primena *Open SDN* modela [2]

SDN je tehnologije koje podržava zajednica, koja propagira rešenja otvorenog koda i primenu otvorenih standarda [13]. Bolje razumevanje rešenja otvorenog koda, usko je povezano sa ciljevima *Open SDN* modela. Ovi ciljevi su programibilnost primenom kontrolne ravni i apstrakcija mrežnih aplikacija kroz odvajanje kontrolne ravni od ravni za prosleđivanje paketa. Postizanje ovih ciljeva znači komunikacijsku nezavisnost od dobavljača i uspostavljanje interoperabilnosti između fizičkih i virtuelnih uređaja. Time je moguće obezbediti stalnu vidljivost mrežnih tokova i kreirati jedinstveni i programabilni okvir za upravljanje uređajima, što na kraju dovodi do automatizacije upravljanja.

Implementacija ovog SDN promoviše otvorenost i primenu [14]:

- otvorenih standarda – opšte dostupne specifikacije za softver i hardver koji se zajednički razvijaju i održavaju kroz otvorenu zajednicu (npr. *OpenFlow* standard)
- softvera otvorenog koda – aplikacije su dostupne svima tako da se mogu modifikovati i poboljšati (npr. *OpenDaylight* softver za implementaciju funkcija kontrolnog sloja)
- otvorenog hardvera – dizajn fizičkog objekta koji je licenciran tako da ga može kreirati, modifikovati i distribuirati bilo ko, i
- otvorenih API-a i alata za razvoj softvera – kompleti za razvoj softvera (eng. *software development kits*), koji se često koriste u procesu definisanja međusobne komunikacije softverskih entiteta i značajno olakšavaju pisanje softvera.

Implementaciju ovog modela SDN mreže prate izazovi, koji se pre svega odnose na interoperabilnost sa tradicionalnim mrežama. Ozbiljan izazov u praksi predstavlja integracija SDN mreža sa već postojećim mrežama i uređajima, koji nisu dizajnirani za rad u SDN okruženju. Takođe, centralizacija mrežne logike, donosi i izazove u pogledu bezbednosti, jer u SDN kontroler potencijalno predstavlja centralnu tačku napada. Važno je naglasiti na kraju, da preduzeća i organizacije, koje se opredele za ovaj model implementacije SDN funkcionalnosti, često imaju ozbiljan zadatak da obuče osoblje, da razume novu arhitekturu i protokole.

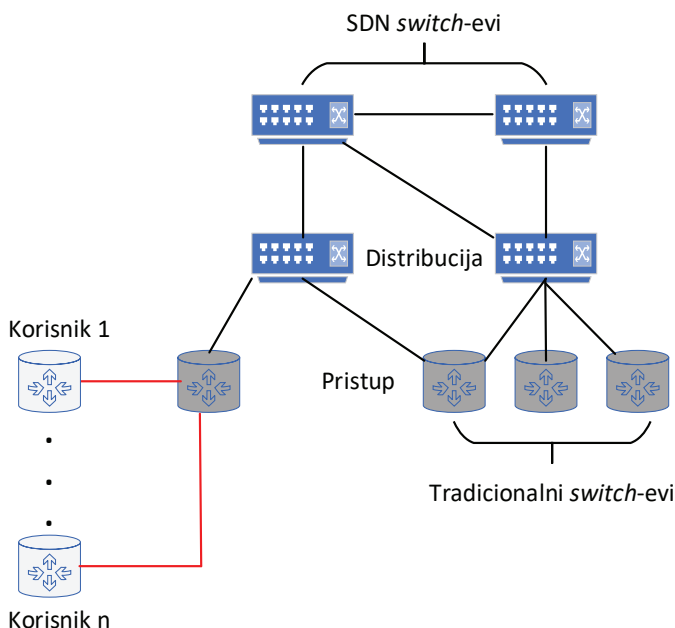
Implementacija Open SDN-a se u praksi može sresti u:

- *data* centrima i cloud infrastrukturi, gde se koristi za automatsko skaliranje resursa, dinamičku alokaciju kapaciteta, i izolaciju virtuelnih mreža za različite klijente
- mrežama telekom provajdera (eng. *Carrier Networks* - CN), gde se koristi za optimizaciju saobraćaja između različitih tačaka mreže i za bržu implementaciju novih usluga
- mrežama preduzeća, gde se koristi da bi se pojednostavilo upravljanje mrežom, smanjili operativni troškovi i poboljšala bezbednost.

U celini posmatrano, *Open SDN* donosi fleksibilnost i efikasnost kroz softversko upravljanje, čime se olakšava dinamičko prilagođavanje mreže zahtevima modernih aplikacija i korisnika. Korišćenje otvorenih protokola i standarda ima za cilj, da se obezbedi interoperabilnost i smanji zavisnost od pojedinih proizvođača mrežne opreme.

2.6.2. Hibridni SDN

Punu implementaciju SDN funkcionalnosti u računarskim mrežama prate brojni tehnički i finansijski izazovi. Uprkos brojnim prednostima SDN tehnologije, u praksi se najčešće srećemo sa hibridnim mrežnim arhitekturama. Razlozi tehničke, ali pre svega finansijske prirode, nameću usvajanje strategije, koja propagira inkrementalnu implementaciju SDN funkcionalnosti. Slobodno se može reći, da hibridna SDN mreža predstavlja srednje rešenje, jer podrazumeva integraciju starih (tradicionalnih) i SDN uređaja koji rade paralelno i koriste prednosti i tradicionalnih i SDN mreža (Slika 14) [15].



Slika 14. Model hibridne SDN mreže

Izgradnja hibridne SDN mreže podrazumeva, zamenu delova tradicionalne opreme, sa virtuelnom infrastrukturom koja podržava *OpenFlow* protokol tj. primenu SDN tehnologije. Osnova arhitekture hibridne SDN mreže je 3C model, koji podrazumeva [15]:

- heterogenost infrastrukture u ravni podataka, kontrolnoj ravni ili u obe ravni,
- komunikaciju između tradicionalnih i SDN uređaja sa ciljem da se omogući distribucija zadataka i deljenje funkcionalnosti u ravni podataka, kontrolnoj ravni ili u obe ravni i
- kombinaciju različitih mrežnih tehnologija, sa ciljem, da se minimizuju troškovi, pojednostavi prelazak na SDN i implementiraju saobraćajne politike, tako da se obezbedi optimalna skalabilnost i velika robusnost.

Drugim rečima, hibridne SDN mreže funkcionišu tako što omogućavaju koegzistenciju tradicionalnih mrežnih protokola i mehanizama (kao što su OSPF, BGP, statičko rutiranje) i SDN upravljanih segmenata (gde se koristi centralizovani kontroler i protokoli poput *OpenFlow* protokola). Ovakav pristup omogućava fleksibilnost u implementaciji novih SDN funkcionalnosti, dok se istovremeno zadržava stabilnost i pouzdanost tradicionalne mreže. Prednost hibridnih SDN mreža je u tome, što organizacijama omogućavaju, da SDN uvedu tamo gde je najpotrebnije, a da istovremeno zadrže tradicionalne tehnologije u delovima mreže gde bi potpuna migracija bila suviše komplikovana ili skupa. Što je još važnije, hibridni SDN obezbeđuje zaštitu investicija. Operatori mogu dodaju SDN funkcionalnosti samo u onim mrežnim segmentima gde je potrebno, dok se u ostalim delovima mreže zadržavaju postojeći uređaji tj. infrastruktura. Oni time štite

prethodna ulaganja u mrežnu opremu. Primena hibridnog SDN-a i edukativno pomaže korisnicima, pre svega inženjerima i administratorima. Stvaraju se uslovi, da se oni postepeno upoznaju sa SDN konceptima i tehnologijama, što može olakšati prelazak na potpunu SDN arhitekturu u budućnosti.

U praksi postoji više modela hibridnih SDN mreža. Jedan od njih je na topologiji zasnovan (eng. *Topology-based*) hibridni model SDN mreže, koji podrazumeva primenu SDN tehnologije na osnovu uvida u topologiju fizičke mreže. Ova integracija omogućava optimizaciju korišćenja resursa, poboljšanje saobraćajnog inženjeringa i preciznu kontrolu nad elementima mreže. Uvid u topologiju mreže, ima za cilj da se SDN primeni, tamo gde je potrebno ponašanje mreže uskladiti sa specifičnim potrebama ili aplikacijama. Pored navedenih prednosti, ovaj hibridni SDN model ima i određene nedostatke. Ovakva integracija SDN funkcionalnosti može izazvati probleme u upravljanju i povećati troškove implementacije i održavanja, stvoriti potencijalne bezbednosne rizike, zavisnost od podrške i standarda dobavljača i ograničiti skalabilnost.

Drugi model je na servisima zasnovan (eng. *Service-based*) hibridni model SDN mreže, koji integriše arhitekturu orijentisanu na usluge sa centralizovanom kontrolnom ravni, koja vrši orkestraciju tih usluga. Strukturiranje mreže u usluge i omogućavanje dinamičke orkestracije, ima za cilj, da se olakša isporuka različitih aplikacija, optimizuje alokacija resursa i omogući postepeno usvajanje SDN-a. Drugim rečima, ovaj model obezbeđuje ravnotežu između stabilnosti sa jedne strane i prednosti programiranja i centralizovanog upravljanja sa druge strane. Pored toga, postoje i određeni nedostaci, koji se odnose na složenost integracije, uključujući ograničenu agilnost i skalabilnost, složenost upravljanja, potencijalna uska grla u performansama, izazove interoperabilnosti i složenost migracije.

Treći model je na klasama zasnovan (eng. *Class-based*) hibridni SDN model implementacije, kojim se optimizuje upravljanje mrežom. Optimizacija se obezbeđuje, tako što se uređaji i saobraćaj kategorišu u klase, kojima se upravlja preko različitih SDN politika. Uloga SDN kontrolera je da primenom različitih protokola, omogući integracije tradicionalnih mrežnih uređaja i SDN komponenti. Osim toga, SDN kontroler ima i druge zadatke, kao što su implementacija politika za bezbednost i obezbeđivanje adekvatnih performansi za svaku klasu, kroz dinamičku alokaciju resursa i QoS mehanizme. Dakle, ovaj model prilagođava upravljanje mrežom specifičnim potrebama, balansirajući između različitih elemenata mreže i tipova saobraćaja. Glavni nedostaci ovog SDN modela se odnose na složenost upravljanja klasama (koriste se različiti protokoli) i potencijalne probleme interoperabilnosti među mrežnim uređajima. Međutim, ako se mreža brzo širi, mogu se pojaviti potencijalni bezbednosni izazovi uzrokovani različitim kontrolama pristupa.

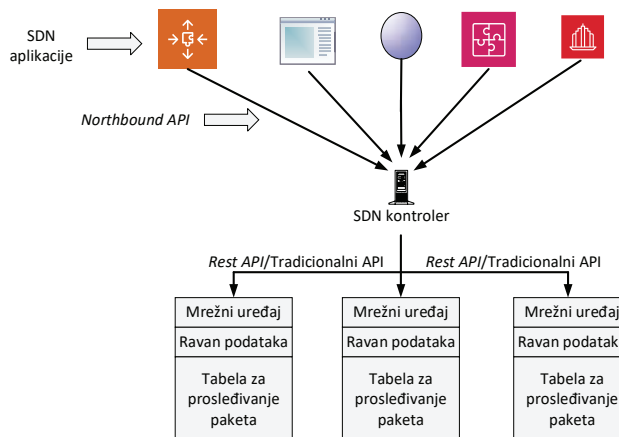
Četvrti model je integrisani hibridni SDN model, koji koristi SDN kontroler za upravljanje SDN i tradicionalnim mrežnim uređajima. Ovaj pristup uvodi programibilnost i automatizaciju u postojeću infrastrukturu, sa ciljem stvaranja prilagodljive, skalabilne i fleksibilne mrežne arhitekture. Ovakva arhitektura je sposobna za dinamičku modifikaciju i efikasno upravljanje mrežnom infrastrukturom. Složenost integracije može dovesti do izazova upravljanja i operativnih poteškoća. To se uglavnom odnosi na upotrebu različitih protokola i standarda širom mrežne infrastrukture, što može uzrokovati probleme interoperabilnosti. Jasno je, da integracija, u kojoj se koriste različite

mrežne tehnologije, može uzrokovati potencijalne bezbednosne propuste, značajna finansijska ulaganja i potrebu za stalnom alokacijom resursa.

Dakle, izgradnja hibridne SDN infrastrukture je praćena određenim ograničenjima. To se pre svega odnosi na složenost upravljanja heterogenostima u kontrolnoj ravni i probleme prevođenja poruka tradicionalnih protokola u poruke SDN protokola i obrnuto. Kao rezultat ovih ograničenja, nastaje proces rekonfiguracije, koji uzrokuje nedoslednost u prosleđivanju paketa, što može dovesti do petlje u prosleđivanju paketa i degradacije performansi u smislu kašnjenja i vremena obrade paketa [16]. Ponekad se problem javlja ako postoje dva ili više kontrolera u hibridnoj SDN infrastrukturi. U ovom slučaju može doći do značajnih kašnjenja, problema sa saobraćajnim inženjeringom i poteškoća u postizanju potrebnog nivoa skalabilnosti, pa čak i bezbednosti. Implementacija adekvatnog saobraćajnog inženjeringa u hibridnom SDN-u predstavlja izazov, posebno imajući u vidu da neki uređaji ne mogu da podrže apstrakciju tokova [17].

2.6.3. SDN preko API-a

SDN preko API-a (eng. SDN *via* API) je model implementacije SDN-a, koji su neki dobavljači ponudili kao rešenje, kojim se pozivaju funkcije na udaljenim uređajima (kao što su *switch*-evi) i to koristeći konvencionalne metode kao što su SNMP protokol (eng. *Simple Network Management Protocol*), CLI (eng. *Command Line Interface*) ili Rest API-a. Na ovaj način, obezbeđujući kontrolne tačke na uređaju, SDN kontroler upravlja udaljenim uređajima pomoću API-ja. Važno je istaći, da ova implementacija SDN-a ne zahteva posebne *switch*-eve (npr. *OpenFlow* standard dobro funkcioniše sa nekim tradicionalnim *switch*-evima) [18]. Velika prednost ove implementacije SDN-a je mogućnost lakog pravljenja softvera za orkestraciju. SDN preko API-ja obezbeđuje povećanu otvorenost i na taj način eliminiše potrebu za vlasničkim (en. *proprietary*) rešenjima. Drugim rečima, on omogućava različitim uređajima da komuniciraju i budu kontrolisani preko standardizovanih interfejsa (Slika 15).



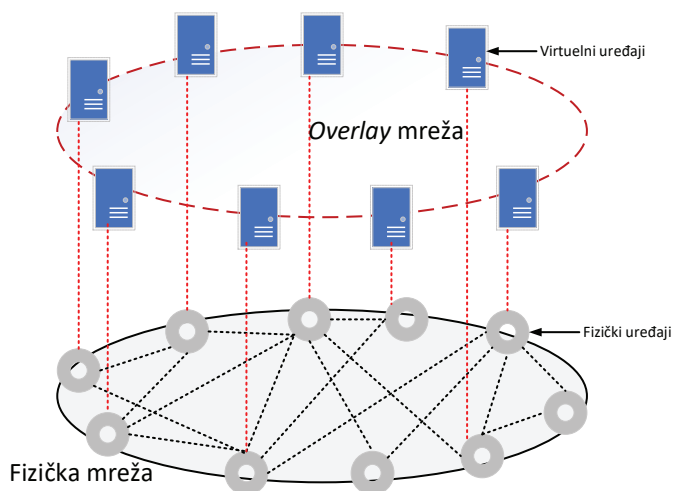
Slika 15. Implementacija SDN funkcionalnosti preko API-a [2]

Ovaj metod implementacije predstavlja korak napred u otvorenosti i pruža mogućnosti za sveobuhvatniju saradnju umesto korišćenja vlasničkih rešenja. U praksi, postoji mogućnost korišćenja API-a, koji su vlasničke prirode (vezani su za određenog proizvođača). Takvi API-i ne moraju biti kompatibilni sa ostatkom softverske infrastrukture, što treba eliminisati. Zato aplikacije koje koriste SDN preko API-a, moraju imati informacije o opremi dobavljača sa kojom bi trebalo da komuniciraju. To je od ogromnog značaja, jer aplikacije moraju da koriste ispravan API. Argument onih koji predlažu ovaj metod implementacije je postizanje SDN ciljeva omogućavanjem:

- da aplikacije i dalje utiču na odluke o prosleđivanju i
- otvorenost API-a, što znači da svako može da kreira aplikacije i koristi API-je (na ovaj način je moguće učiniti mrežu programabilnom, ali ne nužno i fleksibilnom zbog vlasničke prirode southbound API-a).

2.6.4. SDN *overlay* mreže

U realnim okruženjima postoji još jedan model implementacije SDN funkcionalnosti, koji takođe podrazumeva odvajanje kontrolne od ravni podataka. Koncept ovog modela je stvaranje nove, nezavisne mreže „na vrhu” postojeće mreže. To ne znači da tradicionalna mrežna infrastruktura, koja se nalazi ispod nove mreže, ostaje bez kontrolne ravni. Naprotiv, uređaji i dalje imaju lokalnu kontrolnu ravan gde se procesi tradicionalno odvijaju. Ovaj model implementacije pokreće virtuelnu mrežu na vrhu postojeće hardverske infrastrukture, stvarajući dinamičke tunele do različitih lokalnih i udaljenih računarskih okruženja. Virtuelna mreža dodeljuje propusni opseg različitim kanalima i dodeljuje uređaje svakom kanalu, ostavljajući fizičku mrežu nepromenjenom. Tako su virtuelna mreža i osnovna infrastruktura povezane, a paketi se razmenjuju između njih (Slika 16) [19].



Slika 16. Implementacija SDN funkcionalnosti preko *overlay* mreže

Slika 16. prikazuje implementaciju SDN-a preko *overlay* mreže, koja uključuje kreiranje virtuelizovanog mrežnog sloja na vrhu fizičke infrastrukture. SDN kontroler ima ključnu ulogu i omogućava apstrakciju osnovnog hardvera i dinamičku kontrolu nad mrežnim funkcijama. On kreira politike i konfiguracije, što podrazumeva, da *overlay* mreža definiše kako se paketi prosleđuju, obezbeđujući efikasno rutiranje, QoS i mere bezbednosti. Kontroler komunicira sa mrežnim uređajima koristeći *southbound* API za slanje instrukcija visokog nivoa za fizičku infrastrukturu. Iz perspektive korisnika, oni komuniciraju sa *overlay* mrežom i koriste SDN kontroler za upravljanje, bez potrebe za podrškom za osnovne fizičke uređaje. Ova implementacija SDN-a ispunjava osnovne kriterijume, ali ima jedno ograničenje. Uređaji koji su pod virtuelnom mrežom, moraju podržavati protokol, koji se koristi za komunikaciju sa tom mrežom. Primeri ove implementacije su VXLAN (eng. *Virtual eXtensible LAN*) i virtuelizacija mreže pomoću generičke enkapsulacije rutiranja (eng. *Network Virtualization using Generic Routing Encapsulation* - NVGRE).

ZAKLJUČAK

SDN tehnologija predstavlja fundamentalnu promenu u načinu na koji se mreže dizajniraju, upravljaju i kontrolišu. Za razliku od tradicionalnih mreža, gde su funkcije kontrole i prenosa integrisane unutar mrežnih uređaja, SDN odvaja ove funkcije. U SDN mrežama, kontrolna ravan je centralizovana u kontroleru, što omogućava dinamičko upravljanje mrežom i optimizaciju saobraćaja na fleksibilan način. Centralizovana kontrola omogućava mrežnim administratorima brže reagovanje na promene u zahtevima za mrežnim resursima i jednostavniju implementaciju politika mrežne bezbednosti i QoS-a. Ključni aspekti primene SDN-a su:

- centralizovana kontrola i fleksibilnost – omogućava efikasnije upravljanje mrežom i brzo prilagođavanje mrežnih politika,
- automatizacija i orkestracija – olakšava upravljanje mrežnim konfiguracijama, smanjujući manuelne intervencije i potencijal za ljudske greške,
- skalabilnost i inovativnost – obezbeđuje skalabilnije mreže, koje mogu lakše da integrišu nove aplikacije i tehnologije poput NFV tehnologije (eng. *Network Function Virtualization*) i mrežnog *slicing*-a,
- veća transparentnost i kontrola – omogućava precizno praćenje mrežnih performansi i optimizaciju saobraćaja.

SDN danas predstavlja ključnu tehnologiju za modernizaciju mreža, omogućavajući veću agilnost i spremnost za digitalne transformacije u različitim industrijama. Uprkos izazovima u integraciji sa postojećim mrežama i potencijalnim bezbednosnim pitanjima, njegova primena je sve šira i predstavlja budućnost mrežnih arhitektura.



POGLAVLJE 3



VIRTUELIZACIJA U INFRASTRUKTURI SDN MREŽA

3. VIRTUELIZACIJA U INFRASTRUKTURI SDN MREŽA

3.1. ISTORIJAT VIRTUELIZACIJE

Istorija virtualizacije je duga više decenija, tokom kojih je ova tehnologija oblikovala moderno računarstvo i mrežne arhitekture. Začetak ove tehnologije se veže za 60-te godine prošlog veka i pojavu tzv. *mainframe* računara. Kompanija IBM je je započela razvoj tehnologije, koja je imala za cilj, da omogući deljenje resursa *mainframe* računara između više korisnika. Razvijen je *time-sharing* koncept, koji je omogućio, da više korisnika istovremeno deli resurse IBM System/360 sistema [2]. Kasnije, sa početkom 70-ih godina, IBM je razvio *hypervisor* tehnologiju za svoj System/370, koja je omogućila kreiranje virtuelnih mašina (eng. *virtual machine* - VM). Sistem je bio sposoban, da simulira više instanci operativnog sistema na istom hardveru. Ovo je bio prvi komercijalni sistem za virtualizaciju. On je omogućio korisnicima da pokreću više instanci IBM-ovih operativnih sistema na jednom fizičkom računaru.

Početkom 80-ih i 90-ih godine, došlo je do uspona x86 virtualizacije, koja je postala izuzetno popularna sa pojavom x86 arhitekture. Tada su se pojavili i prvi alati za virtualizaciju PC računara, kao što su *SoftWindows* i *VMware* (prvi uspešno implementirao virtualizaciju na x86 platformi i omogućio kreiranje i upravljanje virtuelnim mašinama). Početkom 2000-ih, dolazi do procvata u IT industriji, što se posebno odrazilo na značaj virtualizacije, koja je postala ključna tehnologija u IT industriji, posebno za servere i *data centre*. *VMware* i drugi proizvođači, kao što su Microsoft (sa *Hyper-V*), počeli su da nude sveobuhvatna rešenja za virtualizaciju, kojima se omogućava konsolidacija servera i smanjuje potreba za fizičkim hardverom.

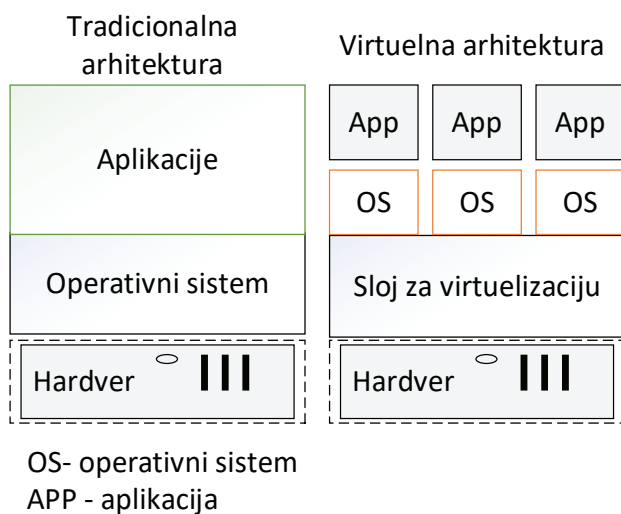
Važno je napomenuti, da su 90-ih godina prošlog veka, još uvek bila dominantna *proprietary* rešenja i model upravljanja jedan server - jedna aplikacija. Ovakva rešenja nisu dozvoljavala, da se softverske aplikacije izvršavaju na hardveru drugog proizvođača. U tom momentu je neracionalno korišćenje (pre svega serverskih) resursa bila realnost. Pojava tehnologije virtualizacije i mogućnost, da se na jednoj računarskoj mašini istovremeno pokrene više virtuelnih servera (sa različitim operativnim sistemima), je u najvećoj meri uticala na smanjenje zavisnosti od proizvođača hardvera. Tehnologija virtualizacije je postala osnova razvoja novog modela računarstva tj. *cloud computing*-a.

Uvođenjem *cloud computing*-a, virtualizacija je postala centralna komponenta za izgradnju moderne infrastrukture, čiji vlasnici, kao što su AWS, Azure i Google Cloud, koriste virtualizaciju za obezbeđivanje resursa ili servisa na zahtev (eng. *resource or service on demand*). Upravljanje samo virtuelnom infrastrukturom računarskih mreža postaje pravi izazov, ali i podsticaj za primenu SDN-a, kao novog koncepta umrežavanja. Razvojem SDN umrežavanja, centralizacija funkcije upravljanja i uvođenje znatno višeg nivoa programabilnosti, predstavljaju ključne komponente za softverizaciju mreže i izgradnju inteligentnih okruženja kao što 5G i 6G mreže. Sam proces virtualizacije, dobija novu dimenziju početkom ove decenije, kada počinje razvoj nove tehnike virtualizacije, koja se pojavljuje pod nazivom kontejnerizacija. Virtuelni entiteti, poznati kao kontejneri (*Docker* i *Kubernetes*), postali su popularni zbog svoje efikasnosti i brzine implementacije. Oni se posebno koriste danas za realizaciju mikroservisne

arhitekture i lako skalabilnih i prenosivih aplikacija. Štaviše, danas mnoge organizacije sve više primenjuju hibridne i multicloud strategije, kombinovanjem virtuelizacije sa kontejnerizacijom i drugim tehnologijama, kako bi se postigla maksimalna fleksibilnost u rešavanju zahteva iz mreže.

3.2. KONCEPT VIRTUELIZACIJE

Virtuelizacija je tehnološki proces koji omogućava kreiranje virtuelnih verzija računarskih resursa (serveri, operativni sistemi, skladišta za podatke ili mrežni resursi), koristeći softver umesto fizičke opreme. To znači da jedan fizički resurs, poput servera ili mrežnog uređaja, može biti podeljen na više virtuelnih instanci koje funkcionišu kao nezavisni resursi (Slika 17).



Slika 17. Tradicionalna i virtuelna arhitektura [20]

U suštini, virtuelizacija predstavlja koncept, kojim se aplikacije i njihove osnovne komponente apstrahuju od hardvera na kojem se izvršavaju. Osnova koncepta je proces deljenja resursa računarskog sistema u više zasebnih (virtuelnih) okruženja. Primenom metoda, kao što su hardversko i softversko particionisanje, *timesharing* ili emulacija, računarske resurse je moguće „grupisati“ u više zasebnih računarskih okruženja, sa sopstvenim operativnim sistemima, koji se simultano izvršavaju na jednoj fizičkoj mašini. Razlog primene koncepta virtualizacije koji se zasniva na apstrakciji računarskih resursa, je povećanje efikasnosti, smanjenje troškova i omogućavanje fleksibilnijeg upravljanja IT okruženjem, posebno u *cloud* i *data* centar okruženjima. Osim namere, da se unaprede performanse računarskog sistema, poveća skalabilnost i ostvari veća pouzdanost u radu, virtuelizacija pomaže da se izgradi jedinstven domen sa aspekta

bezbednosti i upravljanja. Ona omogućava još jedan vid apstrakcije računarskih resursa i stvara uslove za spajanje više razdvojenih, fizičkih celina u veću, logičku (virtuelnu) celinu. Računarski resurs koji nastaje spajanjem tj. agregacijom više fizički razdvojenih računarskih mašina (eng. *grid computing*), operativni sistem vidi kao jedan logički skup računarskih resursa. Drugim rečima, primenom virtuelizacije se stvara logički (virtuelan) pogled na računarske resurse, koji je upadljivo drugačiji od fizičkog.

U realnim okruženjima korisnici imaju različite potrebe i zato se opredeljuju da koriste različite alate, kako bi ostvarili svoje ciljeve. Njihovi ciljevi mogu biti veoma različiti, kao npr. da se [6]:

- bilo kom uređaju u mreži, omogućiti pristup preko Interneta do bilo koje aplikacije preko bilo koje mreže (čak i ako ta aplikacija nikada nije bila dizajnirana da radi sa tom vrstom uređaja),
- aplikacije izoluju jedna od druge, kako bi se omogućio bezbedan rad i veća upravljivost u okruženju,
- izvrši izolacija aplikacija od operativnog sistema i omogućiti da ista funkcioniše i na drugim verzijama operativnog sistema,
- poveća broja korisnika koje aplikacija može da podrži i omogućiti istovremeno pokretanje više instanci na različitim mašinama,
- smanji vreme potrebno za pokretanje aplikacije kroz segmentaciju podataka ili same aplikacije i zadaci delegiraju na više računarskih sistema,
- optimizuje korišćenja jednog sistema, omogućavajući da isti više i inteligentnije radi (smanjuje se vreme u kome procesor ne radi),
- obezbedi redundansa kako bi se uticalo na visoku pouzdanost i dostupnost aplikacija i podataka i dr.

3.2.1. Softver za virtuelizaciju

Softver za virtuelizaciju (eng. *hypervisor*) igra ključnu ulogu u modernim IT okruženjima. On upravlja hardverskim resursima i ima ključnu ulogu u kreiranju virtuelnih mašina (simulirano okruženje koje se često naziva gostujuća mašina) i okruženja u kojem one rade. Njegova svrha je da obezbedi efikasno korišćenje resursa, smanji troškove i omogućiti bržu implementaciju aplikacija. Izbor pravog rešenja za virtuelizaciju zavisi od specifičnih potreba, budžeta i okruženja u kojem se koristi, a sami hipervizori, koji se implementiraju u računarskim sistemima, podeljeni su po tipu i dizajnu [1].

Zadatak hipervizora je da obezbedi određeni stepen apstrakcije računarskih resursa i stvori okruženje u kome program radi isto kao da je na ekvivalentnoj fizičkoj mašini. Drugim rečima, hipervisor ima potpunu kontrolu nad virtuelizovanim resursima. Ovde se mora uzeti u obzir, da se statistički, većina instrukcija operativnog sistema izvršava bez intervencije hipervizora. U praksi postoje tri tipa hipervizora [21]:

- *Bare metal* hipervisor – smešten između hardvera i operativnog sistema. Pokreće se direktno na hardveru i kreira virtuelna okruženja sa svojim operativnim sistemom (npr. *Citrix XenServer*, *VMware ESX* i *Microsoft Hiper-V*),

- hostovani hipervizor – nalazi se na vrhu operativnog sistema fizičke mašine (matični operativni sistem) i upravlja hardverskim resursima. Omogućava da se operativni sistemi virtuelnih mašina instaliraju na hostu, da rade paralelno i da dele resurse (KVM i VirtualBox),
- hibridni hipervizor – uglavnom radi direktno na hardveru, ali koristi operativni sistem domaćina za I/O operacije (*IBM VM/370, VMware Server i VMware Workstation*).

Sa aspekta softverske arhitekture postoje dva dizajna hipervizora [1]. Jedan je tzv, monolitni dizajn, koji karakteriše to da hipervizor poseduje posebne „module“, tj. drajvere koji su svesni hipervizora i time omogućavaju operativnom sistemu virtuelne mašine da komunicira sa hardverom hosta. Drugi tip dizajna se odnosi na mikrokernel hipervizore. Ovi hipervizori koriste operativni sistem hosta (deluje kao *root* (roditeljska) particija) za pristup hardveru i korišćenje širokog spektra postojećih drajvera koji nisu deo hipervizora.

3.2.2. Virtuelne mašine

Danas postoje različite definicije, kojima se opisuje tehnologija virtuelizacije. Najčešće se koristi ona, koja virtuelizaciju opisuje kao postupak kojim se apstrahuju resursi fizičke mašine, tako što se kreiraju međusobno izolovana, virtuelna okruženja. Ova okruženja se nazivaju virtuelnim mašinama ili virtuelnim kontejnerima. Svako od njih koristi sopstveni sloj hardvera, softvera, podataka, mreže i memorije. Ti slojevi su međusobno izolovani, tako da se lako kontrolišu procesi na njima. Njima se može direktno pristupiti, kako bi se vršile izmene i resursi dodeljivali prema potrebi korisnika. Strogo posmatrano, između virtuelnih mašina i virtuelnih kontejnera postoje razlike. One se uglavnom odnose na način na koji su ta virtuelna okruženja kreirana, odnosno na nivo izolacije i rezultat koji se postiže svakom od ovih tehnika.

Osnovne komponente svake virtuelne mašine su [22]:

- matični operativni sistem – operativni sistem fizičke mašine (eng. *Host Operating System*), koji se izvršava direktno na hardveru i poseduje podršku za virtuelizaciju i odgovarajući skup aplikacija
- hipervizor (eng. *hypervisor*) – sloj za virtuelizaciju koji kreira virtuelnu mašinu, dodeljuje resurse toj virtuelnoj mašini, menja njene parametre i na zahtev vrši njeno brisanje
- operativni sistem gosta (eng. *Guest Operating System - GOS*) – operativni sistem koji se izvršava na virtuelnoj mašini, tj. entitetu koji „gostuje“ na fizičkoj mašini i kompatibilan je sa hardverskim resursima koje nudi hipervizor.

U praksi se koriste različite tehnike virtuelizacije, koje imaju za cilj da omoguće, da se na fizičkim mašinama izvršavaju različiti programski kodovi. Zato se sa pojavom prvih virtuelnih mašina, postavilo pitanje: da li procesor fizičke mašine, pored sopstvenog operativnog sistema, može da opslužuje (različite) operativne sisteme virtuelnih mašina? Značaj ovog pitanja je veliki, jer se na virtuelnim mašinama izvršavaju aplikacije, čije funkcionisanje zavisi upravo od operativnog sistema i njegove sposobnosti da angažuje potrebne hardverske resurse (operativni sistem preko svog kernela, komunicira sa drajverima i angažuje hardverske resurse) [23].

Virtuelne mašine se oslanjaju na različite vrste virtualizacije, koje omogućavaju različite nivoe kontrole i efikasnosti:

- potpuna virtualizacija (eng. *full virtualization*) – simulira celokupan hardverski sloj, omogućavajući virtuelnim mašinama da se ponašaju kao fizički serveri (najčešće korišćen tip virtualizacije).
- paravirtualizacija (eng. *paravirtualization*) – delimično modifikuje operativne sisteme unutar virtuelnih mašina, kako bi se smanjilo opterećenje na hipervizoru i poboljšale performanse.
- virtualizacija na nivou operativnog sistema (eng. *containerization*) – omogućava izolaciju aplikacija unutar jednog operativnog sistema, pružajući slične koristi kao virtuelne mašine, ali sa manjom potrošnjom resursa (npr. *docker* i *kubernetes*).

Potpuna virtualizacija je tehnika koja se koristi, da bi se fizička mašina (server) u potpunosti virtualizovala i kreiralo virtuelno okruženje neophodno za funkcionisanje operativnog sistema virtuelne mašine (gosta). Pri tome, operativni sistem virtuelne mašine je u potpunosti izolovan od fizičkog hosta (između njih se nalazi hipervizor), pa se može koristiti na način kako je originalno kreiran. Ovakav pristup virtualizaciji računarskih resursa povećava bezbednost računarskog sistema i značajno utiče na njegovu fleksibilnost i skalabilnost (na jednom fizičkom serveru se može kreirati više virtuelnih instanci, odnosno servera sa sopstvenim operativnim sistemima). Kao rezultat dobija se globalni sistem, sa nekim novim mogućnostima i eliminišu neki nedostaci koji su postojali na fizičkim serverima (npr. iskorišćenost resursa na svakom serveru pojedinačno). Potpuna virtualizacija je u praksi primenjena u rešenjima kao što su: *VMware Server*, *VirtualBox*, *Oracle VM*, *Hyper-V*, *KVM* i dr.

U nekim rešenjima, tehnologija virtualizacije omogućava da virtuelna mašina radi u sličnom, ali ne i istom hardverskom okruženju kao fizička mašina (virtuelnim mašinama se prezentuje interfejs sličan onom koji ima stvarni hardver). Zato je potrebno da se operativni sistem virtuelne mašine portuje (eng. *porting*) tj. modifikuje i prilagodi za rad u virtuelnom okruženju [24], [25]. Modifikacija se vrši rekompajliranjem kernela operativnog sistema ili instalacijom specijalno razvijenog API-a, odnosno paravirtuelizovanih drajvera (para-drajvera). Zadatak paravirtuelizovanih drajvera je da omoguće pristup hipervizoru u realnom vremenu i obezbede komunikaciju između operativnog sistema virtuelne mašine i operativnog sistema fizičke mašine (matični operativni sistem). Operativni sistem virtuelne mašine, podatke o hardveru dobija upravo od operativnog sistema fizičke mašine. Samim tim, simulacija kompletnog hardvera nije potrebna, a performanse virtuelnog sistema su znatno bolje (alokaciju neophodnih resursa vrši hipervizor, što rezultuje gotovo prirodnim izvršavanjem operacija). Potrebno je napomenuti, da je kod rešenja sa primenjenom tehnikom paravirtualizacije (sistemi zasnovani na *UNIX*-u, nekim varijantama *BSD*-a ili *Open Solaris*-u), brzina operativnog sistema gosta, manja od brzine operativnog sistema fizičkog hosta za najviše 10% što je upravo posledica modifikacije operativnog sistema virtuelne mašine.

U realnim okruženjima, a pre svega na cloudu, primenjuje se tehnika virtualizacije na nivou operativnog sistema hosta. Ona se zasniva na korišćenju njegovog kernela za potrebe kreiranje višestrukih i međusobno izolovanih *user-space* instanci [1]. Za ove virtuelne instance se koristi termin kontejneri. Oni se najčešće konfigurišu da rade

kao privatni virtuelni serveri, koji dele kernel tog operativnog sistema i imaju zasebne *root* fajl sisteme [26]. Prednosti ove tehnike virtualizacije su bezbedan rad virtuelnih instanci (međusobno su izolovane), centralizovana administracija i konfiguracija koja se izvršava na operativnom sistemu hosta. U praksi, ovakva rešenja se najčešće susreću kao: *Linux-Vserver*, *FreeBSD Jails*, *Solaris Containers*, *Virtuozzo* i dr.

Danas se sa velikom sigurnošću može konstatovati, da su virtuelne mašine promenile način na koji korisnici upravljaju svojim IT resursima, omogućavajući fleksibilniju i efikasniju upotrebu infrastrukture. Kroz različite tipove virtualizacije i napredne funkcionalnosti kao što su migracija u realnom vremenu i kreiranje *snapshot*-ova, one predstavljaju moćan alat za savremene potrebe poslovanja. Sama tehnologija virtualizacije, postala je značajna za IT stručnjake, koji svakodnevno koriste sledeće prednosti virtualizacije u svom radu:

- efikasno korišćenje resursa – moguće je pokretati više virtuelnih mašina na jednom serveru, maksimalno iskorišćavajući dostupne resurse, čime se smanjuje potreba za dodatnim fizičkim serverima
- izolaciju – virtuelne mašine su izolovana okruženja u kojima se bez rizika za matični operativni sistem mogu testirati nove aplikacije
- skalabilnost – virtuelne mašine se lako kreiraju, repliciraju i skaliraju u skladu sa potrebama, što ih čini fleksibilnim i pogodnim za korišćenje u dinamičkim okruženjima
- migraciju u realnom vremenu i oporavak od katastrofalnih događaja (eng. *disaster recovery*) – virtualne mašine se mogu migrirati u realnom vremenu (eng. *live migration*), sa jednog na drugi fizički server bez prekida u radu, čime se olakšava održavanje i balansiranje opterećenja (moguće je praviti *snapshot*-ove i brzo se vratiti na pređašnje stanje).

3.2.3. Alokacija resursa

Virtualizacija treba da omogući efikasno deljenje hardverskih resursa fizičke mašine, kao što su memorija, CPU, interfejsi i prostor na disku. Hipervizor je upravo zadužen da obezbedi te mehanizme, koji će na optimalan način deliti resurse. Kada se kreira virtuelna mašina, hipervizor dodeljuje određenu količinu memorije i CPU-a [24]. Te CPU-ove, koji su dodeljeni virtuelnoj mašini, njen operativni sistem posmatra kao fizičke CPU-ove. U svakom slučaju, ono što hipervizor dodeljuje virtuelnoj mašini sa aspekta CPU infrastrukture, zavisi od CPU resursa koji su dostupni na nivou matičnog operativnog sistema. Kao primer, može se uzeti fizička mašina, koja koristi Intel-ov Xeon E5-2680v2 CPU i ima deset jezgara/socket-a sa dualnim *thread*-ovima. U ovom slučaju, hipervizor može da ponudi 20 virtuelnih CPU-a virtuelnoj mašini. Slično je sa dodelom memorije virtuelnoj mašini, koju njen operativni sistem vidi kao svoju fizičku memoriju.

Kada prvi put kreira virtuelnu mašinu, hipervizor definiše veličinu prostora na disku koji treba dodeliti njenom operativnom sistemu. U praksi postoje dve metode dodele prostora na disku i to *thick provisioning* i *thin provisioning* metoda [24]. U prvom slučaju, tzv. *thick provisioning* metoda podrazumeva, da se disk neophodan za virtuelnu mašinu

unapred dodeli i rezerviše na operativnom sistemu hosta. Prednost ove metode je u tome, što virtuelna mašina unapred dobija potreban prostor za skladištenje podataka, dok je nedostatak to što se taj prostor više ne može koristiti u druge svrhe. Za razliku od ove metode, *thin provisioning* metoda štedi prostor na disku i sprečava njegovo rasipanje. Cilj ove metode je da se „lažira“ pun kapacitet koji treba obezbediti operativnom sistemu virtuelne mašine, tako što se na matičnom operativnom sistemu kreira i rezerviše manja datoteka. Paralelno sa popunjavanjem unapred dodeljenog prostora, hipervizor počinje da povećava rezervisani prostor i pušta ga da raste do stvarno potrebne veličine.

Važno je istaći, da virtuelne mašine nisu svesne virtualizacije diska i isti posmatraju kao sopstveni, fizički disk. Datoteke koje se kreiraju za potrebe obezbeđivanja virtuelnog skladišnog prostora, mogu biti u nekom od sledećih formata [24]:

- VMDK (eng. *Virtual machine disk*)
- VDI (eng. *Virtual Disk Image*)
- VHD (eng. *Virtual hard disk*).

Ove datoteke predstavljaju prostor na disku, koji je rezervisan za podatke virtuelnih mašina. Međutim, ponekad se koriste i za pakovanje virtuelne mašine za transport preko hostova (sadrže operativni sistem virtuelne mašine, aplikaciju i podatke). Postoje brojni alati, koji mogu da konvertuju disk virtuelne mašine iz jednog formata u drugi. Kao alternativa pomenutoj distribuciji virtuelnih mašina, koristi se metoda pakovanja *image* fajla virtuelne mašine u ISO (eng. *International Organization for Standardization*) format datoteke, koji se lako može preneti i ugraditi kao virtuelni disk druge instance, na drugoj fizičkoj mašini. Potrebno je pomenuti i činjenicu, da se u praksi susrećemo sa različitim vrstama hipervizora, pa se zato sve više koristi otvoreni format (eng. *Open Virtualization Format* - OVF), koji je kompatibilan sa drugim formatima

Virtuelne mašine se kreiraju kao izolovana okruženja i zato je potrebno implementirati mehanizam da se povezivanje i razmenu podataka sa spoljašnjim svetom, ali i sa drugim virtuelnim mašinama koje su kreirane na istoj fizičkoj mašini. U tu svrhu, koriste se mrežne kartice (eng. *Network Interface Cards* - NIC) fizičke mašine, koje se kao i drugi resursi (npr. CPU i memorija) dele između virtuelnih mašina [1]. Postoje različite tehnike, koje mogu da obezbede deljenje mrežne kartice. Jedna od njih je kreiranje virtuelnih kartica (*virtual Network Interface Cards* - vNICs) od strane hipervizora. Ove kartice, hipervizor predstavlja operativnom sistemu virtuelnih mašina, kao da su njihove fizičke mrežne kartice. Zadatak hipervizor je da u tu svrhu izvrši odgovarajuća mapiranja na fizičkoj kartici. Međusobno povezivanje virtuelnih mašina (tj. njihovih virtuelnih mrežnih kartica), kao i povezivanje sa mrežnom karticom fizičke mašine, moguće je uraditi samo preko virtuelnog sviča (eng. *virtual Switch* - vSwitch).

3.2.4. Prednosti i izazovi primene virtuelnih mašina

Virtuelne mašine pružaju brojne prednosti, ali i pored toga je njihova primena praćena sa određenim izazovima [24]:

- performanse – u nekim slučajevima, virtuelizacija može smanjiti performanse u poređenju sa direktnim radom na fizičkom hardveru, posebno kada su u pitanju intenzivne aplikacije,
- kompleksnost upravljanja – u velikim okruženjima sa mnogo virtuelnih mašina, upravljanje resursima, bezbednošću i usklađenošću može biti složeno,
- troškovi licenciranja – korišćenje komercijalnih hipervizora i softverskih alata za upravljanje virtuelnim mašinama može biti skupo.

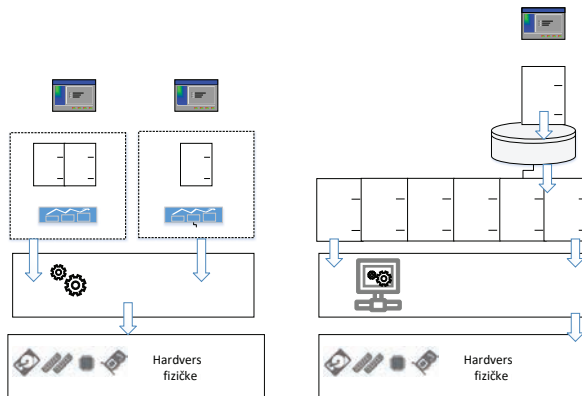
Virtuelne mašine imaju širok spektar primena u praksi. One se koriste za testiranje softvera, gde ih razvojni timovi koriste za testiranje aplikacija u različitim operativnim sistemima bez potrebe za fizičkim hardverom. U cloud okruženjima kao što su AWS, Azure i Google Cloud, korisnici mogu brzo kreirati ili ukloniti virtuelne mašine prema potrebama, omogućavajući plaćanje po osnovu korišćenja, tj. *pay-as-you-go* modela. Međutim, u najvećem broju preduzeća, virtuelizacija se koristi za konsolidaciju više servera na manje fizičkih mašina, što smanjuje troškove hardvera i energije. Jednostavno, virtuelne mašine omogućavaju lako pravljenje *backup*-a i brzo prebacivanje u slučaju kvara fizičkog servera, čime se skraćuje vreme oporavka. Ovde je potrebno naglasiti, da budućnost virtuelnih mašina, uprkos činjenici da su i dalje osnovni deo infrastrukture, zavisi od novih tehnologija kao što su kontejneri i *serverless* arhitekture. Ove arhitekture nude „lakšu“ skalabilnost i manju potrošnju resursa. Ipak, virtuelne mašine ostaju nezamenjive kada su potrebni izolacija i kompatibilnost sa starijim aplikacijama. U budućnosti, možemo očekivati dalji razvoj u smislu bolje integracije sa cloud platformama i optimizacije performansi.

3.3. RAZLIKE IZMEĐU VIRTUELIZACIJE I EMULACIJE

Konceptualno posmatrano, virtualizacija predstavlja skup tehnika kojima se raspoloživi računarski resursi organizuju u zasebna radna okruženja. Kreiranje ovih virtuelnih okruženja može se izvršiti korišćenjem hardverskog ili softverskog particionisanja, emulacijom hardvera, simulacijom softverskih procesa ili apstrakcijom resursa i spajanjem više fizički odvojenih entiteta u jednu logičku celinu. Iz ovoga se može zaključiti, da se emulacija nekada koristi tokom virtualizacije resursa, mada strogo posmatrano između ove dve tehnologije postoje značajne razlike [24].

U emulaciji, poseban softver (emulator) omogućava da se programi ili procesi izvršavaju na platformi koja je drugačija od one za koju su predviđeni. Većina emulatora ima zadatak da simulira neko hardversko okruženje arhitekturu, ali postoje slučajevi kada se i određeni softver (npr. operativni sistem) mora prilagoditi za rad u određenom okruženju [24]. Zato se emulator često posmatra kao prevodilac aplikacije, koja se pokreće na raspoloživom hardveru. U praksi, to bi značilo da hardver, koji emulator treba da simulira (CPU, memorija, disk, I/O procesi), ne sme biti ograničen na osnovni operativni sistema ili da aplikacija napravljena za jedan tip operativnog sistema ili procesora mora da radi i na drugom operativnom sistemu. Dakle, emulator radi kao aplikacija na osnovnom operativnom sistemu, koja se oslanja na mogućnosti deljenja resursa i alokacije tog operativnog sistema na drugu platformu. Pri tome, nije obezbeđen nikakav dodatni mehanizam, koji bi garantovao dostupnost ili razdvajanje resursa, kao što su *storage*-i ili I/O uređaji. Emulacija se često koristi i za pokretanje aplikacije razvijene za jedan operativni sistem na drugom operativnom sistemu ili hardveru.

Nasuprot tome, sam cilj virtualizacije je sasvim drugačiji, jer se zahteva da se omogući deljenje hardverskih resursa između različitih aplikacija ili operativnih sistema. Kao osnovne gradivne elemente, virtualizacija koristi postupak razdvajanja resursa i njihovu međusobnu izolaciju. Na slici 18. su šematski prikazane obe tehnologije i jasno se identifikuje, da emulacija predstavlja sposobnost softvera u računarskom uređaju da simulira ponašanje drugog programa ili uređaja, a virtualizacije tehniku kojom se stvaraju virtuelne instance (okruženja).

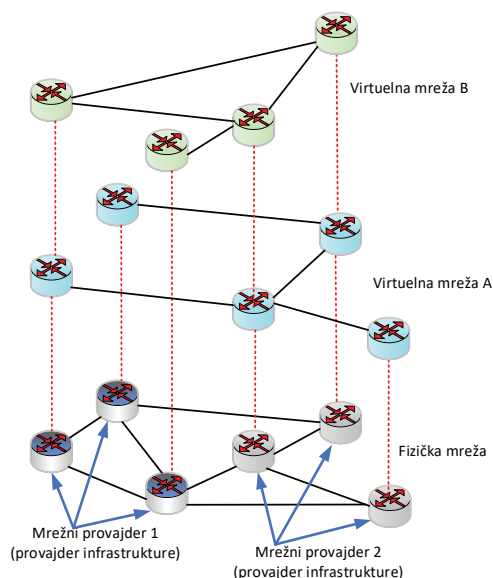


Slika 18. Šematski prikaz virtualizacije i emulacije

3.4. VIRTUELIZACIJA MREŽE

Najveći broj računarskih mreža još uvek poseduje tradicionalnu, hijerarhijski organizovanu infrastrukturu mreže. Ona je prilično heterogena i u najvećoj meri je izgrađena od uređaja različitih proizvođača. To ukazuje na kompleksnost upravljanja mrežnom infrastrukturom, posebno zbog nepostojanja jedinstvenog upravljačkog interfejsa. Jedan od razlog leži u činjenici, da većina proizvođača isporučuje opremu/uređaje, koji za povezivanje koriste sopstvene (eng. *proprietary*) protokole [24]. To usložnjava, a u nekim slučajevima i onemogućava interakciju sa uređajima drugih proizvođača. Poseban izazov za administratore tradicionalnih mreža, predstavlja realizacija korisničkih zahteva, koji se odnose na implementaciju novih i resursno sve zahtevnijih servisa (direktan rezultat razvoja i primene novih, naprednijih informaciono-komunikacionih tehnologija tehnologija). S obzirom, da je veliki broj funkcionalnosti definisan u hardveru, jasno je izražen manjak fleksibilnosti, da se adekvatno odgovori na zahteve korisnika. To utiče na povećanje cene realizacije tih servisa i održavanja postojeće računarske infrastrukture. Drugim rečima, moderni trendovi u računarstvu intenziviraju probleme povezane sa promenama u količini i tipovima mrežnog saobraćaja, zahtevima za većom dostupnošću i potrebom za paralelnom obradom velikih količina podataka.

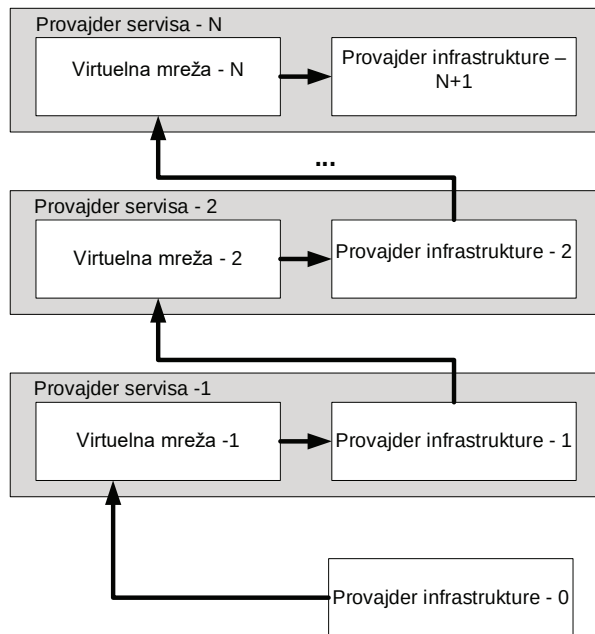
Da bi se zadovoljile potrebe različitih grupa korisnika, pored postojećih tehnika virtuelizacije u mreži (VLAN, VXLAN i *Virtual Private Network* - VPN), primenjuju se i drugi koncepti virtuelizacije računarske mreže [1]. Ovi koncepti podrazumevaju postojanje jedne softverske platforme sa centralizovanom administracijom nad virtuelizovanim mrežnim resursima. Njen zadatak je da omogući, kreiranje više virtuelnih mreža na jednoj fizičkoj mreži ili da kombinacijom virtuelizovanih resursa različitih fizičkih mreža izgradi jednu, virtuelnu mrežu (Slika 19).



Slika 19. Virtuelizacija mreže

Virtuelizacija mreže je posebno osnažena primenom SDN koncepta umrežavanja i razvojem tehnologije virtualizacije mrežnih funkcija, kao što su *firewall*, NAT (eng. *Network Address Translation*) ili DPI (eng. *Deep Packet Inspection*). Njihovom primenom se omogućava veća kontrola i fleksibilnost u mrežnom dizajnu. Primenom SDN koncepta, kontrolna ravan se odvaja od ravni podataka i time se stvaraju neophodni preduslovi za primenu centralizovanog upravljanja mrežom. To znači da se rutiranje i prebacivanje saobraćaja upravljaju putem softverskog kontrolera, umesto putem specijalizovanih mrežnih uređaja. S druge strane, virtualizacijom mrežnih funkcija, unosi se dodatna fleksibilnost i omogućava, da se prethodno pomenute mrežne funkcije pa čak i VPN-ovi implementiraju na virtuelnim mašinama ili kontejnerima, umesto na specijalizovanom hardveru. Time se dodatno smanjuje potreba za posebnim uređajima i omogućava brže postavljanje mrežnih usluga. Virtuelne (logičke) mreže se kreiraju na vrhu fizičke mreže i omogućavaju izolaciju mrežnog saobraćaja između različitih korisnika ili aplikacija.

Ovde je potrebno napomenuti, da se virtualizacijom mreže, na jednoj fizičkoj mreži može napraviti više heterogenih mreža. Ali, virtuelne mreže se mogu graditi i kombinovanjem resursa više fizičkih mreža u jednu virtuelnu mrežu i takođe vršiti prebacivanje virtuelnih mašina sa jedne na drugu mrežu (Slika 20).



Slika 20. Kombinovanje resura različitih provajdera infrastrukture [1]

3.4.1. Infrastruktura virtuelnih mreža

Infrastruktura virtuelnih mreža predstavlja kombinaciju softverskih i hardverskih elemenata koji omogućavaju kreiranje i upravljanje virtuelnim mrežnim okruženjima. To uključuje virtuelne mrežne uređaje, kao što su virtuelni ruteri i *switch*-evi, kao i alate za automatizaciju, kontrolu i monitoring mrežnog saobraćaja. Osnovne komponente ove infrastrukture su:

- virtuelni *switch*-evi – softveri unutar hipervizora koji omogućava povezivanje virtuelnih mašina u okviru iste fizičke mašine i upravljanje mrežnim saobraćajem
- virtuelni ruteri – softveri koji funkcionišu kao virtuelizovane aplikacije i omogućavaju rutiranje saobraćaja između različitih virtuelnih mrežnih segmenata
- *overlay* mreže – logičke mreže kreirane na vrhu fizičke mreže, korišćenjem različitih tehnika i protokola.

Strogo posmatrano, arhitekturu virtuelne mrežne infrastrukture obuhvata sledeće slojeve i komponente koje zajedno omogućavaju fleksibilnost i automatizaciju:

- fizički sloj (eng. *Physical Layer*) – fizička mrežna infrastruktura, uključujući rutere, *switch*-eve i servere, koja obezbeđuje osnovnu hardversku podršku za pokretanje virtuelnih mrežnih elemenata,
- sloj virtuelizacije (eng. *Virtualization Layer*) – kreira i upravlja virtuelnim mrežnim elementima (virtuelni ruteri i *switch*-evi), koristeći različite hipervizore,
- kontrolni sloj (eng. *Control Layer*) – centralizovani softverski sistem vrši upravljanje i kontrolu mrežnog saobraćaja (upravlja rutiranjem i prosleđivanjem saobraćaja na osnovu definisanih politika),
- Sloj orkestracije (eng. *Orchestration Layer*) – alati i platforme (npr. *OpenStack* ili *Kubernetes*) za automatizaciju kreiranja, skaliranja i upravljanja mrežnim resursima u skladu sa potrebama korisnika ili aplikacija.
- Aplikacioni sloj (eng. *Application Layer*) – virtuelizovane mrežne funkcije (VNF-ovi), koje obavljaju specifične zadatke unutar mreže.

3.4.2. Ključne komponente virtuelne mreže

Virtuelna mreža se sastoji od niza komponenti koje omogućavaju kreiranje, upravljanje i optimizaciju mrežnih resursa u virtuelnom okruženju. Ove komponente omogućavaju nezavisnost u radu mrežnih funkcija od fizičkog hardvera, čime se postiže veća fleksibilnost, automatizacija i skalabilnost mrežnog saobraćaja. Ključne komponente virtuelne mreže su [27]:

- virtuelni *ethernet* adapteri koji se nalaze na pojedinačnim virtuelnim mašinama
- virtuelni *switch*-evi koji povezuju virtuelne mašine
- Virtuelni ruteri koji vrše rutiranje saobraćaja između različitih virtuelnih mrežnih segmenata ili između virtuelnih i fizičkih mreža.
- servisna konzola ka spoljnim mrežama.

3.4.2.1. Virtualni *ethernet* adapteri

Virtualni *ethernet* adapteri, ili virtualne mrežne kartice, su softverski implementirani mrežni interfejsi koji omogućavaju virtualnim mašinama da komuniciraju sa virtualnim mrežama [28]. Ove komponente su ključni deo virtualizovane mrežne infrastrukture, jer virtualnim mašinama omogućavaju pristup mreži, sličan onome koji fizičke mrežne kartice omogućavaju fizičkim računarima. Virtualni *ethernet* adapteri omogućavaju virtualnim mašinama sledeće tipove komunikacija:

- *komunikacije unutar hosta* – vNIC omogućava virtualnim mašinama da komuniciraju jedni sa drugima unutar istog fizičkog servera, koristeći virtualne *switch*-eve kao međusobnu tačku povezivanja
- *komunikacije sa eksternom mrežom* – vNIC može da omogući virtualnim mašinama povezivanje sa fizičkom mrežom, preko virtualnog *switch*-a koji je povezan na fizički mrežni interfejs servera. Na taj način virtualna mašina komunicira sa drugim uređajima na internetu
- *komunikacije preko VLAN-ova* – virtualni *ethernet* adapteri mogu biti konfigurisani da pripadaju specifičnim VLAN-ovima, što omogućava logičku izolaciju mrežnog saobraćaja između različitih grupa virtualnih mašina.

Virtualni *ethernet* adapteri obezbeđuju logičku konekciju između virtualnih uređaja, ali i virtualnih i fizičkih uređaja (tzv. *uplink* portovi). Mogu se konfigurisati na više načina, u zavisnosti od potreba virtualne mreže. U tzv. *bridged mode*-u, virtualni adapter se povezuje direktno na fizičku mrežu kroz virtualni *switch* (kao da je fizički uređaj na mreži) i time omogućava virtualnoj mašini da dobije IP adresu iz istog opsega kao i fizički uređaji u mreži. Konfiguracija u *NAT mode*-u se koristi da se virtualni adapter kroz NAT, poveže sa fizičkom mrežom. Time se omogućava, da više virtualnih mašina deli jednu IP adresu, što je korisno za izolaciju virtualnih mašina i kontrolisan pristup spoljnim mrežama. Komunikacija virtualne mašine samo sa host mašinom i drugim virtualnim mašinama koje su kreirane na tom hostu, obezbeđuje se konfiguracijom virtualnih *ethernet* adaptera u tzv. *host-only mode*-u. Time, virtualna mašina nema pristupa eksternoj mreži, što je korisno za testiranje i kreiranje izolovanih mrežnih okruženja.

Drugim rečima, virtualni *ethernet* adapteri su osnovni elementi koji omogućavaju mrežnu konektivnost unutar virtualnih okruženja. Njihova fleksibilnost i sposobnost prilagođavanja različitim mrežnim scenarijima čine ih ključnim delom moderne virtualizovane infrastrukture, posebno u kontekstu *cloud computing*-a i *data* centara. Razumevanje načina na koji oni funkcionišu i poznavanje tehnologija koje ih podržavaju značajno doprinosi efikasnom upravljanju mrežom u virtualizovanim, ali i hibridnim okruženjima. Njihova bezbednost je ključna komponenta virtualnih mreža, posebno u *cloud* okruženjima. To se pre svega odnosi na tzv. *MAC spoofing* zaštitu, gde se virtualni mrežni adapteri konfigurišu da spreče promenu MAC adrese unutar virtualne mašine, što štiti od potencijalnih napada gde virtualna mašina pokušava da se predstavi kao drugi uređaj na mreži. Takođe je važna i zaštita od rada u *promiscuous mode*-u i kontrola, da li virtualni adapter može da prima sav mrežni saobraćaj ili samo onaj koji je namenjen njemu. Ovo je od velikog značaja za sprečavanje neovlašćenog presretanja mrežnog saobraćaja.

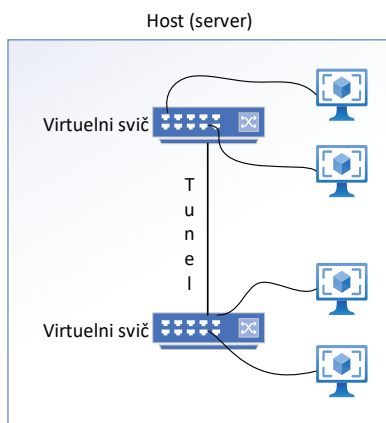
Prednosti virtuelnih *ethernet* adaptera su brojne, pogotovo kada je reč o modernim *data* centrima i *cloud* okruženjima. To se mora posmatrati sa sledećih aspekata:

- fleksibilnost u upravljanju mrežom – mogu se lako konfigurisati i prilagođavati mrežnim potrebama, omogućavajući brzo dodavanje ili uklanjanje mrežnih resursa
- smanjenje hardverskih troškova – omogućavaju virtuelnim mašinama da dele fizičke mrežne resurse, čime se smanjuje potreba za dodatnim fizičkim mrežnim karticama
- brza implementacija i skaliranje – omogućavaju brzu implementaciju novih VM-ova i njihovo povezivanje sa mrežom bez potrebe za fizičkim kablovima ili opremom.

3.4.2.2. Virtuelni *switch*-evi

Virtuelni *switch*-evi su softverski mrežni uređaji koji omogućavaju povezivanje virtuelnih mašina unutar jednog fizičkog hosta (servera) ili između više hostova (servera) u okviru virtuelnog okruženja. Oni igraju ključnu ulogu u virtuelizovanim mrežama, jer virtuelnim mašinama omogućavaju da međusobnu i sa spoljnim mrežama komuniciraju na način sličan tradicionalnim, fizičkim svičevima (Slika 21). Njihove osnovne funkcije uključuju [29]:

- prosleđivanje saobraćaja – usmeravaju saobraćaj između virtuelnih mašina na osnovu MAC adresa, koristeći tabelu sa MAC adresama za identifikaciju na koju vNIC treba preusmeriti saobraćaj
- segmentaciju mreže – koriste VLAN tehnologiju da omogućese segmentaciju mreže i logičku izolaciju između različitih mrežnih segmenata unutar virtuelnog okruženja.
- upravljanje pristupom fizičkoj mreži – omogućavaju virtuelnim mašinama pristup fizičkoj mreži preko fizičkih mrežnih interfejsa host servera, čime se omogućava komunikacija sa uređajima van virtuelnog okruženja.



Slika 21. Virtuelni *switch*

Virtuelni *switch*-evi obuhvataju širok spektar funkcionalnosti, koje su bitne za upravljanje mrežnim saobraćajem u virtuelnim okruženjima. To se pre svega odnosi na primenu VLAN tehnologije, gde se omogućava izolacija različitih aplikacija ili korisničkih grupa unutar istog fizičkog mrežnog okruženja. U virtuelnim *switch*-evima se implementira 802.1Q protokol za označavanje mrežnog saobraćaja i nesmetan protok saobraćaja različitih VLAN-ova preko istog fizičkog mrežnog linka. Obeležavanje paketa (eng. *tagging*) može se vršiti na tri načina:

- *VGT mode* (eng. *virtual guest tagging*) – unutar virtuelne mašine se instalira 802.1Q drajver i softverska podrška za transparentan prenos frejmova između virtuelne mašine i virtuelnog *switch*-a, koji obeležene pakete (eng. *tagged frames*), bez ikakve promene, propušta dalje u mrežu
- *VST mode* (eng. *virtual switch tagging*) – virtuelni svič dodaje i uklanja obeležja na mrežnim paketima, pre nego što oni budu prosleđeni virtuelnim mašinama
- *EST mode* (eng. *External switch tagging*) – dodavanje i uklanjanje obeležja se obavlja isključivo na eksternom fizičkom sviču.

QoS funkcionalnost na virtuelnim *switch*-evima, omogućava prioritizaciju mrežnog saobraćaja. To je ključno za aplikacije koje zahtevaju određeni nivo performansi, kao što su VoIP ili video konferencija. Virtuelni *switch*-evi mogu kontrolisati propusni opseg za svaku virtuelnu mašinu ili vNIC, čime se postiže optimalno korišćenje mrežnih resursa. Ovde je važno pomenuti i mogućnost replikacije mrežnog saobraćaja sa jednog na drugi port, odnosno tzv. *port mirroring* omogućava. Ovo je korisno za mrežni monitoring i analizu saobraćaja, jer administratorima omogućava da identifikuju probleme sa mrežom ili potencijalne pretnje sa aspekta bezbednosti. Naime, virtuelni *switch*-evi mogu slati kopije saobraćaja sistemima za detekciju/preventivno reagovanje na upade (eng. *Intrusion Detection/Prevention Systems*) radi poboljšanja bezbednosti.

3.4.2.3. Napredne funkcionalnosti virtuelnih *switch*-eva

Jedna od naprednih funkcionalnosti, koje se implementiraju na virtuelnim *switch*-evima je *port-group*. To je logička konfiguracija, koja se koristi za grupisanje mrežnih priključaka (portova) sa zajedničkim mrežnim podešavanjima. U kontekstu virtuelizacije, *port-group* omogućava jednostavno upravljanje mrežnim podešavanjima za virtuelne mašine koje se povezuju na virtuelni *switch*. Kada se kreirate *port-group*, pravi se mesto gde se mogu povezati virtuelne mašine sa specifičnim mrežnim parametrima [30]. Ova funkcionalnost definiše skup mrežnih parametara kao što su:

- *VLAN ID* – identifikacija VLAN-a kojem će pristupiti svi portovi povezani kroz tu *port-group*
- mrežne politike – podešavanja kao što su upravljanje propusnim opsegom (eng. *bandwidth*), definisanje pravila za bezbednost (npr. *promiscuous mode*, promene MAC adresa i sl.) i balansiranje opterećenja
- *Promiscuous Mode* – omogućava virtuelnim mašinama da "slušaju" sav saobraćaj koji prolazi kroz *port-group*, a ne samo saobraćaj namenjen njihovoj MAC adresi.

Još jedna napredna funkcionalnost, koju je moguće primeniti na virtuelnim *switch*-evima je *NIC Teaming* (eng. *Network Interface Card Teaming*). Ona omogućava agregaciju (grupisanje) više fizičkih mrežnih adaptera u jedan logički mrežni interfejs. Koristi se sa ciljem, da se postignu veće brzine mrežnog saobraćaja, balansira opterećenja i obezbedi redundansa u slučaju kvara jednog od fizičkih mrežnih adaptera. Funkcionalnost je korisna u okruženjima koja zahtevaju visoku dostupnost i bolje performanse mrežne komunikacije. *NIC teaming* funkcionalnost se u zavisnosti od mrežnih potreba može konfigurisati na nekoliko načina:

- *Active/Active Mode* – svi adapteri u timu su aktivni i koriste se za prenos saobraćaja, što omogućava maksimalnu iskorišćenost mrežnog kapaciteta (ovaj način je najefikasniji za balansiranje opterećenja, jer svi adapteri u timu učestvuju u mrežnom saobraćaju)
- *Active/Standby Mode* – jedan ili više adaptera je aktivno u svakom trenutku, dok ostali služe kao rezerva u slučaju da aktivni adapter prestane da radi (jedan od rezervnih automatski preuzima njegovu ulogu), što je dobro kada je prioritet stabilnost i otpornost na kvarove, a ne povećanje propusnosti
- *LACP* (eng. *Link Aggregation Control Protocol*) – koristi se standardizovani protokol (IEEE 802.3ad) za upravljanje grupama agregiranih linkova (eng. *link aggregation group* - LAG), koji se dinamički mogu dodavati ili uklanjati iz tima.

Uprkos prednostima koje donosi primena *NIC teaming* funkcionalnosti (veća dostupnost, povećanje propusnosti i balansiranje opterećenja), postoje i nedostaci, koji se odnose na:

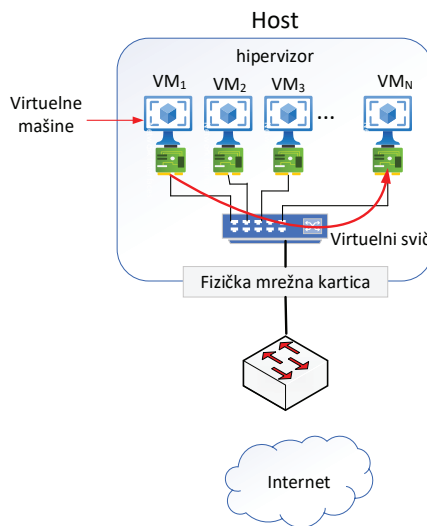
- kompleksnost konfiguracije – *NIC teaming* može biti složen za postavljanje, posebno u okruženjima sa različitim mrežnim uređajima i *switch*-vima
- kompatibilnost sa mrežnom Infrastrukturom – da bi se postigle najbolje performanse, *switch* na koji se povezuju *NIC*-ovi mora podržavati funkcionalnosti kao što je *LACP*
- trošak dodatnog hardvera – korišćenje više fizičkih adaptera može povećati troškove hardvera, posebno u velikim okruženjima.

3.4.2.4. Vrste virtuelnih *switch*-eva

Virtuelni *switch*-evi su u suštini softver, koji omogućava da jedna virtuelna mašina komunicira sa drugom. Njihovo međusobno povezivanje se vrši u skladu sa *single-tier* topologijom, pa nema petlji i nema potrebe za STP protokolom. Osnovna katakteristika *single-tier* topologije, koja se primenjuje u manjim i srednjim virtuelizovanim okruženjima za povezivanje *switch*-eva, je primena jednostavnih i efikasnih rešenja za mrežnu arhitekturu. Iako pruža mnoge prednosti, kao što su niska latencija i jednostavna implementacija, takođe postoje i nedostaci kao što su ograničena skalabilnost i činjenica, da kada glavni virtuelni *switch* otkaže, sve virtuelne mašine koje su povezane na njega prestaju da funkcionišu. Ovde je potrebno naglasiti i to, da je u virtuelnim okruženjima, *ARP* (eng. *Address Resolution Protocol*) protokol nepotreban jer:

- tehnologije kao VXLAN enkapsuliraju saobraćaj na L2 sloju preko L3 sloja, koristeći različite metode rezolucije adrese
- SDN kontroler upravlja IP-to-MAC mapiranjem bez ARP *broadcast*-a.

U praksi, virtuelni *switch*-evi mogu biti klasifikovani na softverske i hardverske virtuelne *switch*-eve, zavisno od toga kako i gde su implementirani i kojim se resursima koriste za prebacivanje mrežnog saobraćaja. Softverski virtuelni *switch*-evi (eng. *Software-Based Virtual Switches*) se obično implementiraju kao deo softverskog sloja hipervizora i koriste CPU i memoriju servera za prebacivanje mrežnog saobraćaja između virtuelnih mašina [31]. Prednosti ovakvog rešenja su pre svega fleksibilnost i prilagodljivost. Naime, softverski *switch*-evi se lako konfigurišu i prilagođavaju potrebama virtuelizovanog okruženja. Duboko su integrisani sa hipervizorom, što omogućava da se na jednostavan način upravlja mrežnim podešavanjima virtuelnih mašina. Potrebno je istaći i niske troškove implementacije, jer se oslanjaju na resurse hosta i ne zahtevaju dodatni hardver (Slika 22).



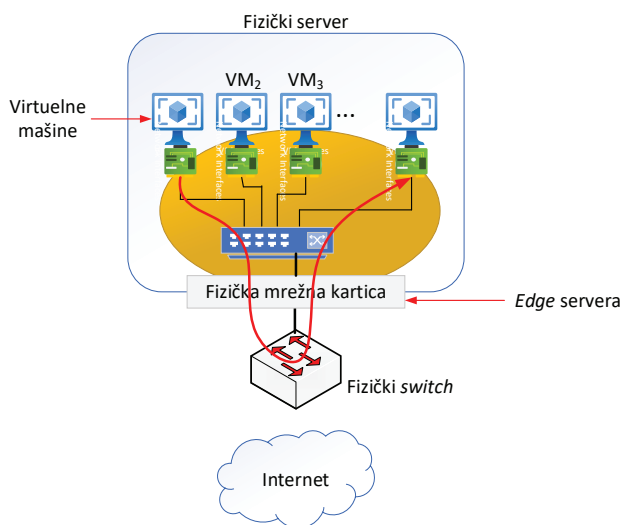
Slika 22. Softverski virtuelni *switch*

Na osnovu slike 22 i prethodno izrečenog se može zaključiti, da je softverski virtuelni *switch*, koji se implementira u softveru hipervizora, sposoban da upravlja i koordinira sa saobraćajem virtuelnih mašina i da istima obezbedi pristup (deljenje) fizičkoj mrežnoj kartici. Sama mrežna konektivnost je smeštena između host servera i *edge*-a *switch*-a, tako da mrežni adapteri servera sada upravljaju konekcijama koje su ranije postojale i između *edge* i *core switch*-eva. Razdvajanje tokova podataka se radi kreiranjem VLAN-ova.

Uprkos brojnim prednostim, softverski virtualni switch ima i određene nedostatke; koji se odnose na:

- performanse – može doći do ograničenja u performansama jer koristi CPU resurse hosta, posebno u okruženjima sa velikim protokom podataka
- opterećenje hostova – korišćenje CPU i memorije za mrežno prebacivanje može uticati na performanse ostalih aplikacija koje se pokreću na hostu.

Za razliku od softverskih, hardverski virtualni *switch*-evi omogućavaju virtuelnim mašinama, da komuniciraju između sebe i sa spoljnim mrežama, koristeći fizičke resurse mrežnog hardvera (Slika 23) [32]. Ova vrsta virtuelnog *switch*-a obezbeđuje visok nivo performansi i nisku latenciju, jer se prebacivanje mrežnog saobraćaja vrši na nivou hardvera, umesto na nivou softverskog sloja. Često se implementiraju kao deo specijalizovanih mrežnih uređaja, kao što su napredni mrežni adapteri ili fizički *switch*-evi, koji sadrže funkcije za virtuelizaciju. Za prosleđivanje saobraćaja koriste hardverske resurse, što smanjuje opterećenje CPU servera i omogućava bolju propusnost i brzinu prenosa podataka. Osim toga, ovi *switch*-evi smanjuju latenciju u komunikaciji između virtuelnih mašina, što je ključno za aplikacije osetljive na kašnjenje. U veliko broju slučajeva, oni su dizajnirani za rad u velikim okruženjima sa velikim brojem virtuelnih mašina, omogućavajući efikasno upravljanje mrežnim resursima.



Slika 23. Hardverski virtuelni *switch*

Za realizaciju hardverskih virtuelnih *switch*-eva, ogroman značaj ima SR-IOV (eng. *Single Root I/O Virtualization*) tehnologija [32], koja omogućava efikasnu virtuelizaciju I/O resursa, poboljšavajući performanse i smanjujući latenciju u virtuelizovanim okruženjima. Ključna karakteristika ove tehnologije je to što virtuelnim mašinama, omogućava da direktno komuniciraju sa fizičkim uređajem bez prolaska kroz složeni sloj softverske

virtualizacije. Ovo se postiže korišćenjem "virtuelnih funkcija" koje su mapirane na fizičke resurse. Svaka virtuelna funkcija može biti dodeljena jednoj ili više virtuelnih mašina, omogućavajući im da koriste resurse bez značajnog preopterećenja. Budući da se saobraćaj preusmerava direktno na fizički hardver, SR-IOV smanjuje opterećenje procesora i poboljšava ukupne performanse mreže, omogućavajući veće brzine prenosa podataka. U suštini, eliminacijom slojeva virtualizacije i omogućavanjem direktne komunikacije između virtuelnih mašina i fizičkog hardvera, SR-IOV smanjuje latenciju, što je posebno važno za aplikacije osetljive na kašnjenje.

Uprkos pomenutim prednostima, hardverskih virtuelnih *switch*-evi koji se baziraju na SR-IOV tehnologiju, imaju sledeće nedostatke.

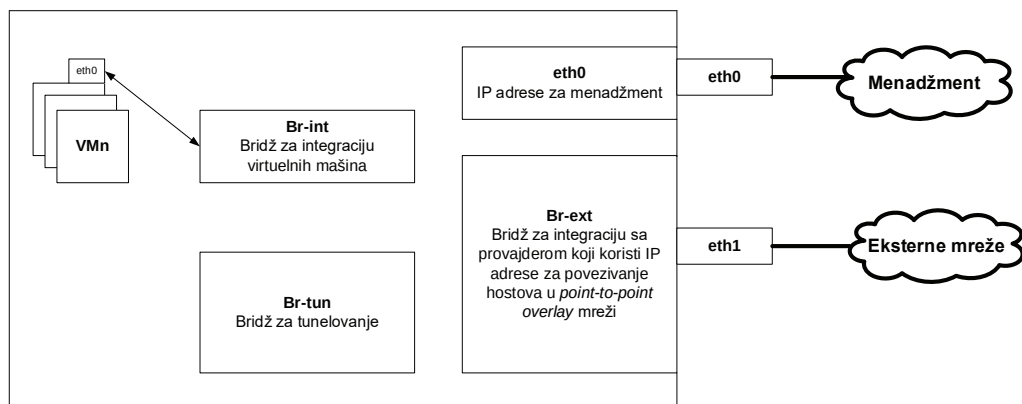
- složenija konfiguracija – postavljanje i upravljanje SR-IOV okruženjem može biti složeno i zahteva pažljivo planiranje i konfiguraciju
- ograničenje u pogledu povezivanja – broj virtuelnih funkcija koje fizički uređaj može da podrži može biti ograničen, što može predstavljati problem u velikim okruženjima
- ograničena podrška – svi mrežni adapteri ili uređaji ne podržavaju ovu tehnologiju, što može ograničiti njenu upotrebu u nekim infrastrukturnim okruženjima.

3.4.2.5. Open vSwitch

Open vSwitch (OVS) je *open-source* virtuelni svič dizajniran za upravljanje virtuelnim mrežama u skalabilni virtuelnim okruženjima. On omogućava lako povezivanje virtuelnih mašina i fizičkih servera, kao i pružanje naprednih funkcionalnosti mrežnog upravljanja. Ovo je posebno važno u data centrima, cloud okruženjima i drugim kompleksnim mrežnim infrastrukturnim postavkama. Naime, OVS omogućava virtualizaciju mrežnih resursa i omogućava dinamično kreiranje i upravljanje virtuelnim mrežama, a poseduje i napredne funkcionalnosti, kao što su podrška za standardne protokole, upravljanje kvalitetom usluge i analitika mrežnog saobraćaja. Kada je reč o podršci za standardne protokole, potrebno je naglasiti da OVS podržava standardne mrežne protokole, kao što su *OpenFlow* i VLAN, čime se omogućava interoperabilnost sa raznim mrežnim uređajima [33]. On poseduje i vlastiti API, koji a programerima i administratorima omogućava upravljanje mrežnim resursima putem skripti ili aplikacija, čime se olakšava automatizacija i integracija sa drugim sistemima. QoS funkcionalnosti se obezbežuje postavljanjem politika koje omogućavaju prioritizaciju mrežnog saobraćaja i pomažu u održavanju stabilnosti i performansi u mrežnom okruženju. U tom smislu, posebno je važno to, što OVS može da prati mrežni saobraćaj i pruža analitičke informacije (korisno za dijagnostiku i optimizaciju mrežnih performansi).

Na slici 24, prikazana je arhitektura OVS-a. Nju čine kontrolna i ravan podataka. Kontrolna ravan upravlja logikom i odlukama o usmeravanju, dok ravan podataka obavlja stvarno prebacivanje paketa. Važno je napomenuti, da OVS može raditi kao virtuelni svič u virtualizovanim okruženjima kao što su *KVM*, *Xen* i *VMware*, ali takođe može da funkcioniše kao fizički svič u tradicionalnim mrežnim okruženjima. To ga čini primenljivim u modernim *data* centrima i *cloud* infrastrukturnim okruženjima. Koristi se i u testnim i razvojnim okruženjima, kako bi se simulirale kompleksne mrežne strukture

bez potrebe za fizičkom opremom. Zahvaljujući svom API-u, OVS se može lako integrisati u alate za automatizaciju i orkestraciju, što omogućava brže i efikasnije upravljanje mrežnim resursima.



Slika 24. Open vSwitch

Uprkos prednostima, kao što su fleksibilnost, visoke performanse i činjenica da je *open-source* rešenje (OVS je besplatan i omogućava zajednici da doprinosi njegovom razvoju i poboljšanju), postoje i određeni nedostaci:

- složenost konfiguracije – iako pruža brojne funkcionalnosti, njegova konfiguracija i upravljanje mogu biti složeni, posebno za manje iskusne korisnike
- potreba za dodatnom opremom – da bi se postigle optimalne performanse, može biti potrebna dodatna oprema ili resursi, kao što su procesorska snaga i memorija.

3.4.2.6. Virtuelni ruteri

Virtuelni ruter (eng. *virtual router*) je softverski bazirana instanca rutera, koja funkcioniše unutar virtuelizovanog okruženja. Njegovom primenom se omogućava implementaciju mrežnih funkcionalnosti, koje su tipično dostupne na hardverskim ruterima, ali na virtuelnoj platformi, pružajući veću fleksibilnost, skalabilnost i ekonomičnost. Virtuelni ruteri se mogu lako kreirati, konfigurisati i upravljati, što omogućava brzu adaptaciju na promene u mrežnim zahtevima. Mogu se skalirati dodavanjem novih instanci rutiranja, bez potrebe za fizičkom opremom. Sve instance rutiranja su međusobno izolovane, svaka ima svoju konfiguraciju i svoje polise i optimalno dimenzionisane resurse [34].

Termin instanca rutiranja, koristi se za označavanje odvojenih i nezavisnih entiteta, koji upravljaju složenim usmeravanjem i izolacije mrežnog saobraćaja. To uključuje sledeće funkcije:

- segmentaciju mreže - virtuelni ruter se deli na više logičkih, izolovanih segmenata, sa sopstvenim tabelama rutiranja

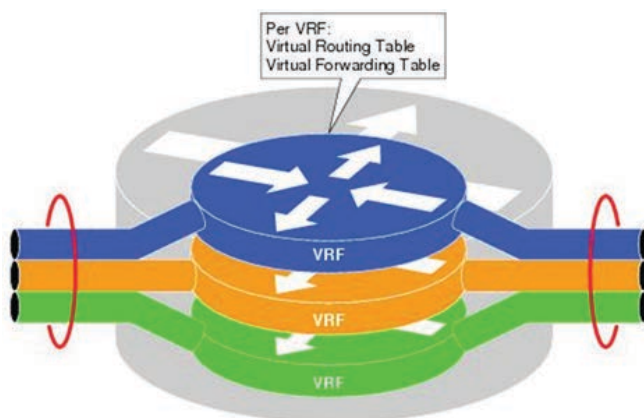
- *multitenancy* pristup – obezbeđuju se odvojene instance rutiranja za različite tenante ili organizacione celine u okviru zajedničke infrastrukture
- kontrolu polisa – primenjuju se različite politike rutiranja, bezbednosna pravila i podešavanja kvaliteta usluge za svaku instancu rutiranja
- rutiranje među instancama – olakšava se rutiranje saobraćaja između različitih instanci, ali uz kontrolisanu komunikaciju između njih.

U kontekstu mrežnog usmeravanja, vrste instanci se obično odnose na različite tipove instanci rutiranja instanci, koje se koriste za upravljanje rutama unutar mrežnih uređaja. U praksi se koristi nekoliko vrsta instanci:

- *Default Routing Instance* – koristi se kada nema specifičnih podešavanja ili pravila za određene rute (sve rute koje nisu eksplicitno dodeljene drugim instancama obično padaju u ovu instancu)
- VRF instanca (eng. *Virtual Routing and Forwarding*) – omogućava kreiranje više virtuelnih rutera unutar jednog fizičkog rutera, gde svaki ima sopstvenu tabelu rutiranja, što omogućava izolaciju saobraćaja između različitih mrežnih segmenata (često se koristi na cloudu u multitenant okruženjima)
- MPLS (eng. *Multiprotocol Label Switching*) instance rutiranja – koriste se za upravljanje rutama u MPLS mrežama. MPLS omogućava označavanje paketa za brže prebacivanje kroz mrežu, a različite MPLS instance mogu da se koriste za različite servise ili klase usluga
- BGP (eng. *Border Gateway Protocol*) instance – koriste se za upravljanje rutama između različitih autonomnih sistema (AS). Ove instance omogućavaju različite rute za različite provajdere ili klijente
- OSPF (eng. *Open Shortest Path First*) instance – koriste se za upravljanje saobraćajem između različitih oblasti (eng. *Area*) unutar autonomnog sistema
- Instance statičkog rutiranja – koriste se za upravljanje statičkim rutama
- VPN instance – omogućavaju kreiranje sigurnih, šifrovanih tunela za prenos podataka između različitih lokacija ili korisnika. Ove instance omogućavaju izolaciju saobraćaja i privatnost za korisnike
- *Application-Based Routing Instances* – koriste se za upravljanje saobraćajem na osnovu aplikacijskih zahteva ili pravila, omogućavajući optimizaciju mrežnog saobraćaja prema specifičnim aplikacijama ili uslugama.

U *data* centrima i u *cloud* okruženju se veoma često primenjuje tzv. VRF instanciranje (Slika 25). Ova tehnologija je pogodna u velikim virtuelizovanim okruženjima jer omogućava:

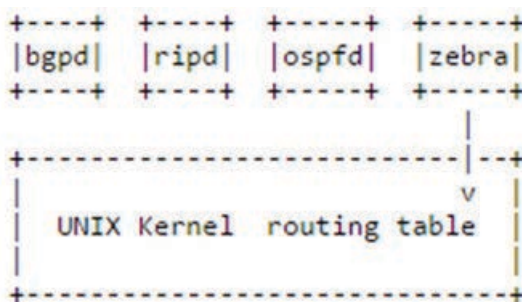
- kreiranje više instanci i tabela rutiranja i njihov simultan rad u ruteru
- izolaciju saobraćaja različitih instanci
- dinamičku alokaciju resursa
- bezbednost u mreži bez potrebe za enkripcijom i autentifikacijom
- da se za korisnike kreiraju posebni VPN-ovi pa se ova tehnologija često naziva i VPN rutiranje i prosleđivanje
- međusobnu komunikaciju različitih VRF instanci rutiranja.



Slika 25. VRF instanciranje [35]

U *Linux* okruženjima se često koristi *Quagga open-source* softverski paket koji implementira različite protokole za dinamičko rutiranje saobraćaja [36]. Njegova fleksibilnost, podrška za različite protokole i *open-source* priroda su faktori koji utiču na njegovu primenu u izgradnji efikasnih i robusnih mrežnih infrastruktura bez značajnih troškova. Naime, *Quagga* podržava nekoliko ključnih protokola za usmeravanje, uključujući OSPF, BGP, RIP i IS-IS (*Intermediate System to Intermediate System*) protokole. Posедуje modularnu arhitekturu, koja se sastoji od više *daemon*-skih procesa, od kojih svaki upravlja određenim protokolom usmeravanja. Ovo omogućava fleksibilnost i mogućnost konfiguracije različitih protokola nezavisno. Za razliku od tradicionalnih softvera za rutiranje koji obezbeđuju jedan programski proces za sve funkcionalnosti protokola rutiranja, *Quagga* osim *daemon*-a koji daju podršku za pojedine protokole rutiranja, poseduje i tzv. Zebra kernel menadžer (Slika 26). Ovaj menadžer upravlja:

- redistribucijom ruta iz različitih protokola
- promenama u kernelu tabele rutiranja i
- dodavanjem *daemon*-a za nove protokole rutiranja.

Slika 26. Arhitektura *Quagga open-source* softverskog paketa [37]

U praksi se koristi nekoliko *mode*-ova administracije *Quagga open-source* softverskim paketom. Ovi *mode*-ovi se odnosi se na različite režime rada koji omogućavaju mrežnim administratorima da konfiguriraju, upravljaju i prate različite protokole usmeravanja. *Quagga* koristi CLI (eng. *Command Line Interface*) za administraciju, a unutar ovog interfejsa postoje različiti modovi koji olakšavaju upravljanje rutiranjem. Glavni *mode*-ovi administracije su:

- privilegovani *mode* (eng. *Privileged Mode*) – administratori imaju pristup svim komandama i funkcionalnostima, uključujući mogućnost podešavanja i upravljanja različitim protokolima usmeravanja. Ovaj *mode* se obično aktivira nakon ulaska u osnovni *mode* i zahteva odgovarajuće autentifikacione informacije.
- korisnički *mode* (eng. *User Mode*) – korisnici mogu pregledati informacije o rutiranju i statusu protokola, ali nemaju mogućnost menjanja konfiguracija. Ovaj *mode* je pogodan za brzo pregledanje informacija bez rizika od neželjenih izmena.
- konfiguracijski *mode* (eng. *Configuration Mode*) – administratori mogu da menjaju konfiguraciju rutera i protokola, ali si sve izmene privremene dok se ne sačuvaju.
- protokol *mode* (eng. *Protocol Mode*) – omogućava administratorima da upravljaju specifičnim protokolima usmeravanja.
- VRF *mode* (eng. *Virtual Routing and Forwarding Mode*) – omogućava upravljanje i konfiguraciju VRF instanci. U ovom režimu, administratori mogu postavljati i upravljati izolovanim rutama za različite VRF instance

ZAKLJUČAK

Virtuelizacija mreže nastavlja da evoluirati sa tehnološkim napretkom u oblasti veštačke inteligencije (AI) i automatizacije. AI se koristi za predikciju saobraćajnih tokova i automatsku optimizaciju mrežnih resursa, dok se napredne orkestracione platforme integrišu sa SDN i NFV rešenjima kako bi se olakšalo upravljanje velikim mrežama. U budućnosti, možemo očekivati dalju integraciju virtuelizacije mreže sa konceptima kao što su *edge computing* i *Internet of Things* (IoT). Ovde je potrebno istaći, da se virtualne mreže trenutno primenjuju, od data centara do cloud i infrastrukture naprednih 5G i 6G mreža.

U *data* centrima, virtuelizacijom mreže se omogućava optimizacija resursa, brza i efikasna izolacija saobraćaja između različitih aplikacija i olakšava migracija radnih opterećenja između servera. Sa aspekta *clouda*, provajderi primenjuju ovu tehniku i koriste SDN i NFV koncepte, kako bi omogućili korisnicima da kreiraju vlastite virtualne mreže i konfiguriraju bezbednosne politike na jednostavan način. Pomenute činjenice deluju kao okidač za virtuelizaciju mreže, koja je ključan instrument za razvoj arhitekture 5G i budućih 6G mreža, je omogućavaju dinamičko upravljanje mrežnim resursima i pružanje različitih servisa kroz koncept mrežnog (eng. *network*) *slicing*-a.

Jasno je, da virtuelizacija mreže nudi brojne prednosti, ali sa njenom primenom nastaju i određeni tehnički izazovi. Osnovne prednosti su:

- fleksibilnost i agilnost – virtuelizacija mreže omogućava brzu promenu konfiguracija i postavljanje novih mrežnih servisa. Na primer, preduzeća mogu brzo kreirati virtuelne mrežne segmente za različite aplikacije ili korisnike.
- smanjenje troškova – korišćenjem standardnog hardvera umesto specijalizovanih mrežnih uređaja, kompanije mogu značajno smanjiti operativne i kapitalne troškove.
- centralizovano upravljanje – primenom SDN-a, administratori mogu centralno upravljati mrežom putem softverskog kontrolera, što olakšava implementaciju bezbednosnih pravila i optimizaciju saobraćaja.
- skalabilnost – virtuelizacija mreže omogućava skaliranje resursa prema potrebama, što je ključno u dinamičnim okruženjima kao što su *data* centri.

Kao što je prethodno navedeno, virtuelizaciju mreže osim prednosti, prate i određeni tehnički izazovi, kao što su:

- kompleksnost – upravljanje SDN i NFV okruženjima može biti kompleksno, posebno za korisnike koji nemaju iskustva sa virtuelizacijom.
- bezbednost – virtuelizacija mreže omogućava veću kontrolu, ali i uvodi nove potencijalne ranjivosti u softverskom sloju, kao što su napadi na SDN kontrolere.
- performanse – implementacija mrežnih funkcija na standardnom hardveru može izazvati pad performansi u poređenju sa specijalizovanim uređajima, posebno u slučajevima kada se zahteva velika propusna moć.



POGLAVLJE 4



KONTEJNERIZACIJA U SDN MREŽAMA

4. KONTEJNERIZACIJA U SDN MREŽAMA

4.1. UVOD U KONTEJNERIZACIJU

Programeri često imaju potrebu, da svoju aplikaciju (njen kod) postave na udaljenu lokaciju, tj. van lokalnog računara na kojem su razvili aplikaciju. Udaljena lokacija može biti fizički server u *data* centru ili virtuelni server u cloudu (npr. AWS, Azure, ili Google Cloud), kome korisnici treba da pristup putem interneta ili preko mreže. Međutim, premeštanje koda iz lokalnog okruženja na kojem je aplikacija razvijena, na produkciono okruženje gde će biti dostupna korisnicima, u nekim slučajevima je praćeno problemima tokom pokretanja. Pokretanje koda na udaljenom računaru može doneti različite izazove i probleme. Jedan od najčešćih problema sa kojima se možemo susresti tokom ovog procesa su razlike u okruženju. Na primer, udaljeni server može koristiti drugačiji operativni sistem od onog na lokalnom računaru (npr. u produkcionom okruženju je *Linux*, dok je razvoj rađen na *Windows-u*), što može uzrokovati razlike u komandama, konfiguracijama ili ponašanju aplikacije. Ponekad su problem razlike u verziji softvera (različite verzije programskog jezika, biblioteka ili zavisnosti na lokalnom računaru i udaljenom serveru) i konfiguraciji okruženja u pogledu podešavanja varijabli, *PATH* varijable ili konfiguracionih fajlova može uticati na to, da aplikacija ne radi kako je očekivano. Problemi u funkcionisanju mreže, kao što se velika latencija ili nestabilnost konekcije između lokalnog računara i udaljenog servera, takođe mogu ometati prenos fajlova ili komunikaciju sa serverom. Ako pravila na *firewall-u* ili NAT-u nisu pravilno podešena, to će otežati pristup i zato je neophodno postaviti pravilna pravila, koristiti SSH tunelovanje kako bi se osigurala bezbedna komunikacija, a potrebno je i da specifični portovi na serveru, koje aplikacija koristi, budu otvoreni.

Pored pomenutih postoje i drugi problemi, čije rešavanje programerima oduzima mnogo vremena. Zato se opravdano postavlja pitanje: kako rešiti ove probleme? Jedno od rešenja je automatizacija pomoću CI/CD (eng. *Continuous Integration/Continuous Delivery*) ili ređe *Continuous Deployment*. To podrazumeva upotrebu različitih alata (npr. *Jenkins*, *GitLab CI/CD*, ili *GitHub Actions*) i automatizaciju različitih procesa u razvoju, testiranju i isporuci softvera. CI/CD prakse omogućavaju brže i efikasnije puštanje novih verzija aplikacija, smanjujući rizik od grešaka i manualnih intervencija. Drugo, korišćenje skripti (npr. *bash skripti* ili *Dockerfile-a*) za instalaciju zavisnosti i konfiguraciju, može smanjiti greške vezane za razlike u okruženjima. Treći važan korak u rešavanju problema je njihova identifikacija pre postavljanja aplikacije u produkciono okruženje. Zato je potrebno testirati aplikacije u okruženju koje simulira uslove produkcijskog servera. Na kraju, kao rešenje se nameće i upotreba tzv. kontejnera. Kontejnerizacija primenom *Docker* tehnologije omogućava kreiranje konzistentnog okruženja, što znači da će aplikacija raditi isto na lokalnom računaru i na udaljenom serveru. Pravilno upravljanje ovim aspektima, može značajno da smanjiti rizike i poboljša iskustvo vezano za postavljanje i pokretanje aplikacija na udaljenoj lokaciji (eng. *deployment*).

Tehnika kontejnerizacije je održivo rešenje za mnoštvo problema. Ova tehnika obezbeđuje da se softverski kod isporučuje sa odgovarajućim konfiguracionim datotekama, bibliotekama i zavisnostima potrebnim za pokretanje koda na bilo kojoj

infrastrukturi. Kontejneri su virtuelni entiteti koji su efikasniji od virtuelnih mašina u pogledu prenosivosti i potrošnje resursa. Mnogi za njih koriste izraz "lagani". Svaka aplikacija može biti povezana sa operativnim sistemom bez procesa koji zahtevaju dodatne troškove, jer aplikacije dele isti kernel. Kontejneri poseduju i druge prednosti u odnosu na virtuelne mašine, kao što su manji zahtevi za kapacitetom i kraće vreme pokretanja. Ostale prednosti uključuju mogućnost smanjenja troškova servera i licenciranja.

4.2. KONCEPT KONTEJNERIZACIJE

Kontejnerizacija je metoda za pakovanje aplikacija i njihovih zavisnosti u izolovana okruženja, koja omogućavaju dosledno pokretanje i rad aplikacije, bez obzira na to gde se nalaze (na lokalnom računaru, serveru ili u cloudu). Ovakva okruženja se nazivaju kontejnerima i ona dele osnovne resurse operativnog sistema hosta (kao što je kernel), ali imaju sopstveni fajl sistem, mrežne interfejske i procesni prostor. To ukazuje na manju veličinu kontejnera u odnosu na virtuelne mašine, jer nije potrebno uključivati dodatni kernel ili upravljačke programe, što je kod virtuelnih mašina čest slučaj. Deljenjem kernela, kontejneri mogu direktno pristupati resursima hosta (CPU, RAM ili hard disk), uz zadržavanje izolovanog okruženja za aplikacije. Drugim rečima, razvoj tehnologije kontejnerizacije je u mnogome zavisi od razvoja *Linux* kernela.

U kontekstu kontejnera, moramo jasno definisati, odnosno dati odgovor na pitanje: šta se podrazumeva pod procesom? U suštini, proces je specifična instanca aplikacije ili usluge koja se izvršava unutar izolovanog kontejnerskog okruženja. Kada se pokrene kontejner, zapravo se pokreće jedan ili više procesa unutar tog kontejnera, gde svaki proces funkcioniše kao deo izolovanog sistema sa sopstvenim resursima. Ovdje se mora uzeti u obzir, da su procesi osnovni element svakog izvršnog programa, a da kontejneri pružaju okvir koji obezbeđuje izolaciju i kontrolu nad tim procesima. Proces u kontejnerima rade u izolovanom prostoru, što znači da kontejneri ne dele procesni prostor sa host sistemom ili drugim kontejnerima, osim kada je to eksplicitno dozvoljeno. Većina kontejnera pokreće jedan glavni proces (npr. za *web* server ili *mysql* za bazu podataka) i kada se taj proces završi ili se ugasi, kontejner takođe prestaje da radi.

Da bi se kreirala izolovana okruženja u kojima se pokreću procesi, koriste se funkcije ugrađene u kernelu *Linux*-a, kao što su *chroot* mehanizam, izolacija imenskog prostora i kontrolne grupe [38]. *Chroot* je mehanizam koji omogućava promenu korenskog (eng. *root*) direktorijuma (*/*) za određeni proces i njegove pod-procese. Ova promena stvara izolovano okruženje za taj proces, tako da on vidi novi korenski direktorijum kao svoj „korenski fajl sistem“. Promene u osnovnom direktorijumu prate određeni proces i impliciraju promene u svim njegovim pod-procesima, što znači, da kada se koristi *chroot* mehanizam, proces i svi njegovi pod-procesi mogu pristupiti samo fajlovima unutar novog korenskog direktorijuma (kao da je taj direktorijum ceo fajl sistem). Na primer, ako se *chroot* postavi na */home/user/chroot*, za procese unutar tog okruženja */home/user/chroot* će izgledati kao korenski direktorijum (*/*) i time se ograničava pristup fajl sistemu na jedan direktorijum. Taj direktorijum, posmatrani proces i njegovi podproces i tretiraju kao osnovni direktorijum (*/*). Ovo se često koristi za stvaranje izolovanih okruženja u svrhe testiranja softvera ili čak i za povećanje bezbednosti.

Važno je napomenuti da *chroot* mehanizam nije bezbednosna funkcija na *Linux*-u. Neki od softvera ga mogu zaobići i direktno pristupiti delovima fajl sistema. U praksi to znači, da uprkos implementaciji *chroot* mehanizma, neki procesi mogu biti u mogućnosti, da vide druge procese i informacije (npr. ID-ove korisnika i grupa), pristupe imenu hosta (eng. *hostname*) i komuniciraju sa njima. Da bi se to sprečilo, koristi se *namespace* mehanizam ili tzv. prostor imena. Ovaj mehanizam omogućava logičko grupisanje i izolaciju resursa kako bi se izbegle kolizije i olakšalo upravljanje u različitim okruženjima (operativni sistemi, mreža ili razvoj softvera). Način na koji se *namespace* mehanizam koristi zavisi od konteksta (od programskih jezika do mrežnih resursa i operativnih sistema). Generalno, *namespace* mehanizam na fleksibilan način izoluje procese, grupe procesa i njihovu međusobnu komunikaciju, tako što za svaki proces kreira *namespace* i definiše resurse kernela, kojima proces može da pristupi. Aplikacija pokrenuta pod *Linux*-om koristi procese i njihove ID-ove definisane njenim *namespace*-om, koji nisu vidljivi drugim aplikacijama. Štaviše, aplikaciji se može dodeliti poseban mrežni *namespace*. Time joj se omogućava, da ima sopstveni plan rutiranja i organizaciju mrežnog steka. To znači, da će interfejs ili ruta kreirana u aplikaciji koja koristi određeni mrežni *namespace* biti vidljiva drugim aplikacijama na istom hostu, samo ako imaju zajednički *namespace*. U kontekstu mreža, *namespace* se često koristi za organizaciju imena u DNS-u (eng. *Domain Name System*). DNS *namespace* je hijerarhijska struktura koja organizuje imena domena na internetu. Na vrhu te hijerarhije su TLD domeni (eng. *Top Level Domain*) kao što su *.com*, *.org*, i drugi, a ispod njih se nalaze drugi domeni i poddomeni.

U *Linux*-u postoje nekoliko vrsta *namespace*-ova, koji omogućavaju izolaciju različitih resursa [20], kao što su:

- *Process ID (PID) namespace* – omogućava izolaciju procesa tako da procesi unutar jednog *namespace*-a ne mogu videti procese u drugom *namespace*-u
- *Network namespace* – svaki *network namespace* ima svoje mrežne interfejse i IP adrese, što omogućava da kontejneri imaju sopstvene mrežne konfiguracije.
- *Mount namespace* – izoluje mount tačke fajl sistema, tako da svaki *namespace* može imati različitu strukturu fajl sistema.
- *Unix time-sharing (UTS) namespace* – omogućava da različiti *namespace*-ovi imaju različita imena hosta (eng. *hostname*).
- *Inter-Process Communication (IPC) namespace* – izoluje međuprocesnu komunikaciju, tako da procesi u različitim *namespace*-ovima ne mogu međusobno komunicirati putem zajedničkih IPC mehanizama.

Namespace mehanizam omogućava individualni pregled sistemskih resursa za svaki proces. Ovakav metod izolacije, daje pristup neograničenom korišćenju resursa (npr. *mount namespace* ne može ograničiti prostor na disku koji proces troši). Da bi se kontrolisala dodela resursa na nivou svakog procesa ili skupa procesa, potrebno je da se koristi mehanizam kontrolne grupe (eng. *Control group - Cgroup*). Ovaj mehanizam je ugrađen u *Linux* kernel i on omogućava da se procesi grupišu u hijerarhijsku strukturu, čime se omogućava upravljanje resursima kao što su CPU, memorija, diskovni I/O i *bandwidth* linkova. U praksi to znači, da administratori mogu koristiti ovaj mehanizam,

da bi ograničiti koliko CPU vremena ili memorije određena grupa procesa može da koristi, čime se sprečava prekomerno korišćenje resursa koje može negativno da utiče na druge procese ili aplikacije u sistemu. Štaviše, *Cgroup* omogućava praćenje potrošnje resursa po grupama procesa, što je korisno za analizu performansi i optimizaciju aplikacija. Administratori dobijaju mogućnost, da prate koliko resursa koriste različite grupe procesa i identifikuju potencijalne probleme ili uska grla. Dakle, ovaj mehanizam omogućava merenje korišćenja resursa, definiše različite prioritete korišćenja, pa čak i ograničava korišćenje resursa.

Cgroup-ovi koriste virtuelni fajl sistem (*cgroupfs*), koji administratorima, pa čak i korisnicima omogućava da konfigurišu i upravljaju *Cgroup*-ovima putem fajlova i direktorijuma. Važno je napomenuti, da su *Cgroup*-ovi organizovani u hijerarhijske strukture, što znači da se resursi mogu dodeliti grupama procesa, koje mogu imati podgrupe (eng. *subgroups*). Svaka *Cgroup* može imati aktivirane različite kontrolne mehanizme, zavisno od resursa koji se žele kontrolisati (Slika 27), kao što su:

- *cpu* – za upravljanje CPU vremenom
- *memory* – za upravljanje korišćenjem memorije ili
- *blkio* – za upravljanje I/O operacijama na blok uređajima.

Kreiranje cgroup-a:

```
bash
mkdir /sys/fs/cgroup/memory/my_cgroup
```

Postavljanje limita na memoriju:

```
bash
echo "512M" > /sys/fs/cgroup/memory/my_cgroup/memory.limit_in_bytes
```

Dodavanje procesa u cgroup:

```
bash
echo <PID> > /sys/fs/cgroup/memory/my_cgroup/cgroup.procs
```

Slika 27. Kreiranje *cgroup*-a i postavljanje limita

Na prvi pogled izgleda da su virtuelni kontejneri i virtuelne mašine slični. Međutim, sličnosti su minimalne, jer virtuelne mašine rade u hipervizorskom okruženju gde svaka virtuelna mašina ima svoj operativni sistem sa pridruženim binarnim datotekama, bibliotekama i aplikacijama. To ukazuje na ključnu razliku između njih – virtuelna mašina troši više sistemskih resursa, što je posebno vidljivo kada nekoliko virtuelnih mašina radi na istom fizičkom serveru. Za razliku od virtuelnih mašina, kontejneri su manji,

oni dele isti operativni sistem i njegov kernel. Za pokretanje kontejnera je potrebno svega nekoliko sekundi. U odnosu na virtuelne mašine, postoje sledeće razlike, koje utiču na njihovu primenu [24]:

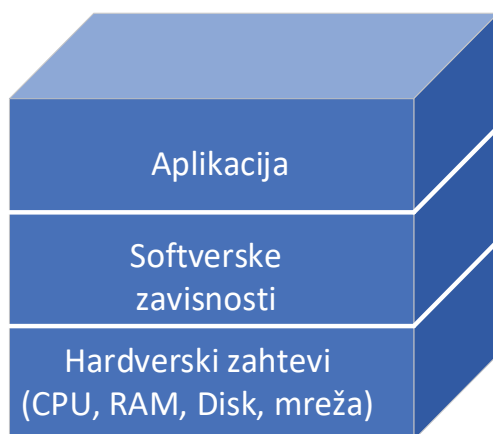
- deljenjem kernela operativnog sistema fizičke mašine (umesto korišćenja hipervizorskog softvera), virtuelizacija se implementira na nivou procesa (*namespace* i *Cgroup* mehanizmi omogućavaju pristup preko API-a kao što je *LXC runtime via API*)
- direktnim pokretanjem aplikacija na operativnom sistemu hosta se štede resursi, jer se koriste isto jezgro, binarne datoteke i biblioteke
- smanjenom potrošnjom resursa se obezbeđuje veća propusnost za operacije čitanja/pisanja memorije i bolja iskorišćenost CPU-a (obezbeđuju se performanse slične fizičkom hostu)
- neke aplikacije se ne mogu pokrenuti iz matičnog operativnog sistema
- deljenje kernela i biblioteka smanjuje bezbednost (npr. ako aplikacija kontejnera izazove pad kernela, to će uticati na sve kontejnere koji rade na fizičkoj mašini).

Kontejnerizacija se značajno razlikuje od drugih tehnika virtuelizacije, jer se umesto virtuelizacije celokupnog hardvera, kreiraju izolovana okruženja sa ograničenim pristupom sistemskim resursima na operativnom sistemu. Ograničeni pristup sistemskim resursima predstavlja osnovnu karakteristiku koja razlikuje kontejnere od virtuelnih mašina. Zato se danas virtuelni kontejneri u računarskim okruženjima koriste za [39]:

- migraciju postojećih aplikacija u modernija mrežna okruženja kao što je cloud – postojeće aplikacije se mogu preneti u kontejner uz minimalne modifikacije, stvarajući preduslove za visoku prenosivost,
- efikasnu podršku distribuiranim aplikacijama, na način da se mikroservisi mogu lakše izolovati, primeniti i skalirati,
- izgradnju, testiranje i implementaciju virtuelnih okruženja, koristeći iste image fajlove kontejnera,
- lakše raspoređivanje poslova i zadataka koji se ponavljaju, što je veoma korisno za pozadinske procese, kao što su ETL (eng. *Extracts, Transforms, and Loads data*) funkcije ili *batch* poslovi (eng. *jobs*) u automatizaciji dugotrajnih ili repetitivnih procesa koji se mogu obavljati u pozadini, bez potrebe za neposrednim nadzorom (posebno korisni u situacijama kada je potrebno obraditi veliku količinu podataka ili izvršiti kompleksne operacije).

4.3. VRSTE VIRTUELNIH KONTEJNERA

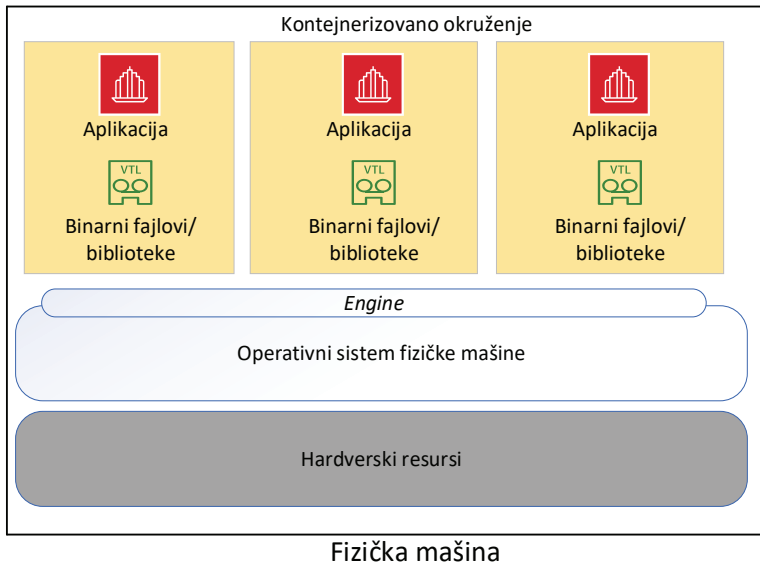
Virtualni kontejneri omogućavaju pokretanje aplikacija u izolovanom okruženju, čime se obezbeđuje bolje iskorišćenje resursa i veća portabilnost u odnosu na tradicionalne virtualne mašine. U njima se mogu instalirati i pokretati različite aplikacije, od kojih se neke izvršavaju samostalno (tzv. *standalone* aplikacije), dok su druge aplikacije u stvari operativni sistemi. Shodno ovoj činjenici, kontejneri se mogu podeliti u dve osnovne kategorije: kontejnere aplikacija (eng. *App container*) i kontejnere operativnog sistema (eng. *OS container*). Na Slici 28. je prikazana struktura kontejnera aplikacija [40]:



Slika 28. Struktura kontejnera sa aplikacijom

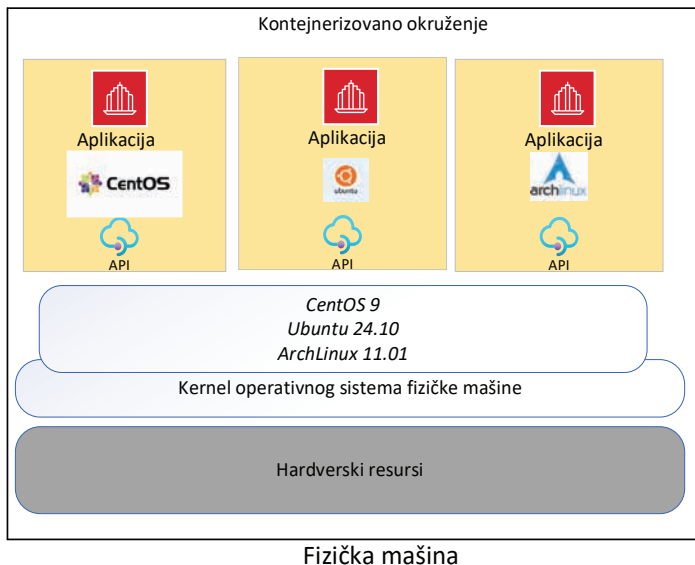
Kontejner aplikacija je samostalan, tzv. *all-in-one* paket za softversku aplikaciju. On sadrži „upakovane“ binarne aplikacije, softverske zavisnosti i hardverske zahteve potrebne za pokretanje aplikacije. Odmah, nakon instalacije u sistem, kontejner aplikacija se pokreće koristeći lokalni hardver i operativni sistem. To znači, da ovi kontejneri direktno pokreću aplikaciju, koristeći binarne fajlove i biblioteke matičnog operativnog sistema i njegovog kernela. Oni poseduju svoj *namespace* za potrebe umrežavanja, kao i tačke priključenja na disk (eng. *mount points*). Važno je napomenuti, da su kontejneri aplikacija projektovani da pokreću samo jednu uslugu.

Primenom ove vrste kontejnera, moguće je izolovati različite servise koji rade na matičnom operativnom sistemu. Tako se može napraviti grupa kontejnera aplikacija, gde svaka aplikacija obezbeđuje neki servis. Ovakav pristup arhitekturi softvera je veoma popularan i često se naziva arhitekturom mikroservisa. Svaki servis ima na raspolaganju sopstvene resurse i funkcioniše nezavisno, što znači da može zasebno da raste i bude skaliran. Štaviše, svaki servis može da se menja ili nadograđuje bez uticaja na druge servise, sa kojima može i da komunicira pomoću odgovarajućeg API-a. Dakle, modularni pristup izgradnji softverske arhitekture omogućava, da se kontejneri aplikacija, koji poseduju solidnu izolovanost, skalabilnost i mogućnost brzog oporavka, jednostavno povežu i time omogući spajanje mikroservisa u jedan softver (Slika 29) [41]



Slika 29. Virtuelno okruženje sa kontejnerizovanim aplikacijama

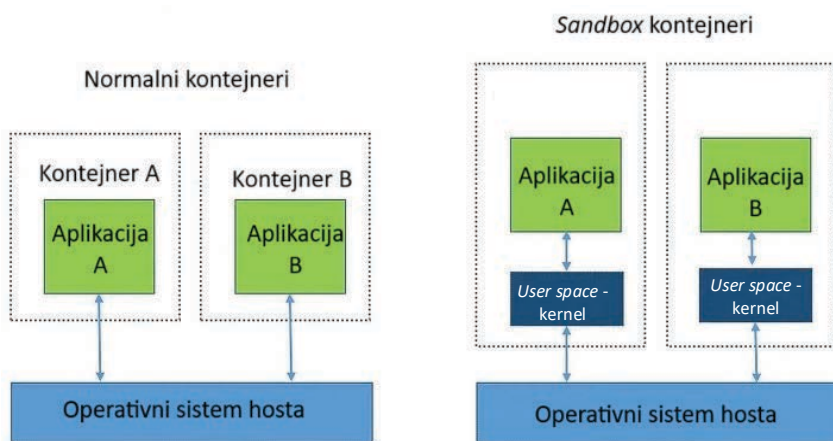
Kada je aplikacija u kontejneru operativni sistem, taj kontejner se naziva OS kontejner. Kao i kod virtuelne mašine, operativni sistem koji radi u virtuelizovanom okruženju se naziva se operativni sistem gosta i on pokreće aplikacije unutar kontejnera (Slika 30.) [24].



Slika 30. Virtuelno okruženje sa kontejnerima za operativne sisteme

Ovde je važno objasniti razlog, zbog koga se umesto direktnog pokretanja aplikacije prvo pokreće operativni sistem gosta. Neophodno je poći od činjenice, da aplikacije u kontejneru koriste kernel, biblioteke i binarne fajlove matičnog, tj. operativnog sistema fizičke mašine. Međutim, neke od njih mogu zahtevati veći stepen nezavisnosti. U praksi to podrazumeva potrebu, da se obezbedi pristup drugačijem skupu biblioteka ili sistemskih binarnih fajlova, od onog, koji se nalazi u matičnom operativnom sistemu. Zato se u kontejnerima instalira operativni sistem gosta, koji sadrži biblioteke i sistemske binarne fajlove neophodne za pokretanje određene aplikacije. Ovime se eliminiše zavisnost od matičnog operativnog sistema, ali se kernel i dalje deli između matičnog i operativnog sistema gosta. Drugi razlog, za implementaciju kontejnera za operativne sisteme, predstavlja potreba da se obezbedi bolja izolacija, bezbednost i nezavisnosti za aplikaciju u kontejneru. Takođe, važno je ukazati na činjenicu da operativni sistem u kontejneru i matični operativni sistem moraju biti kompatibilni, upravo iz razloga što dele kernel.

U praksi se mogu susresti još neke vrste kontejnera, kao što je tzv. *Sandbox* kontejner. Ovo je specijalna vrsta virtuelnih kontejnera, koji se koriste za izolaciju aplikacija i procesa u cilju poboljšanja bezbednosti. Cilj je da se obezbedi okruženje u kome se aplikacije mogu pokrenuti odvojeno od glavnog sistema, minimizirajući rizik od neželjenih efekata ili napada na ostatak sistema (Slika 31) [42].



Slika 31. Virtuelno okruženje sa *Sandbox* kontejnerima [42]

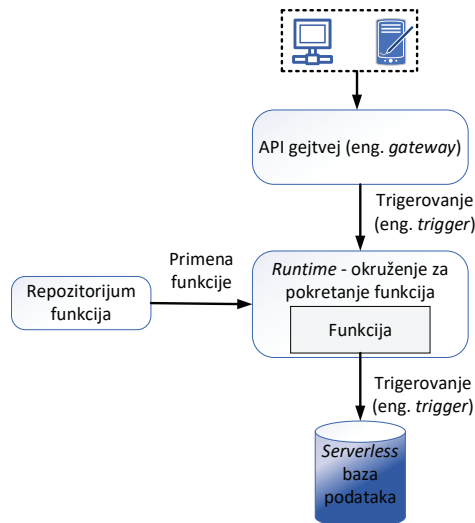
Jedna od važnih karakteristika *Sandbox* kontejnera je to, da su izolovani su od glavnog operativnog sistema. To znači da čak i ako aplikacija unutar kontejnera sadrži grešku ili ranjivost, ona neće moći lako da kompromituje ostatak sistema. Drugo, ovi kontejneri omogućavaju ograničavanje resursa kao što su CPU, memorija i mrežni pristup, što dodatno smanjuje potencijalni uticaj aplikacije na ostatak sistema. Takođe, koriste i dodatne mere sa aspekta bezbednosti, kao što su *seccomp* filteri, izolacija na nivou jezgra i virtuelizacija, kako bi poboljšali bezbednost okruženja. U realnim okruženjima se mogu sresti kao:

- *gVisor* – Google-ov alat koji implementira bezbednosni sloj između aplikacije i jezgra *Linux* operativnog sistema i simulira systemske pozive unutar kontejnera, čime se smanjuje direktan pristup aplikacije osnovnom sistemu,
- *Firecracker* – Amazonova platforma otvorenog koda optimizovana za kreiranje mikrovirtuelnih mašina i specijalno dizajnirana za *serverless* okruženja, gde je brzina pokretanja ključna,
- *Kata Containers* – kombinuju karakteristike kontejnera i virtuelnih mašina kako bi se obezbedili visoki nivo izolacije i brzina kontejnera.

Sandbox kontejneri su posebno korisni za:

- testiranje neproverenih aplikacija ili koda – programeri mogu da ih pokrenu u *Sandbox* kontejnerima bez rizika za glavni sistem,
- izolaciju aplikacija korisnika – u cloud okruženjima, aplikacije različitih korisnika mogu biti pokrenute u izolovanim kontejnerima kako bi se sprečili potencijalni napadi između korisnika,
- eksperimentisanje sa novim tehnologijama – *Sandbox* kontejner omogućava da se novi softver isproba bez uticaja na proizvodno okruženje.

U realnim okruženjima se često sreću i tzv. *Function-as-a-Service* (FaaS) kontejneri (Slika 32) [43]. Ovi kontejneri predstavljaju izolovana okruženja, u kojima se na zahtev pokreću male, specifične funkcije ili zadaci, bez potrebe za održavanjem kompletnog servera. Ova arhitektura je ključna u *serverless computing*-u, gde korisnici ne brinu o infrastrukturi već samo pišu i izvršavaju kod. Ovo FaaS kontejnere čini idealnim za obavljanje malih, kratkotrajnih zadataka, koji zahtevaju visoku skalabilnost i nisku latenciju.



Slika 32. FaaS kontejneri [43]

Ključne karakteristike FaaS kontejnera su:

- *event-driven priroda* – često se pokreću kao odgovor na specifične događaje, kao što su HTTP zahtevi, promene u bazi podataka, ili druge aplikacione notifikacije,
- kratkotrajnost – dizajnirani su da izvrše zadatak u što kraćem roku, nakon čega se „zatvaraju“ i to omogućava uštedu resursa (kontejneri nisu aktivni nakon obavljenog zadatka),
- automatska skalabilnost – omogućavaju pokretanje dodatnih instanci kontejnera po potrebi, skalirajući se prema opterećenju,
- plaćanje po osnovu korišćenja – korisnici plaćaju samo za vreme izvršavanja funkcija, što omogućava ekonomičnost u odnosu na tradicionalni hosting gde je potrebno stalno održavanje resursa.

Primeri FaaS platformi su:

- *AWS Lambda* – platforma koja podržava različite programske jezike i omogućava integraciju sa ostalim AWS servisima,
- *Google Cloud Functions* – platforma koja omogućava brzu izradu, pokretanje i skaliranje funkcija kao odgovor na događaje u *Google Cloud* ekosistemu.
- *Azure Functions* – platforma koja omogućava razvoj i pokretanje *serverless* funkcija u Azure okruženju.
- *OpenFaaS* – *open-source* rešenje za pokretanje FaaS aplikacija na kontejnerskim platformama kao što su *Kubernetes* i *Docker Swarm*.

Prednosti upotrebe FaaS kontejnera su pre svega efikasnost (pokretanje samo kada je potrebno smanjuje potrošnju resursa) i brzina razvoja (programeri mogu brže razvijati i pokretati funkcije, što ubrzava proces razvoja aplikacija). Izolacija i sigurnost koju nude FaaS kontejneri je dodatna prednost, jer smanjuje rizik od grešaka, koje bi uticale na druge delove aplikacije. Zato se ovi kontejneri često koriste za obradu podataka u realnom vremenu (npr. analiza podataka koji dolaze iz IoT uređaja), automatsko skaliranje mikroservisa (pokretanje dodatnih instanci servisa tokom opterećenja, u okviru ETL procesa i dr).

U modernim računarskim okruženjima se često koriste kontejneri dizajnirani za pokretanje aplikacija i servisa na "ivici" mreže (*edge*), bliže korisnicima, uređajima ili izvorima podataka. Za razliku od konvencionalnih kontejnera, koji se obično pokreću u centralizovanim data centrima ili cloud okruženjima, *edge* kontejneri omogućavaju obradu podataka na mrežnim čvorištima (bliže krajnjim korisnicima ili IoT uređajima) [44]. Njihove ključne karakteristike su:

- niska latencija – nalaze se bliže izvorima podataka ili korisnicima i omogućavaju brži odziv aplikacija (važno za aplikacije u realnom vremenu, kao što su video nadzor, autonomna vozila ili pametna mrežna oprema),
- smanjenje saobraćaja koji se šalje u cloud ili centralne servere, čime se smanjuje opterećenje mreže i troškovi prenosa podataka,

- skalabilnost i fleksibilnost – dinamičko dodavanje i skaliranje kontejnera u zavisnosti od potreba i specifičnosti lokacija,
- povećana bezbednost i privatnost – smanjuju potrebu za slanjem osjetljivih podataka u centralizovane *data centre*, čime se poboljšava privatnost korisnika i smanjuje rizik od napada.

Primeri upotrebe *edge* kontejnera u realnim okruženjima su:

- IoT aplikacije – prikupljanje i analiza podataka sa IoT uređaja u realnom vremenu, kao što su senzori u pametnim gradovima ili industrijskim postrojenjima
- Autonomna vozila – vozila sa autonomnim sistemima koriste *edge* kontejnere za brzo procesiranje podataka sa senzora, omogućavajući donošenje odluka u realnom vremenu
- Pametni gradovi – obrada podataka sa saobraćajnih kamera, pametne rasvete i drugih infrastrukturnih sistema u blizini lokacija gde su podaci generisani
- Medicina – telemedicinske usluge i obrada podataka sa medicinskih uređaja na lokaciji, pre slanja važnijih podataka u cloud radi dublje analize.

4.4. DOCKER PLATFORMA

Svaki virtuelni kontejner se može naći u *rest* ili *running* stanju [58]. U *rest* stanju, koje se ponekad naziva i stanjem mirovanja, kontejner se ponaša kao fajl (npr. *Container image* fajl) ili skup fajlova na disku (tzv. *Container Repository*). Da bi se kontejner pokrenuo iz ovog stanja, neophodno je startovati njegov *engine*. Zadatak *engine*-a je da raspakuje fajlove i pošalje metapodatke kernelu operativnog sistema. Kernel šalje API poziv i inicira dodatnu izolaciju i priključivanje (eng. *mounting*) kopija fajlova koji su predstavljali *Container Image*. Važno je znati, da se kontejneri nakon pokretanja ponašaju kao običan *Linux* proces.

U praksi se mogu sresti različiti *engine*-i kontejnera (LXD, CRI-O, *Docker* i dr.), koji su uglavnom zasnovani na OCI (eng. *Open Container Initiative*) standardima. Većina njih zapravo ne vrši pokretanje kontejnera, već se oslanja na alate kao što je *runc*, koji treba da [45]:

- preko API-ja pošalju odgovore na zahteve korisnika,
- omoguće raspakivanje i proširivanje *image* fajla kontejnera (*Container Image*) na disku korišćenjem tzv. *Graph* drajvera,
- pripreme tačke priključenja (eng. *mount point*) za kontejner,
- obezbede metapodatke koji će biti prosleđeni odgovarajućem *Container Runtime* okruženju, kako bi se kontejner pokrenuo.

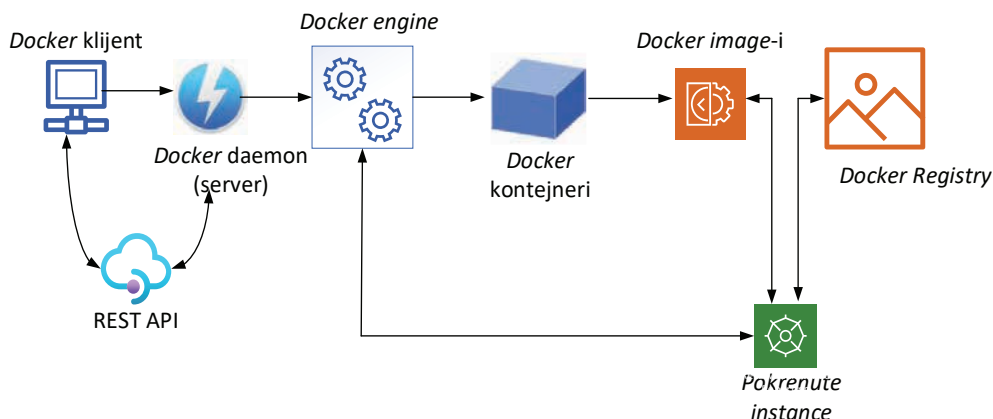
Dakle, kontejneri se mogu posmatrati kao virtuelna okruženja za izvršavanje aplikacija, koja rade na vrhu kernela operativnog sistema. Njihovom primenom se obezbeđuje izolovano okruženje i deljenje datoteka sa matičnim operativnim sistemom. Međutim, njihova replikacija i prenos sa jednog na drugi host, mogu predstavljati potencijalni izazov. Da bi se ovi procesi shvatili, mora se pravilno razumeti funkcionisanje gradivnih blokova virtuelnog kontejnera, kao što su:

- konfiguracioni fajlovi kojim se definiše njegovo okruženje,
- skup fajlova na disku koji predstavljaju njegov *namespace* za priključenje i
- izvršni fajlovi i biblioteke koje pripadaju aplikaciji.

Ovo se pre svega odnosi na činjenicu, da se u kontejneru nalaze i određene biblioteke matičnog operativnog sistema i binarni fajlovi. Zato, prenos kontejnera na drugu lokaciju u mreži nije tako jednostavan. Neki *engine*-i kontejnera, kao što je LXC, ne nude rešenje kako fajlove kontejnera i okruženje pakovati zajedno - kao grupu. Pre nego što je nastao OCI standard i sa njim i *runc* alat, LXC se koristio kao *Runtime* za kontejner, da bi se [46]:

- komuniciralo sa kernelom u momentu startovanja kontejner procesa,
- podesila *cgroup* funkcija,
- ažurirala *SELinux* polisa (bezbednosna arhitektura koja koristi labele da bi se sprovela kontrola pristupa za aplikacije, procese i fajlove na sistemu) i *AppArmor* polisa (bezbednosna arhitektura kojom se kreiranjem profila utiče na pristup pojedinih procesa kernelu),
- obezbedio pristup tačkama priključenja, u skladu sa onime što je definisao *engine* kontejnera i
- pristupilo metapodacima *engine*-a kontejnera.

Međutim, potrebe korisnika su brzo prevazišle ono što je LXC mogao da ponudi. Zato je razvijen *libcontainer* alat koji je sastavni deo *Docker engine*-a. *Docker* je počeo da se razvija 2013. godine pod imenom *dot-cloud* i on je trebao da obezbedi metod pakovanja kontejnera, koji bi bio efikasan sa aspekta prenosivosti, replikacije i kontrole verzija kontejner [2, 46]. Ideja je bila, da se *Docker* pokrene kao jedan proces (*Docker engine*), koji bi radio sa *Docker image* fajlovima. Na Slici 33. je prikazano, kako se u jedan *Docker image* fajl mogu zajedno spojiti aplikacija, potrebni binarni fajlovi i zavisnosti, koji se zatim mogu preneti i replicirati.



Slika 33. Arhitektura Docker-a

Docker je platforma otvorenog koda, koja omogućava implementaciju, pokretanje, ali i modifikaciju kontejnera. Ona omogućava upravljanje kontejnerima, kao standardizovanim i izvršnim komponentama, koje kombinuju izvorni kod aplikacije sa bibliotekama operativnog sistema i zavisnostima potrebnim za pokretanje tog koda u bilo kom okruženju. Ovakva platforma je veoma pogodna za razvoj i primenu distribuiranih aplikacija, posebno u situacijama, kada se aplikacije prebacuju na cloud ili u hibridna multicloud okruženja. Koristeći mogućnosti koje već postoje u *Linux*-u i drugim operativnim sistemima, kontejneri se mogu kreirati i bez korišćenja *Docker* platforme. Opredeljenje za primenu *Docker* platforme je lakša, brža i bezbednija kontejnerizacija, koja pruža sledeće prednosti:

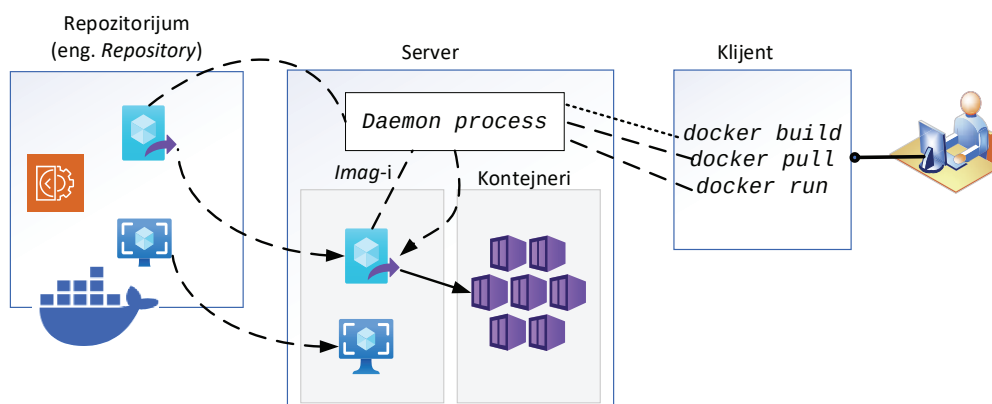
- prenosivost (eng. *portability*) – omogućava korisnicima da naprave ili instaliraju složenu aplikaciju na bilo kojoj mašini,
- automatizacija – korisnici mogu lako da automatizuju svoj rad i uštede vreme izbegavajući pojedine zadatke koji se ponavljaju uz pomoću *cron job*-a i *Docker* kontejnera,
- velika zajednica – poseduje *Slack* kanal do ogromnog broja programera na različitim lokacijama na *web*-u.

Međutim, postoje i određeni nedostaci, koji se odnose na sledeće:

- brzina – uprkos činjenici da je pokretanje aplikacije iz *Docker* kontejnera brže nego na virtuelnoj mašini, ono je i dalje znatno sporije od pokretanja aplikacija na fizičkom serveru,
- *Docker* nije namenjen za pokretanje aplikacija koje zahtevaju grafički korisnički interfejs (eng. *Graphical User Interface* - GUI) – korisnici moraju poznavati rad sa komandne linije (npr. upozorenja specifična za operativni sistem, česta ažuriranja i dr.),
- bezbednost – *Docker* radi na matičnom operativnom sistemu, što znači da zlonamerni softver, skriven u kontejnerima, može pronaći put do fizičke mašine.

Arhitekturu *Docker* platforme, čine četiri ključne komponente (Slika 34) [24], [47]:

- *Docker* klijent – zadužen za kreiranje, upravljanje i pokretanje kontejnerskih aplikacija i komunikaciju sa jednim ili više servera
- *Docker* server (*Docker daemon*) – čeka na REST API zahteve *Docker* klijenta i vrši upravljanje *image*-ima, kontejnerima, mrežom i volumenima. Da bi upravljao *Docker* servisima, jedan *Docker daemon* može komunicirati sa drugim *Docker daemon*-ima
- *Docker image* – *read-only template*, sa uputstvima *Docker* serveru, koja se odnose na kreiranje *Docker* kontejnera. Ovi fajlovi se mogu preuzeti sa *web* lokacije *Docker Hub*-a ili bilo kog skladišta *Docker image*-a, ali mogu biti i modifikovani od strane korisnika i poslati na *Docker* server
- *Docker registry* (tj. *Docker Hub*) – aplikacija otvorenog koda na strani servera koja skladišti i distribuira *Docker image* fajlove.



Slika 34. Funkcionisanje *Docker*-a

Docker platforma koristi klijent-server arhitekturu za komunikaciju. *Docker* klijent razgovara sa *Docker daemon*-om, koji obavlja poslove izgradnje, pokretanja i distribucije *Docker* kontejnera. Pri tome, klijent i *daemon* mogu da rade na istoj fizičkoj mašini ili se klijent može povezati sa udaljenim *daemon*-om. Komunikacija između ovih komponenta *Docker* platforme se odvija koristeći REST API, preko UNIX soketa ili mrežnog interfejsa. U praksi se često koristi i *Docker compose* alat, koji omogućava da se definišu uputstva za kreiranje multikontejnerskih aplikacija. Ova platforma nije jedino rešenje koje se koristi. U praksi, postoje još neka rešenja za pakovanje i prenos kontejnera. Jedno od njih je *Rocket* (Rkt) rešenje, koje poseduje mehanizam za pokretanje aplikacija u *Linux*-u. Zasniva se na otvorenoj *App Container* specifikaciji, koja definiše format *image*-a za pakovanje kontejnera. Važno je istaći, da se danas koristi i OCI format, koji svi alati za kontejnere mogu da koriste i prilagođavaju [48].

4.5. ORKESTRACIJA

Termin orkestracije se danas često pominje u modernim računarskim okruženjima, kao što su cloud, 5G ili buduće 6G mreže. Njen značaj je veliki, jer ona treba da obezbedi automatizaciju procesa alokacije resursa i implementacije usluga širom mreže i time omogući efikasno korišćenje resursa i dinamičnu, pouzdanu isporuku usluga. U širem smislu, pod orkestracijom se podrazumeva upotreba određenog skupa alata, koji su konfigurisani da obezbede optimalno izvršavanje aplikacija i upravljanje nad njihovim stanjem. Jedna od ključnih uloga orkestracije je i koordinacija aplikacija, tako da se aplikacije izvršavaju u tačno definisanom prostoru i vremenu, čime se unapređuje sistem i njegove performanse.

Često se postavlja pitanje, gde se očekuje najveći doprinos od orkestracije? Logičan odgovor je u mrežama sa velikim brojem međusobno povezanih servisa/kontejnera, kao što je npr. Google (koristi preko milion servera, a nedeljni broj pokrenutih servisa se meri u milijardama). U takvom okruženju, ručno praćenje, pojedinačno podešavanje i instalacija softvera, su praktično neizvodljivi. Iako su kontejneri po svojoj prirodi lagani i kratkotrajni, njihovo pokretanje u produkciji može predstavljati ogroman napor, posebno ako se vrši na manuelan način. To se može objasniti na primeru mikroservisa, koji su obično izolovani u posebnim kontejnerima, gde se pokretanje kontejnerizovane aplikacije može pretvoriti u upravljanje stotinama ili hiljadama kontejnera. Orkestracijom kontejnera se značajno se smanjuje operativna složenost upravljanja (automatizuje se veliki broj operacija), pa je očigledno, zašto je orkestrator glavni upravljač celog sistema. Njegova uloga se ne ogleda samo u procesu podizanja sistema i aplikacija, već i u pogledu održavanja stanja i visoke dostupnosti celog sistema.

Orkestracija kontejnera se može tretirati kao automatizacija velikog broja operativnih aktivnosti, koje su potrebne za pokretanje kontejnerskog radnog opterećenja i usluga. Spekter tih aktivnosti je veoma širok i uključuje proces upravljanja životnim ciklusom kontejnera (obezbeđivanje, primena, skaliranje, umrežavanje, uravnoteženje opterećenja i sl.). To omogućava organizacijama da iskoriste prednosti rada u kontejnerizovanom okruženju, kao što su:

- pojednostavljenje operacija - najvažnija prednost, jer kontejneri unose veliku količinu složenosti, koja može brzo i lako da izmakne kontroli bez orkestracije kontejnera,
- otpornost - alati za orkestraciju kontejnera mogu se automatski ponovo pokrenuti ili skalirati, povećavajući otpornost i
- dodatna sigurnost - orkestracija kontejnera pomaže u zaštiti aplikacija u kontejnerima, smanjujući ili eliminišući mogućnost greške izazvane ljudskim faktorom prilikom instanciranja.

Ulogu i značaj orkestracije kontejnera su prepoznale velike kompanije, a pre svih Google svojim internim *Borg* projektom (postavio temelje *Kubernetes* platforme) i *Microsoft Service Fabric* projektom [49]. Ovo nisu jedini projekti, jer pored njih postoje još i *Docker Swarm*, *Apache Mesos Marathon*, *Nomad* i dr.

4.5.1. *Kubernetes* platforma

Kubernetes je *open-source* platforma za orkestraciju kontejnera, koja automatizuje razmeštanje, skaliranje i upravljanje različitim tipovima kontejnera i njihovim aplikacijama. Karakteriše je portabilnost aplikacija i konzistentnost u njihovom radu na različitim infrastrukturama. Imajući u vidu, da kontejneri predstavljaju dobro rešenje za povezivanje i pokretanje aplikacija, veoma je važno, da se njima efikasno upravljanje u produkcionom okruženju. Zato se velika pažnja, posvećuje razvoju sistema, koji bi automatizovano upravljao kontejnerima i osigurao njihov rad bez zastoja (npr. ako se kontejner „spusti“, mora se pokrenuti drugi kontejner) [50].

Ova platforma omogućava, da se u programskom kodu kompletno opišu svi elementi jedne aplikacije spakovane u kontejnere, kao i infrastruktura potrebna za njeno funkcionisanje. Ovako koncipirana, ona obezbeđuje brojne funkcionalnosti, kao što su [51]:

- otkrivanje servisa i balansiranje saobraćaja, odnosno komunikaciju između svih servisa u sistemu na transparentan način, putem DNS zapisa,
- balansiranje saobraćaja koji se prosleđuje nekom servisu, korišćenjem različitih algoritama,
- orkestracija sistema za skladištenje podataka (sistemi za skladištenje podataka su apstrahovani), tako da aplikacije koje koriste te sisteme ne zavise od tipa sistema za skladištenje podataka,
- automatizovana ažuriranja verzija aplikacija, sa mogućnošću jednostavnog i brzog vraćanja na prethodnu verziju aplikacije u slučaju grešaka (*rollout* i *rollback*),
- optimizovano korišćenje raspoloživih resursa sistema, tj. balansiranje kontejnera u zavisnosti od potreba za zauzećem procesorskog vremena i sistemske memorije,
- automatizovani oporavak sistema u slučaju grešaka prilikom pokretanja i izvršavanja kontejnerizovanih aplikacija i
- upravljanje nad aplikativnim konfiguracijama sa mogućnošću šifrovanja, tako da se konfiguracije aplikacija mogu jednostavno i brzo iskoristiti i replicirati na veći broj različitih aplikacija (npr. parametri za pristup serveru baze podataka).

Veliki broj funkcionalnosti i distribuirana arhitektura, gde se komponente arhitekture mogu zameniti bez kritičnog uticaja na druge komponente, omogućili su da se na *Kubernetes* platformi izgrade različite softverske distribucije i platforme. Svaka *Kubernetes* distribucija predstavlja set aplikacija i servisa povezanih u celinu, čije su komponente podešene tako da zadovoljavaju potrebe korisnika. Upraksi postoji 69 sertifikovanih *Kubernetes* distribucija, od kojih neke imaju za cilj, da budu što približnije standardnoj *Kubernetes* verziji, tako što inkorporiraju dodatne funkcionalnosti (drajveri za mrežne servise, diskove i sl.) nad neizmenjenom *Kubernetes* platformom. Druge distribucije podrazumevaju prilagođavanje *Kubernetes* platforme određenim zahtevima, kao što su brzina, lakoća upotrebe, integracija sa eksternim servisima i sl. Najpoznatije *Kubernetes* distribucije su *Amazon Elastic Kubernetes Distro*, *Cisco Container Platform*, *IBM Cloud Private*, *k3s*, *Rancher Kubernetes*, *Red Hat OpenShift*, *Oracle Cloud Native Environment* i *VMware Tanzu Kubernetes*.

Brojne kompanije nude svojim korisnicima *Kubernetes* platforme kao servise, gde se apstrahuju procesi fizičke instalacije, upravljanja i održavanja *Kubernetes* klastera, a korisnicima predstavljaju samo oni servisi koji se tiču njihovih poslovnih potreba (podizanje kontejnera, podešavanje privilegija, mreže i sl.). Trenutno postoji 48 sertifikovanih vendorskih platformi, od kojih se mogu izdvojiti: *Microsoft Azure Kubernetes Service* (AKS), *Alibaba Cloud Container Service*, *Amazon Elastic Kubernetes service* (EKS), *Google Kubernetes Engine* (GKE), *Huawei Cloud Container Engine* (CCE), *IBM Cloud*, *Oracle Container Engine*, *Red Hat OpenShift Dedicated* i *Tencent Kubernetes Engine* (TKE).

U praksi, često dolazi do mešanja pojmova *Kubernetes* platforma i *Kubernetes* distribucije, pa je veoma važno da se istaknu razlike između ova dva pojma. *Kubernetes* platforma pruža osnovnu funkcionalnost orkestracije kontejnera, dok njene distribucije dodaju specifične alate i optimizacije za proizvodna okruženja, omogućavajući lakšu integraciju i održavanje u poslovnim sistemima. Ključne razlike su:

- dodatne funkcionalnosti – distribucije obično uključuju dodatne funkcionalnosti i alate koji nisu deo osnovne *Kubernetes* platforme
- prilagođenost – distribucije su često prilagođene za specifične cloud provajdere ili *on-premise* okruženja, sa optimizacijama za te platforme
- podrška i održavanje – za razliku od osnovne *Kubernetes* platforme, distribucije često dolaze sa podrškom vendara, što olakšava održavanje i rešavanje problema.

Kubernetes je inicijalnu popularnost stekao, kroz usku povezanost sa *Docker* tehnologijom. Danas se koristi i za orkestraciju kontejnera u multicloud okruženju. Ovaj vid orkestracije kontejnera, odnosi se na upotrebu alata za orkestraciju upravljanja kontejnerima u višestrukim infrastrukturnim cloud okruženjima. Za to postoje brojni razlozi, ali jedan od osnovnih su optimizacija troškova infrastrukture, fleksibilnost, portabilnost i skalabilnost (kao što je dinamičko proširenje clouda iz lokalnog okruženja, kada je to potrebno). Drugim rečima, multicloud okruženja i kontejneri se sve više koriste zbog portabilnosti i mogućnosti pokretanja u bilo kom okruženju.

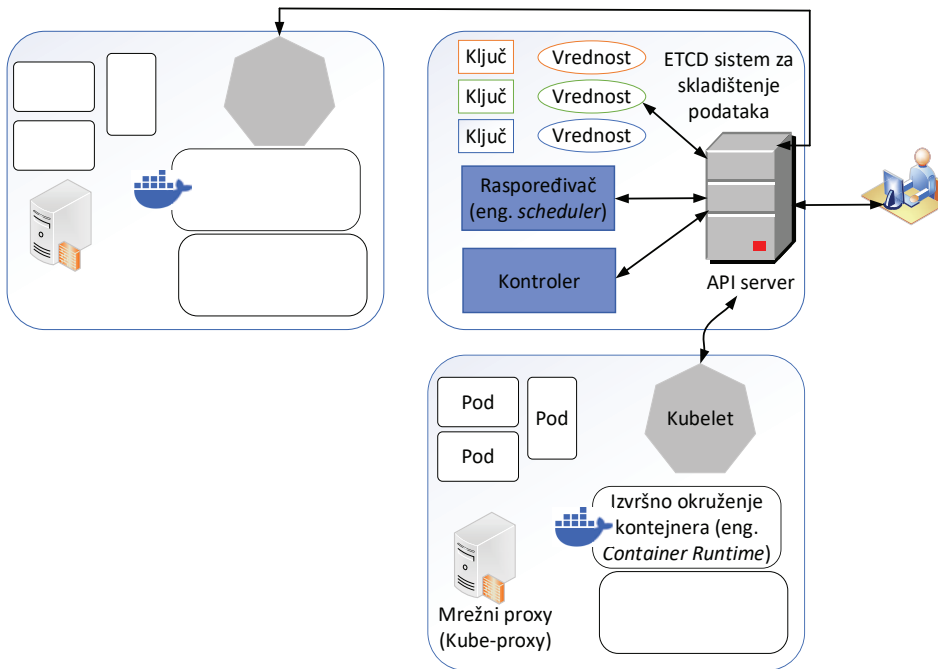
4.5.2. Arhitektura *Kubernetes* platforme

Potreba da kompleksnu aplikaciju učinimo visoko dostupnom, skalabilnom, prenosivom i primenljivom u malim modulima nezavisno, dovela je do nastanka *Kubernetesa*. Osnovu platforme čini klijent-server (eng. client-server) koncept, sa klasterom, koji se sastoji od:

- jednog glavnog čvora (eng. *Master node*), koji se brine o stanju klastera i kontejnera koji su mu povereni na izvršavanje i
- više radnih čvorova (eng. *Worker Node*) na kojima se pokreću kontejneri [52].

Arhitektura *Kubernetesa* je kompleksna i sastoji se od više komponenti. Na najvišem nivou apstrakcije se nalaze dve glavne komponente (Slika 35):

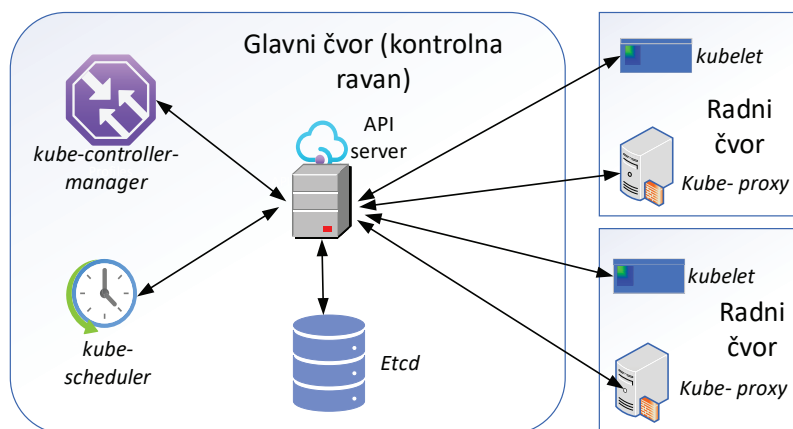
- Glavni čvor – upravlja *Kubernetes* klasterom
- Radni čvor – izvršava konkretne korisničke aplikacije.



Slika 35. Kubernetes arhitektura i komponente

Klaster u *Kubernetesu* predstavlja skup resursa (servera, *storage*-a, mrežnih resursa i dr.), koji rade zajedno i čine jednu celinu. Dizajniran je tako, da se skalira horizontalno, što znači da se resursi dodaju i oduzimaju po potrebi (u zavisnosti od opterećenja i zahteva aplikacija). Može se pokrenuti u *cloud* okruženju, na lokalnoj infrastrukturi, kao i u hibridnom okruženju (*on-premise*, *cloud* i VM). Za njegovo pokretanje je potrebno obezbediti infrastrukturu (*bare-metal* serveri, virtuelne mašine, *cloud* resursi itd.), instalirati *Kubernetes* softver i podesiti različite konfiguracije i postavke.

Glavni čvor predstavlja kontrolnu ravan (Slika 36) i ima osnovni zadatak da se brine o kontejnerima i pratećoj infrastrukturi klastera. On je glavni čvor i nadležan je za „zakazivanje“ podova (eng. *Pods*) i nadzor radnih čvorova [53], obezbeđivanje potrebnih resursa (uključuje svaki kontejner i pod), kao i pokretanje i monitoring stanja kontejnera i podova i njihovo restartovanje po potrebi. Ovaj čvor ima mogućnost, da smanji i poveća broj kontejnera u podu u zavisnosti od opterećenja, ali i da omogući njihovu međusobnu komunikaciju sa „spoljnim svetom“ (nadležan je za konfigurisanje mreže).



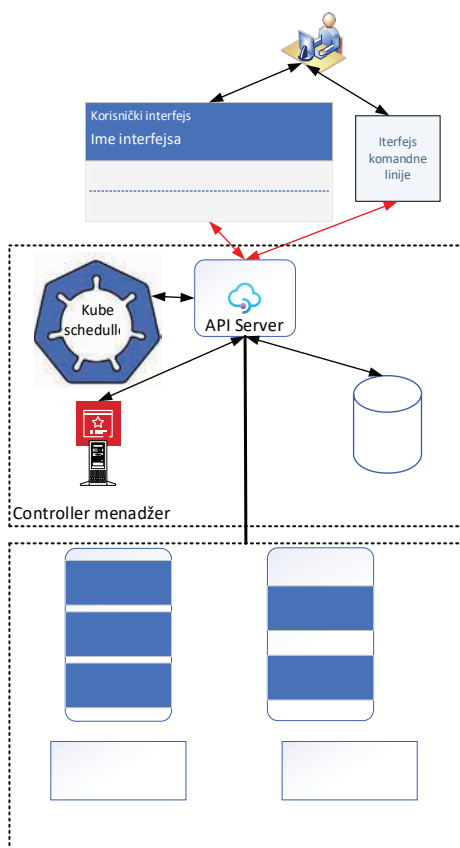
Slika 36. Kubernetes - arhitektura glavnog čvora

Na slici 36. su prikazane osnovne funkcionalnosti kontrolne ravni. Jasno se vidim da se one ostvaruju kroz izvršavanje sledećih servisa [54]:

- *kube-apiserver* – centralna komponenta kontrolne ravni, koja omogućava da se putem REST API poziva šalju zahtevi API serveru za kreiranje, ažuriranje ili brisanje Kubernetes resursa. Stanje servera se čuva u distribuiranoj *etcd* bazi podataka, što omogućava API serveru da bude servis bez stanja (eng. *stateless service*).
- *etcd* – distribuirana baza podataka, čije su osobine: jednostavan interfejs (interakcija je moguća putem HTTP protokola), hijerarhijska organizacija podataka u ključ-vrednost (eng. *key-value*) formatu, mogućnost nadgledanja izmena nad podacima i dr.
- *kube-scheduler* – ključna komponenta kontrolne ravni, koja prati nove, tek kreirane podove i bira „najoptimalnije“ radne čvorove (njegov zadatak je da proveri sve radne čvorove i utvrdi, da li su sposobni da odgovore na sve zahteve poda) na kojima će ih zakazati i pokrenuti. Nakon razvrstavanja radnih čvorova, njegov zadatak je da o svom izboru obavesti API serveru. U slučaju da ne postoji pogodan radni čvor, pod neće biti raspoređen i njegovo stanje se označava kao nerešeno (eng. *pending*).
- *kube-controller-manager* – predstavlja servis koji konstantno nadgleda klaster i odgovara na događaje koji se dešavaju. Sastoji se od više različitih, usko specijalizovanih kontrolera koji imaju tačno definisan opseg i metodu rada. Njegov konkretan zadatak je da uvidom u željeno stanje klastera i praćenjem trenutnog stanja, nakon uočenih izmena između dva stanja, klaster dovede u željeno stanje.
- *cloud-controller-manager* – opcionalna komponenta sistema koja se koristi kod kontrolera koji zavise od cloud servisa. Princip rada se svodi na prethodno navedeni *controller manager*, osim što se kontrolna petlja odnosi na resurse koji nisu u direktnoj vezi sa samom Kubernetes platformom, već zavise od cloud provajdera (npr. proces otkrivanja novih čvorova se dobija kroz cloud API).

Slično glavnom čvoru i radni čvor ima instalirane servise, koji predstavljaju ključne komponente, kojima se realizuju određene funkcionalnosti (Slika 37.) [2]:

- *Kubelet* – prati promene na API serveru i izvršava zadatke koje API server objavljuje za njega. Nakon izvršenja nekog zadatka, bilo da je rezultat uspešno ili neuspešno završen, on šalje izveštaj o statusu nazad na API server
- *Kube-proxy* – zadužen je za mrežnu komunikaciju između aplikacija koje se izvršavaju na čvorovima, kao i za mrežnu komunikaciju aplikacija sa spoljnim korisnicima
- Izvršno okruženje (*Container Runtime - CR*) – aplikacija koja upravlja životnim ciklusom kontejnera. Aplikacija je vođena specifikacijom izvršnog okruženja, kako bi pozivi bilo kojih *Runtime*-ova koji prate specifikaciju mogli biti izvršeni. Ukoliko postoji direktna implementacija nekog rešenja, unutar samog *Kubernetes* koda, održavanje celog projekta zavisi od te implementacije. Iz tog razloga, *Kubernetes* podržava više CR-ova kao što su *containerd* i *CRI-O*, a mnogi proizvođači prave i svoje rešenje.

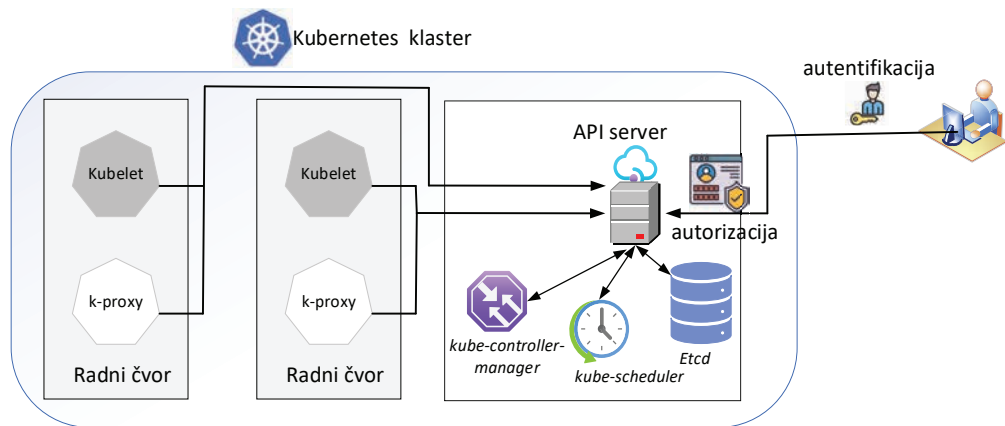


Slika 37. *Kubernetes* - arhitektura radnog čvora

4.5.3. Objekti u Kubernetes platformi

U *Kubernetes* platformi, objekat je osnovna jedinica koja predstavlja stanje neke komponente klastera. Može se posmatrati i kao zapis sa instrukcijama, kojima se API serveru „nalaže“ kako da rukuje sistemskim resursima i komunicira sa određenim komponentama. U *Kubernetes* platformi, postoje različiti objekti i svaki od njih ima određene karakteristike (npr. vrsta resursa, konfiguracija i željeno stanje). One se koriste za opisivanje aplikacija, njihovih resursa, i načina na koji bi trebalo da se ponašaju unutar klastera [55]. Strukturu svakog objekta čine dve ugnježdene celine, *spec* kojom se opisuje se željeno stanje objekta i *status* celina sa trenutnim vrednostima stanja objekta. Potrebno je napomenuti, da se u *Kubernetes* platformi mogu instancirati (pokrenuti) objekti (servisi), koji pripadaju različitim domenima. Da bi se objekti, koji pripadaju različitim domenima, međusobno izolovali i obezbedila nezavisnost u njihovm radu, koriste se različite *namespace* funkcionalnosti (npr. u *namespace* A, moguće je kreirati objekat X, ali i u *namespace* B moguće je kreirati novi objekat X, koji se suštinski razlikuje od objekta X iz A).

Upravljanje *Kubernetes* objektima se vrši kroz klijent-server interakciju prema *RESTful* API modelu. Klijenti koriste *kubectl* CLI alat ili neku aplikaciju i šalju zahteve u YAML (eng. *Yet Another Markup Language*) ili JSON formatu na API server (centralni deo kontrolne ravni u klasteru) (Slika 38).



Slika 38. Upravljanje objektima u *Kubernetes* platformi

Klijenti sa slike 38. (koriste najčešće najčešće *kubectl* ili neki drugi alat), u svojim zahtevima definišu željeno stanje aplikacija i infrastrukture, koji se mogu odnositi na kreiranje, čitanje, ažuriranje, ili brisanje *Kubernetes* objekata. Pre nego što započne obradu zahteva, API server autentifikuje korisnika i proverava njegove akreditive, da bi utvrdio da li korisnik ima dozvolu za željenu operaciju. Nakon sprovedenog postupka autorizacije, API server vrši validaciju zahteva kako bi se uverio da je zahtevan objekat ispravno definisan. U praksi to znači, da API server poseduje module za validaciju,

modifikaciju i primenu pravila prilikom obrade zahteva klijenata. Ovi moduli posreduju između autentifikacije/autorizacije i upisa u bazu podataka (*etcd*). Svaki od prijemnih kontrolera može primeniti različita pravila ili ograničenja na objekte koji se kreiraju ili ažuriraju. Zato je važno, da u objektima postoje obavezna polja, sa precizno definisanim značenjem, kao što su:

- *apiVersion* - koje označava verziju *Kubernetesa*,
- *kind* - koje označava vrstu objekta,
- *metadata* - u kome se unose vrednosti vezane za konkretan objekat, kao što su ime objekta, *namespace*, oznake (eng. *labels*) i anotacije. Oznake se koriste prilikom pronalaženja i filtriranja *Kubernetes* objekata, dok se anotacije koriste opisno i nemaju mogućnost filtriranja po njima i
- *spec* - u kome se opisuje željeno stanje.

Ako je zahtev validan, API server upisuje ažurirani objekat u *etcd* distribuiranu bazu podataka klastera. Ova baza podataka čuva sve podatke o stanju klastera u formi ključ-vrednost zapis. Kada je zahtev obrađen, API server vraća odgovor klijentu sa informacijama o statusu operacije. Važno je napomenuti, da informacije API server može poslati i drugim, zainteresovanim komponentama u klasteru. Te informacije, komponente kao što je *Scheduler*, koriste da bi izvršili potrebne akcije (npr. da dodeli pod nekom čvoru u klasteru). Kada klijent primi informacije on može da proveri stanje objekta, logove ili druge detalje da bi pratio napredak.

Gradivni blokovi svakog *Kubernetes* objekata su resursi, neophodni da bi se napravio npr. kontejner ili pod (npr. CPU, memorija i disk veličina i I/O, mrežni propusni opseg i sl.). U *Kubernetes* terminologiji, pojmovi resurs i objekat su blisko povezani, ali postoje razlike. Naime, *Kubernetes* resurs se odnosi na apstrakciju ili definiciju, koja opisuje određeni entitet unutar *Kubernetes* klastera. *Resurs* definiše tipove podataka i ponašanja u YAML ili JSON fajlovima, kao što su *pod*, *Service*, *Deployment*, *ConfigMap*, i dr. Resursi su deklarirani i predstavljaju ono što korisnik može definisati u klasteru. S druge strane, *Kubernetes* objekat je instanca resursa, odnosno konkretan entitet koji postoji unutar klastera kao rezultat primene resursa. Kada korisnik kreira objekat u klasteru, koristi resurs kao predložak za stvaranje tog objekta. Na primer, *Deployment* resurs opisuje kako bi aplikacija trebalo da bude postavljena i skalirana, dok je *Deployment* objekat instanca tog resursa koja postoji u klasteru i upravlja podovima prema pravilima definisanim u resursu. Važno je naglasiti, da je upravljanje *Kubernetes* resursima od suštinskog značaja, jer omogućava kontrolu svakog resursa koji je potreban aplikaciji, u bilo kom trenutku (npr. aplikacija pravi mnogo HTTP zahteva i treba ograničiti njenu dostupnu memoriju, tako da ne „pojede“ sav RAM i sruši ceo klaster). Takođe, *Kubernetes* resurs se može posmatrati kao tzv. *lightweight* objekat koji predstavlja jedan entitet klastera i uključuje informacije o interakciji sa tim entitetom i metapodatke kao što su njegovo ime i labela.

4.5.4. Vrste objekata u Kubernetes platformi

U praksi postoji više vrsta *Kubernetes* objekata, kao što su *Pod*, *ReplicaSet*, *Deployment*, *StatefulSet*, *Job*, *CronJob*, *Service*, *ConfigMap*, *Secret*, *Volume* i dr. *Pod* predstavlja najjednostavniji objekat u *Kubernetes* sistemu, koji se sastoji od jednog ili više kontejnera koji dele isti mrežni prostor (IP adresu) i skladišne resurse. S obzirom na ovakvu konfiguraciju, svi kontejneri unutar jednog poda moraju biti smešteni na istom radnom čvoru. Njihova međusobna komunikacija se odvija kroz lokalni mrežni prostor, bez dodatne mrežne latencije. Oni dele isti *namespace* i isti set resursa, kao što su mrežni interfejsi, IP adrese i memorija, pa ih *Kubernetes* tretira kao deo iste mrežne jedinice. Dakle, svaki pod ima jedinstvenu IP adresu i mrežne portove, preko kojih se odvija komunikacija između ostalih objekata u sistemu. Osim toga, svaki pod ima i svoje skladište podataka (eng. *storage*). Ono može biti perzistentno ili neperzistentno (privremeno). Unutar perzistentnog skladišta, podaci se ne brišu nakon gašenja poda, dok se podaci uskladišteni u privremena skladišta gube nakon gašenja poda.

Važno je istaći, da se podovi mogu nalaziti u jednom od više sledećih stanja:

- *Running* – pod je uspešno pokrenut i aktivan,
- *Succeeded* – pod je uspešno pokrenut i završio je svoje instrukcije,
- *Pending* – pod još uvek nije potpuno pokrenut,
- *Unknown* – pod se nalazi u nepoznatom stanju i
- *Failed* – pod nije uspeo da se pokrene zbog greške.

Pod se može pokrenuti kao samostalni resurs, ali se to ne preporučuje, jer se time gube sve prednosti koje nudi *Kubernetes*. Umesto pokretanja pojedinačnog poda, poželjno je koristiti koncept *ReplicaSet*-a [56]. U praksi često postoje opravdani razlozi zbog kojih je neophodno da postoji više kopija (replika) nekog poda. To su uglavnom sledeći razlozi:

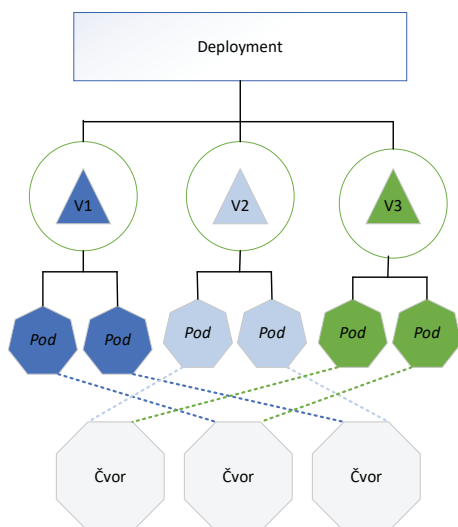
- redundantnost – postojanjem više kopija poda na različitim fizičkim serverima, u slučaju pada nekog servera, ostale kopije nastavljaju sa radom,
- skaliranje – kada postoji više kopija poda, sistem može da prihvati i obradi veći broj zahteva i
- balansiranje saobraćaja – postojanjem više kopija poda, na različitim serverima, može vršiti balansiranje saobraćaja i time doći do ravnomerne opterećenosti servera.

Za replikaciju poda u *Kubernetes* sistemu je nadležan *ReplicaSet* objekt. On predstavlja proces koji omogućava kreiranje i pokretanje više kopija istog poda. Podovi mogu biti grupisani u jednu celinu koja se može lako referencirati, a pored njihovog grupisanja, *ReplicaSet* vodi računa i broju kopija svakog poda. Naime, u slučaju nastanka greške, kada pod prelazi u *failed* stanje, broj definisanih kopija se smanjuje i dolazi do situacije da sistem nije u željenom stanju. Da bi sistem vratio u definisano (prvobitno) stanje, *ReplicaSet* će kreirati i pokrenuti novi pod. Isto važi i u slučaju da se konfiguracija *ReplicaSet* objekta promenila, tj. da je definisani broj kopija poda manji od broja trenutno pokrenutih kopija – određeni broj podova će biti zaustavljen, kako bi se zadovoljilo definisano stanje.

Unutar *spec* sekcije *ReplicaSet* objekta, korišćenjem selektor objekta, biraju se podovi sa odgovarajućim labelama, kojima će se upravljati. Na ovaj način, moguće je da se određeni podovi tokom rada izoluju (uklone iz *ReplicaSet*-a) ili da se u postojeći *ReplicaSet* dodaju već postojeći podovi koji su definisani izvan njega. Važno je istaći, da se izolacija određenog poda koristi i kada se želi ispitati njegovo stanje, a da se pri tome ne utiče na rad ostalih podova. Prednost kod dodavanja postojećih podova u *ReplicaSet* se ogleda u mogućnosti pravljenja dodatnih kopija bez potrebe za stopiranjem rada poda koji se dodaje. Dakle, zadatak *ReplicaSet*-a je, da vodi računa o broju replika. Iako veoma koristan, *ReplicaSet* nije idealno rešenje za postavljanje aplikacija, jer ga nije moguće pravilno ažurirati. Naime, on je vezan za određeni image i njegovu verziju, tj. nakon ažuriranja *ReplicaSet* objekta, on se obriše i napravi se novi, tako da je stanje podova izgubljeno [57].

Deployment je objekat najvišeg nivoa za upravljanje aplikacijama. Njegova uloga je da omogući automatsko upravljanje i održavanje skupa instanci (podova) aplikacije. U praksi to znači, da je *Deployment* treba da obezbedi (Slika 39):

- automatsko skaliranje skaliranje aplikacije, tako što se na lako dodaju ili uklanjaju instance aplikacije (podovi), kako bi se ista prilagodila opterećenju,
- postepeno ažuriranje aplikacija (eng. *rolling updates*), bez prekida u radu aplikacije (npr. kada se objavi nova verzija aplikacije, *Deployment* menja jedan po jedan stari pod sa novim i obezbeđuje da aplikacija uvek ostane dostupna)
- povratak na prethodnu, stabilnu verziju aplikacije (eng. *rollback*), u slučaju problema sa ažuriranjem nove verzije
- automatsko pokretanje novog poda, ukoliko neki od postojećih podova postane nestabilan ili prestane da radi (eng. *self-healing*), kako bi se održao željeni broj instanci.



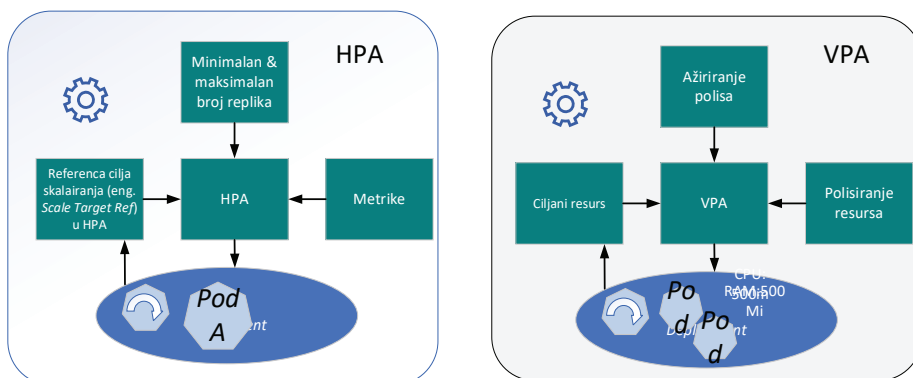
Slika 39. Prikaz rada *Deployment* objekta

Još jedan važan zadatak *Deployment*-a je, da u pozadini upravlja *ReplicaSet*-ovima. To znači, da kroz *Deployment* objekt, *ReplicaSet* dobija funkcionalnosti replike i direktnog rada sa podovima (u slučaju prestanka rada nekog poda unutar *Deployment*-a, zadatak *ReplicaSet*-a je da podigne novi pod).

Glavna namena *Deployment* objekta je da obezbedi jednostavan prelazak sa jedne na drugu verziju aplikacije. To se ne odnosi samo na ažuriranje aplikacije novom verzijom, već i na vraćanje na prethodne verzije aplikacije (npr., ukoliko je uočena greška u nekoj novoj verziji, moguće je uraditi povratak na prethodnu verziju). Metod prelaska sa jedne na drugu verziju aplikacije, naziva se strategijom [58]. U zavisnosti od zahteva korisnika, u praksi se koristi više strategija, od kojih svaka ima svoje prednosti i mane. Najčešće korišćene strategije su:

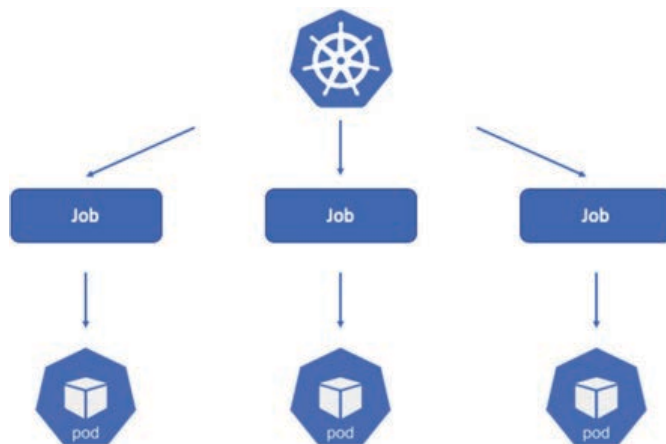
- *Recreate* – najjednostavnija strategija kod koje se sve instance podova ugase, a nakon toga se pokrenu nove.
- *Ramped* – podrazumevana strategija u *Kubernetes* sistemu (poznata kao *rolling update*), gde se nakon početka ažuriranja *Deployment*-a, pokreće i novi *ReplicaSet* (sa novom verzijom aplikacije). Broj replika se u novom *ReplicaSet*-u kontrolisano povećava, a u starom smanjuje. Nakon što broj replika u starom *ReplicaSet*-u padne na nulu, a u novom dostigne željeni broj, stari *ReplicaSet* se briše i ostaje samo novi.
- *Blue/green* – ima za cilj, da se simultano kreiraju dva okruženja, plavo i zeleno. Nakon testiranja novog okruženja (plavog), na *load balancer*-u se saobraćaj rutira sa starog (zelenog) okruženja na novo okruženje.
- *Canary* – deo korisnika se preusmerava na novu verziju okruženja, dok ostatak korisnika pristupa staroj verziji (udeo novog i starog okruženja je moguće menjati).

U praksi često postoji potreba za skaliranjem broja replika podova i automatskim podešavanjem njihove konfiguracije na osnovu merenja resursa koje podovi trenutno koriste (Slika 40) [59]. Za automatsko skaliranje broja replika je zadužen *Horizontal Pod Autoscaler* (HPA) objekt, koji povećava ili smanjuje broja podova, u skladu sa trenutnim stanjem sistema (npr. podacima o zauzetosti procesora, sistemske memorije i ostalih resursa). Zadatak HPA je da obezbedi dovoljno resursa za izvršavanje aplikacija, tako da nema prekomernog trošenja resursa i degradacije performansi. Za automatsko podešavanje konfiguracije podova se koristi *Vertical Pod Autoscaler* (VPA). Njegov zadatak je da kroz tzv. *Custom Metrics* API dobije uvid u trenutno korišćenje resursa i prilagodi korišćenje CPU i RAM resursa na zahtevane vrednosti za određeni pod. Time se smanjuje preveliko i neefikasno korišćenje resursa, zastoje u aplikacijama i „veliki“ računi na cloudu. VPA se može konfigurisati i da koristi različite algoritme za skaliranje, kao npr. *predictive modeling*, ali i da funkcioniše samo na određenim podovima i u određenom *namespace*-u.



Slika 40. HPA & VPA u Kubernetes-u [59]

Za izvršavanje zadataka na *Kubernetes* platformi je zadužen *Job* objekt [60]. Njegov zadaci su pokretanje aplikacije koja unosi inicijalne podatke u sistem, pokretanje integracione skripte za povezivanje određenih delova sistema i sl. *Job* objekat u *Kubernetes*-u instancira pod. Jedna od njegovih glavnih karakteristika je da se nakon uspešnog izvršavanja, on sam zaustavlja, odnosno prelazi u stanje *completed*. Usko posmatrano, ovaj objekat se koristi za izvršavanje jednokratnih zadataka na klasteru. Od njega se očekuje da obezbedi pouzdan i deterministički način izvršavanja zadataka, bez potrebe za ručnim upravljanjem. U zavisnosti od potreba korisnika, *Job* pruža mogućnost da se zadatak izvršava ne samo serijski, već i paralelnom modu. *Job* objekti se paralelno pokreću, kroz podešavanje atributa *completions* i *parallelism* unutar samog objekta. Kroz atribut *completions* se podešava potreban broj uspešnih pokretanja, dok se konačno ne prebaci u stanje *completed*. Vrednost atributa *parallelism* označava broj podova koje je potrebno paralelno pokrenuti (Slika 41) [61].



Slika 41. Job objekti u Kubernetes platformi [61]

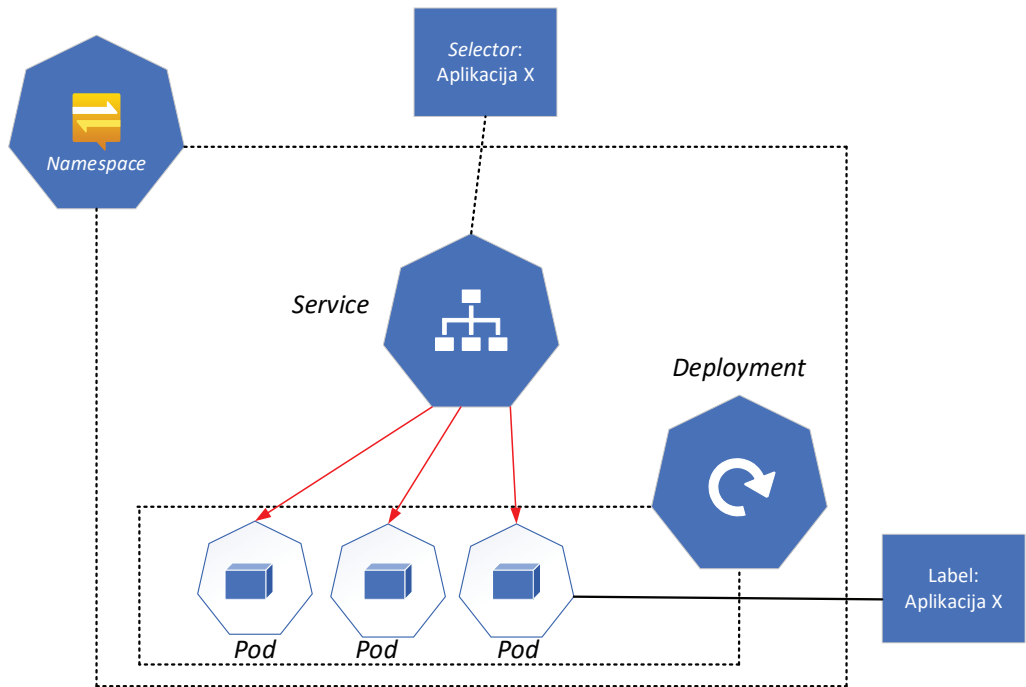
Iako *Job* objekat ima široku primene, sve više se postavljaju zadaci kojima se zahteva da se definiše vreme izvršavanja i mogućnost ponavljanja. Zahvaljujući *CronJob* objektu, moguće je implementirati funkcionalnosti kao što su: pokretanje aplikacije koja vrši *backup* podataka u tačno definisanom vremenskom trenutku ili pokretanje aplikacije koja periodično ubacuje sveže podatke na *cache* server [60]. *CronJob* objekat u pozadini instancira *Job* objekat, tako da on ima sve funkcionalnosti, kao što je mogućnost paralelnog pokretanja više podova i postizanje broja uspešnih pokretanja. U suštini, *CronJob* se može posmatrati i kao mehanizam za zakazivanju poslova koji se u *Kubernetesu* pokreću periodično prema unapred definisanim vremenskim intervalima, sa ciljem da se:

- omogući automatizovana obrada podataka, periodično čišćenje skladišta, generisanje izveštaja i dr.
- definiše vremenski raspored i specifikacija posla
- u specifikaciji posla definiše *Docker* kontejner i komanda koja se izvršava u okviru kontejnera.

Usmeravanje saobraćaja ka logičkim skupovima podova ili nekim eksternim resursima je još jedna važna funkcionalnost na *Kubernetes* platformi. U tu svrhu se koristi *Service* objekat [2], koji je nastao iz potrebe da svaki pod ima IP adresu, koja se nakon svakog novog pokretanja može menjati. Ovaj problem najviše dolazi do izražaja, kod onih podova koji zavise od mrežne komunikacije sa drugim podovima. Drugim rečima, *Service* je *Kubernetes* objekat koji definiše stalnu IP adresu i DNS ime za grupu replika aplikacije. On funkcioniše na način da se:

- drugim aplikacijama na klasteru omogućava komunikacija sa grupom replika kao sa jedinstvenom entitetom (bez obzira koliko replika trenutno radi),
- balansira opterećenje i zahtevi za pristup aplikaciji raspoređuju na različite replike (da bi se obezbedile optimalne performanse) i
- obezbeđuje bezbednost komunikacije enkripcijom i verifikacijom identiteta.

Unutar *Service* objekta se definiše *selector* atribut, preko koga se kreira logički skup podova na koje će *Service* objekat referencirati. U atributu *ports* se definiše protokol (TCP, UDP, SCTP, HTTP, PROXY), preko koga će se prosleđivati saobraćaj, krajnja tačka komunikacije koja će biti dostupna ostalim servisima unutar sistema i krajnja tačka komunikacije (eng. *target port*) na koju će biti prosleđen saobraćaj (Slika 42).



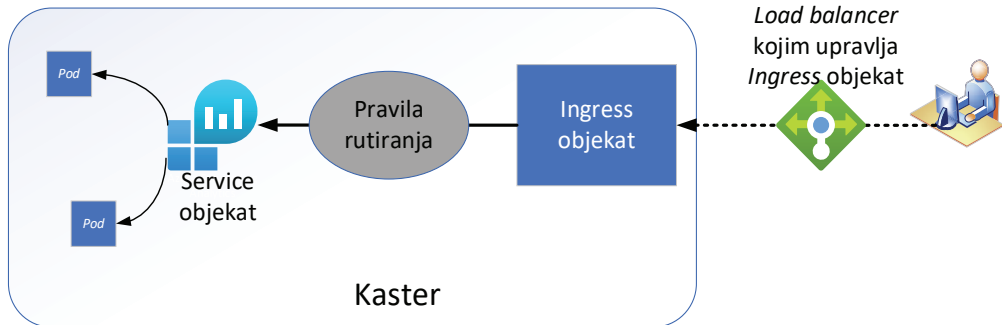
Slika 42. Primena Service objekta na Kubernetes platformi

Ukoliko se u *Service* objektu ne navede *selector* atribut, logički skup podova neće biti definisan, a sav saobraćaj biće preusmeren na mrežnu lokaciju koja je definisana kroz *ports* atribut i *Endpoint* objekat (ovaj objekat je potrebno ručno napraviti). Instancirani *Service* objekat dobija virtuelnu IP adresu, čija vrednost zavisi od vrednosti polja *atribut type*. Ovaj atribut može imati jednu od sledećih vrednosti [62]:

- *ClusterIP* – podrazumevana vrednost, ukoliko *atribut type* nije definisan. *Service* objekat dobija internu IP adresu iz opsega klastera, što znači da joj može pristupiti samo *resurs* iz internog klastera.
- *NodePort* – *Service* instanca dobija adresu *Kubernetes* čvora i nasumičnu krajnju tačku komunikacije. Jasno je, da je tada *Service* instanca je vidljiva na mreži i van *Kubernetes* sistema. Takođe, *Service* instanca se propagira na sve čvorove sistema, pa iz tog razloga nije potrebno znati na kom se čvoru nalaze podovi na koje *Service* referencira. Ovaj proces je moguć, jer *kube-proxy* servis prosleđuje saobraćaj na pravi čvor na kome se pod nalazi i na krajnju tačku komunikacije.
- *LoadBalancer* – u ovom slučaju, *Service* dobija IP adresu van *Kubernetes* sistema, odnosno, mrežna komunikacija ka logičkom skupu podova na koje *Service* referencira će biti ostvarena izvan *Kubernetes* sistema (ovo je često javna IP adresa koja je dostupna preko interneta).

- *ExternalName* – specijalni slučaj *Service* objekta, gde se umesto *selector*-a koriste DNS zapisi, koji najčešće referenciraju servise van *Kubernetes* sistema.

U praksi je često potrebno, da se omogući spoljašnji pristup aplikacijama na klasteru. U tu svrhu, na *Kubernetes* platformi se uglavnom koristi *Ingress* objekat. On treba da omogući komunikaciju između aplikacija na klasteru, ali i definisanje pravila za rutiranje zahteva za aplikacijom na osnovu URL-a, imena hosta ili nekog drugog atributa (Slika 43) [2].



Slika 43. Primena *Ingress* objekta na *Kubernetes* platformi [63]

Pitanje skladištenja i razmene podataka je od velikog značaja za rad na *Kubernetes* platformi. U tu svrhu se koristi posebna vrsta objekta koji se naziva *ConfigMap* [64]. U ovom se objektu mogu skladištiti različite informacije, kao što su npr. podaci o konfiguraciji neke aplikacije, predstavljeni u vidu fajlova ili u ključ-vrednost formatu. Nakon definisanja *ConfigMap* objekta, moguće je koristiti njegove vrednosti prilikom konfiguracije podova ili drugih objekata. Generalno, namena *ConfigMap* objekta se najbolje može objasniti kroz potrebu određenog poda, da se prilikom instanciranja podese određeni parametri potrebni za pokretanje aplikacije. Unutar *ConfigMap* objekta se npr. može definisati vrednost parametra adresa_baze_podataka, dok se u definiciji poda referencira naziv *ConfigMap* objekta i način prosleđivanja parametra kontejneru. *ConfigMap* se uglavnom koristi za skladištenje ne-strukturirane konfiguracije koja se može koristiti u aplikaciji. Može se koristiti za konfigurisanje aplikacijskih postavki, povezivanje parametara sa okruženjima, nizovima i putanjama do datoteka i sl. On omogućava izdvajanje konfiguracije iz aplikacijskog koda, što olakšava upravljanje aplikacijama i smanjuje količinu koda koji je potreban za promenu konfiguracije. Inače, on se obično definiše kao tekstualni YAML fajl, koji se može kreirati ili ažurirati u *Kubernetes* klasteru.

U praksi postoje tri načina korišćenja *ConfigMap* vrednosti i to kroz promenljive okruženja (eng. *environment variables*), argumente komandne linije (kada postoji potreba da se aplikacija konfiguriše kroz argumente komandne linije) i prezentovanje vrednosti *ConfigMap* objekta kroz fajl sistem koji se pridodaje podu. Kada se *ConfigMap* objekat prezentuje kroz fajl sistem, u slučaju ažuriranja objekta, izmene se automatski propagiraju na svaki pod koji ih koristi i nije potrebno restartovati kontejner. Svaki od ova tri načina korišćenja ima svoje prednosti, a izbor se svodi na preferencije i način

implementacije aplikacije unutar kontejnera. S obzirom na to, da ih može koristiti više različitih objekata, a njihov sadržaj se može prezentovati na različite načine, *ConfigMap* objekti nude fleksibilnost prilikom konfiguracije sistema.

Secret je objekat, vrlo sličan *ConfigMap* objektu, koji po pravilu sadrži neke osetljive informacije, kao što su lozinke, tokeni za pristup i sl. Korišćenjem *Secret* objekta uklanja se potreba za čuvanjem ovih osetljivih informacija unutar kontejnera, odnosno unutar aplikacionog koda. Postoji više tipova *Secret* objekta: *opaque*, *basic-auth*, *ssh-auth*, *tls* i dr. Moguće je definisati i svoje tipove. Svi tipovi su izvedeni od *Opaque* tipa, odnosno, svaki tip dodaje svoje atribute nad *Opaque* tipom, kako bi se, na primer, u slučaju *ssh-auth* tipa olakšala komunikacija sa SSH serverom i klijentom [62]. Kao što je slučaj i sa *ConfigMap* objektom, postoje tri načina korišćenja: kroz promenljive okruženja, argumente komandne linije i kroz fajl sistem koji se prezentuje podu. Takođe, kod prezentovanja kroz fajl sistem, nakon ažuriranja *Secret* objekta, izmene će biti automatski propagirane na fajl sistem i kontejner će moći da vidi te izmene bez ponovnog pokretanja samog kontejnera.

Važno je napomenuti, da je ovaj objekat dizajniran da bude odvojen od aplikacije i da omogući čuvanje osetljivih informacija na centralnom mestu [65]. Za razliku od *ConfigMap*-a, on čuva informacije koje su kodirane i zaštićene od neovlašćenog pristupa. Može se koristiti za pružanje informacija za različite resurse u *Kubernetesu* (*Deployment*, podovi ili *Service*) i ono što je najvažnije može biti kreirani ručno ili generisan od strane aplikacije koja koristi *Kubernetes* API.

Ovde je važno istaći, da je lokalni disk unutar poda kratkotrajan (eng. *ephemeral*), odnosno sve informacije koje se smeste na fajl sistem, nakon gašenja poda se gube (pokreće se novi pod sa novim fajl sistemom). Rešenje ovog problema na *Kubernetes* platformi je uvođenje različitih tipova logičkih diskova (eng. *Volume*), kroz apstrakciju zvanu *Container Storage Interface* (CSI). Svaki logički disk čija implementacija zadovoljava ovaj interfejs, može se predstaviti kao objekat unutar sistema. Nakon predstavljanja logičkih diskova sistemu, jedan pod simultano može koristiti sve njemu prezentovane diskove. Objekat *PersistentVolume* (PV) predstavlja deo prostora na logičkom disku koji se kreira od strane administratora i dostupan je za potraživanje od strane korisnika, kroz pravljenje zahteva, tj. objekta *PersistentVolumeClaim* (PVC).

PersistentVolume se može predstaviti na statički i dinamički način. Dinamički način podrazumeva korišćenje apstrakcije *StorageClass*, preko koje se zahtev za prostor na disku (PVC), dinamički mapira preko atributa iz zahteva, dok se statički *PersistentVolume* kreira ručno od strane administratora. Neki od tipova PV su: *azureDisk*, *azureFile*, *csi*, *glusterfs* i dr.

PersistentVolumeClaim je objekat preko koga se zahteva pristup disku. U zahtevu se navodi količina diska i tip pristupa. Tipovi pristupa su [62]:

- *ReadWriteOnce* - disk je predstavljen za čitanje i pisanje na jednom čvoru,
- *ReadWriteMany* - disk je predstavljen za čitanje i pisanje na više čvorova,
- *ReadOnlyMany* - disk je predstavljen samo za čitanje na više čvorova i
- *ReadWriteOncePod* - disk je predstavljen za čitanje i pisanje na samo jednom podu.

4.6. KONTEJNERIZACIJA U SDN OKRUŽENJU

Primena kontejnerizacije u SDN okruženju ima za cilj da se poboljša fleksibilnost, skalabilnost i efikasnost mrežnih resursa, ali i da se omogući jednostavnije upravljanje mrežnim funkcijama i uslugama. U kombinaciji sa SDN-om, kontejneri mogu doneti brojne prednosti u mrežnom okruženju, uključujući ubrzanje implementacije novih usluga, bolju izolaciju i lakše održavanje mrežnih infrastrukture. Ključni aspekti implementacije ove tehnologije u SDN okruženju su:

- *modularnost i fleksibilnost* – SDN omogućava centralizovano upravljanje mrežnim uređajima, pa je u ovom okruženju u kontejnere moguće implementirati različite mrežne funkcionalnosti (npr. *firewall*, *load balancing* ili VPN) u odvojene module. Ti moduli se mogu lako rasporediti, skalirati i ažurirati i to čak bez potrebe za velikim promenama u mrežnoj infrastrukturi.
- *brza implementacija i orkestracija* – *Kubernetes* i drugi alati za orkestraciju kontejnera mogu se koristiti za automatsko raspoređivanje i upravljanje mrežnim funkcijama u SDN okruženju. To omogućava dinamičko skaliranje i brzo pokretanje novih mrežnih funkcionalnosti, kao i jednostavno ažuriranje postojećih. SDN kontroleri mogu koristiti orkestratore kontejnera za upravljanje mrežnim servisima na nivou aplikacija i infrastrukture, čime se olakšava integracija novih mrežnih funkcija.
- *izolacija i bezbednost* – kontejneri omogućavaju bolju izolaciju aplikacija i servisa, što omogućava efikasno upravljanje bezbednošću i lakše rešavanje potencijalnih problema. U SDN okruženju, kontejneri omogućavaju bolju segmentaciju mreže, čineći implementaciju sigurnosnih politika jednostavnijom i bržom.
- *poboljšana skalabilnost* – kontejnerizacija omogućava lako skaliranje mrežnih servisa u zavisnosti od opterećenja (npr. u slučaju velike potražnje za određenim mrežnim servisima, broj instanci koje obavljaju tu funkciju može se automatski povećati). SDN upravlja dinamičkim dodeljivanjem resursa, a kontejneri omogućavaju efikasno skaliranje resursa u mreži.
- *automatizacija i smanjenje kompleksnosti* – korišćenjem *Kubernetes* platforme, moguće je automatizovati pokretanje, ažuriranje i skaliranje mrežnih funkcionalnosti u SDN okruženju, čime se smanjuje kompleksnost u administraciji i održavanju mreže. Primenom *DevOps* metodologije razvoja softvera u SDN okruženju, mogu se brže isporučivati nove mrežne funkcionalnosti i to putem kontinuirane integracije i kontinuiranog isporučivanja (CI/CD), što omogućava brže testiranje i implementaciju novih mrežnih servisa.
- *upravljanje mrežnim funkcijama kao uslugom* – kontejnerizacija u SDN okruženju može biti korišćena za mrežne funkcije kao usluge (NFV), gde se mrežne funkcije kao što su ruteri, *firewall*, NAT i dr. pokreću u kontejnerima, što omogućava bolje korišćenje resursa i brže raspoređivanje mrežnih usluga.
- *optimizacija performansi* – kontejnerizacija omogućava lakše upravljanje resursima (npr. CPU, memorija, mrežna propusnost) u mrežnim aplikacijama. U SDN okruženju, gde je kontrola nad mrežnim resursima centralizovana, ovo omogućava bolju optimizaciju performansi i efikasniji rad mrežnih funkcija.

4.7. PRIMENA KONTEJNERIZACIJE U REALNIM SDN OKRUŽENJIMA

U realnim SDN okruženjima, kontejnerizacija se primenjuje na nekoliko ključnih mesta kako bi se poboljšala fleksibilnost, skalabilnost i efikasnost mrežnih funkcionalnosti. Najčešće se koristi za implementaciju mrežnih funkcija kao usluge. Naime, mrežne funkcije se umesto na tradicionalnim fizičkim uređajima ili virtuelnim mašinama, raspoređuju u kontejnerima. To omogućava efikasnije korišćenje resursa i lakše skaliranje mrežnih funkcija prema potrebama mreže. Na ovo rešenje se sve više opredeljuju telekom operatori, koji kontejnerizaciju primenjuju da bi omogućili brzu implementaciju i skaliranje mrežnih funkcionalnosti u skladu sa promenama u saobraćaju i zahtevima korisnika. Drugi razlog, koji je telekom provajdere naterao na korišćenje kontejnera u SDN okruženju je dinamičko skaliranje i upravljanje opterećenjem. Naime, SDN infrastruktura može efikasno upravljati dinamičkim skaliranjem mrežnih funkcija na osnovu saobraćaja i performansi. *Kubernetes* i drugi orkestratori omogućavaju automatsko skaliranje broja kontejnera koji pokreću određene mrežne funkcije, u zavisnosti od trenutnih zahteva mreže. Primer za ovo su data centri i cloud okruženja, gde se kontejnerizacija koristi za dinamičko povećanje broja instanci za funkcije kao što su *load balancing* ili DNS usluge u vreme visokog opterećenja.

U SDN arhitekturama, kontejneri se koriste za implementaciju kontrolera i centralizovano upravljanje mrežnim uređajima i uslugama. Raspoređivanjem SDN kontrolera u kontejnere, omogućava se lakše upravljanje, ažuriranje i skaliranje. Kao primer može poslužiti primena *OpenDaylight* ili ONOS (eng. *Open Network Operating System*) SDN kontrolera u kontejnerima, da bi se omogućilo jednostavnije skaliranje i ažuriranje mrežnih politika i konfiguracija. Fleksibilnost, koja potiče od implementacije ne samo kontrolera, već i mrežnih funkcija u kontejnerima, predstavlja značajnu prednost u odnosu na tradicionalnu implementaciju na hardverskim uređajima. Korišćenjem kontejnera se npr. virtuelne mrežne funkcije (eng. *Virtual Network Functions* – VNFs) mogu pokrenuti u visoko efikasnim i lako skaliranim kontejnerskim okruženjima. Kao primer mogu poslužiti telekomunikacione mreže, koje koriste VNF implementacije za pružanje usluga kao što su bezbednost (*firewall*), optimizacija saobraćaja i kontrola kvaliteta usluge, gde se brzo i efikasno vrši upravljanje kontejnerima, putem platformi kao što je *Kubernetes*.

U SDN okruženjima, monitoring mreže i analitika apodataka su od ogromnog značaja za optimizaciju performansi. Kontejnerizacija omogućava brzu i laku implementaciju alata za monitoring i analizu mrežnog saobraćaja, kao što su *Prometheus* i *Grafana*. Ovi alati se koriste, da bi se pratili i analizirali tokovi paketa u realnom vremenu (npr. *Facebook* koristi *Prometheus* i *Grafana* za praćenje performansi mreže u svom SDN okruženju). Korišćenje kontejnerizovanih instanci omogućava skaliranje resursa za praćenje mreže i generisanje metrika u realnom vremenu. Konkretno, kontejnerizovane aplikacije za analizu saobraćaja se koriste u realnim SDN okruženjima, da bi se pratile performanse i brže pronalazile greške ili problemi u mrežnoj infrastrukturi. Efikasan monitoring i analitika podataka su važan preduslov za efikasnije upravljanje mrežnim politikama i bezbednosnim funkcijama, kao što su autentifikacija, autorizacija i šifrovanje. Ove funkcije mogu biti implementirane u kontejnerima i brzo raspoređene, testirane i ažurirane u SDN okruženju. Kao primer može da posluži činjenica, da kontejneri omogućavaju brzo implementiranje novih sigurnosnih pravila, kao što su zaštita od DDoS napada, u realnom vremenu, bez prekida mrežnih usluga.

Kontejnerizacija u SDN okruženjima često koristi za testiranje i razvoj mrežnih usluga i funkcionalnosti. Korišćenjem kontejnera, inženjeri mogu brzo postaviti okruženje za testiranje novih mrežnih funkcionalnosti ili simulaciju novih scenarija bez potrebe za skupim fizičkim uređajima. Dobro urađeno testiranje i rezultati do kojih se tom prilikom dolazi, u SDN okruženjima se može iskoristiti za automatizaciju mrežnog saobraćaja. To je posebno važno, u slučajevima kada je potrebno dinamički usmeravati saobraćaj između različitih resursa u SDN okruženju. Korišćenje kontejnera omogućava lako implementiranje automatizovanih rešenja za dinamičko preusmeravanje saobraćaja u zavisnosti od trenutnih mrežnih uslova. Primer je automatizovani *load balancing* u SDN mrežama, koji u velikom broju slučajeva koristi kontejnere za brzo dodavanje novih instanci servisa koji balansiraju saobraćaj prema potrebama u realnom vremenu.

U *edge computing* arhitekturama, kontejnerizacija u SDN okruženju omogućava upravljanje mrežnim funkcijama, koje se implementiraju blizu krajnjih korisnika. Korišćenjem *Docker* kontejnera za implementaciju SD-WAN (eng. *Software-Defined Wide Area Networking*) funkcionalnosti, operatori mogu optimizovati mrežni saobraćaj između IoT uređaja, smanjujući latenciju i povećavajući efikasnost. Specifičan slučaj je *Cisco*-ovo rešenje, koje koristi kontejnerizovane funkcije u SD-WAN rešenju za IoT uređaje u industrijskim postavkama. Korišćenje kontejnerizacije omogućava brzo dodavanje novih mrežnih funkcionalnosti za uređaje koji se nalaze na ivici mreže (eng. *edge*).

ZAKLJUČAK

Primena kontejnerizacije u SDN okruženju poboljšava efikasnost, skalabilnost, sigurnost i fleksibilnost mrežnih funkcija, omogućavajući lakšu implementaciju i održavanje mrežnih usluga. Korišćenjem alata za orkestraciju kontejnera, kao što je *Kubernetes*, u kombinaciji sa SDN kontrolerima se značajno unapređuje upravljanje mrežnim funkcijama i omogućava brži razvoj i dinamička alokacija mrežnih resursa. U SDN okruženjima, kontejnerizacija se konkretno primenjuje u industrijama kao što su telekomunikacije, data centar, cloud infrastruktura, IoT, 5G/6G mreže, i mrežni bezbednosni sistemi. U praksi postoje brojni primeri prednosti, ali i nedostataka primene kontejnerizacije u realnim SDN okruženjima.

Prednosti primene kontejnerizacije u SDN okruženjima su vezani za:

- fleksibilnost i agilnost – laka i brza implementacija novih mrežnih funkcija i servisa i dinamičko menjanje mrežne konfiguracije u realnom vremenu, što je korisno za prilagođavanje promenama saobraćajnog opterećenja u mreži,
- skalabilnost – jednostavno skaliranje mrežnih funkcionalnosti u zavisnosti od opterećenja, koje se primenom *Kubernetes* platforme može automatizovati,
- izolacija i bezbednost – svaka mrežna funkcija može biti izolovana u svom kontejneru, što poboljšava bezbednost i omogućava bolje upravljanje pristupom između različitih mrežnih servisa,
- efikasno korišćenje resursa – kontejneri su „lakši“ od virtuelnih mašina, sa manjim zahtevima za memoriju i procesorskim kapacitetom, pa se mrežni resursi mogu bolje iskoristiti, čime se smanjuje potreba za skupljim fizičkim uređajima,

- brže inovacije i testiranje – mrežni servisi mogu biti brzo implementirani i testirani u kontejnerima pre nego što se primene u produkciji, što ubrzava razvoj i inovacije u SDN okruženju,
- automatizacija i orkestracija – korišćenje alata kao što su *Kubernetes* i *Docker* omogućava visoku automatizaciju upravljanja mrežnim resursima, smanjujući potrebu za manuelnim intervencijama i povećavajući efikasnost u operacijama.

Naravno da postoje nedostaci primene kontejnerizacije u SDN okruženjima, koji se pre svega odnose na:

- kompleksnost upravljanja – upravljanje velikim brojem kontejnera i njihovom komunikacijom može postati kompleksno, jer konfiguracija, praćenje i održavanje velikog broja kontejnera zahteva značajne resurse i stručnost,
- latenciju u mreži i moguće degradacije performansi – korišćenjem kontejnera se poboljšava efikasnost korišćenja resursa, ali postoji rizik od povećanja latencije u mrežama kada se koristi veliki broj kontejnera (mrežna komunikacija između kontejnera, posebno kada su raspoređeni u različitim fizičkim lokacijama, može uticati na performanse),
- izazove sa aspekta bezbednosti – kontejneri omogućavaju izolaciju, ali ako nisu pravilno konfigurisani i ako postoje propusti u *image* fajlovima to može dovesti do problema (npr. ako mrežni kontejneri nisu pravilno izolovani, postoji rizik da napadači mogu kompromitovati mrežne funkcije ili pristupiti osetljivim podacima),
- oslanjanje na orkestraciju – ako orkestracija nije pravilno postavljena ili ako orkestrator ne funkcioniše ispravno, cela mreža može doći u problem sa performansama i dostupnošću,
- potrebu za dodatnom infrastrukturom – iako kontejnerizacija koristi manje resursa nego virtualne mašine, za efikasno upravljanje kontejnerima potrebno je investirati u dodatnu infrastrukturu, kao što su orkestratori i monitoring alati. Takođe, potrebna je infrastruktura za integraciju sa postojećim SDN kontrolerima i mrežnim uređajima.



POGLAVLJE 5



NFV KONCEPT U SDN MREŽAMA

5. NFV KONCEPT U SDN MREŽAMA

5.1. UVOD

Razvoj novih ICT tehnologija je veoma brz i značajno utiče na primenu novih koncepata umrežavanja. To podrazumeva redizajn tradicionalnih, hijerarhijski organizovanih mreža i implementaciju fleksibilnijih i skalabilnijih rešenja, koja uključuju primenu novog koncepta upravljanja mrežnom infrastrukturom. Rad u okruženjima sa virtuelnim mašinama i kontejnerima postao je realnost, koju karakteriše brzo i lako skaliranje unutar infrastrukture i odsustvo potrebe za dodatnom nabavkom hardvera. Umesto korišćenja opreme, specijalizovane za primenu određene mrežne funkcije, primenjuje se koncept virtuelizacije mrežnih funkcija (eng. *Network Function Virtualization* - NFV). Ovaj koncept omogućava, da se mrežne funkcije implementiraju i pokreću na standardnoj računarskoj infrastrukturi, čime se smanjuju troškovi i povećava fleksibilnost.

NFV koncept je deo šire strategije, koja ima za cilj, da se u mreže „unese“ znatno viši nivo programabilnosti i da se one približe "softverski definisanim" arhitekturama. Osnova koncepta je razdvajanje mrežnih funkcija od hardvera na kojem rade i stvaranje uslova, da se mrežne funkcije, kao što su ruteri, *switch*-evi, *firewall*-ovi, *load balancer*-i i dr. mogu pokretati kao virtuelizovani servisi na deljenoj infrastrukturi. Čitav proces se zasniva na virtuelizaciji resursa, korišćenjem tehnologija kao što su virtualne mašine (VMs) ili kontejneri. Dakle, suština NFV koncepta leži u primeni mrežnih funkcija u formi softverskih aplikacija, koje se pokreću na standardnim serverima. Umesto ulaganja u skupe specijalizovane uređaje, mrežne funkcije se mogu pokretati na standardnim x86 serverima i lako prilagođavati promenama u opterećenju mreže.

Odsustvo potrebe za specijalizovanim hardverom i dodatnim nabavkama, pomaže operaterima da smanje kapitalne i operativne troškove. Oni dobijaju mogućnost, da brže i lakše implementiraju nove mrežnih usluge, jer to ne zahteva fizičku implementaciju novih uređaja. Sa primenom NFV koncepta i clouda, dolazi i nova poslovna filozofija *pay-as-you-go*, koja pomaže i klijentima da smanje troškove. Zato organizacije ubrzano prelaze sa korišćenja namenskog hardvera, sa unapred implementiranim softverom, na primenu naprednih softverskih rešenja na standardnom hardveru. Danas se može reći, da je to deo savremenih društveno-ekonomskih trendova, kojim se analiziraju potencijalni problemi koji prate implementaciju naprednih servisa u tradicionalnim računarskim mrežama i pronalaze nova rešenja.

Ovde se mora imati u vidu da, pored brojnih prednosti NFV koncepta i drugih tehnologija povezanih sa softverski definisanim umrežavanjem, postoje i određeni rizici koji usporavaju njihovu širu primenu, pre svega u mrežama operatora. Jedan rizik se odnosi na postojanje više standarda i inicijativa otvorenog koda (Otvorena platforma za NFV, *Open-Source MANO*, *Open Network Automation Platform*, *European Telecommunications Standards Institute* – ETSI, i *Metro Ethernet Forum*) koji podstiču razvoj NFV. Za širu primenu NFV koncepta neophodan je jasan arhitektonski pravac, koji bi obezbedio prihvatljive uslove za sve provajdere i operatore. Drugi rizik je bezbednost u složenim okruženjima sa implementiranim NFV konceptom. Naime, neophodno je primeniti odgovarajuća rešenja za zaštitu fizičkog sloja, virtuelizovanog sloja i aplikacija. Ovaj

zahtev dodatno komplikuje primenu NFV koncepta u modernim okruženjima. Na kraju, treba uzeti u obzir, da prelazak sa fizičke na virtuelnu infrastrukturu, može dovesti do degradacije performansi mrežnih funkcija. Zato je neophodno kontinuirano pratiti performanse virtuelizovanih mrežnih funkcija i izbegavati potencijalna uska grla u saobraćaju (npr. virtuelni *switch* može biti potencijalno usko grlo između virtuelnih mašina i mrežnih servisa za različit saobraćaj). Da bi se promene pravilno sagledale, neophodno je razumeti značaj prelaska sa hardverskog pristupa na virtuelizovani mrežni pristup zasnovan na softveru [66].

5.2. OSNOVE NFV KONCEPTA

Razvoj informacionih tehnologija i pojava novih oblika komunikacija predstavljaju osnovnu pokretačku snagu u razvoju savremenog računarstva. Da bi se omogućilo kontinuirano praćenje modernih trendova, potrebno je pronaći adekvatne odgovore na brojne izazove. Ovi izazovi se uglavnom odnose na mogućnost jednostavne implementacije novih i sve zahtevnijih usluga i izgradnje fleksibilnijeg i skalabilnijeg okruženja. To je moguće samo kroz kontinuirani razvoj i primenu naprednih softverskih tehnologija, kao što je tehnologija virtuelizacije računarske infrastrukture i mreže u celini [2].

Danas je teško konstatovati, da postoji računarska mreža u kojoj nije primenjena virtuelna (logička) verzija bar nekog uređaja, bilo da su u pitanju serveri, operativni sistemi, procesori, skladišta podataka, *switch*-evi ili ruteri. Svoju širu primenu, tehnologija virtuelizacije je posebno našla u data centrima, čija se fizička infrastruktura sastoji velikog broja serverskih aplikacija, koje su velikoj meri implementirane na virtuelnim serverskim mašinama koje rade na deljenom hardveru. Ovde je važno naglasiti, da je i sam NFV koncept izgrađen na tehnici virtuelizacije servera i primeni virtuelizovanih mrežnih servisa. Njegova implementacija u mreži, zasniva se na dva glavna principa:

- razdvajanje softvera od hardvera i
- implementaciji mrežnih funkcija nezavisno od lokacije u mreži (npr. *firewall*, *load balancer* i drugi mrežni servisi, mogu se implementirati na bilo kojoj lokaciji sa virtuelizovanom računarskom infrastrukturom).

NFV koncept je tehnologija koja omogućava izgradnju specifičnog ekosistema, koji obuhvata virtuelne mrežne uređaje, alate za upravljanje i infrastrukturu koja softverska rešenja integriše sa generičkim hardverom. Zamena specijalizovanih fizičkih uređaja, sa jednim ili više softverskih programa, koji istu mrežnu funkciju obavljaju na generičkom računarskom hardveru, odnosno na sistemima, uređajima i komponentama, široko dostupnim na tržištu (eng. *Commercial-Off-The-Shelf* - COTS hardver), značajno pomaže telekom operatorima [66]. Naime, u tradicionalnim računarskim mrežama, telekom operateri u potpunosti kontrolišu hardver i softver koji obavlja određenu mrežnu funkciju. Problem nastaje onda kada se rapidno menjaju zahtevi korisnika i kada je neophodno dinamički vršiti promene u mreži i brzo realizovati usluge. Rešenje za ove probleme, telekom operateri nalaze u primeni NFV tehnologije, uz bitnu pretpostavku da oprema na lokaciji ima potrebne resurse. Primenom virtuelizacije računarske infrastrukture otvaraju se neke nove mogućnosti u pogledu konfigurisanja i upravljanja

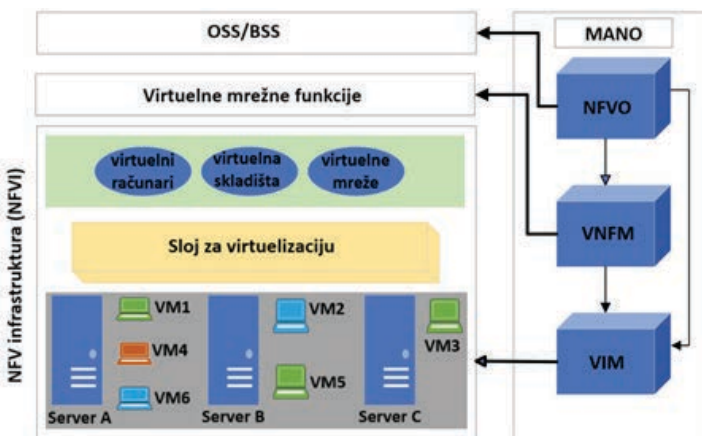
mrežom. Moguće je implementirati otvorene softverske interfejske i izgraditi fleksibilno i skalabilno okruženje sa neophodnim nivoom agilnosti za rešavanje svih korisničkih zahteva, primenu inovativnih rešenja i izgranju novih, optimalnijih i efikasnijih mrežnih arhitektura [67].

5.3. ETSI NFV PROGRAMSKI OKVIR (eng. *FRAMEWORK*)

U tradicionalnim računarskim mrežama, mrežne funkcije se obično realizuju kao kombinacija softverskih i hardverskih rešenja određenog proizvođača. To neumitno vodi ka „zaključavanju“ infrastrukture operatora u specifičnom hardveru i softveru (tzv. *vendor lock-in*), odnosno ka velikom stepenu zavisnosti od pojedinih proizvođača. Prirodno rešenje, za izbegavanje ovakvog problema, je primena tehnologije virtuelizacije. Ona predstavlja iskorak ka realizaciji fleksibilnije i skalabilnije računarske infrastrukture, jer uvodi sledeće novine u tradicionalnu mrežnu arhitekturu [68]:

- mrežni elementi više nisu skupovi integrisanih hardverskih i softverskih entiteta, jer se softver razvija odvojeno od hardvera i obrnuto,
- fleksibilna i skalabilna primena mrežnih funkcija – visok nivo fleksibilnosti u pogledu distribucije i korišćenja infrastrukturnih resursa, omogućava da se različite mrežne funkcije obavljaju istovremeno na deljenom hardveru i
- dinamičko obavljanje aktivnosti u mreži – kontrola operativnih parametara mrežnih funkcija odvija se kroz granularnu kontrolu i praćenje stanja mreže.

Strateški cilj primene NFV tehnologije je implementacija mrežnih funkcija na virtuelnoj infrastrukturi, koja se kreira primenom različitih tehnika virtuelizacije na standardnom (široko korišćenom) i relativno jeftinom hardveru [13]. Na slici 44., prikazan je programski okvir koji predstavlja osnovu za rad na standardizaciji NFV arhitekture [69].



Slika 44. Struktura NFV programskog okvira [70]

Ovaj programski okvir uključuje standarde, koji definišu upravljanje virtuelizovanim mrežnim funkcijama, odnose, međusobne zavisnosti i protok podataka između njih, kao i proces alokacije potrebnih resursa. U tom smislu, ETSI ISG je podelio uloge u okviru programskog okvira i kategorisao tri funkcionalna bloka [70]:

- NFVI (eng. *Network Functions Virtualization Infrastructure*) blok – osnovni gradivni blok celokupne arhitekture, u kojem su grupisani hardver za hostovanje virtuelnih mašina, softver za virtuelizaciju i virtuelizovani resursi.
- VNF (eng. *Virtual Network Functions*) blok – virtuelni entiteti (virtuelne mašine i kontejneri) sa softverski implementiranim mrežnim funkcijama.
- MANO (eng. *Management And Network Orchestration*) blok – blok koji kroz stalnu interakciju sa NFVI i VNF blokovima, obezbeđuje efikasno upravljanje i orkestraciju svih resursa u virtuelizovanim data centrima, sa fokusom na dinamičku alokaciju resursa u skladu se zahtevima iz mreže.

Za bolje razumevanje NFV programskog okvira, mora se uzeti u obzir činjenica, da se NFV tehnologija u fiksним i mobilnim mrežama može primeniti na različite funkcije u kontrolnoj u ravni podataka [71]. U tom smislu, postoje brojna rešenja, kao što su:

- softverski implementirana dubinska inspekcija paketa (DPI) – omogućava naprednu analizu sadržaja paketa i saobraćaja, jednostavnije mehanizme za primenu, ažuriranje, testiranje i skaliranje resursa nakon promene opterećenja i višedimenzionalno izveštavanje [72],
- translacija IP adresa na nivou operatera – korisnicima se omogućava pristup Internetu i deljenje jedne ili više javnih IP adresa,
- virtuelizacija servisa u mrežnim okruženjima [73],
- virtuelizacija mreža za isporuku sadržaja (eng. *Content Delivery Network* - CDN) – omogućava se lakše širenje i skaliranje usluga isporuke sadržaja, kao i ponovno korišćenje hardvera za instaliranje drugih aplikacija za isporuku usluga [74],
- virtuelizacija jezgra mobilne mreže – fleksibilnije se upravlja mrežom i stvaraju se uslovi koji omogućavaju opsluživanje veće količine saobraćaja uz bolje korišćenje resursa (uključuje uštedu energije, hardversku konsolidaciju i podršku za *multitenancy* pristup, gde jedna softverska instanca sa svojom infrastrukturom pruža uslugu većem broju korisnika i bržem konfigurisanju novih servisa) [75] i
- koordinisana implementacija cloud tehnologija u organizacijama – omogućava da se usluge na zahtev i mrežni resursi prilagode potrebama organizacije (uopšteno govoreći, različite verzije usluge koegzistiraju na deljenom hardveru).

5.3.1. Komponente NFV arhitekture

5.3.1.1. NFVI blok

NFVI blok predstavlja osnovnu komponentu NFV arhitekture, čiji je zadatak, da obezbedi resurse (fizičke i virtualne) potrebne za implementaciju, pokretanje i upravljanje virtualizovanim mrežnim funkcijama. Pored fizičkih resursa, kao što su serveri, skladišta za podatke i mrežni uređaji, sastavni deo NFVI infrastrukture je i softver koji omogućava virtualizaciju pomenutih resursa i efikasno izvršavanje VNF-ova. Generalno, uloga NFVI bloka je da obezbedi potreban nivo fleksibilnosti i skalabilnosti u NFV okruženju, što podrazumeva fleksibilnu alokaciju resursa na zahtev i mogućnost da operatori brzo šire ili smanjuju kapacitet mreže u skladu sa potrebama. Na takvoj virtualnoj infrastrukturi, moguće je implementirati različite vrste virtualizovanih mrežnih funkcija, kao što su mrežna bezbednost, monitoring, ali i efikasno upravljanje saobraćajem, što vodi optimalnijem korišćenju resursa (smanjuje se potreba za fizičkim uređajima i optimizuje potrošnja energije i veličina prostora koji se koristi za smeštaj opreme).

Dakle, prednosti primene NFVI bloka su značajne:

- smanjenje troškova – korišćenjem COTS hardvera i virtualizacije, NFVI omogućava operatorima da smanje kapitalne i operativne troškove uzrokovane korišćenjem specijalizovane opreme,
- veća agilnost – virtualizacija resursa omogućava brže reagovanje na promene u mrežnim zahtevima i lakšu implementaciju novih mrežnih funkcija,
- mogućnost automatizacije procesa i efikasnije upravljanje infrastrukturom – smanjuje se uloga ljudskog faktora i
- šira skalabilnost - lako skaliranje mrežnih funkcija u skladu sa promenama u opterećenju i potrebama korisnika.

Da bismo sagledali prednosti ovog rešenja, moramo poći od trenutnog stanja. Svoj cilj, da zadovolje trenutne potrebe, operatori često realizuju instalacijom značajnih hardverskih resursa, koji bi po njima trebala da „pokriju“ i buduće potrebe za proširenjima. Međutim, u praksi, posebno u mrežama sa tradicionalnom arhitekturom to nije moguće ostvariti. Instalirani hardver često ne može da zadovolji rastuće potrebe, pa se mora vršiti nadogradnja postojećeg, instalacija novog hardvera ili promena svrhe korišćenja resursa u skladu sa novim zahtevima. Rešenje je izbor skalabilnog hardvera, koji se može zameniti bez uticaja na aplikacije i hostovane mrežne funkcije. Telekom operateri imaju za cilj, da usklade resurse, što je više moguće sa trenutnim potrebama. Oni nameravaju da u početku obezbede infrastrukturu sa značajnim hardverskim resursima kako bi izbegli potrebu za budućim promenama (proširenjima). U praksi, instalirani hardver često ne može da zadovolji rastuće potrebe. Iz tog razloga, moramo da nadogradimo hardver ili promenimo korišćenje zajedničkih resursa prema novim zahtevima, što svakako nije racionalno rešenje. Rešenje je slobodan izbor skalabilnog hardvera, sa kojim se može zameniti postojeći hardver bez uticaja na aplikacije i mrežne funkcije. To znači mogućnost, da se kreira virtualna infrastruktura i da se na njoj instaliraju softverske aplikacije, odnosno mrežne funkcije.

Izbor hardvera je složen proces, vezan za nameru telekom operatera, da smanji troškove. Iako se COTS hardver pokazao kao najbolje rešenje, situacija je komplikovana jer brojni proizvođači zahtevaju veću kompatibilnost između svoje i COTS opreme. Ova činjenica direktno utiče na izbor opreme jer se mora garantovati pouzdan rad i podrška dobavljača. Takođe, moramo uzeti u obzir izbor operativnog sistema fizičke mašine i hipervizora kao sloja virtuelizacije, ako se odlučimo na implementaciju virtuelnih mašina kao virtuelnih entiteta. U slučaju COTS hardvera, potrebno je uzeti u obzir sledeće:

- vrstu dostupne tehničke podrške,
- troškove licenciranja,
- troškove nabavke,
- mogućnost nadogradnje,
- stabilnost u radu i
- mogućnost interakcije sa softverima otvorenog koda.

Odabir lokacije na kojoj će biti instalirana NFVI je takođe važan faktor. Idealno bi bilo instalirati NFVI na geografski različitim lokacijama, u područjima sa velikom koncentracijom korisnika. Za nesmetano funkcionisanje infrastrukture mora postojati geografska redundansa i potreban nivo fleksibilnosti u njenom obezbeđivanju. Fleksibilnost u obezbeđivanju redundanse je neophodna, da bi se eliminisali problemi uzrokovani kvarom mrežnog uređaja. U tradicionalnim mrežama, redundansa se implementira na nivou uređaja, što u slučaju kvara uređaja (na primer diskova na ruterima) dovodi do prekida saobraćaja (mrežna funkcija postaje nedostupna). Problem se rešavao, tako što su se unapred instalirali redundantni uređaji ili kreirala rezervna putanja, što je skupo, a ponekad i složeno.

U okviru NFVI infrastrukture, redundantnost se obezbeđuje na nivou komponente, tako da su šanse za gubitak mrežne funkcije usled kvara jedne komponente minimalne. Mrežna funkcija, kao što je npr. ruter, može se implementirati kao virtuelizovana mrežna funkcija na serveru, koja koristi RAID (eng. *Redundant Array of Independent Disks*) tehnologiju. Kvar na jednom od diskova neće uticati na mrežnu funkciju na drugim diskovima. Pored redundanse na nivou hardvera, NFV koncept podrazumeva i redundantnost za virtuelne mašine ili kontejnere. U tom smislu, mogu se implementirati različita rešenja za virtuelizaciju, a sve sa ciljem da se kreira robustan dizajn mrežne infrastrukture. Ovde je važno ukazati i na značaj samog izbora lokacije za NFVI infrastrukturu. Idealno bi bilo, radi redundanse, kreirati NFVI infrastrukturu na različitim lokacijama i obezbediti geografsku redundansu, ali i tamo gde je velika koncentracija korisnika.

Hardversku infrastrukturu tradicionalnih mreža čine uređaji čiji životni ciklus počinje kupovinom i uglavnom se završava kvarovima ili nakon određenog perioda kada provajderi ostanu bez ugovora o podršci i komponentata za popravku. Zato je njihova zamena neophodna kada dođu do kraja životnog ciklusa, kako bi se smanjila mogućnost kvara. Očekivanja operatera su, da uređaji treba da vrate uložena sredstva u tom roku, što je ponekad nemoguće. Slične procene vremena i prakse mogu se primeniti na NFVI infrastrukturu, jer operativni sistemi fizičke mašine, hipervizori i virtuelizovane mrežne

funkcije, takođe imaju životni vek. Nakon što životni vek istekne, potrebna je nadogradnja ili ispravka grešaka i obnavljanje ugovora o podršci. U procesu projektovanja NFVI infrastrukture, potrebno je uzeti u obzir faktore kao što su programska podrška, životni ciklus virtuelizovanih mrežnih funkcija, ažuriranje softvera hipervizora i zamena *switch*-eva i servera. Koordinacija između ovih zahteva, treba da omogućiti provajderima da izbegnu potencijalne zamke i efikasno izvrše nadogradnju.

5.3.1.2. VNF blok

Virtuelizovane mrežne funkcije su druga važna komponenta NFV arhitekture. To su softverske aplikacije, tj. mrežne funkcije implementirane u softveru, koje se izvršavaju na standardizovanom hardveru umesto na specijalizovanim uređajima. Zato se često i nazivaju softverski definisanim mrežnim funkcijama, čija je uloga da obavljaju različite zadatke unutar mreže. To se pre svega odnosi na funkcije kao što su: softverski *firewall* za kontrolu i filtriranje mrežnog saobraćaja, softverski *load balancer* za raspodelu mrežnog saobraćaja između više servera radi optimizacije resursa, softveri za detekciju i prevenciju upada u mrežu, virtuelizovani NAT za transnslaciju mrežnih adresa i dr.

Virtuelizovane mrežne funkcije se pokreću na virtuelnim mašinama ili kontejnerima u okviru NFVI infrastrukture. Za njihovo pokretanje su zaduženi hipervizor ili kontejnerski softver, koji omogućavaju da se na istom hardveru pokrene i više virtuelizovanih mrežnih funkcija. Alokacijom resursa i upravljanjem životnim ciklusom virtuelizovanih mrežnih funkcija se bave različiti alati za orkestraciju (npr. NFV *Orchestrator*). Dakle, VNF je deo NFV kocepta, koji se fokusira na softver bez primene specijalizovanog hardvera za obradu paketa. Paketi podataka se inherentno obrađuju u softveru. Naime, kod koji obrađuje pakete je deo softvera virtuelizovanih mrežnih funkcija i izvršava se na CPU jedinici servera. U njega se implementiraju specijalne softverske tehnike, koje treba da kompenzuju nedostatak specijalizovanog hardvera. Cilj je da se korišćenjem naprednijih algoritama, unapredi obrada paketa kako bi ista radila bolje od tehnika obrade zasnovanih na hardveru. Važno je istaći i važnost kašnjenja i *jitter*-a za performanse prenosa podataka, posebno za usluge koje zahtevaju prenos u realnom vremenu, kao što su glas i multimedija. Stoga je obezbeđivanje virtuelne CPU jedinice korisno za svaku virtuelizovanu mrežnu funkciju, kako bi se izbegla degradacija performansi obrade paketa. Ovde se mora naglasiti i to, da virtuelizacija unosi dodatni overhead, koji može da uspori obradu paketa i utiče na protok podataka u manjoj ili većoj meri. Imajući u vidu, da se NFVI infrastruktura kreira na deljenoj fizičkoj infrastrukturi, onda je jasno da sloj virtuelizacije deluje kao posrednički interfejs između fizičkog i virtuelnog hardvera.

Komunikacija između virtuelizovanih mrežnih funkcija je takođe važna za stabilno funkcionisanje čitavog mrežnog okruženja. Ona se obično odvija preko hipervizora. Za komunikaciju sa fizičkim mrežnim karticama (pNIC) se koriste drajveri virtuelnog interfejsa [76]. Ova komunikacija utiče na performanse brzine obrade podataka. Kao alternativno rešenje, možemo koristiti paravirtualizaciju, što podrazumeva da su pNIC drajveri uključeni u VNF softver, čime se smanjuje degradacije performansi i utiče na fleksibilnost kompletnog rešenja.

Vreme potrebno za pokretanje fizičke mašine, odnosno za učitavanje operativnog sistema u glavnu memoriju ili RAM hosta, veće je od vremena potrebnog za pokretanje virtuelne mašine sa VNF-om ili za ponovno pokretanje sistema. S obzirom, da VNF radi u virtuelnom okruženju, vreme izvršenja navedenih radnji zavisi od faktora kao što su tehnika virtuelizacije i karakteristike virtuelnih mašina ili virtuelnih kontejnera [77]. Za razliku od ovih faktora, koji imaju predvidljiv uticaj na VNF-ove, postoje i faktori čiji bi uticaj mogao biti teži za predviđanje, kao što su opterećenje CPU-a fizičke mašine, dostupna memorija na čvrstom disku i vreme odziva orkestratora. Moramo napomenuti da je vreme potrebno za pokretanje, ponovno pokretanje ili uklanjanje pokrenutog kontejnera kraće od vremena potrebnog za pokretanje ili ponovno pokretanje virtuelnih mašina. Pojedinačni faktori ili kombinacije mogu ozbiljno uticati na vreme potrebno za instanciranje ili uklanjanje VNF-a. Pravovremeno instanciranje VNF-a je ključno da NFV postigne željene karakteristike. To se prvenstveno odnosi na dostupnost i primenu VNF-a u vreme kada je potrebno trenutno kreiranje ili uklanjanje VNF-ova. U takvim okolnostima, samo nekoliko dodatnih sekundi ili milisekundi može izazvati ozbiljan problem u dizajnu mreže.

5.3.1.3. MANO blok

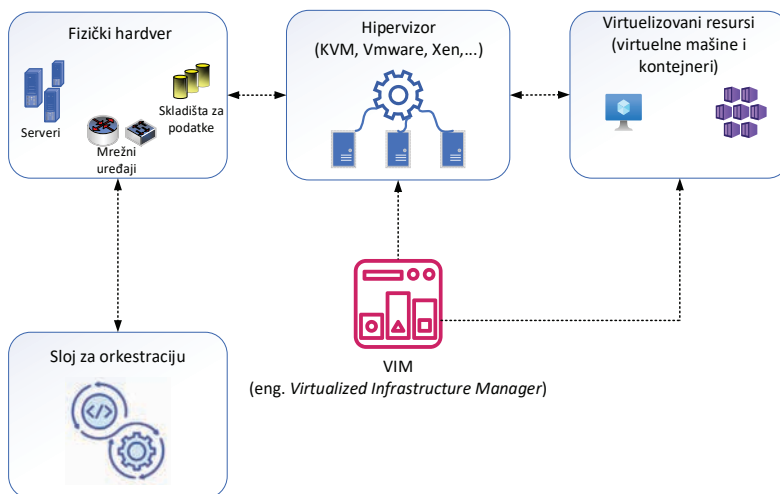
Sa virtuelizacijom mrežnih funkcija, u računarske mreže se unosi visok nivo programabilnosti. To zahteva primenu novog metoda upravljanja mrežnom infrastrukturom, ali i mehanizama za orkestraciju resursa i usluga. Primenom virtuelizacije, softverska implementacija mrežnih funkcija je u velikoj meri postala nezavisna od računarske memorije i mrežnih resursa, što je omogućilo izgradnju novih virtuelnih entiteta (VNF instance) i uspostavljanje različitih veza u okviru NFV infrastrukture (NFVI). Većina mrežnih usluga, u modernim mrežnim infrastrukturom, obezbeđuje se povezivanjem virtuelnih entiteta, koji koriste virtuelizovane resurse. Ovi resursi su generalno povezani sa virtuelnim mašinama, koje se mogu posmatrati kao kontejneri koji pokreću programe i izvršavaju aplikacije na softverskim računarskim resursima (umesto fizičkog računara). Dakle, stvara se virtuelno okruženje na fizičkom hardveru, koje funkcioniše kao virtuelni računarski sistem sa svojom centralnom procesorskom jedinicom (CPU), memorijom, mrežnim interfejsima i skladištem podataka (sa skladištem organizovanim na blok ili pristup fajlu). Činjenica je, da u praksi, ali dosta ređe, postoje i primeri da se određena mrežna usluga realizuje povezivanjem virtuelnih entiteta sa mrežnim funkcijama koje se realizuju tradicionalno, odnosno korišćenjem namenskog hardvera. U takvim slučajevima, mrežni servisi se mogu u potpunosti ili delimično realizovati povezivanjem virtuelizovanih i nevirtuelizovanih resursa.

Za implementaciju funkcija virtuelne mreže, od posebnog je značaja orkestracija virtuelnih resursa na virtuelnim mašinama, što podrazumeva dodelu potrebnih virtuelnih resursa za oslobađanje onih koji više nisu potrebni. Orkestracija je složen zadatak, posebno imajući u vidu različite zahteve i ograničenja (npr. neke VNF instance zahtevaju malo kašnjenje, dok druge zahtevaju obezbeđivanje velikog propusnog opsega između učesnika u komunikaciji). Ovde treba napomenuti, da raspodela potrebnih resursa ima dinamički karakter, posebno ako se ima u vidu da se potrebe VNF instanci za resursima menjaju i da se na njih treba reagovati skoro u realnom vremenu. Drugim rečima, NFV koncept se oslanja na proces orkestracije usluge kao na model definicije usluge koji koristi VNF instance i primenjuje različite topologije njihovog povezivanja.

Ključna komponenta u okviru NFV arhitekture, koja definiše kako se upravlja virtualizovanim mrežnim funkcijama, vrši orkestracija i iste integrišu u mrežne servise je MANO blok. On se sastoji od tri osnovne funkcionalne komponente:

- VIM (eng. *Virtualized Infrastructure Manager*) – upravlja fizičkom i virtuelnom infrastrukturom (virtuelne mašine, skladišni kapaciteti i mrežne konekcije), tako što vrši nadgledanje infrastrukture i obezbeđuje potrebne resursa za virtualizovane mrežne funkcije i mrežne servise i komunicira sa hipervizorima.
- VNFM (eng. *VNF Manager*) – upravlja životnim ciklusom virtualizovanih mrežnih funkcija, tako što upravlja njihovom konfiguracijom, instanciranjem i statusom, kao i rešavanjem zadataka vezanih za skaliranje i monitoring performansi svake pojedinačne virtualizovane mrežne funkcije.
- NFVO (eng. *NFV Orchestrator*) – orkestrira mrežnim servisima i upravlja životnim ciklusom mrežnih servisa, tako što implementira mrežne servise korišćenjem različitih virtualizovanih mrežnih funkcija, upravlja resursima iz različitih domena ili platformi kako bi mrežni servisi ispunili zahteve, ali i vrši koordinacija između VIM-ova i VNFM-ova.

VIM je ključna komponenta u okviru NFV arhitekture, jer je odgovorna za upravljanje virtualizovanom infrastrukturom, koja je neophodna za uspešan rad virtualizovanih mrežnih funkcija. To se pre svega odnosi na upravljanje virtuelnim mašinama ili kontejnerima, mrežnim resursima (virtuelne mreže, portovi, tuneli) i resursima za skladištenje podataka (diskovi i baze podataka). Jasno je, da VIM ima snažnu interakciju upravo sa slojem za virtualizaciju (Slika 45), tj. sa hipervizorima (npr. KVM, VMware, Xen i dr.) i kontejnerskim softverom, jer su oni nadležni za kreiranje i upravljanje virtuelnim resursima. To ukazuje, da posredno, VIM ima važnu ulogu i u obezbeđivanju kontinuirane interakcije sa fizičkom infrastrukturom. Ova interakcija podrazumeva nadgledanje i upravljanje fizičkim serverima, mrežnim uređajima i skladišnim sistemima.



Slika 45. VIM u NFV arhitekturi

Potrebno je naglasiti, da je VIM zadužen za orkestraciju resursa, koja podrazumeva automatizovano upravljanje resursima kako bi se obezbedila efikasnost, skalabilnost i visoka dostupnost. U ovom procesu, VIM ima više zadataka. Prvi zadatak je vezan za prikupljanje podataka o dostupnim resursima na svim nivoima infrastrukture, odnosno podatke o fizičkim resursima (serveri, mrežni uređaji i skladišta za podatke), hipervizorskim resursima (kapacitet procesora, memorije, diska), virtuelizovanim resursima (virtuelne mašine, kontejneri, virtuelne mreže, virtuelizovane baze podataka i dr.). Ovi podaci mu omogućavaju, da stekne jasan uvid u infrastrukturu i raspoloživost resursa i započne automatizovanu orkestraciju resursa. Na osnovu ovih podataka, VIM može da započne alokaciju resursa virtuelnim mašinama i kontejnerima, kao što su CPU, memorija, skladište i mrežni kapacitet. Ovaj proces je dinamičan i zavisi od opterećenja u sistemu i potreba aplikacija. Da bi pravilno rasporedio resurse i smanjio preopterećenja, VIM vrši optimizaciju resursa, koristeći odgovarajuće algoritme. To bi značilo, ukoliko je jedan server prekomerno opterećen, VIM mora da izvrši migraciju virtuelne mašine na drugi server koji je manje opterećen. Poseban zadatak VIM ima u pogledu upravljanja virtuelnim mrežama. Naime, VIM omogućava dinamičko kreiranje i konfiguraciju virtuelnih mreža (npr., VLAN-ova, ili VPN-ova) i time obezbeđuje optimalno usmeravanje saobraćaja između različitih delova infrastrukture.

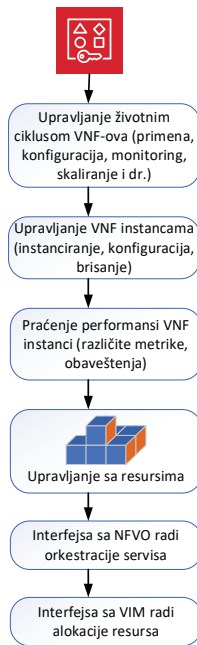
Ovde je potrebno naglasiti, još jednu bitnu funkciju VIM-a, koja se odnosi na kreiranje i skaliranje virtuelnih mašina i kontejnera. On omogućava, da se virtuelni entiteti brzo kreiraju na osnovu šablona ili potreba korisnika. Naime, kroz API-je, korisnici mogu zahtevati resurse, koje im VIM dodeljuje prema unapred definisanim politikama. To znači, da u skladu sa potrebama korisnika ili aplikacija, VIM može automatski, da skalira kapacitete virtuelnih entiteta ili infrastrukturu, tako što dodaje više resursa i pokreće dodatne virtuelne entitete da bi zadovoljio povećane zahteve (pri tome, VIM stalno prati performanse infrastrukture i virtuelnih entiteta). Na osnovu prikupljenih podataka, on kreira izveštaje i obezbeđuje administratorima uvid u stanje resursa. To je od velikog značaja, jer oni preko VIM-a mogu primeniti različite politike za upravljanje resursima. Ovim polisama se određeni resursi dodeljuju specifičnim aplikacijama ili korisnicima sa višim prioritetima ili se za kritične aplikacije rezervišu mrežni i procesorski kapaciteti i obezbeđuju stabilne performanse.

Poseban značaj ima komunikacija sa VIM-om. Administratori mogu pristupiti VIM-u, preko API-a i grafičkih interfejsa, koji pružaju mogućnost jednostavnog upravljanja resursima, automatskog skaliranja i orkestracije, kao i praćenja resursa. S druge strane, komunikacija sa NFV orkestratorom i VNFM-om se odvija preko standardizovanih API-a, kao što su OpenStack API ili Kubernetes API. Iz navedenog se jasno zaključuje, da VIM povezuje fizičku i virtuelnu infrastrukturu i omogućava apstrakciju fizičkih resursa kako bi se predstavili kao virtuelni resursi koji se lako mogu koristiti i orkestrirati. Što je još važnije, on obezbeđuje efikasno skaliranje mrežnih funkcija prema zahtevima i omogućava rad sa različitim platformama (*OpenStack, Kubernetes, VMware* i dr.).

VNFM (eng. *Virtualized Network Functions Manager*) je sledeća važna komponenta u MANO arhitekturi. Ona se koristi za upravljanje životnim ciklusom virtualizovanih mrežnih funkcija u NFV okruženjima. Ona je zadužena za efikasno upravljanje virtuelnim mrežama i omogućava pružanje, upravljanje i održavanje virtualizovanih mrežnih funkcionalnosti na osnovu zahteva korisnika ili mrežnih operatora (Slika 46).

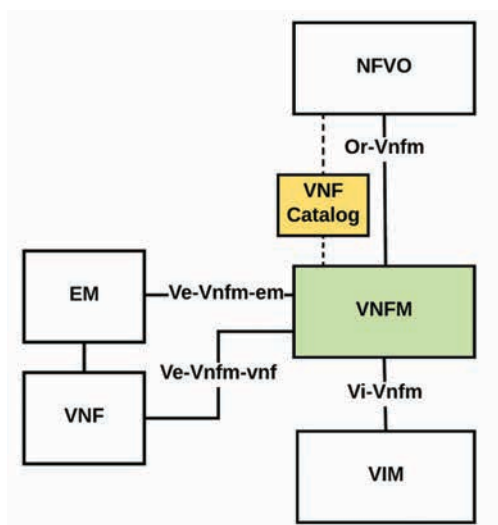
To znači, da VNFM upravlja svim fazama životnog ciklusa virtuelizovanih mrežnih funkcija, uključujući:

- instalaciju – procenjuje i postavlja virtuelizovane mrežne funkcije na virtualizovanu infrastrukturu,
- konfiguraciju parametara kojima se uređuje rad virtuelizovanih mrežnih funkcija u skladu sa politikama i zahtevima,
- pokretanje virtuelizovanih mrežnih funkcija,
- praćenje performansi i statusa virtuelizovanih mrežnih funkcija u realnom vremenu,
- automatsko skaliranje virtuelizovanih mrežnih funkcija prema potrebama mreže (dodavanje ili smanjenje instanci) i
- deaktiviranje i brisanje virtuelizovanih mrežnih funkcija kada više nisu potrebne.



Slika 46. NFVM u NFV arhitekturi

Dakle, VNFM upravlja životnim ciklusom VNF instanci, uključujući instalaciju, skaliranje, održavanje i gašenje. U tom procesu, on koristi tzv. *element manager* (EM) za detaljan nadzor i upravljanje životnim ciklusom pojedinačnih instanci VNFs. Naime, EM prikuplja podatke o performansama i statusu VNF instanci, dok VNFM te podatke koristi za donošenje odluka o skaliranju, migraciji ili gašenju VNF-a. Drugim rečima, EM je odgovoran za upravljanje pojedinačnim uređajem ili komponentom unutar infrastrukture, kao što su fizički uređaji ili VNF instance. On se fokusira na detalje upravljanja uređajem, uključujući konfiguraciju, monitoring, i otklanjanje grešaka na nižem nivou (Slika 47).



Slika 47. Interakcija VNFM sa *element manager*-om (EM) [78]

VNFM je visoko apstrahovana funkcija koja interaguje sa širim ekosistemom i često je integrisan sa drugim MANO komponentama, kao što su VIM i NFVO. On koristi VIM za obezbeđivanje resursa potrebnih za pokretanje virtuelizovanih mrežnih funkcija. Integraciju sa NFVO, koristi da bi orkestrirao celokupan životni ciklus svih virtuelizovanih mrežnih funkcija i upravljao resursima na nivou celokupne mreže. U praksi se često može sresti rešenje, gde je VNFM integrisan i sa OSS (eng. *Operational Support Systems*) sistemom za perativno upravljanje, praćenje i podršku mrežnim uslugama i infrastrukturnim resursima, kao i sa BSS (eng. *Business Support Systems*) sistemom za upravljanje mrežnim servisima i naplatom. Integracija i interfejsi ka svim ovim komponentama predstavljaju vezu, preko koje VNFM prikuplja podatke o performansama i statusu virtuelizovanih mrežnih funkcija i šalje administratorima statističke izveštaje o njihovima performansama, ali i upozorenja u slučaju problema ili prekida. To administratorima omogućava da prate kvalitet usluge i izvrše optimizaciju mreže u skladu sa dogovorenim SLA (eng. *Service Level Agreement*).

Virtuelizacijom mrežnih funkcija, u računarske mreže se unosi mnogo viši nivo programabilnosti. To podrazumeva primenu novog modela upravljanja mrežnom infrastrukturom, ali zahteva orkestraciju resursa i usluga. Za razliku od tradicionalnih mreža, gde su mrežne funkcije vezane za određeni hardver, u savremenim računarskim mrežama se grade virtuelni entiteti (tzv. VNF instance) i uspostavljaju odgovarajuće veza u okviru NFVI infrastrukture. Za njihovu implementaciju je od posebnog značaja orkestracija virtuelnih resursa na virtuelnim mašinama, što podrazumeva dodelu novih resursa ili oslobađanje onih za kojima je prestala potreba. Proces orkestracije je složen zadatak, posebno imajući u vidu različite zahteve i ograničenja vezane za topologiju povezivanja VNF instanci i potrebu za dinamičkom alokacijom resursa, jer se potrebe

VNF instanci za resursima menjaju skoro u realnom vremenu. Sistemi za upravljanja i orkestraciju u okviru NFV MANO bloka su prvenstveno razvijeni za upravljanje na agilni i fleksibilan način. U tom smislu, ETSI ISG je usvojio odgovarajuća uputstva, koja opisuju zahteve i standarde koje softver i hardver NFV MANO sistema treba da zadovolji u pogledu realizacije virtuelizovanih mrežnih funkcija [79].

Ključne funkcije koje treba da obezbedi NFVO su:

- optimalno upravljanje infrastrukturnim resursima za potrebe VNF instanci,
- automatsko kreiranje, konfiguracija i upravljanje mrežnim servisima koji se sastoje od jedne ili više VNF instanci,
- automatizacija životnog ciklusa mrežnih usluga – pokretanje, skaliranje, migracija i deaktivacija,
- integracija i komunikacija sa drugim MANO komponentama (VNFM i VIM),
- upravljanje *end-to-end* mrežnim servisima koji se „prostiru“ kroz više domena ili *data centara*,
- monitoring i analitika i
- automatska orkestracija u pogledu definisanja mrežnih usluga i njihovih zavisnosti.

Da bi operateri primenili kvalitetno rešenje, neophodno je definisati ključne indikatore performansi (eng. *Key Performance Indicators* - KPI) MANO sistema. To je jedini način, na osnovu kojeg bi se mogla izvršiti validna procena performansi predloženog rešenja. U realnim okruženjima to predstavlja ozbiljan izazov, posebno kada se uzme u obzir dinamika i fleksibilnost usluga koje pružaju VNF instance [80]. Zato se često vrši klasifikacija ključnih indikatora na funkcionalne i operativne. Funkcionalni indikatori performansi se odnose na metrike, koje se koriste za merenje performansi, efikasnosti i pouzdanosti funkcija koje pružaju različite MANO komponente (NFVO, VNFM, VIM). Ove metrike služe kao osnov za evaluaciju funkcionalnosti upravljanja virtualizovanim mrežnim funkcijama i mrežnim servisima. Primer za to su:

- količina angažovanih resursa (eng. *resource footprint*),
- spisak različitih VIM platformi (rešenja za upravljanje virtuelnom infrastrukturom) kojima MANO sistem može da upravlja,
- broj virtuelnih menadžera infrastrukture kojima jedan MANO sistem može efikasno da upravlja,
- maksimalni broj VNF instanci koje MANO sistem može nadgledati i upravljati u okviru NFV infrastrukture,
- podrška za DevOps (procedure i alati koji povećavaju isporuku aplikacija i usluga u poređenju sa tradicionalnim procesom razvoja softvera), upravljanje VNF *image*-ima i integrisano praćenje.

Za razliku od funkcionalnih, operacioni KPI se odnose na metrike koje mere operativnu efikasnost, stabilnost i pouzdanost MANO sistema u svakodnevnom radu. Oni su usmereni na procese, performanse infrastrukture i mogućnosti automatskog reagovanja MANO komponenti u realnom vremenu. To se pre svega odnosi na:

- prosečno vreme potrebno za obavljanje osnovnih operacija, kao što su aktivacija mrežnog servisa, skaliranje ili gašenje VNF-a,
- maksimalan broj operacija koje MANO sistem može da obradi u određenom vremenskom intervalu,
- dostupnost sistema (eng. *system availability*) – koliko vremena su MANO komponente dostupne za upotrebu,
- prosečno vreme između dva kvara na MANO sistemu (eng. *Mean Time Between Failures* – MTBF),
- prosečno vreme potrebno da se kvar otkloni (eng. *Mean Time to Repair* - MTTR),
- maksimalan broj istovremenih zahteva koje MANO može obraditi bez degradacije performansi,
- vreme potrebno za horizontalno dodavanje VNF instanci ili vertikalno dodavanje resursa (skaliranje mrežnog servisa ili VNF instance),
- koliko operacija MANO sistem izvršava automatski bez ručne intervencije,
- procenat uspešno automatski otklonjenih grešaka u sistemu,
- prosečno kašnjenje u izvršavanju zahteva, kao što su orkestracija resursa ili aktivacija servisa,
- procenat izgubljenih paketa tokom prenosa podataka između MANO komponenti,
- procenat iskorišćenosti dostupnih infrastrukturnih resursa (CPU, memorija, skladište),
- procenat uspešnih alokacija resursa u odnosu na ukupan broj zahteva,
- procenat operacija koje nisu uspešno završene i
- preciznost predviđanja i izvršenja potreba za resursima.

5.4. REFERENTNE TAČKE U NFV ARHITEKTURI

U ETSI NFV programskom okviru postoje jasno definisane referentne tačke, koje predstavljaju logičke ili bolje rečeno standardizovane granice između različitih funkcionalnih blokova unutar ETSI NFV arhitekture. One ne označavaju fizičke ili konkretne veze, već standarde koji omogućavaju da komponente NFV programskog okvira međusobno komuniciraju. Drugim rečima, referentne tačke u NFV arhitekturi definišu standardizovane granice i pravila za interakciju između različitih sistema i komponenti u NFV mrežama. Prema ETSI NFV standardima, neke od ključnih referentnih tačaka u NFV arhitekturi uključuju:

- *Ve-Vnfm* – definiše komunikaciju između VNF instanci i VNF menadžera i upravlja se životnim ciklusima VNF-ova, uključujući implementaciju, skaliranje, ažuriranja i gašenje.
- *Vi-Vnfm* – predstavlja standard za komunikaciju između VIM-a i VNF menadžera, kojim se šalju zahtevi menadžeru VNF instance da dodeli resurse na osnovu informacija o konfiguraciji i stanju virtuelizovanih hardverskih resursa.
- *Or-Vnfm* – označava standard za komunikaciju između NFV orkestratora (NFVO) i VNFM-ova, koji upravljaju životnim ciklusom pojedinačnih VNF instanci.
- *Or-Vi* – omogućava komunikaciju između NFV Orkestratora i VIM-a, kako bi se obezbedilo upravljanje i alokacija fizičkih i virtuelizovanih infrastrukturnih resursa.
- *Vnfm-Vnfm* – predstavlja standard za međusobnu komunikaciju između različitih VNF menadžera, što omogućava koordinaciju među različitim mrežnim funkcijama.
- *Vi-Vi* – standard koji definiše međusobnu komunikaciju između različitih VIM-ova u slučaju da NFV infrastruktura koristi više različitih VIM sistema za upravljanje resursima.

U načelu, referentne tačke služe kao apstraktni model za komunikaciju, dok se za stvarnu stvarnu implementaciju tih komunikacija u mreži koriste različiti interfejsi.

ZAKLJUČAK

Primena NFV u SDN okruženjima donosi značajne prednosti u smislu fleksibilnosti, skalabilnosti i smanjenja troškova, ali sa sobom nosi i izazove vezane za integraciju, performanse, sigurnost i kompatibilnost sa postojećom infrastrukturom. Da bi ove tehnologije bile uspešno implementirane i ušle u širu upotrebu, potrebno je prevazići ove izazove uz pomoć pažljivog planiranja, razvoja standarda i usklađivanja različitih tehnologija i protokola. Dodatnu prednost ovoj integraciji, daje mogućnost implementacije automatizovanog upravljanja mrežnim funkcijama i resursima čime se poboljšava sveukupna efikasnost mrežnih okruženja i omogućava lakša implementacija svakog novog servisa i promptna reakcija na potrebe tržišta.

Međutim, uprkos brojnim prednostima, integraciju ova dva mrežna koncepta prate i sledeći izazovi:

- kompleksnost povezivanja, jer je potrebno integrisati različite arhitekture i slojeve,
- povećanje latencije i pogoršanje performansi, kao rezultat primene različitih tehnika virtualizacije, pogotovo ukoliko infrastruktura nije optimizovana,
- Bezbednosni napadi na virtualizovane mrežne funkcije i centralizovano upravljanje, ako isti nisu pravilno zaštićeni,
- kompatibilnost sa postojećom infrastrukturom mreže – migracija sa tradicionalnih sistema na SDN/NFV arhitekturu može biti skupa i složena i
- nedostatak interoperabilnosti – različite implementacije mogu otežati interoperabilnost između rešenja različitih vendora.



POGLAVLJE 6



KONTROLNA RAVAN U SDN MREŽAMA

6. KONTROLNA RAVAN U SDN MREŽAMA

6.1. UVOD

Razdvajanje kontrolne od ravni podataka i njena centralizacija, predstavljaju ključan koncept na kome se zasniva SDN tehnologija. Razvoj ovog koncepta je povezan sa evolucijom arhitekture računarskih mreža i potrebom da se obezbedi novi model upravljanja. Na njegovu brzinu, presudno su uticali pojava novih tehnologija i servisa, kao i eksponencijalni rast broja korisnika i saobraćaja u mrežama. Razvoj je išao postepeno, u fazama. U ranim mrežama, kontrolna i ravan podataka su bile čvrsto integrisane u mrežnim uređajima. Svaki uređaj je donosio sopstvene odluke o rutiranju i prosleđivanju paketa. Komunikacija između uređaja zasnivala se na primeni dinamičkih protokola za rutiranje saobraćaja, sa distribuiranom arhitekturom, kao što su RIP (eng. *Routing Information Protocol*), OSPF ili IS-IS protokol. Primenu ovih protokola, pratila sa ograničenja vezana za nedostatak globalnog pogleda na mrežu i teškoće u upravljanju velikim i kompleksnim mrežama.

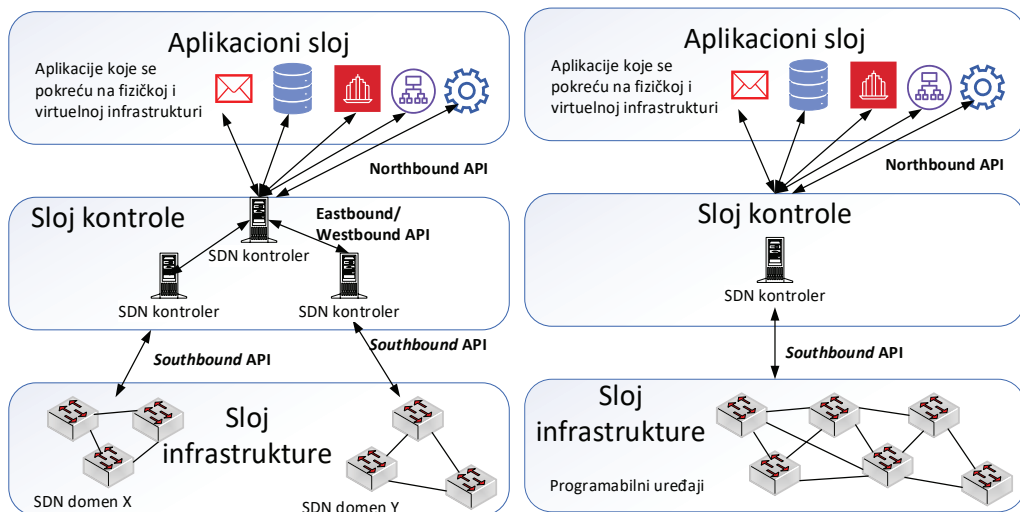
Početkom 90-ih godina prošlog veka, pojavili su se prvi koncepti centralizovane kontrole, koji su razvijani u akademskim projektima, kao što je RCP (eng. *Routing Control Platform*). Rađeni su eksperimenti, koji su imali za cilj, da se implementira centralizovano upravljanje rutiranjem i olakša donošenje odluka. Paralelno se dešavao napredak u razvoju hardvera i softvera. Povećana je procesorska snaga računarskih uređaja i implementirani su distribuirani računarski sistemi, što je otvorilo mogućnost, da se nađe rešenje za centralizovano upravljanje mrežama. Tada je postalo sasvim jasno, da statička (manuelna) konfiguracija i slaba fleksibilnost tradicionalnih mreža predstavljaju glavne probleme u razvoju računarskih mreža i primeni novih vidova komunikacija i servisa. Tradicionalne mreže nisu mogle da zadovolje zahteve korisnika u velikim računarskim okruženjima, kao što su *data* centri, gde je neophodno vršiti brze promene u konfiguracijama i lakše implementirati nove servise.

Strogo posmatrano, razvoj SDN koncepta je započeo 2000-te godine sa razvojnim projektima, kao što je bio 4D projekt (eng. *Decision, Dissemination, Discovery, and Data planes*) [81], koji se bavio iznalaženjem rešenja za odvajanje kontrolne od ravni podataka. Fokus projekta je bio, na donošenju centralizovanih odluka koje bi bile distribuirane uređajima u ravni podataka. Nekoliko godina kasnije, na Stanford univerzitetu je razvijen *OpenFlow*, prvi protokol za komunikaciju između centralizovane kontrolne ravni i ravni podataka. Ovaj protokol je postao osnovna pokretačka snaga za dalja istraživanja i implementaciju SDN-a. Do komercijalne primene ove tehnologije, došlo je kada su kompanije, kao što je Google, počele sa korišćenjem SDN tehnologije i primenom SDN kontrolera, sa ciljem da optimizuju mreže u *data* centrima. Razvijeni su standardi za *northbound* i *southbound* API-i, koji su omogućili kontrolnoj ravni da centralizovano upravlja mrežnim uređajima. Time se došlo do bolje iskorišćenosti raspoloživih resursa, jednostavnije konfiguracije i veće fleksibilnosti u radu sa narastajućom količinom novih i sve sadržajnijih zahteva iz mreže.

Na današnjem stepenu razvoja ove tehnologije, može se jasno identifikovati trend, koji vodi ka širokoj primeni veštačke inteligencije (eng. *Artificial Intelligence* - AI) i mašinskog učenja (eng. *Machine-Learning* - ML) u kontrolnoj ravni, ali i ka odvajanju upravljačkog od kontrolnog dela na njoj. Moderne SDN mreže danas poseduju brojne napredne funkcionalnosti, kao što su centralizovana kontrolna ravan koristi) za donošenje odluka i automatizacija mrežnih operacija (npr. orkestracija virtuelnih mreža na deljenoj fizičkoj infrastrukturi). Sa *data* centara, primena SDN tehnologije se proširila na WAN mreže (npr. SD-WAN mreže), cloud i mobilne mreže (npr. 5G/6G). Sve ovo je uticalo na pojavu još naprednijih rešenja, koja se zasnivaju na decentralizaciji u centralizovanoj kontrolnoj ravni, što uključuje primenu *eastbound* i *westbound* API-a za komunikaciju između SDN kontrolera unutar i između različitih domena. Danas se veliki broj rešenja i zasniva na hibridnom modelu, gde centralizovana kontrola koristi lokalne odluke za bolje skaliranje i otpornost.

6.2. ARHITEKTURA KONTROLNE RAVNI

U SDN implementacijama, kontrolna ravan predstavlja ključnu komponentu, jer upravlja tokovima paketa sa infrastrukturnih uređaja (fizičkih ili virtuelnih) i sarađuje se aplikacija na najvišem sloju arhitekture. Njen mozak je centralizovani mrežni operativni sistem, koji pruža funkcionalnosti, koje poseduje i osnovni operativni sistem (pokretanje programa, upravljanje I/O operacijom, bezbednost, funkcije vezane za topologiju, prosleđivanje najkraće putanje i dr.). Danas u realnim okruženjima, gde je implementirana SDN tehnologija, postoje dva pristupa u organizaciji kontrolne ravni, tj. dve arhitekture – centralizovana i distribuirana (Slika 48) [2].



Slika 48. Centralizovana i distribuirana kontrolna ravan SDN mreže

U centralizovanoj arhitekturi kontrolne ravni, postoji jedan kontrolni entitet (kontroler), koji prima sve informacije iz mreže, a zatim na osnovu njih donosi odluke o usmeravanju saobraćaja i primeni različitih politika. Prednost ovako koncipirane arhitekture se pre svega ogleda u jednostavnosti upravljanja, tj. u centralizovanom donošenju odluka, čime se omogućava jednostavno praćenje saobraćaja i administracija u mreži. Ove prednosti su ostvarene, pre svega, zahvaljujući činjenici da kontroler poseduje globalni uvid u mrežnu infrastrukturu. Ovi je osnovna prednost, koja proističe iz centralizacije kontrolne ravni i činjenice da kontroler ima pristup svim informacijama o mreži i da je sposoban, da optimalno usmerava saobraćaj. Kao centralni i jedinstven element kontrole, on lako uvodi nove funkcije i ažuriranja, što je dodatna prednost. Međutim, u određenim slučajevima, centralizacija kontrolne ravni može da nosi i određene izazove. Jedan od njih je skalabilnost, jer sa rastom mreže, jedan kontroler može postati usko grlo zbog preopterećenja. Ako je taj kontroler i fizički daleko od mrežnih uređaja, to može dovesti do kašnjenja u komunikaciji.

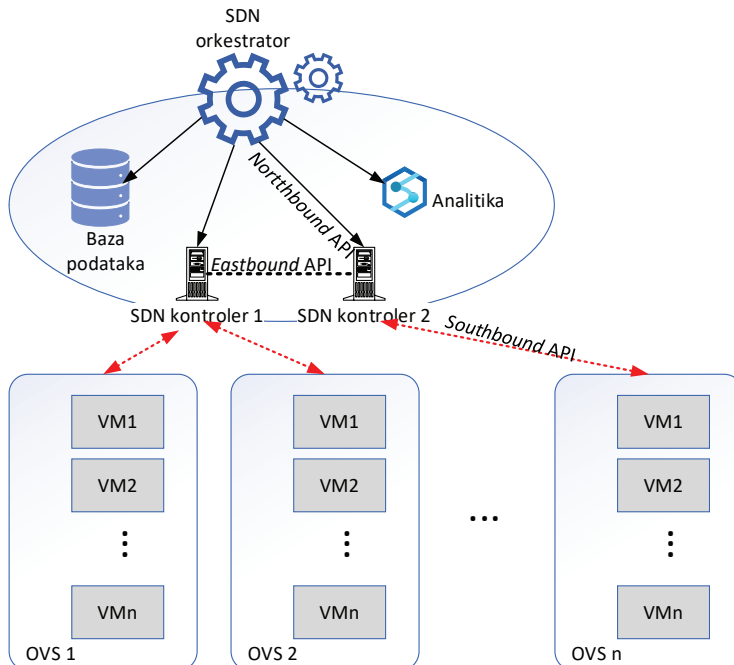
U distribuiranoj arhitekturi kontrolne ravni, zadaci koji se odnose na kontrolu mreže, podeljeni su između više kontrolera. Svaki kontroler upravlja svojim segmentom mreže i sprovodi odgovarajuće politike, koje se odnose na rutiranje saobraćaja i kvalitet servisa. Koherentnost u ponašanju mreže se obezbeđuje kroz efikasnu komunikaciju i koordinaciju aktivnosti između kontrolera. Distribuirana arhitektura u kontrolnoj ravni ima brojne prednosti, koje se pre svega odnose na mogućnost podele zadataka između više kontrolera, čime se smanjuje opterećenje na pojedinačnim uređajima i omogućava bolja skalabilnost. Sami kontroleri se mogu implementirati bliže mrežnim uređajima, što predstavlja dodatnu prednost, jer se smanjuje kašnjenje u komunikaciji. Na kraju, ono što je možda najvažnije, ako jedan od kontrolera ne funkcioniše, ostali kontroleri preuzimaju njegove zadatke (stvora se otpornost na greške). Uprkos brojnim prednostima, distribuirana organizacija kontrolne ravni nosi određene izazove, kao što je kompleksnost upravljanja. Između kontrolera mora da postoji koordinacija, što u velikim mrežama može predstavljati izazov. Naime, veći broj kontrolera može dovesti do većeg protoka podataka između njih samih (veći *overhead* saobraćaj). To povećava složenost sveukupne arhitekture, ali i troškove.

Dakle, izbor između centralizovane i distribuirane kontrolne ravni zavisi od specifičnosti zahteva koji dolaze iz mreže (npr. skalabilnost, performanse i složenost upravljanja). Centralizovani pristup je jednostavniji, ali može proizvesti određena ograničenja u velikim mrežama, dok distribuirani pristup nudi veću skalabilnost i otpornost na greške, ali je složeniji za implementaciju i održavanje.

6.3. KOMPONENTE KONTROLNE RAVNI

Kao i u tradicionalnim mrežama, komponente kontrolne ravni su i u SDN mrežama odgovorne za ključne funkcionalnosti. To su entiteti koji donose odluke o upravljanju mrežnim saobraćajem. Suštinska prednost SDN arhitekture leži u tome, što su ove komponente centralizovane i odvojene od ravni podataka, što omogućava veću fleksibilnost i skalabilnost u mreži. Osnovne komponente kontrolne ravni su (Slika 49):

- SDN kontroler – donosi odluke o rutiranju, balansiranju opterećenja, QoS, bezbednosti i drugim funkcijama, koje u formi instrukcija preko *southbound* interfejsa prenosi mrežnim uređajima u ravni podataka.
- SDN orkestrator – kreira globalni uvid u stanje svih mrežnih resursa i koordinira sa njihovom alokacijom, kako bi se optimizovale mrežne performanse.
- baze podataka sa informacijama o stanju mreže (eng. *Network State Database*) – čuvaju informacije o stanju mreže (topologija, resursi i trenutno opterećenje) i koriste se za analitiku i donošenje odluka u realnom vremenu
- interfejsi (API-ji) – koriste se za komunikaciju između kontrolne i ravni podataka (*Southbound* interfejs), aplikacione i kontrolne ravni (*Northbound* interfejs), kao i za komunikaciju na distribuiranoj SDN kontrolnoj ravni (*Eastbound* interfejs) jednog domena ili za komunikaciju između SDN kontrolera koji pripadaju različitim domenima (*Westbound* interfejs).



Slika 49. Osnovne komponente kontrolne ravni SDN mreže

6.3.1. SDN kontroler

SDN kontroler (eng. *Software-Defined Networking Controller*) je centralna komponenta u SDN mreži, koja upravlja njenom logikom. To znači, da je njegov osnovni zadatak prosleđivanje instrukcija uređajima u ravni podataka, o tome, kako će saobraćaj teći kroz mrežu. Za tu komunikaciju, SDN kontroler koristi *Southbound* API-je, dok za interakciju sa aplikacijama i orkestratorom koristi *Northbound* API-je. Praktično posmatrano, SDN kontroler izvršava brojne zadatke, od kojih je sigurno najvažniji centralizovano upravljanje infrastrukturu mreže. Da bi upravljanje u SDN mrežama bilo efikasno, ključnu ulogu ima saradnja između SDN kontrolera i SDN orkestratora i u tom smislu, potrebno je naglasiti, da oni imaju različite uloge i funkcije i da se njihova podela odgovornosti svodi na nivo upravljanja i apstrakcije u mreži.

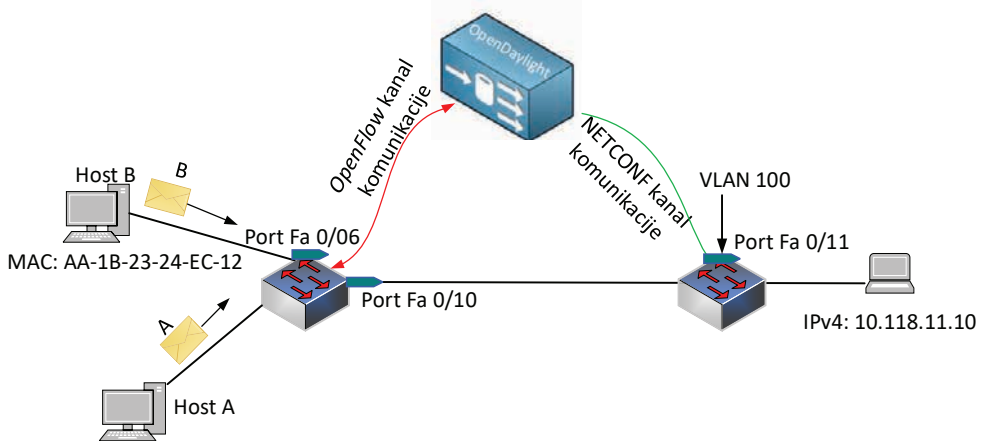
SDN kontroler upravlja operativnim detaljima mreže na nižem nivou, direktno komunicirajući sa mrežnim uređajima. On kontinuirano prikuplja podatke o mrežnim uređajima i saobraćaju, radi njihovu analizu, kako bi došao do informacija kao što su latencija, propusnost ili gubici paketa. Drugim rečima, SDN kontroler poseduje ažurnu mapu (topologiju) mreže i blagovremeno prosleđuje mrežnim uređajima instrukcije, kako da urede svoje tabele tokova (eng. *flow table*). Njegov zadatak, je da „prenese“ odluke o optimalnim putanjama za različite tipove saobraćaja u mreži, u skladu sa politikama, koje su definisane od strane orkestratora. Operativna bezbednost u mreži se garantuje, kroz implementaciju pravila kontrole pristupa i detekciju potencijalnih pretnji, kao i kroz enkripciju komunikacije i primenu IPS/IDS sistema i dr.

U realnim okruženjima postoje različite implementacije SDN kontrolera, koje se mogu svrstati u dve grupe kontrolera – centralizovane i distribuirane. Između ove dve grupe kontrolera postoje razlike u načinu na koji upravljaju mrežom i pružaju kontrolne funkcije. Njihove karakteristike direktno utiču na performanse, skalabilnost, otpornost i složenost mreže. Karakteristike centralizovanih SDN kontrolera su pre svega vezane za njihovu arhitekturu, tj. činjenicu da postoji jedan centralni entitet (kontroler) koji upravlja celom mrežom, sa kojim komuniciraju svi uređaji u mreži. To ukazuje na jednostavnost upravljanja, jer se sa jednog mesta implementiraju različite politike i prati stanje mreže. Dakle, kontroleri sa centralizovanom arhitekturom imaju potpunu vidljivost i globalni pregled mreže, što omogućava optimalno donošenje odluka. Međutim, u slučaju velike količine saobraćaja, oni mogu predstavljati uska grla u saobraćaju i to u velikim mrežama. Štaviše, centralizovani kontroler je tačka potencijalnog kvara eng. (*Single Point of Failure*), pa se često u SDN mrežama koriste redundantni kontroleri (*backup*) za povećanje pouzdanosti. Upotreba ovih kontrolera, svodi se na male mreže i *data centre* sa ograničenim brojem uređaja.

Kao primer implementacije centralizovane kontrolerske platforme, u realnim okruženjima se za širok spektar aplikacija koristi *OpenDaylight* (ODL) kontroler (u praksi se mogu sresti i druge centralizovane kontrolerske platforme, kao što su *Beacon*, *Maestro*, *Rosemary*, *NOX-MT*, *Pox*, *Floodlight* i *Meridian*). To uključuje upravljanje mrežom, orkestraciju, automatizaciju i razvoj novih mrežnih servisa. Kontroler je otvorenog koda i razvijen je unutar *Linux Foundation* projekta, sa osnovnim karakteristikama, kao što su centralizovana kontrola i brza konfiguracija mrežnih servisa putem REST API-ja [82].

Njegova komunikacija sa uređajima u mrežnoj infrastrukturi se odvija preko *Southbound* interfejsa. Ona uključuje primenu *OpenFlow*, *NETCONF*, *SNMP* i drugih protokola. Najčešće se koristi *OpenFlow* protokol, čiji zadatak je da omogućiti, da se *switch*-evima pošalju instrukcije za kreiranje i ažuriranje zapisa u tabelama tokova. Ovi zapisi se koriste za procesiranje dolaznih paketa i izvršavanje različitih akcija (npr. prosleđivanje ili odbacivanje paketa, menjanje zaglavlja paketa i dr.). Dodatno, korišćenjem ovog protokola, kontroler može dobiti informacije koje se tiču stanja interfejsa, brzine portova, postojanja grešaka i statistike saobraćaja. Ove informacije, kontroler koristi za donošenje odluka o rutiranju saobraćaja u realnom vremenu, kao i za optimizaciju mrežnih resursa i otklanjanje zagušenja. Za upravljanje konfiguracijom i statusom mrežnih uređaja, *OpenDaylight* kontroler koristi *NETCONF* protokol, koji mu omogućava da dinamički menja konfiguraciju *switch*-eva (npr. VLAN ID ili IP adrese, tako što preuzima YANG modele konfiguracije i automatski ih menja i prilagođava potrebama mreže. Suštinski posmatrano, *OpenDaylight* kontroler poseduje logiku, kojom upravlja svičevima, a koja se sastoji iz tri komponente:

- centralizovana kontrola,
- topološko mapiranje – *OpenDaylight* koristi LLDP (eng. *Link Layer Discovery Protocol*) i slične protokole, da bi automatski „otkrio“ mrežnu topologiju i saznao kako su svičevi povezani, što je veoma važno za optimizaciju putanja,
- segmentacija mreže – kreiranjem VLAN-ova ili implementacijom *network slicing* koncepta vrši se izolacija mrežnog saobraćaja i stvaraju uslovi, da se kreiraju odvojeni saobraćajni tokovi za različite klijente.

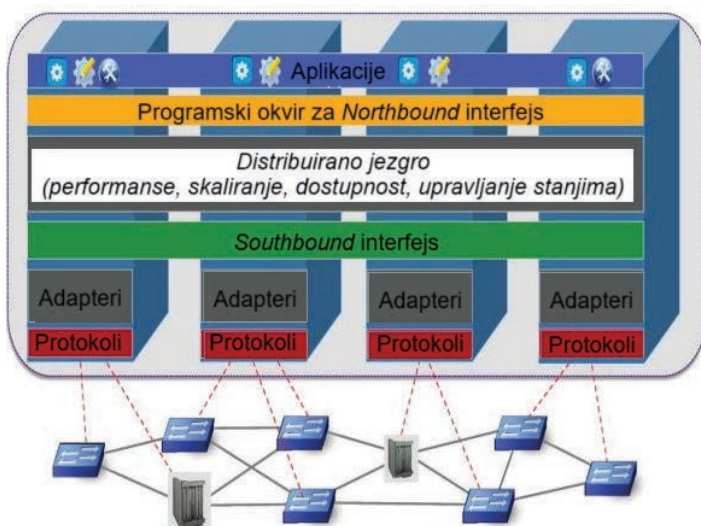


Slika 50. Scenario primene *OpenDaylight* kontrolera [82]

Na slici 50., prikazan je jedan scenario primene *OpenDaylight* kontrolera u SDN mreži. Nakon što je korišćenjem LLDP protokola, otkrio topologiju mreže (detektovao *switch*-eve i njihovu trenutnu konfiguraciju), kontroler se sa *switch*-evima A i C povezuje putem *OpenFlow* kanala. U konkretnom slučaju, *OpenDaylight* kontroler šalje instrukcije *switch*-u A, kojima mu nalaže da saobraćaj sa IP adrese 192.168.10.15 prosledi na port Fa 0/10, a da frejmove sa MAC adresom AA-1B-23-24-EC-12 uputi na port Fa 0/6. Pomenute instrukcije u formi zapisa *switch* A mora da ugradi u svoju tabelu toka i da se u skladu sa njima ponaša. S druge strane, kontroler šalje instrukciju *switch*-u B sa zahtevom za kreiranje VLAN-a 100 na portu Fa 0/11. *Switch* B će postupiti po nalogu i o izvršenom obavestiti kontroler slanjem potvrde.

Za razliku od centralizovanih SDN kontrolera, SDN kontroleri sa distribuiranom arhitekturom međusobno dele funkcije. Zadatak komponenti koje čine distribuiranu arhitekturu je da zajednički upravljaju mrežnim resursima. Njihov osnovni cilj je da se obezbedi potreban nivo skalabilnosti, otpornosti i smanji latencija u velikim ili geografski raspodeljenim mrežama, sa velikim brojem uređaja. Svaki od kontrolera, koji čine distribuiranu arhitekturu, donosi odluke na osnovu lokalnog uvida u mrežu, čime se smanjuje kašnjenje. Prekid na jednom kontroleru, neće uticati na funkcionisanje cele mreže, jer će ostali kontroleri preuzeti kontrolu nad njegovim domenom. To ukazuje na složenost upravljanja i potrebu, da između kontrolera postoji sinhronizacija, kako bi se zadržao konzistentan prikaz mreže. U realnim okruženjima se danas mogu sresti *Fleet*, *Onix*, *SmaRtLight*, *HyperFlow* i *Pane* kontroler, ali se zbog velike skalabilnosti, otpornosti na greške i podrške, koju poseduje za mreže velikih razmera, poput telekomunikacionih mreža i data centara, najčešće koristi ONOS kontroler.

ONOS (eng. *Open Networking Operating System*) je kontroler otvorenog koda, koji je razvijen od strane ONF-a (eng. *Open Networking Foundation*). Osnovna karakteristika mu je distribuirana arhitektura, koja se ogleda u mogućnosti, da se kreira klaster sa više instanci kontrolera. Ovaj klaster omogućava deljenje stanja mreže između pomenutih instanci, čime se povećava skalabilnost i otpornost na greške. Visoka dostupnost u klasteru se obezbeđuje tako, što u slučaju otkaza jedne instance kontrolera, ostale instance kontrolera automatski preuzimaju njenu funkciju. Da bi to bilo moguće, potrebno je da postoji konzistentnost podataka na instancama, što se obezbeđuje korišćenjem protokola za replikaciju poput Raft-a. Važno je pomenuti, da ONOS kontroler poseduje podršku za različite protokole *Southbound* interfejsa (*OpenFlow*, *NETCONF*, *P4Runtime*, *BGP* i dr.) i *Northbound* interfejs (REST API- i gRPC). Na slici 51. prikazana je arhitektura ovog kontrolera [83].



Slika 51. Arhitektura Onos kontrolera [83]

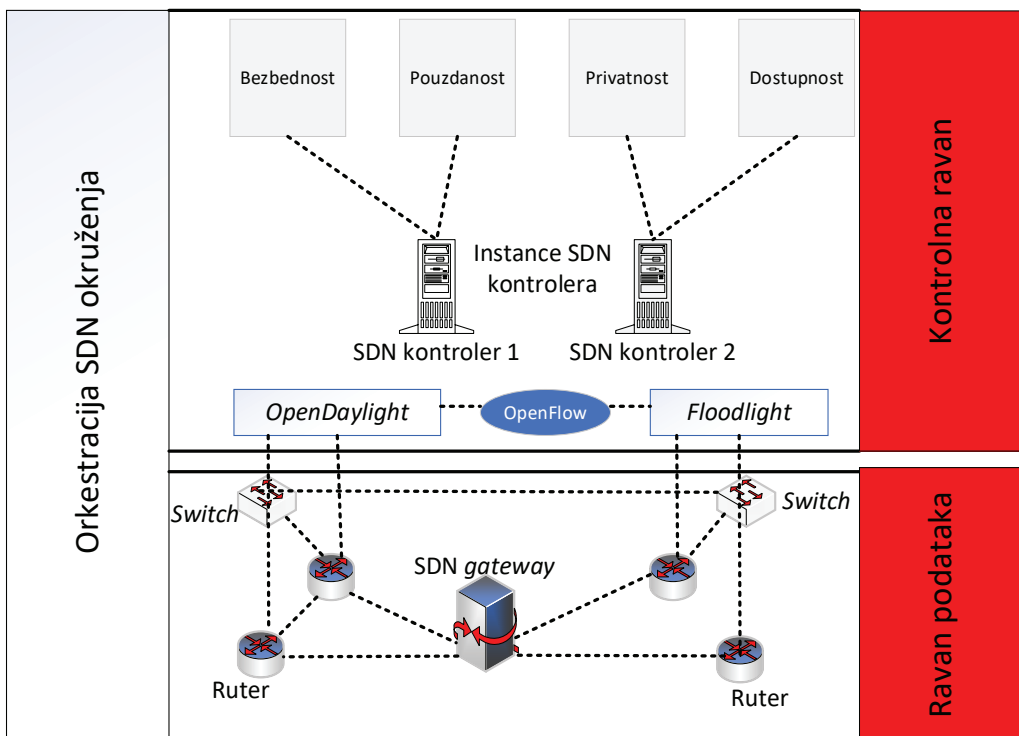
Kao što se može videti, ONOS je modularan sistem sa jasno definisanim slojevima. Osnovna komponenta arhitekture je jezgro kontrolera, koje je zaduženo za obezbeđivanje osnovnih funkcionalnosti, kao što su otkrivanje topologije, upravljanje tokovima i skladištenje podataka. Ova komponenta sadrži distribuiranu bazu podataka, koja je od ključnog značaja za deljenje stanja između različitih instanci kontrolera. Komunikacija ovih instanci sa mrežnim uređajima na sloju infrastrukture, odvija se preko Southbound interfejsa i protokola, kao što su *OpenFlow*, *NETCONF*, *SNMP* i *P4Runtime*. Za razliku od ovog interfejsa, komunikacija instanci kontrolera sa aplikacijama se odvija posredstvom REST API-a ili drugih mehanizama. Zadatak ovih mehanizama je da omoguće aplikacijama upravljanje funkcijama, kao što su QoS, rutiranje saobraćaja i bezbednost. Potrebno je napomenuti još jednu važnu činjenicu, da se na ONOS kontrolerskoj platformi mogu razvijati aplikacije prilagođene za specifične potrebe, poput upravljanja saobraćajem ili mrežne analitike.

6.3.2. SDN orkestrator

SDN Orkestrator je komponenta u arhitekturi SDN mreža, koja je zadužena za automatsko upravljanje, koordinaciju i konfiguraciju različitih mrežnih uređaja, usluga i resursa. Štaviše, može se reći, da ova komponenta treba da omogući dinamičko i koordinisano upravljanje mrežom kao celinom. Njena osnovna funkcija vezana je za procese, koji se tiču koordinacije rada ostalih komponenata SDN mreže i njihovu automatizaciju. Naime, orkestrator ima direktan pristup bazi podataka u kojoj se nalaze sve informacije o stanju mreže (topologija, resursi i trenutno opterećenje). On sprovodi analitiku nad tim podacima i donosi odluke u realnom vremenu. Ovim odlukama se omogućava automatizovano upravljanje mrežnim resursima (virtuelni uređaji, tokovi

podataka i virtualne mreže) i koordinacija aktivnosti različitih kontrolera, kako bi se osigurao optimalan rad mreže. Značaj orkestratora za odlučivanje u realnom vremenu je ogroman, jer on omogućava brzo postavljanje novih mrežnih politika i dinamičku dodelu resursa (npr. kreiranje VLAN-ova, VPN-ova, QoS i polisa za bezbednost). Štaviše, orkestrator je u mogućnosti, da automatizuje proces implementacije mrežnih funkcionalnosti u skladu sa potrebama korisnika, tj. njegovih aplikacija.

Sama realizacija orkestratora je široko programabilna, jer orkestrator treba da koordiniše komunikaciju različitih SDN kontrolera, uređaja i aplikacija u jedinstven SDN sistem. Značaj ove koordinacije u komunikacijama je ogroman, jer se kroz njih obezbeđuje kontinuirano praćenje stanja mreže i statistika saobraćaja, kao i blagovremeno ažuriranje podataka u bazi. Kroz pristup bazi podataka, orkestrator dobija kvalitetan uvid u stanje mreže i može da analizira njene performanse. Primenom različitih metoda analitike, koje se najčešće zasnivaju na veštačko inteligenciji, orkestrator donosi odluke o optimizaciji saobraćaja, balansiranju opterećenja ili detekciji grešaka u složenim okruženjima, kao što su npr. IoT mreže [84] (Slika 52). Drugim rečima, orkestrator omogućava centralizovano upravljanje mrežnim politikama i pravilima na višem nivou, kao što su politika pristupa, enkripcija ili QoS politika.



Slika 51. Orkestracija SDN okruženja [84]

Upotreba SDN orkestratora u praksi postaje sve veća, pogotovo u *data* centrima i *cloud* infrastrukturi. Razlog leži u činjenici, da orkestracija omogućava brzu i efikasnu implementaciju virtualnih okruženja, različitih politika bezbednosti i upravljanje saobraćajem u skalabilnim i fleksibilnim okruženjima, kao što su *cloud* okruženja. Njihova primena raste u naprednim 5G/6G mobilnim mrežama. Zadatak orkestratora je, da vrše koordinaciju između različitih baza, čime omogućavaju efikasno upravljanje saobraćajem, mrežnim funkcijama i kvalitetom usluge. U IoT okruženjima, njegova uloga je da upravlja velikim brojem IoT uređaja, omogućavajući automatsku segmentaciju mreže i optimizaciju saobraćaja između uređaja i aplikacija. Međutim, u velikom broju slučajeva, kada je reč o primeni orkestratora, dolazi do mešanja i pogrešnog tumačenja njegove i uloge SDN kontrolera. Zato je dobro, da se razlike naglase i da se ukaže, da je funkcija orkestratora automatizacija i koordinacija mrežnih usluga (kroz definisanje mrežnih politika i usluga), dok je funkcija kontrolera upravljanje mrežnim uređajima (topologijama) i tokovima.

6.3.3. Interfejsi u SDN arhitekturi

Uloga *Northbound* interfejsa, u SDN okruženjima, je da omogući komunikaciju kontrolne ravni sa korisničkim aplikacijama, mrežnim servisima i orkestratorom. Njegova osnovna funkcija je da pruži apstrakciju mreže za aplikacije, omogućavajući im da upravljaju mrežnim resursima bez poznavanja detalja o samim uređajima. Drugim rečima, ovaj interfejs omogućava aplikacijama apstraktan način interakcije sa mrežom, korišćenjem jednostavnih komandi i politika, kao što je npr. prioritizacija saobraćaja. Posebno je važna interakcija, koju ovaj interfejs obezbeđuje između orkestratora i kontrolera, kako bi se izvršila automatizacija zadataka poput upravljanja mrežnim protokom, konfiguracije i skaliranja resursa. Definišu se politike, kao što su QoS, bezbednost i rutiranje, koje SDN kontroler prevodi u konkretne instrukcije mrežnim uređajima. Za ostvarivanje pomenutih vidova interakcije, *Northbound* interfejs često koristi REST API, *gRPC* i druge standardizovane forme komunikacija.

Drugi važan interfejs u SDN mrežama, koji omogućava komunikaciju između SDN kontrolera i mrežne infrastrukture je *Southbound* interfejs. Njegov zadatak je da prenese odluke kontrolera na mrežne uređaje i omogući kontroleru prikupljanje podataka sa tih uređaja. U praksi to znači, da kontroler preko ovog interfejsa, šalje instrukcije mrežnim uređajima kojima se definiše njihovo ponašanje (usmeravanje ili filtriranje saobraćaja, upravljanje propusnim opsegom ili QoS politikama, implementiraju ACL pravila i dr.). Važno je naglasiti, da preko ovog interfejsa idu poruke i u suprotnom smeru ka kontroleru. Naime, kroz *southbound* interfejs idu podaci o stanju mreže, što uključuje informacije o topologiji, performansama uređaja i statistici saobraćaja. To su konkretno informacije, kojima se kontroler informiše o korišćenju linkova i portova, pojavljivanju novih uređaja i veza u mreži ili nastanku grešaka na pojedinim virtuelnim ili fizičkim uređajima (različiti alarmi). Obrada ovih informacija ima veliki značaj za funkcionisanje celokupne SDN mreže. Iz tih informacija, dobijaju se zaključci (informacije), koji se koriste da se mreža dinamički rekonfiguriše (bez potrebe za ručnom intervencijom). Primer je automatsko dodavanje novih pravila rutiranja u slučaju promena topologije ili reakcija na promene u saobraćaju u formi preusmeravanja tokova kroz manje opterećene veze.

Da bi pojednostavio komunikaciju kontrolera sa uređajima na sloju infrastrukture, *Southbound* interfejs apstrahuje specifične detalje mrežnih uređaja. Na taj način, on omogućava kontroleru da radi (komunicira) sa različitim uređajima nezavisno od toga ko je njihov proizvođač. U tu svrhu se koristi *OpenFlow*, kao standardizovan protokol, koji se može implementirati na uređajima različitih proizvođača. Pored *OpenFlow* protokola, *Southbound* interfejs koristi i druge protokole, kao što su: *OVSDB* (eng. *Open vSwitch Database*), *NETCONF*, *OF-CONFIG* (eng. *OpenFlow Configuration Protocol*), *SNMP* i drugi protokoli.

Komunikacija između više kontrolera unutar istog administrativnog domena je veoma važan zadatak unutar SDN arhitekture. On zahteva postojanje interfejsa, preko koga bi se vršila sinhronizacija podataka i koordinisano donosile odluke o mrežnoj topologiji, saobraćaju i politikama. Naime, kontroleri moraju deliti informacije o mrežnoj topologiji, što uključuje informacije o povezivanju između uređaja i njihovim performansama. Oni moraju biti u mogućnosti, da koordinirano usmeravaju pakete, naročito u složenim mrežnim scenarijima gde saobraćaj mora da bude balansiran ili optimizovan. Osim saobraćajnih politika, važno je da kontroleri razmenjuju i podatke o pravima pristupa i drugim segmentima politike bezbednosti, kako bi se obezbedila doslednost u implementaciji tih politika. Ova vrsta komunikacije je ključna za implementaciju kontrolera sa visokom dostupnošću ili distribuiranih SDN arhitektura, gde više kontrolera radi zajedno, kako bi obezbedili efikasno upravljanje mrežom, balansiranje opterećenja i optimizaciju mrežnih resursa. U tu svrhu, SDN mreže poseduju *Eastbound* interfejs, koji se uglavnom i koristi za horizontalnu komunikaciju između SDN kontrolera unutar istog administrativnog domena. Ovaj interfejs donosi nekoliko ključnih mogućnosti u distribuisanim SDN sistemima, kao što su veća skalabilnost, visoka dostupnost i otpornost u slučaju kvara jednog od kontrolera (omogućuje da drugi kontroleri preuzmu odgovornost za upravljanje mrežom), balansiranje opterećenja, kao i sinhronizaciju i doslednost (kontroleri mogu deliti informacije o stanju mreže, što omogućava da svi kontroleri rade sa istim podacima i donose usklađene odluke). Njegov sastavni deo su sledeći protokoli:

- BGP-LS (eng. *Border Gateway Protocol - Link State*) – koristi se za distribuciju informacija o mrežnoj topologiji između kontrolera,
- XMPP (eng. *Extensible Messaging and Presence Protocol*) – omogućava razmenu poruka između SDN kontrolera, čime se omogućava sinhronizacija stanja mreže i koordinacija odluka i
- RESTful API – omogućava jednostavnu razmenu informacija između kontrolera i koordinaciju njihovih aktivnosti.

U SDN mrežama su česti slučajevi, da treba obezbediti interdomenske servise i to kada se usluga proteže preko više mrežnih domena (krajnje tačke servisa se nalaze u različitim domenima). Tada je neophodno, da u SDN mreži postoji koordinacija resursa i politika između kontrolera koji upravljaju tim domenima. Veoma je čest slučaj, da pružao usluge ima više isturenih segmenata, koji imaju samostalno upravljanje. U ovu svrhu, SDN arhitektura poseduje još jedan, tzv. *Westbound* interfejs. Ovaj interfejs igra ključnu ulogu u omogućavanju saradnje i koordinacije između mreža ili mrežnih segmenata, kojima upravljaju različiti entiteti, odnosno različiti provajderi usluga ili organizacije.

Westbound interfejs se implementira, da bi se omogućila razmena informacija između nadležnih SDN kontrolera, jer resursi mreža moraju biti koordinisani, posebno kad su u pitanju linkovi za povezivanje, deljenje kapaciteta ili kada treba optimizovati saobraćaj između domena. Što je još važnije, ovaj interfejs treba da obezbedi interoperabilnost između različitih SDN kontrolera (koji mogu koristiti različite protokole ili imati različite arhitekture). Međutim, primenu *Westbound* interfejsa prate i određeni izazovi:

- interoperabilnost – različiti administrativni domeni često koriste različite SDN platforme, što otežava standardizaciju
- privatnost podataka – provajderi su oprezni u deljenju podataka o unutrašnjoj mrežnoj infrastrukturi, što može da komplikuje međusobno povezivanje
- bezbednost – potrebna je implementacija jakih bezbednosnih mehanizama, kako bi se sprečila potencijalna zloupotreba ili neovlašćen pristup.

6.4. FUNKCIONALNOSTI KONTROLNE RAVNI

Kontrolna ravan ima centralno mesto u SDN mrežama. Ona upravlja radom mreže, tako što njena logika sa jednog mesta u mreži vrši centralizovanu kontrolu i upravljanje, na osnovu globalnog uvida u celokupnu mrežnu infrastrukturu. Time je olakšano donošenje odluka, koje je usklađeno sa trenutnim stanjem i potrebama korisnika. Ovde treba uključiti i zadatke koordinacije saobraćaja i optimizacije resursa, što mrežu čini fleksibilnom, skalabilnom i lakom za prilagođavanje promenama. Jednostavno, dinamičko upravljanje saobraćajem je od velikog značaja, jer je neophodno da se brzo reaguje na skoro svaku promenu u saobraćaju i zahtevima korisnika, a da se istovremeno očuva bezbednost mreže i efikasnost u njenom radu.

Upravo centralizovano upravljanje mrežom, predstavlja ključnu karakteristika kontrolne ravni, koja se zasniva na funkcionalnosti, da se iz jedne tačke donosi odluke za celu mrežu. Ova centralizacija pruža globalni pregled mreže, što olakšava donošenje odluka koje su zasnovane na „informacijama u realnom vremenu“, optimizaciju protoka podataka i bržu reakciju na probleme ili zahteve korisnika. Jasno je, da se time pojednostavljuje implementacija različitih politika u mreži, kojima se bezbednost podiže na kvalitativno viši nivo i omogućava fleksibilno prilagođavanje mreže zahtevima u realnom vremenu. Ovo je posebno značajno danas, kada se radi u dinamičnim okruženjima sa promenljivim opterećenjima i zahtevima.

Za efikasnost centralizovanog upravljanje mrežom, jako je bitna još jedna funkcionalnost kontrolne ravni, a to je topološko otkrivanje u SDN mrežama. Ova funkcionalnost se zasniva na sposobnosti kontrolera da prikuplja informacije o mrežnim uređajima, njihovim vezama i stanju mreže i da održava ažurnu sliku mrežne topologije. To ukazuje na njen krucijalni značaj, za donošenje kvalitetnih odluka u vezi sa usmeravanjem saobraćaja, optimizacijom resursa, implementacijom politika i detekcijom problema u mreži. Naime, podaci o topologiji predstavljaju preduslov, da kontroler definiše i dinamički ažurira pravila za usmeravanje i upravljanje tokovima podataka. Samo tako je moguće, da se mreža brzo adaptira na promene, kao što su dodavanje novih uređaja ili prekidi saobraćaja, čime se poboljšava fleksibilnost i otpornost mreže.

Evidentno je, da usmeravanje paketa i upravljanje tokovima, predstavlja ključnu funkcionalnost kontrolne ravni u SDN mrežama. Kao što je prethodno rečeno, kontroler na osnovu informacija o mrežnoj topologiji i stanju uređaja, donosi odluke o optimalnom putu za svaki tok podataka, a zatim svoje odluke u formi odgovarajućih pravila distribuira mrežnim uređajima. Ono što je još važnije, kontroler poseduje mogućnost, da pravila konfiguriše i menja u realnom vremenu, na osnovu trenutnih zahteva iz mreže, koristeći protokole poput OpenFlow protokola. Ovim pravilima se definiše kako i gde podaci treba da teku, često zasnovano na različitim kriterijumima kao što su IP adresa, protokol, portovi, itd.

Važno je naglasiti, da upravljanje tokovima obezbeđuje granularnu kontrolu saobraćaja. Ono se često vrši kroz implementaciju QoS (npr. prioritetizacija određenih tipova saobraćaja) i drugih politika. Primena politika u mreži (eng. *policy enforcement*), administratorima mreže daje mogućnost efikasnog upravljanja resursima, garantuju potreban nivo bezbednosti u mreži i samo funkcionisanje mreže u skladu sa projektovanim uslovima. Kontroler u tome ima ključnu ulogu, jer on treba da dinamički prilagodi ove politike, kako bi se brzo i efikasno reagovalo na promene u opterećenju, napade ili promene u poslovnim potrebama. U realnim okruženjima, ove politike obuhvataju različite aspekte mreže, kao što su bezbednost, kvalitet servisa i *multitenancy* pristup (segmentacija saobraćaja između različitih korisnika ili aplikacija, kako bi se omogućila izolacija i bezbednost unutar mreže).

Današnje mreže karakteriše eksponencijalni rast saobraćaja, što zahteva da se promene u mrežnoj infrastrukturi vrše na brz i efikasan način, kako bi se sačuvao visok nivo performansi i obezbedila optimalna upotreba mrežnih resursa. To znači da je u kontrolnoj ravni SDN mreža, neophodno, da postoje rešenja, kojima se obezbeđuje visok nivo skalabilnosti i kontinuirano vrši balansiranje opterećenja. Upravo je to zadatak SDN kontrolera, da kroz dinamičku konfiguraciju mreže omogući jednostavno proširenje dodavanjem novih uređaja bez značajnih promena u fizičkoj infrastrukturi. Podatke koje dobija iz mreže (npr. opterećenje portova ili brzina veza), kontroler koristi da dinamički preusmeri saobraćaj ka manje opterećenim delovima mreže. Time se poboljšava ukupna efikasnost mreže i smanjuje latencija, a mrežu brže odgovara na promenljive zahteve.

Promptnost u reakcijama kontrolera na sve veći broj zahteva u modernim mrežama, praćenje stanja i analitika u mreži, ukazuju na značaj funkcija u kontrolnoj ravni SDN mreža. Jasno je, da kontroler mora da prati performanse mreže u realnom vremenu i prikuplja podatke o saobraćaju, opterećenju pojedinih resursa i uređaja, kako bi mogao da detektuje i rešava probleme. Dakle, kontrolna ravan i njeni entiteti imaju zadatak, da analiziraju ove informacije kako bi se optimizovalo usmeravanje saobraćaja, balansiralo opterećenje i poboljšale performanse mreže. Poseban segment u okviru ove funkcionalnosti kontrolne ravni, predstavlja zahtev, da se omogućući pravovremena detekcija bezbednosnih pretnji i obezbedi izveštavanje administratora mreže (time se doprinosi stabilnosti i efikasnosti mreže) [85].

6.5. BEZBEDNOST NA KONTROLNOJ RAVNI

Bezbednost na kontrolnoj ravni je ključan faktor za očuvanje integriteta i poverljivosti podataka, ali i njihove, kao i dostupnosti uređaja u SDN mrežama. Ova ravan je odgovorna za donošenje odluka o usmeravanju saobraćaja, implementaciji različitih mrežnih politika i interakciji sa uređajima na sloju infrastrukture. Ako ona nije adekvatno zaštićena, onda može postati meta napada, što može dovesti do ozbiljnih problema u bezbednosti cele SDN mreže.

Imajući u vidu, da kontroler predstavlja centralnu tačku upravljanja SDN mrežom, jasno je da on predstavlja najčešću metu napada. Napadači žele da dobiju pristup kontroleru, kako bi mogli da manipulišu različitim funkcionalnostima u mreže, uključujući usmeravanje saobraćaja, primenu politika ili čak onemogućiti rad mreži. Sami napadi mogu biti različite prirode i uključivati DDoS napade (eng. *Distributed Denial of Service*), napade na mehanizme za autentifikaciju ili mogu biti usmereni na preuzimanje kontrole nad kontrolerom. Druga važna meta napada su protokoli, kao što je npr. *OpenFlow*. Oni omogućavaju komunikaciju između kontrolera i mrežnih uređaja, pa je motiv napadač usmeren na mogućnost da manipuliše porukama između kontrolera i uređaja. Cilj ovakvih napada je, da se promene tokovi podataka, izazovu preopterećenja na uređajima i linkovima i neovlašćeno pristupi mreži. Sam pristup mreži i nedostatak efikasnih mehanizama za autentifikaciju i autorizaciju, može predstavljati ozbiljan problem za bezbednost mreže. Ako komunikacija između kontrolera i uređaja nije adekvatno zaštićena, napadači mogu preuzeti kontrolu nad uređajima. To praktično znači, da bi napadači mogli manipulirati saobraćajem ili izvoditi napade tipa *man-in-the-middle* (napadač pokušava da presreće i potencijalno manipuliše komunikacijom između kontrolera i mrežnih uređaja kako bi mogao, da pristupi, menja ili blokira poruke koje se razmenjuju). Važno je ukazati, da napadi mogu biti usmereni i na interfejs između kontrolera. U distribuiranoj organizaciji kontrolne ravni, kontroleri međusobno komuniciraju i razmenjuju podatke. Napadi na *Eastbound* ili *Westbound* interfejsa mogu ugroziti koordinaciju među kontrolerima i destabilizovati mrežu.

Mogućnost potencijalnih napada iziskuje potrebu za povećanje bezbednosti na kontrolnoj ravni. U tom smislu, bezbedna komunikacija između kontrolera i mrežnih uređaja predstavlja imperativ. Ona se najčešće štiti enkripcijom, kako bi se sprečila neovlašćena manipulacija saobraćajem i zato se za šifrovanje podataka koriste protokoli, kao što su TLS (eng. *Transport Layer Security*) ili *IPsec*. Kao dodatna mera bezbednosti, u SDN mrežama se implementiraju mehanizmi za autentifikaciju i autorizaciju. Snažna autentifikacija za uređaje koji komuniciraju sa kontrolerom, kao i striktna politike autorizacije, mogu sprečiti neovlašćeni pristup. Upotreba ključeva, sertifikata i drugih mehanizama treba da omoguće, da samo ovlašćeni uređaji komuniciraju sa kontrolerom. Sigurno je, da implementacija pravila za detekciju sumnjivog saobraćaja, napada i anomalija može pomoći u prepoznavanju i ublažavanju potencijalnih pretnji po bezbednost mreže. Iz toga razloga, na kontrolerima se implementiraju različiti alati, koji analiziraju podatke o mrežnim aktivnostima i kreiraju politike, čija primena ima za cilj zaštitu od napada. Ovde je važno napomenuti, da se u SDN mrežama sa distribuiranom kontrolnom ravni koristi redundansa kontrolera, da bi se smanjio rizik od napada na kontrolnu ravan. Ako jedan kontroler bude napadnut, drugi će preuzeti njegovu funkciju,

čime se mreža održava operativnom. Jasno je, da kontinuirano praćenje saobraćaja i aktivnosti u mreži, uz analizu i identifikaciju potencijalnih pretnji, igra esencijalnu ulogu u brznoj detekciji napada i adekvatnoj reakciji u realnom vremenu.

ZAKLJUČAK

Kontrolna ravan u SDN mrežama predstavlja ključni element SDN mreže, koji omogućava centralizovano ili distribuirano upravljanje mrežnim resursima i omogućava visok nivo fleksibilnosti i programabilnosti mreže. Zadužena je za donošenje odluka o usmeravanju saobraćaja, implementaciju mrežnih politika, otkrivanje topologije mreže, kao i za analizu i praćenje performansi, čime stvara uslove za lakše prilagođavanje mreže potrebama korisnika i brže reagovanje na promene u saobraćaju ili infrastrukturi. Međutim, funkcionisanje kontrolne ravni u SDN mrežama prate određeni izazovi, kao što su skalabilnost, otpornost na greške i bezbednost. Ovi izazovi su često vezani za centralizovanu arhitekturu kontrolne ravni, koja može predstavljati usko grlo, dok distribuirana kontrola nudi veću otpornost i bolju skalabilnost, ali sa povećanom složenošću u upravljanju i koordinaciji među kontrolerima.

Bezbednost kontrolne ravni je takođe od presudne važnosti, jer napadi na kontroler ili preuzimanje komunikacije između kontrolera i mrežnih uređaja može ozbiljno ugroziti integritet mreže. Iako SDN omogućava značajnu agilnost i optimizaciju mreže, kako bi se u potpunosti iskoristili svi njegovi potencijali, potrebno je obezbediti odgovarajuće bezbednosne mere i dizajnirati kontrolnu ravan koja može da odgovori na izazove skalabilnosti i otpornosti. Kontrolna ravan, njene funkcionalnosti i komponente, kao što su kontroler i orkestrator, od presudnog su značaja za dalji razvoj mrežnih tehnologija i implementaciju naprednih funkcionalnosti kao što su automatizacija i optimizacija u realnom vremenu.



POGLAVLJE 7



PROTOKOLI ZA KOMUNIKACIJU NA KONTROLNOJ RAVNI

7. PROTOKOLI ZA KOMUNIKACIJU NA KONTROLNOJ RAVNI

7.1. UVOD

Ključnu ulogu u omogućavanju komunikacije između kontrolne i ravni podataka, kao i u orkestraciji mrežnih funkcija, imaju protokoli kontrolne ravni SDN mreže. Njihov je zadatak da omoguće centralizovano upravljanje i optimizaciju procesa u SDN mrežama. Time se postiže veći stepen fleksibilnosti u odnosu na tradicionalne mreže i vrši automatizacija mrežnih operacija. Konkretno, ovi protokoli omogućavaju kontroleru da konfiguriše mrežne uređaje, dinamički usmerava saobraćaj i centralizovano upravlja mrežnim resursima. Njima se prenose informacije o mrežnom saobraćaju, stanju uređaja i topologiji mreže, koje omogućavaju da kontroler stekne globalan uvid u stanje mreže. Cilj je da se stvori preduslov za centralizovano donošenje odluka u pogledu rutiranja i prosleđivanja saobraćaja, kao i primenu polisa kojima se garantuje kvalitet servisa i bezbednost saobraćaja u realnom vremenu (prema trenutnim zahtevima iz mreže). Ovde je potrebno naglasiti, da protokoli kontrolne ravni imaju važnu ulogu u praćenju performansi i grešaka, jer olakšavaju prikupljanje statistike o saobraćaju i performansama mreže kako bi se detektovali problemi i donosile optimalne odluke.

Ključni zadaci koje treba da ispune ovi protokoli su:

- instalacija pravila za prosleđivanje paketa na mrežnim uređajima,
- dinamičko usmeravanje paketa kroz mrežu preko optimalnih ruta,
- implementacija različitih politika kvaliteta servisa i bezbednosti saobraćaja,
- praćenje stanja mreže, očitavanje statistika o saobraćaju (broju paketa ili bajtova koji se prenose u skladu sa određenim pravilom) i otkrivanje grešaka,
- upravljanje konfiguracijom uređaja (VLAN-ovi, portovi i tuneli) i
- ažuriranje topologije mreže u realnom vremenu.

Protokoli na kontrolnoj ravni se u praksi uglavnom kategorišu u skladu sa njihovim funkcionalnostima, a ređe prema komunikacionom pravcu ili primeni. Cilj kategorizacije je da se omogući lakši izbor odgovarajućih protokola za različite potrebe u SDN okruženju. Sa aspekta funkcionalnosti, ovi protokoli se dele na:

- protokole za kontrolu ravni podataka – omogućavaju komunikaciju između SDN kontrolera i mrežnih uređaja kako bi se implementirala različita pravila i razmenjivale informacije,
- protokole za upravljanje konfiguracijom – koriste se za konfiguraciju mrežnih uređaja i održavanje njihove funkcionalnosti,
- protokole za razmenu informacija o topologiji mreže – pomažu da kontroler otkrije i ažurira svaku promenu u topologiji mreže,
- protokole za prenos politika koje se definišu na kontroleru i implementiraju na mrežnim uređajima.

Ranije je navedeno, da se prema komunikacionom pravcu ovi protokoli mogu podeliti na *Northbound*, *Southbound*, *Eastbound* i *Westbound* protokole. *Northbound* protokoli omogućavaju komunikaciju između kontrolera i aplikacija koje definišu mrežne servise ili politike, dok *Southbound* protokoli obezbeđuju komunikaciju između kontrolera i mrežnih uređaja. *Eastbound* i *Westbound* protokoli se koriste za komunikaciju između različitih kontrolera, kako bi se njihove aktivnosti koordinisale i obezbedio potreban nivo skalabilnosti. Na kraju, protokoli kontrolne ravni se mogu podeliti i prema primeni i to na:

- protokole za *data centre* – fokusirani na upravljanje resursima i saobraćajem u virtuelnim okruženjima,
- protokole za WAN mreže (eng. *Wide Area Network*) i
- protokole za implementaciju bezbednosnih politika i praćenje saobraćaja.

7.2. PROTOKOLI ZA KONTROLU RAVNI PODATAKA

Osnovna uloga protokola za kontrolu ravni podataka u SDN mrežama je da omoguće dinamičko upravljanje saobraćajem i komunikaciju između centralizovane kontrolne ravni i distribuirane ravni podataka sa mrežnim uređajima poput *switch*-eva i rutera. Time se stvaraju ulovi za dinamičko upravljanje mrežnim saobraćajem, što uključuje implementaciju pravila za prosleđivanje saobraćaja (tokova), optimizaciju performansi mreže, bezbednost i analizu saobraćaja. Za razliku od tradicionalnih mreža, gde se upravlja paketima na osnovu jednog ili kombinacije atributa iz zaglavlja paketa (npr. mrežni prefiks odredišta, odredišna MAC adresa ili kombinacija IP adresa i TCP/UDP brojeva portova), kontroleri u SDN mrežama upravljaju tokovima paketa koristeći i druge informacije, kao što su protokol (TCP, UDP, ICMP itd.), prioritet ili kvalitet usluge, ToS (eng. *Type of Service*) i DSCP (eng. *Differentiated Services Code Point*) parametri, VLAN ID, TCP/UDP brojevi sekvenci i potvrda (*acknowledgment*) i sl. Dakle, na mnogo granularniji način, kontroler je sposoban da instalira nova pravila prosleđivanja paketa, ažurira ili ukloni postojeća pravila, ali i da prati promene u mreži i definiše različite politike u pogledu kvaliteta servisa (npr. prioritet za određeni saobraćaj) [86].

Protokoli za kontrolu ravni podataka omogućavaju i prikupljanje statističkih podataka o saobraćaju, kao što su informacije o broju paketa, bajtovima, vremenu kašnjenja i greškama. Ove informacije mogu se koristiti za praćenje performansi mreže, dinamičko preusmeravanje saobraćaja u skladu sa trenutnim stanjem mreže, kao i za detekciju i rešavanje problema kao što su zagušenja, greške ili promene u topologiji mreže. Time se postiže veća fleksibilnost u realizaciji zahteva iz mreže i optimizuju njene performanse, jer je kontroler sposoban da optimizuje korišćenje mrežnih resursa (npr. propusni opseg i latencija) i politike rutiranja prilagodi promenama u topologiji mreže (npr. raspodeli saobraćaj između više linkova ili preusmeri isti u skladu sa trenutnim stanjem mreže) [87]. Na kraju, važno je da se naglasi, da protokoli za kontrolu ravni podataka omogućavaju i implementaciju politika, koje se tiču bezbednosti u mreži. To uključuje enkripciju saobraćaja, korišćenje VPN tunela i primenu ACL pravila, kako bi se onemogućio neovlašćen pristup saobraćaju između uređaja. Najpoznatiji protokol za kontrolu ravni podataka je *OpenFlow* protokol, koji kontroleru omogućava da upravlja uređajima u mreži, tako što prati statistiku saobraćaja, definiše pravila za usmeravanje tokova (eng. *flow rules*) i dinamički preusmerava saobraćaj u realnom vremenu.

7.2.1. OpenFlow protokol

OpenFlow protokol je ključna komponenta u SDN arhitekturi, koja omogućava direktnu komunikaciju između kontrolnog sloja (SDN kontrolera) i mrežnih uređaja. Razvijen je na Stanford univerzitetu, sa ciljem, da se da se standardizuje komunikacija između kontrolne i ravni podataka, omogućiti centralizovano upravljanje mrežnim uređajima i jednostavnije programiranje mreže. Inače, proces standardizacije upravljanja mrežama i definisanje arhitektura koje omogućavaju modularnost i fleksibilnost u mrežnim funkcijama je započeo još sredinom 1990-ih godina sa pojavom IEEE P1520 inicijative [88]. Ova inicijativa se pojavila kao odgovor na potrebu da se:

- izvrši standardizacija programabilnih interfejsa – obezbedi interoperabilnost između različitih mrežnih uređaja i softverskih platformi,
- podrži inovativni pristup – razdvajanjem mrežnih funkcija od osnovne infrastrukture stvaraju se uslovi za brže uvođenje novih protokola i servisa i
- omogućiti centralizovano upravljanje infrastrukturom i postavi osnova za koncept programabilnih mreža.

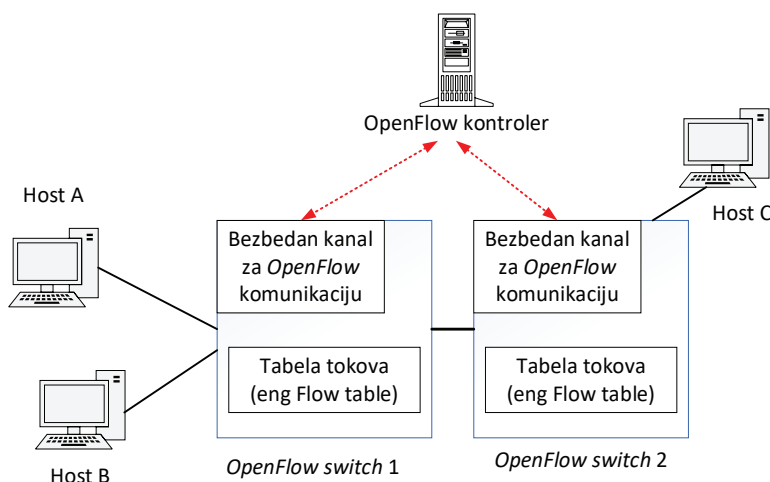
Kao rezultat inicijative za uvođenje većeg stepena programabilnosti u računarskim mrežama, početkom 2000-ih godina je prvi put predstavljena *SoftRouter* arhitektura [89]. *SoftRouter* arhitektura je koncept razvijen u okviru istraživanja o programabilnim mrežama i predstavlja pokušaj razdvajanja kontrolne i ravni podataka, što čini temelj za razvoj SDN mreža. Prvi put je predstavljena ideja centralizacije kontrolne ravni i njene implementacije u softverskom entitetu koji se naziva *SoftRouter* kontroler, kao i implemenetacije jednostavnijih mrežnih uređaja, čiji je zadatak prosleđivanje paketa na osnovu uputstava kontrolera. Centralizacijom kontrolne ravni se stvaraju uslovi za centralizovano upravljanje uređajima od strane kontrolera, koji odluke o rutiranju, preusmeravanju i upravljanju mrežnim saobraćajem donosi na osnovu globalnog uvida u mrežu. Ono što je još važnije je činjenica, da ova arhitektura predviđa implementaciju interfejsa za komunikaciju između kontrolera i mrežnih uređaja, koji više nisu odgovorni za donošenje odluka o rutiranju, što pojednostavljuje njihovu funkcionalnost. Dakle, ključna karakteristika *SoftRouter* arhitekture je široka programabilnost sa ciljem da se omogućiti dinamičko prilagođavanje mreže zahtevima aplikacija i korisnika.

IETF je 2004. godine objavio prve standarde vezane za razvoj *ForCES* (eng. *Forwarding and Control Element Separation*) arhitekture [90], koja se zasniva na razdvajanju kontrolne i ravni podataka u uređajima. Ova arhitektura je razvijena kao odgovor na potrebu za većom fleksibilnošću i programabilnošću mreža, sa ciljem eliminacije ograničenja tradicionalnih mrežnih uređaja, kod kojih su kontrolni i podatkovni slojevi čvrsto integrisani. Definisan je protokol koji omogućava komunikaciju (razmenu podataka i kontrolnih informacija) u realnom vremenu između softvera namenjenog za kontrolu (eng. *Control Element* - CE) i elementa zaduženog za prosleđivanje saobraćaja (eng. *Forwarding Element* - FE), kao što su switch ili ruter). Samo upravljanje je zamišljeno, da se izvršava na centralizovan način tako što bi kontrolni elemenat upravljao sa više elemenata za prosleđivanje.

Razvoj *OpenFlow* protokola je takođe zasnovan na potrebi da se standardizuje komunikacija između kontrolne i ravni podataka i stvore uslovi, da softverske aplikacije tj. kontroleri mogu da programiraju tabele tokova na različitim *switch*-evima. Međutim, uprkos činjenici, da obe arhitekture omogućavaju veći nivo programabilnosti mreže, u praksi postoje određene razlike, koje se upravo odnose na načine kako se ta programabilnost ostvaruje. Naime, *ForCES* arhitektura omogućava veću fleksibilnost i razdvajanje slojeva u mreži, dok je *OpenFlow* arhitektura usmerena na centralizovano upravljanje *switch*-evima u SDN okruženju. U suštini, *OpenFlow* arhitektura se zasniva na protokolu, kojim se definiše način na koji kontroler komunicira sa *switch*-evima, kako bi ih programirao, upravljao saobraćajem i definisao pravila za prosleđivanje paketa.

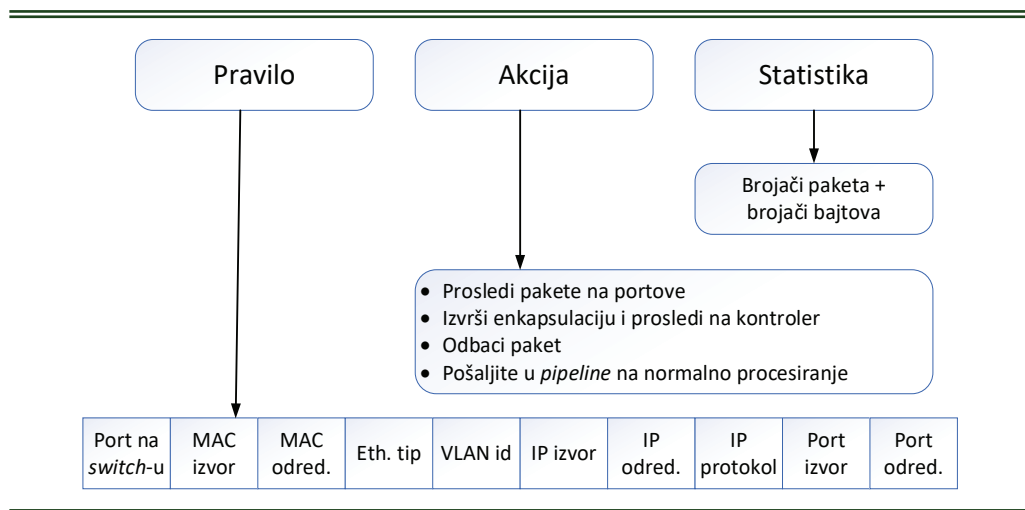
Generalno, *OpenFlow* protokol omogućava da *switch*-evi budu programirani i kontrolisani sa jednog centralnog mesta (kontrolera) u mreži (Slika 52), čime omogućava efikasno upravljanje saobraćajem, fleksibilnost u mrežnim operacijama i podršku za napredne mrežne funkcije [91]. Njegove osnovne karakteristike su:

- centralizovano upravljanje mrežnim uređajima (*switch*-evima) putem kontrolera, koji na njima dinamički menja pravila za prosleđivanje saobraćaja,
- programabilnost tabela prosleđivanja koje kontroler ažurira u realnom vremenu i definiše kako će paketi biti prosleđeni u mreži, na osnovu različitih kriterijuma (npr. IP adrese, MAC adrese, portovi, i dr.),
- slanje komandi koje mogu uključivati dodavanje, uklanjanje ili modifikovanje pravila u tabelama za prosleđivanje,
- podrška za implementaciju naprednih funkcija kao što su *QoS*, *load balancing* i filtriranje saobraćaja i
- dinamičko prilagođavanje zahtevima iz mreže u realnom vremenu, čime se postiže fleksibilnost u upravljanju, što je od posebnog značaja u velikim i složenim mrežama.



Slika 52. *OpenFlow* komunikacija u SDN okruženju [91]

Kao što se vidi na slici 52, arhitektura *OpenFlow* protokola ima dve osnovne komponente. One omogućavaju programabilnost mrežnih *switch*-eva i centralizovano upravljanje mrežnim saobraćajem. Prva komponenta je *OpenFlow* kontroler, centralizovan softverski entitet koji upravlja mrežnim uređajima, tako što donosi odluke o prosleđivanju saobraćaja i šalje *switch*-evima pravila za njihovu primenu. *OpenFlow switch*-evi predstavljaju drugu važnu komponentu ove arhitekture, koja prima instrukcije sa kontrolera i dinamički ažurira tabele prosleđivanja u realnom vremenu. Oni obavljaju operacije kao što su upisivanje, ažuriranje i brisanje pravila u tabelama tokova u skladu sa instrukcijama koje im šalje kontroler [92]. Važno je napomenuti, da *switch*-evi mogu posedovati jednu ili više tabela tokova, u kojima se nalaze zapisi koji definišu kako *switch* treba da postupi sa paketima (npr. prosleđivanje, filtriranje, preusmeravanje paketa). Svaki zapis u tabeli sadrži kriterijume za *matching* (npr. IP adresu), akcije (npr. prosleđivanje paketa ili modifikacija) i druge informacije (Slika 53).



Slika 53. *OpenFlow* tabela [92]

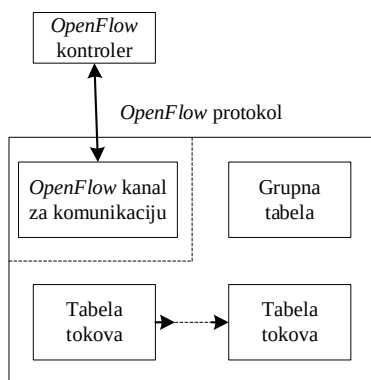
Unosi tokova (eng. *Flow entries*) su osnovni entiteti u tabelama prosleđivanja, koji definišu pravila za obradu mrežnog saobraćaja. Svaki unos predstavlja jedno pravilo, koje određuje šta će *switch* raditi sa paketima koji odgovaraju određenim kriterijumima (npr. IP adresa, MAC adresa, port, itd. Komponente svakog unosa u tabeli tokova su:

- polja za upoređivanje (eng. *match fields*) – sadrže kriterijume za identifikaciju paketa, koje *switch* koristi, da bi sa njima uporedio karakteristike dolaznog paketa i odredio koje pravilo treba primeniti
- prioritet (eng. *priority*) – definiše redosled obrade pravila kada više unosa odgovara istom paketu (pravilo sa višim prioritetom ima prednost)
- brojači (eng. *counters*) – beleže statistike za svaki unos, uključujući broj obrađenih paketa, količinu prenesenih bajtova i vreme od poslednje upotrebe pravila

- akcije (eng. *actions*) – definišu šta treba da se uradi sa paketima koji odgovaraju pravilu (npr. prosleđivanje paketa na određeni izlazni port, modifikacija paketa, odbacivanje paketa ili slanje paketa kontroleru)
- *Cookie* – identifikacija unosa radi upravljanja tokom operacija kao što su ažuriranje i brisanje pravila
- *Idle Timeout* (vreme nakon kojeg se pravilo briše ako nije bilo korišćeno) i *Hard Timeout* (maksimalno vreme nakon kojeg se pravilo automatski briše, bez obzira na njegovu upotrebu)
- ID tabele (eng. *table ID*) – identifikuje tabelu u kojoj se pravilo nalazi, jer *OpenFlow switch*-evi mogu imati više tabela za prosleđivanje.

Ovde je važno objasniti kako funkcioniše proces prosleđivanje paketa na osnovu unosa u tabeli tokova. On započinje čim prvi paket jedne konekcije stigne na *switch*. Tada se pokreće pretraga na osnovu podataka iz zaglavlja paketa i traži odgovarajući unos u tabeli tokova. Ukoliko se pronađe pravilo koje odgovara poljima iz zaglavlja paketa, izvršava se definisana akcija. U suprotnom, ako odgovarajuće pravilo nije pronađeno, paket se prosleđuje kontroleru ili odbacuje (u zavisnosti od konfiguracije).

U realnim okruženjima, *OpenFlow switch*-evi često poseduju više tabela tokova i u stanju su da koriste hijerarhijsku obradu pravila, čime se omogućava fleksibilnija i efikasnija obrada mrežnih paketa. Umesto da sva pravila nalaze u jednoj tabeli, kreira se lanac povezanih tabela, gde svaka tabela sadrži pravila koja se primenjuju na pakete. Paketi se obrađuju sekvencijalno od prve tabele, a zatim se na osnovu definisanih akcija prosleđuju u ostale tabele na dodatnu obradu (kreira se tzv. *pipeline*). Implementacija više povezanih tabela tokova ima za cilj da se smanji složenost svake pojedinačne tabele tokova. Time se na fleksibilan način implementiraju složene politike upravljanja saobraćajem i obezbeđuje modularnost (pravila mogu biti organizovana u nezavisne tabele za različite funkcionalnosti, kao što su QoS, NAT i sl.). Važno je napomenuti, da *OpenFlow* arhitektura omogućava i napredne akcije, kao što su *load balancing*, *multicast/broadcast* i *failover* mehanizam. U tom smislu, na *switch*-evima se pored više tabela tokova može implementirati i grupna tabela (eng. *group table*), kao što je prikazano na slici 54.



Slika 54. Komponente *OpenFlow switch*-eva [2]

Grupna tabela se koristi za definisanje skupova akcija koje se mogu primeniti na pakete, omogućavajući funkcionalnosti koje ne mogu biti ostvarene samo korišćenjem tabele tokova. Svaki unos u ovoj tabeli je identifikovan pomoću *Group* ID-a i sadrži polja tip grupe (eng. *group type*) i segment (eng. *buckets*) tj. lista definisanih akcija. Polje tip grupe određuje način primene akcija i definiše sledeće grupe:

- *ALL* – paket se kopira i prosleđuje na sve navedene izlaze (*multicast* ili *broadcast*)
- *SELECT* – paket se prosleđuje jednom od dostupnih izlaza prema ponderisanom algoritmu (npr. za *load balancing*)
- *INDIRECT* – jedan segment akcije se primenjuje na sve pakete
- *FAST FAILOVER* – paket se prosleđuje na prvi dostupan izlazni port iz liste (koristi se za redundansu i preusmeravanje saobraćaja u slučaju kvara).

Još jedna tabela sa često sreće na *OpenFlow switch*-evima u realnim okruženjima. To je metering tabela [93], koja omogućava implementaciju kontrole protoka i upravljanja saobraćajem kroz mehanizme, kao što su ograničenje brzine (eng. *rate limiting*) i QoS politike. Svaki unos u ovoj tabeli identifikuje se pomoću *Meter* ID-a i sadrži polja *Rate* (definiše se granica brzine protoka) i *Bands* (akcije koje treba primeniti kada protok premaši određeni prag za svaki opseg). Ova tabela se u praksi često uvodi u interakciju sa tabelom tokova. Naime, pravila u tabelama tokova mogu koristiti akciju *Meter Action*, koja se obično referencira na *Meter* ID u *Metering* tabeli. Paketi se mere prema definisanim opsezima u *Metering* tabeli, a zatim se primenjuju odgovarajuće akcije. Primenom ove tabele mogu se ostvariti sledeći benefiti:

- fleksibilnost – definišu se različite politike za različite tipove saobraćaja,
- efikasnost – ograničava se protok i smanjuje zagušenje u mreži,
- tretiranje saobraćaja na osnovu prioriteta nivoa i
- detekcija i zaštita od napada.

Ključni elementi *OpenFlow* komunikacije između kontrolera i *switch*-eva su poruke. One omogućavaju upravljanje mrežnim saobraćajem, konfiguraciju uređaja i prenos informacija o mrežnom stanju. Ove poruke se mogu podeliti na u kategorije:

- poruke koje kontroler šalje na *switch* – omogućavaju da kontroler konfiguriše i upravlja *switch*-em,
- poruke koje *switch* šalje na kontroler – obaveštenja o stanju mreže ili zahtev za određivanje akcije i
- asinhronne poruke – prenose događaje ili informacije koje nisu direktno inicirane od strane kontrolera.

U praksi se koriste različiti tipovi OpenFlow poruka [94], kao što su:

- *Hello* – poruka kojom se uspostavlja veza između kontrolera i *switch*-a i koja sadrži informacije o verziji *OpenFlow* protokola koju uređaj podržava.
- *Features Request/Reply* – poruka kojom se kontroler informiše o konfiguraciji *switch*-a (npr. broj portova, podržane tabele i dr)
- *Set Config* – poruka kojom kontroler prosleđuje konfiguraciju *switch*-u
- *Get Config Request/Reply* – poruka kojom *switch* informiše kontrolera o trenutnoj konfiguraciji
- *Flow Mod* – poruka koja se koristi za dodavanje, modifikaciju ili brisanje unosa u tabeli tokova
- *Group Mod* – poruka kojom se omogućava upravljanje unosima u grupnim tabelama
- *Port Mod* – poruka koja se koristi se za konfigurisanje ili promenu stanja portova
- *Stats Request/Reply* – poruka kojom se kontroleru isporučuju statističke informacije o *switch*-u, portovima, ili saobraćaju
- *Queue Stats* – poruka koja sadrži informacije o statusu redova za planiranje saobraćaja
- *Packet-In* – asinhrona poruka koju *switch* šalje kontroleru kada ne postoji odgovarajuće pravilo u tabeli tokova
- *Packet-Out* – asinhrona poruka koju kontroler šalje *switch*-u sa akcijom koju nalaže (npr. prosleđivanje na port).
- *Flow Removed* – asinhrona poruka koju *switch* šalje kontroleru, kojom ga obaveštava da je iz tabele tokova uklanio neka pravila (npr. zbog vremena isteka)
- *Port Status* – asinhrona poruka koju *switch* šalje kontroleru, kojom ga obaveštava o promenama stanja portova (npr. port je postao neaktivan)
- *Error* – asinhrona poruka koju *switch* šalje kontroleru, kojom ga obaveštava o greškama.

7.2.2. Primena *OpenFlow* protokola

OpenFlow protokol ima široku primenu u različitim scenarijima unutar SDN arhitekture i to posebno u kontekstu upravljanja mrežom. Njegovom primenom se stvaraju uslovi da se poboljša fleksibilnost, smanje troškovi i poveća skalabilnost u mreži. Ona je najčešća u *data* centrima i *cloud* infrastrukturi, gde se *OpenFlow* protokol koristi pre svega za dinamičko upravljanje saobraćajem i kreiranje pravila za rutiranje i prebacivanje saobraćaja između servera u *data* centru. Osim toga, on se koristi i za orkestraciju resursa i balansiranje opterećenja. Štaviše, u multitenant mrežama, ovaj protokol pomaže u izolaciji i prilagođavanju mrežne infrastrukture za višestruko korišćenje od strane različitih korisnika. Kada je u pitanju optimizacija mrežnog saobraćaja, *OpenFlow* protokol se

koristi za balansiranje opterećenja (*OpenFlow* kontroleri analiziraju mrežni saobraćaj i šalju pakete kroz optimalne puteve, čime se izbegavaju zagušenja) i ostvarivanje željenog kvaliteta servisa (implementacijom pravila u tabelama tokova, omogućava se prioritarno tretiranje kritičnog saobraćaja). Njegova primena je evidentna i sa aspekta bezbednosti u mreži. Naime, ovaj protokol omogućava brzo preusmeravanje ili blokiranje zlonamernog saobraćaja, tako što se funkcionalnost *firewall*-a može realizovati implementacijom pravila u tabelama tokova (filtriranje paketa na osnovu IP adresa, portova, ili protokola). Poseban vid bezbednosti se može ostvariti segmentacijom mreže u virtualne LAN-ove, čija se izolacija obezbeđuje dinamičkim definisanjem pravila u tabeli tokova.

Danas se *OpenFlow* protokol sve više primenjuje u mrežama ISP i telekom operatora. On se u ovim mrežama koristi za dinamičko rutiranje saobraćaja, jer su *OpenFlow* kontroleri sposobni da upravljaju rutama u realnom vremenu na osnovu saobraćajnih zahteva. Štaviše, on je pogodan za brzu implementaciju novih usluga, kao što su VPN-ovi ili optimizovane rute za kritičan saobraćaj kao što je VoIP, IPTV ili neki drugi servis. Mogućnost, da se brzo i efikasno odgovori na zahteve iz mreže je razlog, zbog koga se ovaj protokol primenjuje u IoT i modernim 5G/6G mrežama. U ovim mrežama se *OpenFlow* protokol koristi kao podrška za mrežni *slicing*, tj. koristi se za kreiranje i upravljanje segmentima mreže specifičnim za IoT aplikacije ili korisničke zahteve. On omogućava prilagodljivo upravljanje uređajima u realnom vremenu kroz kontrolu i optimizaciju mrežnih resursa.

Generalno, postoje brojne prednosti implementacije *OpenFlow* protokola, ali su osnovne:

- centralizovano upravljanje – jedan kontroler upravlja saobraćajem na svim *switch*-evima, čime se smanjuje kompleksnost,
- fleksibilnost – pravila se mogu brzo menjati i prilagođavati trenutnim potrebama mreže,
- smanjenje troškova – može se koristiti standardni hardver sa centralizovanim inteligentnim upravljanjem,
- bolja optimizacija mrežnih resursa i
- skalabilnost – lako se prilagođava rastu mreže i povećanom obimu saobraćaja.

7.3. PROTOKOLI ZA UPRAVLJANJE KONFIGURACIJOM MREŽNIH UREĐAJA

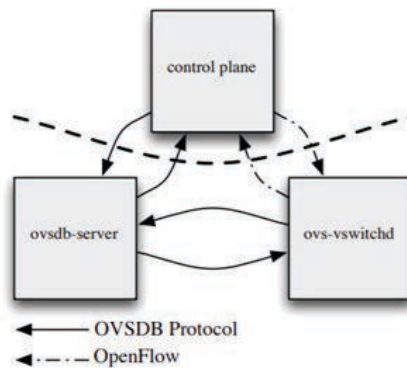
Mrežni uređaji koji se koriste u SDN mrežama su najčešće *switch*-evi, koji podržavaju rad sa *OpenFlow* protokolom. U praksi, to su najčešće softverski, virtualni *switch*-evi, kao što je *Open vSwitch* (OVS), a ređe hardverski *switch*-evi sa podrškom za ovaj protokol. Oni predstavljaju ključna komponentu u modernim mrežama zasnovanim na SDN arhitekturi i široko se koriste u data centrima, cloudu i okruženjima gde je potrebno obezbediti visok stepen fleksibilnost i automatsko upravljanje mrežnim resursima. Obično se implementiraju na standardnim x86 serverima i omogućavaju napredno prosleđivanje paketa, koje se vrši u skladu sa instrukcijama *OpenFlow* kontrolera. Podržavaju tehnologije poput VLAN, VXLAN, GRE, i dr., za enkapsulaciju i tuneliranje, kao i razne mrežne protokole i standarde. Ono što karakteriše ove uređaje su visoke performanse, koje se obezbeđuju kroz integraciju sa kernelom (npr. *Linux kernel datapath*), što omogućava brzo prosleđivanje paketa. Performanse su dodatno poboljšane kroz podršku za *DPDK* (eng. *Data Plane Development Kit*) dodatno poboljšava performanse. Naime, *DPDK* značajno poboljšava performanse *Open vSwitch*-eva jer omogućava visokopropusnu obradu mrežnih paketa u korisničkom prostoru umesto u kernelu. Time se smanjuje kašnjenje i povećava brzina prosleđivanja paketa, što je ključno za mreže sa visokim zahtevima u pogledu performansi, kao što su data centri.

U savremenoj infrastrukturi *data* centara, radna opterećenja (eng. *workload*) i virtualne mašine više nisu statične i mogu biti veoma dinamične u velikim i gustim serverskim klasterima. Zato je potrebno pratiti stanja svih virtualnih entiteta u infrastrukturi, kako bi ih učinili lako prepoznatljivim i kako bi se omogućila njihova jednostavna migracija između različitih hostova, kada je to potrebno. Važno je napomenuti, da stanje mrežnog uređaja može biti bilo koja informacija, poput tradicionalnog „mekog stanja“, kao što je unos u L2 TCAM tabelu, stanje prosleđivanja na L3 sloju, politike rutiranja, ACL liste ili konfiguracija monitoringa. U SDN mrežama se koriste protokoli koji omogućavaju razmenu informacija između kontrolne (SDN kontrolera) i ravni podataka, kako bi se centralizovano i automatizovano upravljalo konfiguracijom mrežnih uređaja. Dakle njihov zadatak je, da obezbede osnovu za konfiguraciju, monitoring i optimizaciju SDN mreža, omogućavajući skalabilnost, fleksibilnost i automatizaciju. U tu svrhu se najčešće koristi *OVSD* protokol, koji omogućava konfiguraciju i upravljanje mrežnim uređajima, kroz dinamičko dodavanje novih *bridge*-va, portova i interfejsa ili njihovo uklanjanje. Pored ovog protokola, u realnim okruženjima se u manjem obimu, mogu sresti i sledeći protokoli:

- *NETCONF* (eng. *Network Configuration Protocol*) protokol – omogućava kontroleru na transakciji zasnovan (eng. *transaction-based*) pristup i konfiguraciju mrežnih uređaja, korišćenjem XML-a i SSH bezbednog prenosa podataka.
- *RESTCONF* (eng. *Representational State Transfer Configuration Protocol*) protokol – koristi se za prikupljanje statističkih podataka o stanju uređaja, saobraćaju i performansama mreže, daljinsko upravljanje konfiguracijama uređaja i automatizaciju mrežnih funkcija

7.3.1. Open vSwitch Database Management protokol

OVSDB (eng. *Open vSwitch Database Management Protocol*) je protokol koji se koristi za upravljanje konfiguracijom i statusom OVS instanci u mreži. U realnom okruženju, ovaj protokol se koristi zajedno sa *OpenFlow* protokolom i njegov osnovni zadatak je da omogući kreiranje, ažuriranje i brisanje logičkih elemenata poput *bridge*-va, portova, interfejsa i drugih resursa OVS-ova (Slika 55). OVSDB komunikacija između klijenata (SDN kontrolera) i OVS-a radi konfiguracije i upravljanja mrežnim resursima je definisana dokumentom RFC 7047 [95].



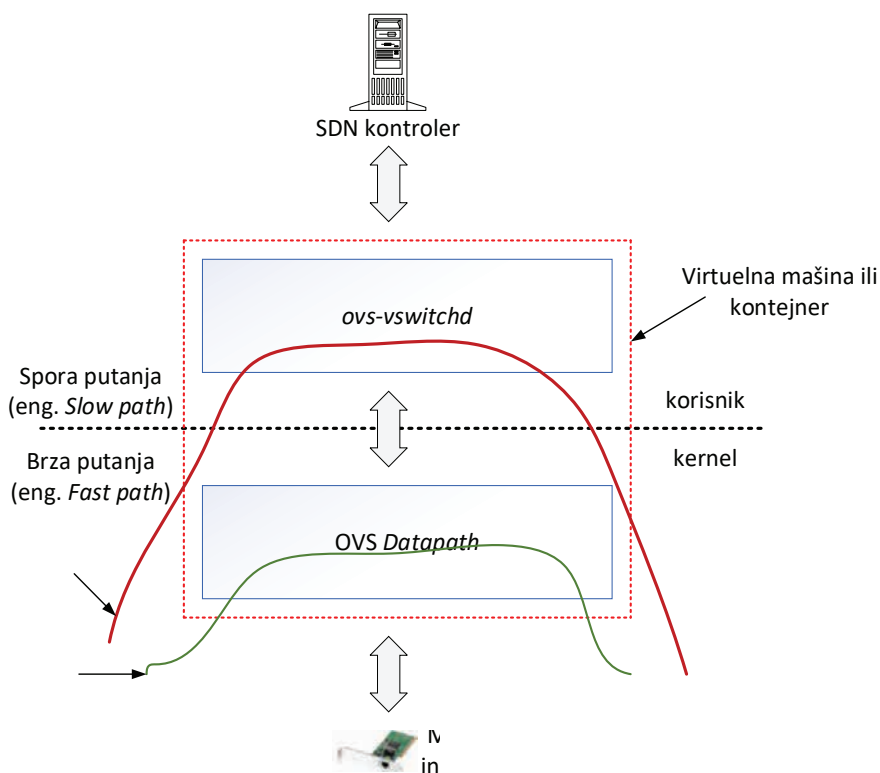
Slika 55. OVSDB dijagram toka podataka [95]

OVSDB komunikacija prikazana na slici 55., zasnovana je na JSON-RPC 2.0 protokolu, koji omogućava klijentima i serverima razmenu poruka u standardizovanom formatu. Komunikacija započinje tako što klijenti (SDN kontroleri) šalju zahteve (*JSON-RPC Request*) OVSDB serveru za izvršavanje određenih operacija nad bazom podataka. Ovi zahtevi moraju da sadrže tri važna polja: *id* (jedinostveni identifikator za praćenje zahteva), *method* (funkcionalnost koja se poziva na strani OVSDB servera - npr. *transact* metoda za pokretanje operacije/transakcije nad bazom podataka) i *params* (specifikacija operacije/transakcije sa parametrima: *op* - tip operacije, *table* - ciljna tabela u bazi podataka i *row* - podaci za novi red u tabeli). Ovim zahtevima se mogu inicirati sledeće operacije nad bazom podataka:

- *Insert* – dodavanje novih redova (npr. kreiranje novog *bridge*-a),
- *Delete* – uklanjanje postojećih redova,
- *Update* – izmena podataka u postojećim redovima,
- *Select* – čitanje podataka iz baze,
- *Listen* – „poslušaj“ promene u bazi podataka i obavesti klijenta u realnom vremenu,
- *Transact* – omogućava grupisanje više operacija u jednoj transakciji.

Dakle, cilj operacija se odnosi na potrebu da se pročita ili izmeni konfiguracija OVS-a. Štaviše, potrebno je napomenuti, da zahtevi mogu uključivati više operacija koje se izvršavaju atomski (sve operacije u okviru jedne transakcije se izvršavaju kao celina). Drugim rečima, ako neka od operacija u transakciji ne uspe, cela transakcija se poništava i baza podataka ostaje u prethodnom stanju. Ovde je potrebno naglasiti, da se zahtevi klijenta upućuju OVSDB serveru, koji je deo OVS sistema, zadužen za upravljanje bazom podataka sa informacijama o mrežnoj konfiguraciji. On proverava njihovu ispravnost, izvršava odgovarajuću operaciju nad bazom podataka i odgovara sa rezultatima operacija (*JSON-RPC Response*). Ovi rezultati sadrže dva važna polja: *result* - povratne informacije o uspešnom izvršenju operacije i *uuid* - jedinstveni identifikator za uneti red u tabeli.

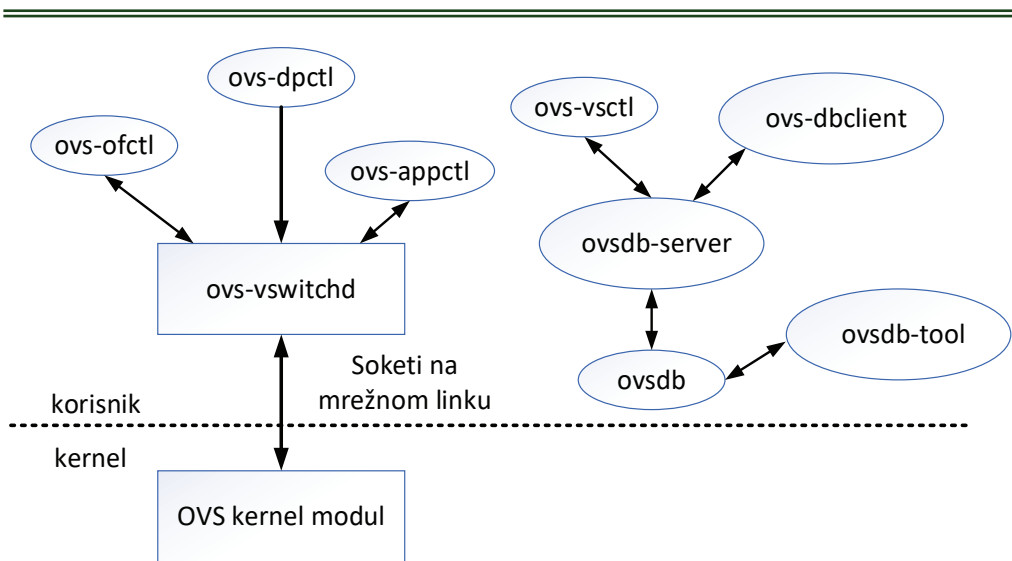
Da bi se prethodno pomenut model komunikacije pravilno shvatio, potrebno je navesti šta su komponente OVS-a i objasniti njihovu ulogu. Klasični dizajn i implementacija OVS-a se sastoji od dve glavne komponente, koje se nalaze odvojeno u korisničkom i kernel prostoru (Slika 56) [96]. Prva komponenta je *ovs-vswitchd daemon* se nalazi u korisničkom prostoru operativnog sistema i obično se naziva „spora putanja“ (eng. *Slow path*). U suštini, to je pozadinski proces koji vrši pretragu konfiguracionih podataka sa servera iz baze podataka da bi podesio putanje podataka. Druga komponenta je OVS *Datapath*, je „brza putanja“ kada tokovi paketa prolaze kroz OVS.



Slika 56. Klasičan dizajn OVS-a [96]

Sa pojavom DPDK-a 2010. godine i njegovom prvom integracijom u OVS 2014. godine, postalo je moguće iskoristiti biblioteku zaobilaženja DPDK kernela za izgradnju OVS putanja podataka. Potrebno je naglasiti da drajveri i mrežni alati takođe koriste DPDK biblioteku sa ciljem, da se aplikacijama koje rade u korisničkom prostoru, omogući da obavljaju mrežne operacije sa performansama koje su obično dostupne samo u kernelu operativnog sistema. Na slici 57. prikazane su i ostale komponente OVS-a, čiji su zadaci:

- *ovsdb-server* baze podataka kojem *ovs-vsctl* postavlja upite kako bi dobio svoju konfiguraciju,
- *ovs-dpctl* – alat za konfigurisanje modula kernela OVS-a i kontrolu pravila prosleđivanja,
- *ovs-vsctl* – program za ispitivanje i ažuriranje konfiguracije *ovs-vswitchd* procesa
- *ovs-appctl* – program koji šalje komande pokrenutim *ovs-vswitchd* procesima
- *ovs-ofctl* – program za ispitivanje i kontrolu OVS-a,
- *ovs-pki* – program za kreiranje i upravljanje infrastrukturom javnog ključa za OVS.



Slika 57. Komponente OVS-a

7.4. PROTOKOLI ZA RAZMENU INFORMACIJA O TOPOLOGIJI MREŽE

Razmena informacija o topologiji mreže je ključna za omogućavanje pravilnog upravljanja mrežom i optimizacije saobraćaja u SDN mrežama. Ovi protokoli omogućavaju da SDN mreža efikasno funkcioniše, pružajući kontroleru globalni uvid u stanje mreže, što je ključno za donošenje odluka o rutiranju, raspoređivanju resursa i implementaciji mrežnih politika. U tu svrhu se koriste različiti protokoli. Poznato je, da je *OpenFlow* najčešće korišćen protokol u SDN mrežama, koji omogućava komunikaciju između kontrolera i mrežnih uređaja. Kontroler koristi ovaj protokol za razmenu informacija o topologiji mreže, kako bi stekao globalni uvid u njeno stanje (npr. informacije koje se odnose na stanje portova, protok podataka na linkovima i povezivanju uređaja). Za mapiranje mrežne topologije, SDN kontroler koristi *LLDP* protokol, tako što *switch*-evi šalju *LLDP* pakete preko svojih portova, a kontroler koristi povratne informacije za identifikaciju susednih uređaja i veza. Za prikupljanje informacija o topologiji tradicionalnih mreža ili SDN mreža u nadležnosti drugih kontrolera, SDN kontroler koristi *BGP-LS* protokol. *BGP-LS* je proširenje standardnog *BGP*-a i koristi se za prenos informacija o stanju mrežnih veza. Ovaj protokol kontrolerima omogućava da međusobno razmenjuju podatke o mrežnim linkovima.

Posebno mesto među protokolima za razmenu informacija o topologiji mreže, u realnim SDN okruženjima zauzimaju *NetFlow* i *OF-Config* protokoli. Prvi se koristi za prikupljanje podataka o saobraćaju, odnosno informacija o protocima i statistici mrežnih uređaja. Ove informacije pomažu kontroleru u donošenju odluka o rutiranju na osnovu topologije mreže. Drugi protokol je *OF-Config* (eng. *OpenFlow Configuration Protocol*) protokol. Ovaj protokol je u načelu dodatni protokol za konfiguraciju *OpenFlow* uređaja. On služi za razmenu informacija o topologiji mreže, tako što omogućava SDN kontroleru da otkrije i konfiguriše mrežne uređaje i veze kako bi se omogućila bolja integracija topoloških podataka.

7.4.1. *NetFlow* protokol

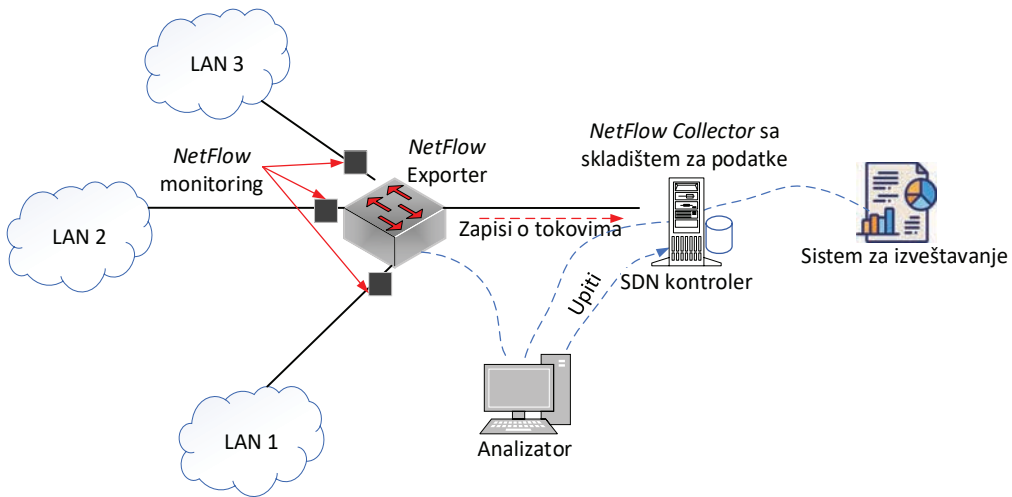
NetFlow protokol je razvijen od strane kompanije *Cisco* sa ciljem, da se kroz monitoring saobraćaja omogući prikupljanje i analiza podataka iz mreže. Njegov osnovni zadatak je, da kontroleru omogući uvid u tokove podataka (eng. *flows*) koji prolaze kroz mrežne uređaj. Informacije koje se prikupljaju upotrebom *NetFlow* protokola, koriste se za optimizaciju mrežnih performansi, unapređenje bezbednosti i efikasnije upravljanje mrežom. Osnovni koncept protokola se zasniva praćenju tokova i beleži podataka o svakom od njih. Ovde se mora naglasiti, da se pod tokom podrazumeva sekvenca paketa, koji dele zajedničke karakteristike, kao što su izvorna i odredišna IP adresa, izvorni i odredišni port, tip servisa (eng. *Type of Service*), ulazni interfejs i sl. Funkcionalnosti koje pruža ovaj protokol su:

- prikupljanje saobraćaja – beleži podatke o svim tokovima u mreži, uključujući broj paketa, bajtova, trajanje toka, i druge statistike,

- analiza stanja mreže – identifikacija različitih obrazaca saobraćaja pomaže u otkrivanju problema u mreži, poput zagušenja ili nastanka nepravilnosti u radu mreže,
- monitoring bezbednosti – otkrivanje anomalija u saobraćaju, koje ukazuju na napade, poput DDoS (eng. *Distributed Denial of Service*) napada ili neovlašćenih aktivnosti,
- optimizacija resursa – obezbeđuje informacije koje se koriste se za planiranje kapaciteta mreže i optimizaciju performansi.

Osnovni elementi u arhitekturi ovog protokola prikazani na slici 58. su [97]:

- *NetFlow Exporter* (NF-E),
- *NetFlow Collector* (NF-C) i
- analitički softver.



Slika 58. NetFlow arhitektura [97]

NetFlow Exporter je komponenta mrežnog uređaja, uređaja, koja prikuplja podatke o mrežnim tokovima (sirove informacije o saobraćaju) i šalje ih *NetFlow collector*-u radi analize i skladištenja. Suštinski, ova komponenta prvo identifikuje tokove na osnovu zajedničkih parametara, a zatim informacije o identifikovanim tokovima beleži u tabeli tokova i to u formi zapisa toka (eng. *Flow record*). Ovi zapisi sadrže informacije, kao što su trajanje toka, broj paketa i bajtova, ali i vremenske oznake (prvi i poslednji paket) i TCP obeležja/zastavice (eng. *flags*). Osnovna funkcija tabele tokova je da omogući brzo prepoznavanje i grupisanje paketa u tokove i time eliminiše potrebu za obradom svakog paketa pojedinačno. Funkcionisanje tabele tokova je u najvećoj meri uslovljeno njenim kapacitetom, koji zavisi od hardverskih resursa uređaja. Naime, ako ima previše tokova,

tabela takova može postati „zagušena“, što dovodi do gubitka podataka. U mrežama sa velikim brojem tokova, visoka frekvencija ažuriranja i brisanja može opteretiti uređaj. To ukazuje na potrebu, da se izvrši optimizacija resursa, tako što bi se ograničio broj tokova koji se istovremeno prate, a time i smanjilo opterećenje mrežnog uređaja. Kada tok postane neaktivan ili kada istekne definisano vreme (eng. *timeout*), zapis se uklanja iz tabele tokova i priprema za izvoz u *NetFlow Collector*.

U SDN mrežama, *NetFlow Collector* se obično nalazi van ravni podataka i integrisan je u kontrolnu ili aplikacionu ravan. Njegova tačna pozicija zavisi od mrežne arhitekture i namene. U realnim okruženjima, na poziciju *NetFlow Collector*-a utiče veličina mreže (u većim mrežama se često koriste distribuirana organizacija *NetFlow Collector*-a), performanse SDN kontrolera (ako SDN kontroler ima ograničene resurse, *NetFlow Collector* se postavlja odvojeno) i analitika zahteva (za dublju analizu, *NetFlow Collector* se integriše sa specijalizovanim aplikacijama u aplikacionoj ravni). Njegov osnovni zadatak je, da komunicira sa *NetFlow Exporter*-ima preko TCP ili UDP protokola i prima zapise o tokovima. Ove zapise, *NetFlow Collector* mora da dekodira iz formata koje koristi *NetFlow Exporter* i da ih organizuje u strukturiranu formu, koja može biti analizirana ili prikazana korisnicima. Ovde je važno napomenuti, da skladištenje podataka podrazumeva smeštaj prikupljenih zapisa u bazu podataka ili u fajl sistem za kasniju analizu (podaci mogu biti skladišteni dugoročno za istorijske analize ili kratkoročno za *real-time* obradu). Time se stvaraju neophodni uslovi za dublju analizu podataka o tokovima (identifikaciju aplikacija koje koriste mrežu, praćenje opterećenja mreže ili otkrivanje anomalija, kao što su DDoS napadi ili neovlašćeni pristup) i otkrivanje korelacija tokova iz više uređaja, čime se dobija kompletan uvid u saobraćaj u mreži.

Važna komponenta u *NetFlow* arhitekturi je i analitički softver. Skladište sa *NetFlow* zapisima je izvor podataka za druge aplikacije, kao što su alati za nadzor mreže, bezbednosni alati ili QoS analitičke platforme. Ovaj softver generiše izveštaje o mrežnim performansama i bezbednosti i može komunicirati sa SDN kontrolerima. Ovi izveštaji mogu sadržati informacije o tome ko su najveći potrošači saobraćaja, koje aplikacije generišu najviše saobraćaja ili ukazati na skrivene detalje o specifičnim tokovima.

Dakle, u SDN mrežama se *NetFlow* protokol koristi za prikupljanje i analizu podataka sa mrežnih uređaja, kako bi kontroler imao detaljniji uvid u saobraćaj i time donosio kvalitetnije odluke o rutiranju saobraćaja, dinamičkoj alokaciji resursa i reakcijama na pretnje u pogledu bezbednosti mreže. U praksi se može sresti više verzija ovog protokola, od kojih su najčešća verzija *NetFlow v5*, koja koristi statički format za tokove, bez fleksibilnosti. Pored toga, često se srće i *NetFlow v9* verzija, koja poseduje određena poboljšanja i uvodi podršku za šablone (eng. *template*) tokova, a koristi se i za prenos podataka u *IPFIX* formatu (eng. *Internet Protocol Flow Information Export*). Uprkos brojnim prednostima, kao što su detaljna vidljivost mrežnog saobraćaja, mogućnost otkrivanja i analize anomalija u saobraćaju, stvaranje preduslova za planiranje kapaciteta mreže i jednostavna integracija alata za analitiku, *NetFlow* poseduje i određena ograničenja:

- generisanje velike količine podataka može da optereti mrežu i *Collector*-e,
- kašnjenja u analizi zbog obrade velike količine informacija i
- kompleksnost konfiguracije u velikim mrežama.

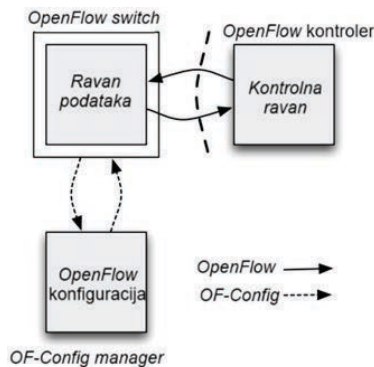
7.4.2. OF-Config protokol

OF-Config protokol ima specifičnu ulogu u razmeni informacija o topologiji SDN mreže. Iako njegova osnovna uloga nije otkrivanje topologije mreže (taj zadatak je često delegiran OpenFlow protokolu ili drugim protokolima), on omogućava konfiguraciju i pružanje ključnih informacija koje pomažu kontroleru da stekne tačan uvid u stanje mreže. Razvijen je od strane ONF-a kao prateći protokol za OpenFlow. Naime, OpenFlow je dizajniran da omogući upravljanje mrežnim saobraćajem kroz definisanje odgovarajućih pravila, dok je zadatak OF-Config protokola da obezbedi postavljanje osnovnih konfiguracija na mrežnim uređajima, upravljanje resursima uređaja (podešavanje interfejsa, alokacija resursa i upravljanje sertifikatima za bezbednu komunikaciju) i da omogući OpenFlow kontroleru komunikaciju sa mrežnim uređajima.

Uloga OF-Config protokola u razmeni topoloških informacija se pre svega ogleda u činjenici, da on obezbeđuje razmenu informacija o fizičkim interfejsima i portovima (npr. status, brzina i tipovi), kreiranim virtuelnim switch-evima i njihovim atributima, kao i o konekcijama (mogućnost da se identifikuju aktivni i povezani portovi). Na osnovu ovih informacija, SDN kontroler sagledava fizičke i logičke aspekte mrežne topologije. Dakle, OF-Config omogućava kontroleru da na osnovu ovih informacija konfigurise OpenFlow uređaje, uključujući IP adrese, portove i sertifikate. Potrebno je naglasiti, da OF-Config ne pruža mrežnim uređajima informacije o njihovim susedima, već samo omogućava da oni budu pravilno konfigurisani, kako bi mogli da šalju ili primaju pakete drugih protokola, koji se koriste za otkrivanje topologije. Drugim rečima, ovaj protokol obezbeđuje uslove za efikasno upravljanje i alokaciju resursa, koje mrežni uređaji mogu koristiti za povezivanje specifičnih portova sa logičkim switch-evima, što pomaže u preciznom mapiranju mrežnih veza i resursa.

Na slici 59. prikazana je OF-Config arhitektura, koju čine [95]:

- OpenFlow switch,
- OpenFlow kontroler i
- OF-Config Manager.



Slika 59. OF-Config arhitektura [95]

Uloga *OpenFlow switch*-a u arhitekturi sa slike 59. je da vrši propuštanje, preusmeravanje ili odbacivanje paketa na osnovu pravila definisanih od strane SDN kontrolera. Konfiguracija resursa (fizički portovi, virtuelni *switch*-evi kreirani unutar fizičkog uređaja, VLAN-ovi, i dr.) se vrši korišćenjem *OF-Config* protokola. Glavna komponenta na *OpenFlow switch*-u koja implementira funkcionalnosti *OF-Config* protokola je *OF-Config manager*. On upravlja fizičkim i logičkim resursima *OpenFlow switch*-a, tako što vrši obradu konfiguracionih zahteva koje šalje SDN kontroler i sinhronizaciju između *OF-Config* i *OpenFlow* funkcionalnosti na uređaju. Uloga kontrolera u *OF-Config* arhitekturi je da vrši osnovnu konfiguraciju uređaja (podešavanje portova, instalacija sertifikata i kreiranje logičkih *switch*-eva), da komunicira sa *OpenFlow switch*-em preko *OpenFlow* protokola i prosleđuje pravila za usmeravanje saobraćaja, kao i da dodeljuje resurse *OpenFlow switch*-evima u skladu sa zahtevima aplikacija ili mrežnim politikama. Važno je napomenuti, da se za pouzdanu i standardizovanu komunikaciju između SDN kontrolera i *OF-Config Manager*-a koristi standardan *NETCONF* protokol, koji koristi XML format za razmenu podataka. Potrebno je istaći i to da je primenom TLS mehanizma enkriptovana komunikacija između kontrolera i *OpenFlow switch*-eva.

Prednosti primene *OF-Config* protokola se odnose na činjenicu, da ovaj protokol omogućava konzistentan i pouzdan način konfiguracije uređaja u heterogenim mrežama, da upotpunjuje funkcionalnosti *OpenFlow* protokola bez preklapanja i da nudi mehanizme za upravljanje sertifikatima i šifrovanom komunikacijom. Međutim, postoje i određena ograničenja, koja se pre svega odnose na zavisnost od *NETCONF* protokola, što može uticati na kompleksnost konfiguracije u mrežama koje nisu prilagođene *NETCONF* protokolu. Dodatna, otežavajuća činjenica je i to, što *OF-Config* protokol nije uvek podržan u svim implementacijama *OpenFlow* protokola.

7.5. PROTOKOLI ZA PRENOS POLITIKA KOJE SE DEFINIŠU NA KONTROLERU I IMPLEMENTIRAJU NA MREŽNIM UREĐAJIMA

Protokoli za prenos politika koje se definišu na kontroleru i implementiraju na mrežnim uređajima imaju zadatak da omoguće efikasnu komunikaciju i koordinaciju između kontrolne i ravni podataka. Oni prenose instrukcije, pravila i politike koje kontroler definiše da bi uticao na rad mrežnih uređaja. Drugim rečima, korišćenjem ovih protokola vrši se distribucija mrežnih politika i omogućava dinamičko prilagođavanje mreže prema trenutnim zahtevima (npr. pogledu skaliranja ili sprečavanja pojave zagušenja na pojedinim linkovima). Ovine se kontroleru omogućava, da kroz implementaciju globalnih politika, na centralizovan način upravlja mrežnim uređajima. Ovo podrazumeva prenos pravila za upravljanje saobraćajem, kao što su prosleđivanje paketa, prioriteti različitih servisa, filtriranje saobraćaja i sl. Prenos ovih pravila podrazumeva, da se izvrši ažuriranje konfiguracije mrežnih uređaja, što uključuje i podešavanje portova, VLAN-ova, tunela, IP adresa i drugih mrežnih parametara.

Drugi važan zadatak, koji se odnosi na primenu ovih protokola je praćenje stanja mreže i telemetrija. Naime, ovi protokoli se koriste i za komunikaciju u smeru od mrežnih uređaja ka kontroleru i prenos informacija o stanju mreže, performansama, zagušenjima u mreži i greškama na uređajima ili linkovima. Prenosom ovih informacija do kontrolera,

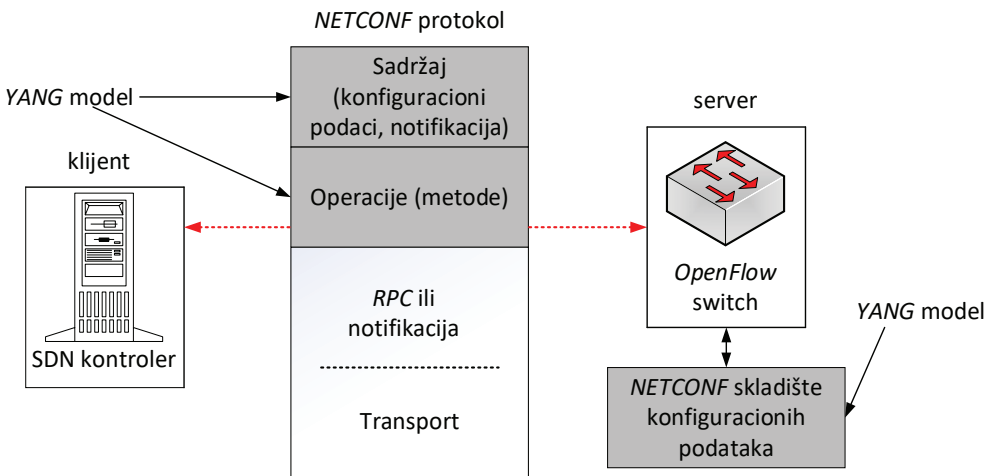
omogućava se donošenje odluka u realnom vremenu (npr. preusmeravanje saobraćaja usled prekida na linku), koje su zasnovane na stanju u mreži. Implementacijom novih pravila ili funkcionalnosti bez potrebe za ručnim podešavanjem svakog uređaja, povećava se fleksibilnost i smanjuje potreba za intervencijom administratora, što ukazuje na značaj primene ovih protokola u procesu unošenja značajno većeg nivoa programabilnosti u mreži.

Pored *OpenFlow* i *OVSD* protokola, u realnim okruženjima se u ovu svrhu koriste i još neki protokoli, kao što su:

- *NETCONF* protokol,
- *gRPC* (eng. *Google Remote Procedure Call*) i
- *P4Runtime* protokol i dr..

7.5.1. NETCONF protokol

NETCONF je standardan protokol razvijen od strane IETF-a u skladu sa RFC 6241 dokumentu. Koristi se u realnim okruženjima, da bi se na fleksibilniji način, korišćenjem XML format podataka, mogla menjati konfiguracija uređaja u SDN mreži. Ovaj protokol je veoma lak za integraciju sa različitim SDN kontrolerima [98]. Za razmenu informacija koristi SSH preko TCP porta 830 i pruža API podršku kontrolerima i aplikacijama za implementaciju SDN-a. S obzirom, da mrežni uređaji čuvaju podatke u YANG strukturi modela podataka, *NETCONF* protokol se može koristiti za preuzimanje i ažuriranje podataka na uređajima. Drugim rečima, mrežni uređaji čuvaju konfiguracione podatke u standardnoj YANG strukturi modela podataka, a kontroler može poslati *RPC* poziv (eng. *Remote procedure call*) da promeni ili preuzme konfiguraciju u XML formatu (Slika 60).



Slika 60. NETCONF protokol u SDN mreži [99]

Kao što se može videti na slici 60., ključne komponente ovog protokola su:

- *NETCONF* klijent – SDN kontroler je klijent koji inicira sesiju i šalje upravljačke zahteve prema mrežnim uređajima, koristeći *NETCONF* za dinamičko prilagođavanje konfiguracija, praćenje stanja mreže i izvršavanje odgovarajućih politika na njima,
- *NETCONF* Server – mrežni uređaj je server koji podržava *NETCONF* protokol i omogućava izvršavanje operacija i odgovaranje na zahteve koje šalje SDN kontroler kao klijent,
- *YANG* modeli – koriste se u kombinaciji sa *NETCONF* protokolom za standardizaciju strukture konfiguracija i podataka o stanju uređaja i
- *XML* – koristi se strukturiranu, proširivu i čitljivu razmenu podataka u funkciji upravljanja mrežnim uređajima.

Pokretanje *NETCONF* sesije počinje kada SDN kontroler (klijent) inicira bezbednu vezu sa *OpenFlow* mrežnim uređajem (*NETCONF* serverom), korišćenjem *SSH* ili *TLS* protokola. Odmah nakon uspostavljanja veze, SDN kontroler se mora predstaviti (autentifikovati) serveru (mrežnom uređaju) korišćenjem korisničkog imena i lozinke ili sertifikata. Osim provere identiteta kontrolera, server mora da izvrši i njegovu autorizaciju, tj. da proveri pravo kontrolera da izvršava određene akcije (npr. modifikacija konfiguracije ili pristup podacima). Nakon uspešne provere, *NETCONF* protokol omogućava kontroleru i mrežnim uređajima da vrše različite operacije, u zavisnosti od njihovih uloga u komunikaciji (kontroler inicira određene operacije, dok mrežni uređaj izvršava iste na osnovu zahteva koje mu šalje kontroler).

Operacije koje inicira SDN kontroler kao *NETCONF* klijent su:

- *get* – omogućava kontroleru da prikupi informacije o konfiguraciji uređaja ili njegovom trenutnom stanju, kao što su IP adrese, konfiguracija interfejsa, status i statistika,
- *edit-config* – ključna operacija za dinamičko upravljanje mrežnim uređajem u SDN okruženju, koju kontroler koristi da bi modifikovao konfiguraciju uređaja (dodavanje novih pravila ili podešavanje parametara mrežnog uređaja),
- *copy-config* – daje mogućnost da se konfiguracija replicira sa jednog na drugi uređaj u mreži,
- *delete-config* – koristi se za brisanje konfiguracija na uređaju, tj. kada je potrebno ukloniti zastarele ili nevažeće konfiguracije,
- *commit* – kontroler koristi ovu operaciju da potvrdi promene koje su izvršene na konfiguraciji uređaja, da bi te promene postale trajne,
- *rollback* – omogućava povratak na prethodnu verziju konfiguracije na uređaju, ako promene koje su izvršene nisu odgovarale ili su dovele do problema,
- *lock* – koristi se za "zaključavanje" konfiguracije uređaja. kako bi se sprečile konkurentne promene dok se izvršavaju ključne izmene i
- *unlock* – otključava konfiguraciju nakon što su izmene urađene.

Operacije koje vrši mrežni uređaj kao *NETCONF* server su:

- *get* – odgovor na zahtev kontrolera za dohvaćanje podataka, gde uređaj dostavlja kontroleru konfiguraciju, statistike, status interfejsa i druge informacije koje kontroler zahteva.
- *edit-config* – izvršava promene koje je zahtevao kontroler, uključujući promene u konfiguraciji uređaja (npr. postavljanje IP adresa, promena ruta, modifikacija QoS pravila i dr.),
- *copy-config* – u skladu sa zahtevom kontrolera vrši kopiranje svoje konfiguracije, npr. u drugi uređaj, kao što je to slučaj u topološkim promenama ili implementaciji novih uređaja,
- *delete-config* – briše konfiguraciju prema zahtevu kontrolera,
- *commit* – potvrđuje primenu novih postavki na uređaju,
- *rollback* – vrši vraćanje na prethodnu konfiguraciju i poništava urađene promene
- *lock* – vrši zaključavanje konfiguracije na zahtev (nijedna konkurentna sesija neće moći da menja konfiguraciju dok se trenutna sesija ne završi) i
- *unlock* – otključava konfiguraciju.

Dakle, *NETCONF* protokol omogućava centralizovano upravljanje SDN mrežnim uređajima, standardizaciju konfiguracija putem *YANG* modela, bezbednu tj. enkriptovanu komunikaciju (*SSH/TLS*) i dinamičku konfiguraciju uređaja. Ono što je važno, on podržava i operacije kao što su *rollback* i *commit*, kao i praćenje stanja uređaja u realnom vremenu. Međutim, primena *NETCONF* protokola u realnim okruženjima, a posebno u velikim mrežama, može biti kompleksna. Razlog za to leži i problemima sa performansama (*XML* format), kao i potrebi da se obezbedi visok nivo skalabilnosti i dodatna infrastruktura za praćenje komunikacije. U nekim slučajevim se može pojaviti i problem interoperabilnosti i to pre svega zbog različitih implementacija pojedinih vendora.

ZAKLJUČAK

Osnovu SDN koncepta predstavljaju protokoli, čija primena ima za cilj da se omogući efikasno i fleksibilno upravljanje mrežnom infrastrukturom. Od karakteristika i mogućnosti protokola, zavisi efikasnost komunikacije između SDN kontrolera i uređaja koji se nalaze u ravni podataka. Drugim rečima, primenom ovih protokola, definiše se način na koji se politike i konfiguracije prenose, primenjuju i prate, čime se stvaraju uslovi za centralizovano upravljanje, automatizaciju procesa i optimizaciju korišćenja mrežnih resursa. U tom smislu, *OpenFlow* je danas jedini standard/protokol koji omogućava direktnu kontrolu nad podacima iz mreže. Njegovom implementacijom, kontroler dobija mogućnost da definiše tokove i pravila na mrežnim uređajima. Drugi protokoli, kao što je npr. *NETCONF* protokol, uz korišćenje *YANG* modela, omogućavaju standardizaciju konfiguracije i praćenje stanja mrežnih uređaja. Ovo je veoma važno za postizanje interoperabilnost u heterogenim mrežama.

Jasno je, da primena ovih protokola donosi mnogobrojne prednosti za mreže u kojima se implementira SDN funkcionalnost. Stvaraju se uslovi, da se u mrežama upravlja na centralizovan način i da se primenom naprednih rešenja baziranih na softveru unese veći nivo programabilnosti. U praksi to znači automatizaciju procesa, smanjenje kompleksnosti konfiguracije i mogućnost brze reakcije na promene u mreži. Ipak, izazovi poput skalabilnosti u velikim mrežama, interoperabilnosti između rešenja različitih proizvođača, kao i bezbednosni rizici, zahtevaju pažljivu implementaciju i održavanje. Zato je neophodno raditi na daljoj evoluciji SDN protokola i to kroz razvoj i standardizaciju ovih protokola, zajedno sa inovacijama u oblasti dalje softverizacije mreže. SDN protokoli danas predstavljaju temelje za izgradnju inteligentnijih mreža, koje bi se lako adaptirale, bile veoma skalabilne i spremne da odgovore na izazove „umreženog“ društva, uključujući integraciju 5G/6G mreža sa IoT aplikacijama i računarstvom na cloud. Efikasnom primenom ovih protokola, SDN mreže dobijaju mogućnost da utiču na budući dizajn mreže, čineći ih fleksibilnijim, bezbednijim i bolje prilagođenim budućim potrebama korisnika i aplikacija.



POGLAVLJE 8



SD-WAN TEHNOLOGIJA

8. SD-WAN TEHNOLOGIJA

8.1. UVOD U SD-WAN

Tradicionalne WAN mreže (eng. *Wide Area Network*) su bile orijentisane na povezivanje korisnika u filijalama ili kampusu, sa aplikacijama koje se nalaze na serverima u *data* centrima podataka. Da bi se obezbedilo pouzdano i bezbedno povezivanje korišćene su namenske veze poput MPLS linkova, što ne funkcioniše u modernim okruženjima, koja se zasnivaju na primeni cloud tehnologije. Kako se u preduzećima sve više koriste cloud servisi, kao što SaaS (eng. *Software as a Service*) i IaaS (eng. *Infrastructure as a Service*), pa čak i kreiraju *multicloud* okruženja (korisnik bira servis cloud provajdera koji mu najviše odgovara), dolazi do ogromnog povećanja saobraćaja [100]. U ovakvoj situaciji, iskustvo sa korisničkim aplikacijama je loše, jer WAN mreže nisu spremne za eksponencijalni rast saobraćaja, koju donosi korišćenje servisa u cloudu. Taj saobraćaj uzrokuje složenost upravljanja, nepredvidljivost performansi aplikacija i ranjivost u zaštiti podataka. Dalje, otvaranje mreža u preduzećima za Internet i cloud izaziva ozbiljne propuste u bezbednosti i stvara probleme vezane za interoperabilnost u pojedinim delovima mreže. Izuzetno je izazovno zaštititi kritičnu infrastrukturu mreže, u trenutcima kada zaposleni, partneri i gosti, pristupaju aplikacijama. Omogućavanje širokopojasnog pristupa na WAN-u čini bezbednosne zahteve akutnijim, stvarajući izazove kako balansirati između potrebe da se zadrži ili unapredi korisničko iskustvo, obezbedi veći nivo bezbednosti i smanji složenost upravljanja.

SD-WAN je mrežna tehnologija, koja koristi SDN funkcionalnost za potrebe upravljanje i optimizacije WAN veza. Na taj način, ova tehnologija pomaže preduzećima i drugim organizacijama, da na efikasniji, fleksibilniji i bezbedniji način povežu svoja udaljene kancelarije sa sedištem preduzeća, data centrima i servisima na cloudu. Osnovu SD-WAN koncepta čini softverski definisano upravljanje, koje se zasniva na odvajanju kontrolne od ravni podataka (*data plane*) i centralizovanoj kontroli nad mrežom. Ono što je karakteristično za ovu tehnologiju je podrška za više transportnih mreža. To znači, da umesto oslanjanja na jednu, specifičnu vrstu mreže, SD-WAN može simultano da koristi različite vrste mrežnih veza za prenos podataka (MPLS, 4G/5G/6G i satelitske veze) i time obezbedi zavidnu fleksibilnost, optimizaciju troškova i bolje performanse u mreži. Istovremeno korišćenje ovih veza znači, da podaci mogu biti usmereni kroz različite transportne mreže na osnovu potreba, prioriteta ili performansi. SD-WAN analizira performanse svake mreže u realnom vremenu i automatski bira najbolju putanju za svaki servis. Ovakav pristup donosi brojne prednosti, kao što su:

- optimizacija troškova – kombinovanjem jeftinijih sa pouzdanim vezama (npr. MPLS), organizacije mogu smanjiti operativne troškove,
- povećana dostupnost – problem prekida u jednoj mreži se rešava automatskim preusmeravanjem saobraćaja na drugu mrežu, koja je u tom trenutku dostupna
- bolje performanse – saobraćaj se dinamički usmerava preko mreže koja najbolje odgovara trenutnim potrebama aplikacija i
- fleksibilnost – mreža se može lako proširiti dodavanjem novih veza bez potrebe za značajnim promenama u infrastrukturi.

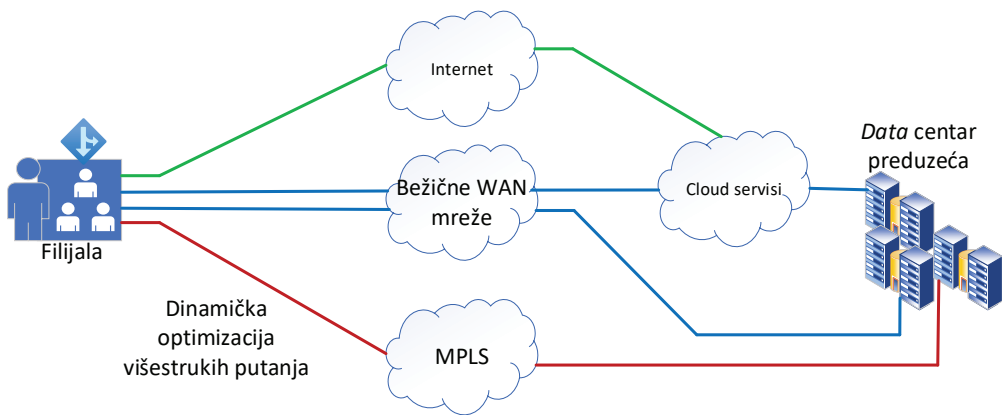
Danas se preduzeća sve više orijentišu na rad u cloud okruženju, a ne retko je i mobilnost ključna komponenta u radu zaposlenih. U takvim okolnostima, potrebno je definisati adekvatno mrežno okruženje, kao i bezbednosnu politiku, koja bi omogućila zaštitu podataka i korisničkih aplikacija u kompleksnim, distribuiranim okruženjima. U tom smislu, da bi se zaštitili podaci i mrežna infrastruktura, u SD-WAN mrežama se ugrađuju sledeći mehanizmi:

- enkripcija saobraćaja – mrežni saobraćaj između lokacija se enkriptuje primenom standardnih protokola (npr. IPSec ili TLS), čime se sprečava neovlašćen pristup,
- *firewall* – pruža zaštitu od neovlašćenog pristupa i napada i
- Segmentacija mreže – logičko razdvajanje različitih delova mreže (npr. IoT podmreža od podmreže poslovnih aplikacija), čime se minimizuju potencijalne štete od napada.

Potrebno je naglasiti, da se u SD-WAN mrežama implementira *Zero Trust* model bezbednosti, koji podrazumeva proveru svake konekcije i autentifikaciju korisnika i uređaja (npr. na osnovu kredencijala, sertifikata i dr.), pre nego što se dozvoli pristup mreži ili podacima. Integracijom mrežnih i funkcija bezbednosti u jedinstven servis, najčešće baziran na cloudu, kreira se tzv. SASE (eng. *Secure Access Service Edge*) arhitektura. Ova arhitektura obuhvata različite tehnologije kao što su CASB (eng. *Cloud Access Security Broker*),SWG (eng. *Secure Web Gateway*), DLP (eng. *Data Loss Prevention*) i ZTNA (eng. *Zero Trust Network Access*). U njenom središtu je centralizovano praćenje mrežnih aktivnosti i analitika podataka iz mreže, a sve sa ciljem da se problemi bezbednosti, kao i drugi problemi, brzo otkrivaju i rešavaju u realnom vremenu. Dakle, kroz integraciju sa cloud okruženjima, SD-WAN ima za cilj da omogućio direktan, optimizovan pristup cloud aplikacijama i servisima, zaobilazeći tradicionalne, centralizovane mrežne tokove.

8.2. SD-WAN ARHITEKTURA

Primena SD-WAN tehnologije ima za cilj, da se redefiniše način na koji se WAN mreže projektuju i da se mrežama u preduzećima i drugim organizacijama, obezbedi veća efikasnost, sigurnost i agilnost u implementaciji IT okruženja, koja se baziraju na modernim tehnologijama i servisima. Arhitektura mreža sa implementiranom SD-WAN tehnologijom se sastoji od nekoliko komponenti, koje zajedno treba da omoguće efikasno, fleksibilno i bezbedno upravljanje WAN mrežama. Osnovu arhitekture čini softverski definisan pristup, koji kontrolnu ravan odvaja od ravni prenosa podataka. Na slici 61. je prikazana arhitektura mreža sa implementiranom SD-WAN tehnologijom.



Slika 61. SD-WAN arhitektura [101]

Kao što je prikazano na slici 61., osnovne komponente arhitekture SD-WAN mreža su:

- *ravan podataka* – obuhvata fizičke i virtualne mrežne uređaje, koji razmenjuju podatke između lokacija preko transportnih mreža (MPLS, širokopojasni internet, 4G/5G/6G ili satelitske veze), enkriptuju saobraćaja radi bezbednosti i sposobni su da detektuju kašnjenja, gubitak paketa i promene protoka,
- *kontrolna ravan* – centralizovana mrežna logika (softver lokalizovan na sajtu preduzeća ili hostovan na cloudu) koja donosi odluke o dinamičkom usmeravanju paketa na osnovu mrežnih performansi, upravlja mrežnim politikama (npr. prioritet servisa ili bezbednost) i obezbeđuje redundansu i otpornost mreže,
- *upravljačka ravan* – sistem za orkestraciju, zadužen za analitiku i monitoring performansi mreže u realnom vremenu, definisanje mrežnih i politika bezbednosti i centralizovano donošenje upravljačkih odluka o svim lokacijama i uređajima,
- *ravan aplikacija* – fokusira se na optimizaciju performansi aplikacija koje koriste mrežu i poseduje funkcije za prepoznavanje tipa saobraćaja i optimizaciju rutiranja za specifične aplikacije (eng. *application-aware routing*) i

- *edge uređaji* – hardverski ili softverski uređaji instalirani na lokacijama filijala, data centrima ili na cloudu, koji upravljaju saobraćajem na lokalnom nivou, vrše enkripciju ili ulogu *firewall*-a i komuniciraju sa kontrolnom ravni radi izvršenja mrežnih politika.

Važno je naglasiti, da je efikasan protok podataka u SD-WAN arhitekturi, moguće opisati kroz tri ključne funkcije: upravljanje saobraćajem, dinamičko rutiranje saobraćaja i primena mehanizama za garantovanje bezbednosti. Za efikasnu realizaciju upravljačkih odluka donesenih na upravljačkoj ravni (orkestracija) zadužena je kontrolna ravan. Ova ravan analizira mrežne performanse za svaki tip servisa (kašnjenje, *jitter* i gubitak paketa) i bira najbolju rutu. Posebnu ulogu u samom rutiranju saobraćaja, koje se vrši u skladu sa dinamikom koja se zahteva iz mreže imaju SD-WAN *edge* uređaji. Ovi uređaji na svakoj lokaciji preusmeravaju saobraćaj u skladu sa centralno definisanim politikama.

Osnovne karakteristike, koje SD-WAN arhitekturu čine jedinstvenom i pogodnom za modernizaciju mreža, naročito u preduzećima i organizacijama koje poseduju više lokacija i aplikacije u cloudu su:

- dinamičko rutiranje saobraćaja – saobraćaj se usmerava na osnovu performansi mreže, tipa aplikacije i definisanih politika, a kritičnim aplikacijama (npr. VoIP ili video konferencije) se daje prioritet, što obezbeđuje visok kvalitet usluge,
- centralizacija upravljačkih funkcija – konfiguracija i preimena mrežnih politika u mreži se vrši sa jedne lokacije, pa administratori mogu upravljati i nadzirati mrežu putem jednog interfejsa, što smanjuje kompleksnost administracije,
- podrška za višestruke transportne mreže – prenos podataka se može vršiti preko različitih transportnih mreža, uz mogućnost da se obezbedi redundansa i otpornost mreže kroz *failover* mehanizme između različitih mrežnih veza,
- bezbednost na mrežnom nivou – enkripcija saobraćaja između lokacija, segmentacija mreže i odvajanje kritičnih aplikacija i podataka, kao i integracija sa *firewall*-ovima i drugim bezbednosnim alatima garantuju bezbednost aplikacija i uređaja uz ZTNA pristup,
- podrška za integraciju sa cloudom – jednostavno povezivanje sa aplikacijama i servisima u cloudu (npr. AWS, Microsoft Azure ili Google Cloud) kroz direktan pristup iz filijala, smanjuje latenciju i zavisnost od centralnog data centra,
- *real-time* monitoring i analitika – trenutni uvid u performanse mreže, saobraćaj i potencijalne pretnje u realnom vremenu omogućava proaktivnu identifikaciju problema i optimizaciju mreže na osnovu podataka.

U realnim okruženjima postoje sledeći modeli SD-WAN modeli implementacije:

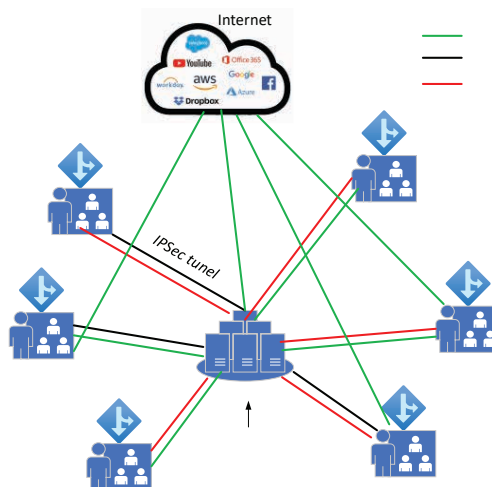
- *On-Premises* SD-WAN – kontrolna i upravljačka ravan tj. orkestracija se nalaze unutar mreže organizacije,
- *Cloud-Hosted* SD-WAN – kontrolna ravan je hostovana u cloudu, da bi se omogućio fleksibilniji i skalabilniji pristup i
- Hibridni model – kombinacija *on-premises* i cloud-hosted komponente, koji se prilagođava potrebama organizacije, obezbeđujući bezbednost i fleksibilnost.

8.3. ON-PREMISES SD-WAN IMPLEMENTACIJA

On-premises SD-WAN implementacija podrazumeva postavljanje i upravljanje SD-WAN infrastrukturom direktno u lokalnom okruženju kompanije, umesto na cloudu. Ovo rešenje pruža organizacijama potpunu kontrolu nad mrežom, čime se povećava sigurnost i omogućava veća prilagodljivost. Naime, ova implementacija se koristi, kada je potrebno zadržati sve podatke unutar mreže kompanije ili kada industrijski standardi ili zakoni ne dozvoljavaju upotrebu cloud rešenja i pre svega kada je neophodno da postoje jedinstvene mrežne politike i konfiguracije. Sama realizacija ovog modela SD-WAN mreža zahteva, da se sprovedu sledeći koraci:

- tačna procena broja lokacija koje treba povezati,
- definisanje prioriteta aplikacija,
- identifikacija potreba za primenom bezbednosnih rešenja, kao što su segmentacija mreže ili napredni *firewall*,
- određivanje kapaciteta veza i redundanse,
- instalacija kontrolne ravni (SDN kontrolera) u *data* centru kompanije, njegova konfiguracija i povezivanje sa *edge* uređajima i ostalom mrežnom infrastrukturom,
- definisanje i implementacija politika za rutiranja i prioriteta za aplikacije,
- implementacija bezbednosnih politika (enkripcija podataka, segmentacija mreže i definisanje pravila pristupa),
- konfigurisanje *Failover* mehanizama za automatizovano prebacivanje saobraćaja u slučaju pada veze i
- implementacija rešenja za monitoring performansi tj. kontinuirano praćenje mreže preko lokalnog kontrolera.

Na slici 62. prikazana je arhitektura *On-Premises* SD-WAN mreže [102].



Slika 62. *On-Premises* SD-WAN arhitektura [102]

Kao što se vidi na slici 62., *On-Premises* implementacija SD-WAN arhitekture uključuje implementaciju nekoliko komponenti, koje zajedno omogućavaju funkcionalnost i efikasnost SD-WAN mreže. Svaka komponenta ima specifičnu ulogu u upravljanju, rutiranju i obezbeđivanju mrežnog saobraćaja. Centralna komponenta koja upravlja svim aspektima SD-WAN mreže, uključujući politike rutiranja, sigurnosne postavke i optimizaciju saobraćaja je SDN kontroler. U ovoj SD-WAN arhitekturi, kontroler je lokalno implementiran u *data* centru. Njegova uloga je, da upravlja *edge* uređajima, definiše i distribuira mrežne politike, kao i da garantuje bezbednost u mreži, kroz autentifikaciju i enkriptovanje komunikacije. Važno je istaći, da lokalna infrastruktura (eng. *On-Premises Infrastructure*), pored smeštaj kontrolera, bezbednosnih komponenti (*firewall*) i lokalnih mrežnih veza, mora da obezbedi redundansu kroz lokalne i regionalne resurse. Ono što je još važnije, na lokalnoj infrastrukturi se implementira i platforma za orkestraciju. U suštini, to je softver za automatizaciju i koordinaciju mrežnih operacija. Njegova uloga je centralizovano upravljanje svim komponentama SD-WAN mreže i automatska distribucija konfiguracija i ažuriranja *edge* uređajima. Ova platforma je povezana sa sistemom za monitoring i alatima za analitiku (*monitoring systems and analytics tools*), čiji je zadatak praćenje performansi mreže (latencija, gubici paketa i propusnost u realnom vremenu), ali i prepoznavanje bezbednosnih pretnji i anomalija u mreži.

Sami *edge* uređaji se instaliraju na svim udaljenim lokacijama (filijalama), koje treba da budu deo SD-WAN mreže. Oni mogu biti fizički ili virtualni uređaji, koji se povezuju na lokalnu mrežu (LAN) putem *Ethernet* portova i na dostupne transportne mreže. Kada se *edge* uređaj poveže na lokalnu infrastrukturu, on odmah pokušava da se autentifikuje kod lokalnog kontrolera koristeći unapred konfigurisane ključeve (eng. *pre-shared keys*) ili sertifikate. Ovime se obezbeđuje, da se samo ovlašćeni uređaji mogu povezati na mrežu. Nakon uspešne autentifikacije, kontroler šalje početne konfiguracije *edge* uređajima, koje uključuju politike rutiranja, pravila za prioritet aplikacija i bezbednosna pravila (*firewall*, enkripcija). Da bi se imala bezbedna komunikacija sa kontrolerom (podrazumeva bezbedno upravljanje uređajem i slanje različitih ažuriranja), neophodno je da se kreira tunel (*IPSec* tunel) između *edge* uređaja i lokalnog kontrolera. Činjenica, da je kontroler lokalno hostovan, utiče na to, da je kašnjenje u komunikaciji između kontrolera i *edge* uređaja minimalno, što omogućava brzu reakciju na promene u mreži. Kroz ovaj bezbedan kanal komunikacije, kontroler šalje *edge* uređaju različite politike, koje se odnose na dinamičko rutiranje saobraćaja, definisanje prioriteta servisima, ali i pravila koja se odnose na enkripciju podataka i identifikaciju aplikacija u cilju optimizacije performansi. S druge strane, *edge* uređaji kontinuirano šalju telemetrijske podatke kontroleru, uključujući informacije o latenciji, gubitku paketa, statusu veze (primarna, sekundarna) ili potencijalnim napadima. Ovi podaci služe kontroleru, da vrši dinamičku optimizaciju saobraćaja, ali i da detektuje i rešava probleme u realnom vremenu.

Dakle, osnovne funkcionalnosti unutar *On-Premises* SD-WAN arhitekture se izvršavaju na sledeći način:

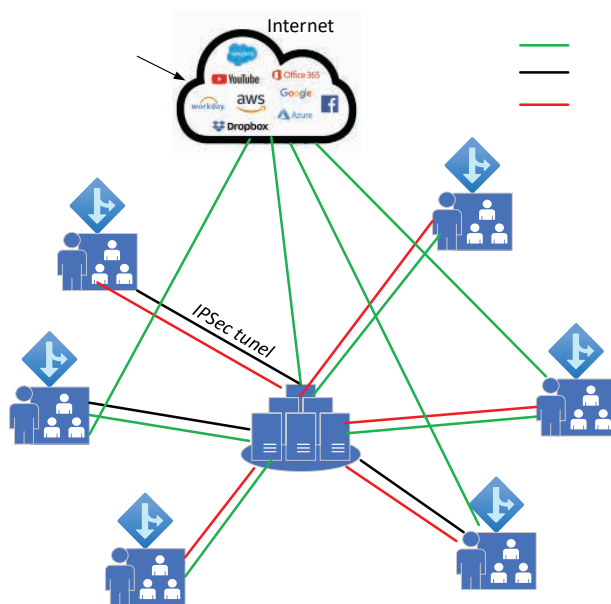
- dinamičko rutiranje i optimizacija saobraćaja se vrši automatski na osnovu trenutnih zahteva iz mreže, tipa aplikacija i unapred definisanih politika (npr. kritične aplikacije dobijaju prioritet u odnosu na druge aplikacijama kako bi se obezbedila niska latencija),
- upravljanje i orkestracija se sprovode od strane administratora kroz centralizovani upravljački interfejs za definisanje pravila i upravljanje mrežom, a sve promene poput dodavanja novih lokacija ili ažuriranja pravila se automatski distribuiraju sa kontrolera na sve *edge* uređaje,
- razmena saobraćaja između *edge* uređaja se vrši raspoloživih mrežnih linkova, a u slučaju pada nekog od linkova, saobraćaj se automatski prebacuje na druge dostupne linkove bez prekida usluge (*failover*),
- podaci se enkriptuju i implementira ZTNA politika kako bi se sprečio neovlašćen pristup u mreži i
- alati za analitiku prate performanse mreže, identifikuju probleme u realnom vremenu i generišu izveštaji, koji pomažu u optimizaciji mreže i unapređenju kvaliteta usluge.

Pored brojnih prednosti, koje nudi ova arhitektura, a koje se odnose na na potpunu kontrolu nad mrežom, politikama i podacima (sve komponente se nalaze lokalno), bezbednost, optimizovane performanse i redundansu na različitim nivoima, postoje i sledeći izazovi:

- visoki troškovi – nabavka i održavanje hardverskih i softverskih komponenti,
- kompleksnost upravljanja – potrebno je tehničko znanje za implementaciju i održavanje mreže,
- skalabilnost – dodavanje novih lokacija može biti složenije i sporije u poređenju sa *cloud-hosted* SD-WAN rešenjima.

8.4. CLOUD-HOSTED SD-WAN

Cloud-Hosted SD-WAN implementacija se primenjuje kod preduzeća i organizacija, koje žele jednostavnu implementaciju, optimizaciju za cloud aplikacije i smanjenje operativnih troškova. Ovu implementaciju SD-WAN arhitekture karakteriše pozicioniranje SDN kontrolera i usluge mrežnog upravljanja u cloud okruženju [100]. Time se preduzećima omogućava, da koriste prednosti SD-WAN tehnologije bez potrebe za lokalnim hostovanjem kontrolera ili infrastrukturnih komponenti, čime se pojednostavljuje implementacija i održavanje. Drugim rečima, instalacijom kontrolera u cloudu, upravljanje politikama, bezbednosnim pravilima i rutiranjem obavlja se sa cloud-a. Da bi izvršili inicijalnu konfiguraciju, administratori pristupaju centralizovanom softveru putem web portala za konfiguraciju i monitoring mreže (Slika 63).



Slika 63. *Cloud-Hosted SD-WAN* arhitektura [102]

Kao što se vidi na slici 63., kontroler sa clouda vrši centralizovano upravljanje i distribuciju pravila rutiranja, bezbednosnih i QoS politika na *edge* uređaje putem bezbedne veze između kontrolera i svakog *edge* uređaja. Prvi zadatak *edge* uređaja je da se registruju na cloud kontroleru koristeći prethodno dodeljene ključeve (eng. *Pre-shared keys*) (prethodno podeljeni ključevi), X.509 sertifikate za autentifikaciju i verifikaciju ili u nekim slučajevima vršeći *Token-based* autentifikaciju. Nakon autentifikacije, kreiraju se bezbedni kanali komunikacije između kontrolera i svakog *edge* uređaja korišćenjem *IPSec*, *TLS* ili nekog drugog protokola za tunelovanje. Važno je napomenuti, da ovi tuneli omogućavaju bezbednu komunikaciju i centralizovan metod slanja konfiguracije sa kontrolera, koja uključuje pravila rutiranja (npr. preferirane puteve kroz *MPLS*, *broad-*

band, 4G/5G/6G mreže i dr.), prioritete servisa i bezbedonosnu politiku. Kontroler ima mogućnost da primeni specifična pravila za različite lokacije ili uređaje, čime se postiže fleksibilnost i optimizacija (npr. u velikim gradovima se koriste MPLS veze, a za udaljene lokacije bežične veze).

Jedan od važnijih zadataka u ovoj arhitekturi je sinhronizacija politika i monitoring. Svaka promena pravila na kontroleru, zahteva da se nova konfiguracija odmah sinhronizuje sa *edge* uređajima. Kontroler može da koristi mehanizme, koji mu omogućavaju, da šalje samo promenjene delove konfiguracije, čime se smanjuje opterećenje mreže. Zadatak *edge* uređaja je da potvrdi prijem i primenu novih politika, ali bez obzira na to kontroler vrši proaktivne provere da bi bio siguran, da svi uređaji koriste najnovije politike.

Optimizacija performansi i bezbednosti se vrši kroz dinamičko rutiranje i primenu različitih bezbednosnih funkcija. Naime, kontroler kontinuirano prikuplja telemetrijske informacije sa *edge* uređaja. On koristi ove informacije, da bi odlučio koje su putanje optimalne za svaki servis. Štaviše, on može da utiče na bezbednost u mreži, tako što distribuira pravila za segmentaciju mreže, enkripciju saobraćaja između *edge* uređaja ili vrši ažuriranje pravila na *firewall*-u. Važna prednost ove arhitekture je i mogućnost automatskog skaliranja i dodavanja novih uređaja. Naime, novi uređaji na *edge*-u mreže se automatski povezuju sa kontrolerom, tako što koriste prethodno definisane šablone konfiguracije sa kontrolera i time se automatski sinhronizuju sa postojećim mrežnim politikama.

Prednosti implementacije *Cloud-Hosted SD-WAN* arhitekture su:

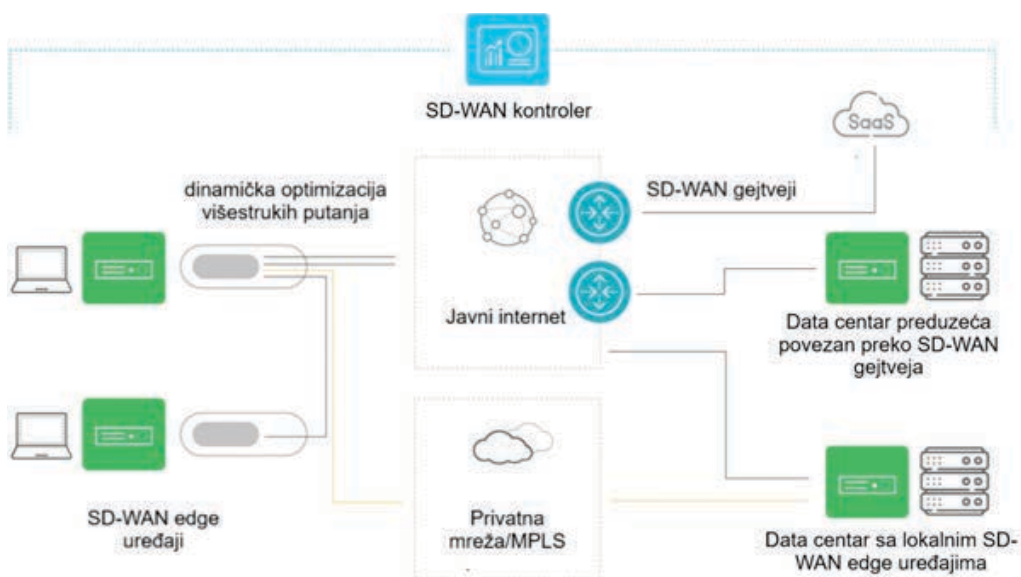
- smanjeni troškovi infrastrukture – nema potrebe za implementacijom i održavanjem kontrolera na lokaciji preduzeća, već se samo plaća usluga cloud provajderu, što smanjuje kapitalne izdatke,
- brža implementacija, jer nema potrebe za instalacijom lokalnog hardvera,
- skalabilnost – dodavanje novih lokacija je jednostavno i brzo
- direktna integracija sa SaaS i IaaS platformama omogućava bolji pristup cloud resursima i
- upravljanje mrežnim uređajima na bilo kojoj lokaciji putem cloud platforme.

Međutim, primena ove arhitekture je praćena i određenim izazovima:

- zavisnost od stabilnosti i bezbednosti infrastrukture cloud provajdera,
- regulatorna ograničenja u nekim domenima društvene delatnosti, koja se tiču hostovanja kontrolera van lokalne infrastrukture
- udaljenost cloud kontrolera može u specifičnim situacijama da utiče na povećanje latencije
- nemogućnost da se u nekim trenucima obezbedi pouzdan i brz internet, što je od ključnog značaja za optimalne performanse ovog rešenja.

8.5. HIBRIDNI MODEL SD-WAN ARHITEKTURE

Hibridni SD-WAN model implementacije kombinuje MPLS i Internet kao transportne mreže kako bi se optimizovale performanse i smanjili troškovi. Ovaj model kombinuje softversko upravljanje i dinamičkog usmeravanja saobraćaja između različitih tipova veza, kao što su *MPLS*, Internet i 4G/5G veze. Naime, kod ovog modela se *MPLS* koristi za transport podataka kritičnih aplikacija, koje zahtevaju nisku latenciju i visoku pouzdanost, dok se za druge aplikacije ili kao backup za *MPLS* koristi javni Internet. Usmeravanje saobraćaja se vrši dinamički u realnom vremenu na osnovu performansi mreže (kašnjenje, gubitak paketa i propusnost). Dakle, hibridni SD-WAN kombinuje različite mogućnosti *MPLS* i SD-WAN tehnologiju za stvaranje kompleksne i efikasne mrežne arhitekture, u kojoj SD-WAN služi kao prekrivač (eng. *overlay*) koji poboljšava fleksibilnost i kontrolu mreže. Koristi softver za upravljanje različitim virtuelnim mrežnim slojevima preko postojećih fizičkih mreža i omogućava preduzećima i drugim organizacijama da dinamički usmeravaju i preusmeravaju mrežni saobraćaj na osnovu specifičnih potreba različitih aplikacija (Slika 64.) [103].



Slika 64. Hibridna SD-WAN arhitektura [103]

Na slici 64. je prikazana hibridna SD-WAN arhitekturu koja kombinuje javni internet i privatnu MPLS mrežu za optimizaciju performansi, sigurnosti i troškova. Centralna komponenta je SDN kontroler, čija je uloga centralizovana kontrola i upravljanje SD-WAN infrastrukturom. On je zadužen da kreira i šalje mrežne politike prema gejtvejima (eng. gateways) i SD-WAN *edge* uređajima, kao i da vrši optimizaciju saobraćaja i automatizaciju mrežnih konfiguracija. Gejtveji se nalaze između javnog interneta i *data* centra

preduzeća i služe za povezivanje SD-WAN *edge* uređaje sa *data* centrima preduzeća, cloud aplikacijama ili javnim internetom. Oni funkcioniraju kao agregaciona tačka za saobraćaj sa SD-WAN *edge* uređaja, koji se optimizuje upravi u skladu sa instrukcijama SDN kontrolera. Važno je istaći, da su gejtveji zaduženi i za bezbednost veza između lokalnih resursa i aplikacija u cloudu. Kada je reč o samom povezivanju gejtveja sa SD-WAN *edge* uređajima i *data* centrima, ono se vrši preko javnog interneta, koji omogućava jeftinu i brzu povezanost. Tamo gde se zahteva stabilan i pouzdan prenos podataka između lokacija i gde je neophodno pojačati bezbednost i garantovati kvalitet usluge, koristi se i privatna mreža sa MPLS-om kao transportnim mehanizmom. Inače, hibridni SD-WAN koristi MPLS zajedno sa javnim internetom za postizanje balansa troškova i performansi. Ovde je važno naglasiti, da su SD-WAN *edge* uređaji krajnje tačke koje povezuju udaljene lokacije sa ostatkom SD-WAN mreže i vrše lokalnu optimizaciju saobraćaja i implementiraju politike kreirane od strane SD-WAN kontrolera. Ovi uređaji kontinuirano prate performanse, kao što su latencija, gubitak paketa i *jitter* za svaki dostupni link (npr. javni internet i MPLS). Ove informacije se dostavljaju kontroleru, koji definiše politike u skladu sa kojima *edge* uređaji dinamički biraju optimalan put za saobraćaj. Štaviše, *edge* uređaji na lokaciji *data* centra mogu lokalno da vrše optimizaciju saobraćaja i direktno primenjuju politike dobijene od kontrolera.

Za SD-WAN arhitekturu je karakteristično, da SDN kontroler može biti smešten na različitim lokacijama u zavisnosti od implementacije [104], pa čak i na dva mesta istovremeno, u zavisnosti od funkcionalnosti i potreba mreže. Kao centralizovana komponenta, SD kontroler može biti instaliran u cloudu ili u lokalnom *data* centru. Kada se kontroler nalazi u javnom ili privatnom cloudu, administratori mu pristupaju preko javnog interneta putem *web* interfejsa ili API-ja. Prednosti ovakve implementacije su brzina i jednostavnost, visoka fleksibilnost i skalabilnost, kao i mogućnost da se mrežom upravlja sa bilo koje lokacije. Priroda podataka i komunikacija često opredeljuju preduzeća, da kontroler instaliraju u *data* centru i da sami održavaju kontroler, mrežne uređaje i infrastrukturne resurse (komunikacija između kontrolera i uređaja se odvija kroz privatnu mrežu ili VPN tunel). Prednosti ovakve implementacije su vezane za visok stepen kontrole nad podacima i komunikacijama, ali su prisutni i izazovi, koji se odnose na troškove za održavanje i skaliranje infrastrukture, kao i manjak fleksibilnosti u poređenju sa implementacijom na cloudu.

Hibridna SD-WAN arhitektura ne mora nužno podrazumevati dva kontrolera, ali se može interpretirati na više načina u zavisnosti od konteksta. U hibridnom modelu kontroler se može kombinovati, tako da se u cloudu ili *data* centru implementira centralizovani kontroler, dok se na *edge* uređajima implementiraju lokalni kontroleri, koji nisu potpuno nezavisni kontroleri. Lokalni kontroleri se na *edge* uređajima implementiraju sa delom funkcija tj. nadležnosti kontrolne ravni. U realnim okruženjima, centralizovani kontroler upravlja globalnim politikama i konfiguracijom mreže, dok lokalni kontroleri na *edge* uređajima, na osnovu globalnih politika dinamički donose lokalne odluke kao što je dinamičko usmeravanje saobraćaja u realnom vremenu preko dostupnih putanja.

U realnim okruženjima se može sresti i rešenje sa dva SDN kontrolera, koji se instaliraju na cloudu i *data* centru. Oni se koriste za upravljanje različitim aspektima mreže, što je čest slučaj, naročito u složenijim implementacijama, kada je potrebno balansirati između centralizovane i decentralizovane kontrole. Kontroler u *data* centru ima zadatak da

upravlja kritičnim segmentima mreže, kao što su povezivanje sa lokalnim aplikacijama, *data* centrima ili resursima koji moraju ostati u kontrolisanom okruženju. S druge strane, kontroler koji se instalira na cloudu upravlja mrežama i saobraćajem između udaljenih lokacija, pružajući uvid u performanse globalne mreže i omogućavajući dinamičko upravljanje saobraćajem između lokalnih mreža i oblaka (npr. SaaS aplikacije). Da bi se postigle optimalne performanse u oba okruženja, porebno je sinhronizovati politike oba kontrolera. To znači, da kontroler u cloudu mora da šalje lokalnom kontroleru informacije o optimalnim putanjama za saobraćaj prema određenim cloud aplikacijama, dok kontroler u *data* centru mora slati podatke o performansama, koji kontroler u cloud treba da iskoristi i izvrši prilagođenje globalne rute i politike za celu mrežu. Dakle, ova izvedba kontrolne ravni omogućava dinamičku optimizaciju saobraćaja, jer se u u zavisnosti od dostupnosti i performansi mrežnih veza u svakom trenutku koristi najbolja putanja.

Generalno, prednosti hibridnog modela sa dva kontrolera su:

- fleksibilnost – preduzeća biraju najbolju putanju za svaki servis, kombinujući lokalnu kontrole za osetljive aplikacije i globalneu optimizaciju u cloudu za brži rast i promene,
- skalabilnost – kontroler na cloudu omogućava lako skladištenje podataka i analizu za velike mreže sa mnogim povezanim lokacijama, dok kontroler u *data* centru omogućava brzu reakciju i visoku kontrolu u lokalnim mrežama,
- bezbednost – kontroler u *data* centru obezbeđuje povezivanje i enkripciju podataka između *edge* uređaja na mrežnim tačkama, dok kontroler na cloudu omogućava globalnu bezbednost sa pristupom bazama podataka i aplikacijama i
- optimizacija troškova – kombinovanjem privatnih mreža i javnog interneta se smanjuju troškovi.

ZAKLJUČAK

SD-WAN tehnologija predstavlja nov pristup u organizaciji, implementaciji i upravljanju WAN mrežama. Njena primena ima za cilj, da se omogući centralizovano upravljanje mrežnim politikama sa clouda ili iz *data* centara u preduzećima. Time se značajno utiče na performanse aplikacija, smanjuju troškovi povezivanja i obezbeđuje veća fleksibilnost u izboru mrežnih putanja za različite servisa. Mogućnost, da se kroz dinamički izbor najboljih putanja u realnom vremenu, izvrši optimizacija mrežnog saobraćaja i poveća efikasnost i pouzdanost mreže, predstavlja značajan benefit za preduzeća. Međutim, praksa je pokazala, da SD-WAN tehnologija kroz implementaciju hibridnog modela pruža još veći nivo skalabilnosti i bezbednosti u mreži. Kombinacijom prednosti privatnih i javnih mreža, preduzeća mogu da koriste privatne mreže poput MPLS-a za kritičan saobraćaj, kako bi se očuvala visoka pouzdanost i performanse, dok za ostale servis mogu da koriste javni internet.

Deljenjem kontrolne ravni kroz implementaciju *on-premises* i *cloud-hosted* kontrolera, moguće je kreirati model SD-WAN arhitekture, koji karakteriše bolje upravljanje lokalnim i globalnim saobraćajem, kao i veća fleksibilnost u mrežnoj infrastrukturi. Time preduzeća

i organizacije, dobijaju mogućnost da na agilniji način vrše povezivanje mrežnih uređaja. Smanjujući potrebu za fizičkim povezivanjem, brže se odgovaraju na promene u poslovanju i rapidno povećanje zahteva korisnika. Pri tome, moguće je postići bolje performanse, smanjiti troškove i unaprediti bezbednost brže, sve uz jednostavan i optimalan model upravljanja u mreži.



POGLAVLJE 9



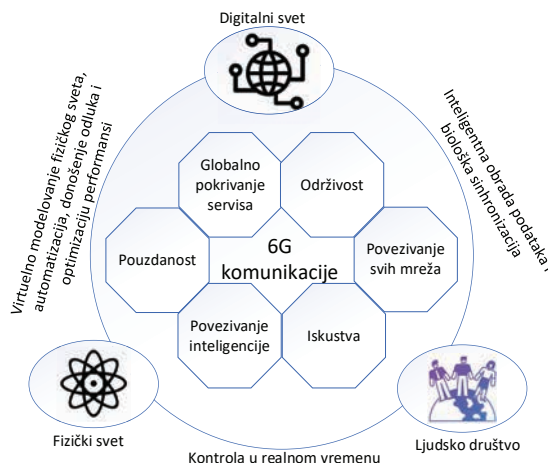
SDN I MREŽNI *SLICING*

9. SDN I MREŽNI SLICING

9.1. MODERNI TRENDOWI U KOMUNIKACIJAMA

Moderni trendovi u komunikacijama donose velike novine, koje se pre svega ogledaju kroz razvoj naprednih tehnologija kao što su 5G i 6G mreže, IoT, veštačka inteligencija i virtualizacija mreža. 5G mreže omogućavaju brzu, pouzdanu vezu sa niskom latencijom, dok se 6G fokusira na integraciju THz opsega (deo elektromagnetnog spektra sa frekvencijama između 0,1 THz i 10 THz, što odgovara talasnim dužinama između 3 mm i 30 μm .), hologramsku komunikaciju i veću energetska efikasnost. S druge strane, IoT omogućava povezivanje milijardi uređaja i kreiranje inteligentnih okruženja od pametnih kuća do industrijskih sistema, dok veštačka inteligencija unapređuje mrežno upravljanje kroz predikciju kvarova, automatizaciju i prilagođavanje infrastrukture zahtevima korisnika u realnom vremenu. Uz to, tehnologije poput SDN-a i NFV-a omogućavaju fleksibilnije i skalabilne mreže koje efikasno koriste resurse.

Danas smo svedoci pojave i drugih tehnologija, koji uključuju razvoj kvantne komunikacije za ultra-pouzdanu razmenu podataka, integraciju cloud *computing*-a i *edge computing*-a za aplikacije u realnom vremenu i satelitske mreže koje omogućavaju globalnu pokrivenost. Fokus se sve više pomera na energetska efikasnost, *cyber* bezbednost i interoperabilnost kroz otvorene standarde, što doprinosi održivosti i bezbednosti modernih mreža. Ovakav tehnološki razvoj omogućava pojavu novih aplikacija kao što su proširena (*Augmented Reality* - AR) i virtuelna stvarnost (*Virtual Reality* - VR), autonomna vozila i metaverzum (koncept digitalnog univerzuma, odnosno virtuelnog sveta koji objedinjuje fizičku i digitalnu stvarnost, omogućavajući korisnicima interakciju kroz personalizovane avatare, razmenu podataka i zajedničke aktivnosti u realnom vremenu). Primenom 6G komunikacija (Slika 65), stvara se osnova za primenu brže, pouzdanije i personalizovane komunikacije u različitim sferama industrije [105].



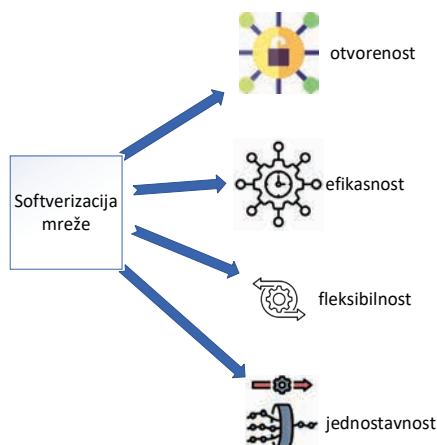
Slika 65. 6G komunikacije [105]

Moderni trendovi zahtevaju izgradnju mrežne infrastrukture, koja ne samo da prati tehnološki razvoj, već je i u stanju da odgovori na buduće zahteve korisnika i industrije. Ovi zahtevi se odnose na potrebu za postojanjem brze, pouzdane, ali i potrebama korisnika prilagodljive mrežne infrastrukture, u koju se lako mogu implementirati napredne tehnologije. Sami zahtevi, predstavljaju potrebu, da se izgradi infrastruktura, koju bi karakterisali:

- velika brzina i propusnost – mreže moraju podržavati velike brzine prenosa od nekoliko Gb/s do više Tb/s,
- minimalna latencija – latencija u mrežama mora biti ispod 1 ms, kako bi se omogućio efikasan rad real-time aplikacija, kao što su autonomna vozila, *cloud gaming* i dr.,
- povezanost velikog broja uređaja – mreže moraju da omoguće masovno povezivanje uređaja, kao npr. kod *massive* IoT okruženja (pametni gradovi, industrijski IoT, logistički sistemi i sl.),
- dinamička skalabilnost – sposobnost mreže da se dinamički prilagodi promenama u opterećenjima,
- bezbednost i privatnost – primena *end-to-end* enkripcije, zaštitu od napada i sigurno rukovanje podacima (npr. bezbednost finansijskih transakcija, privatnost zdravstvenih podataka ili zaštita komunikacije sa autonomnim uređajima),
- inteligentno upravljanje i automatizacija – integracija veštačke inteligencije u sisteme upravljanja treba da omogući prevenciju i automatsko rešavanje neželjenih događaja i optimizaciju resursa,
- energetska efikasnost – razvoj i primena energetski efikasnih mreža koje smanjuju potrošnju energije, posebno u IoT uređajima, ali i baznim stanicama,
- interoperabilnost i fleksibilnost – široka primena otvorenih standarda, kako bi se obezbedila interoperabilnost između različitih uređaja i tehnologija,
- napredne arhitekture mreža – primenom SDN i NFV tehnologija se kreiraju mreže prilagođene zahtevima korisnika,
- pouzdanost i kontinuitet usluga – mreže moraju obezbediti pouzdanost od 99.999% (pet devetki) i toleranciju na greške.

Jasno je, da je modernizaciju mrežne infrastrukture od ključnog značaja za ispunjavanje ovih zahteva. Ona se može izvršiti samo kroz „softverizaciju“ mreže, jer se tako mrežama omogućava, da budu agilnije, ekonomičnije i spremne za nove tehnološke izazove (Slika 66). Ovo je centralni element razvoja mreža sledećih generacija, uključujući 5G, 6G i IoT ekosisteme. Suštinski posmatrano, softverizacija mreže predstavlja proces transformacije tradicionalnih, hardverski baziranih mrežnih sistema u fleksibilne i dinamične mreže koje se upravljaju i optimizuju pomoću softvera. Ovaj pristup se oslanja pre svega na primenu NFV tehnologije, koja omogućava prebacivanje mrežnih funkcija sa specifičnog hardvera na generičke virtuelne servere i SDN koncept umrežavanja, koji razdvaja kontrolni sloj mreže od sloja za prenos podataka, omogućavajući centralizovano i programabilno upravljanje. Ovi aspekti, uz podršku automatizacije i orkestracije,

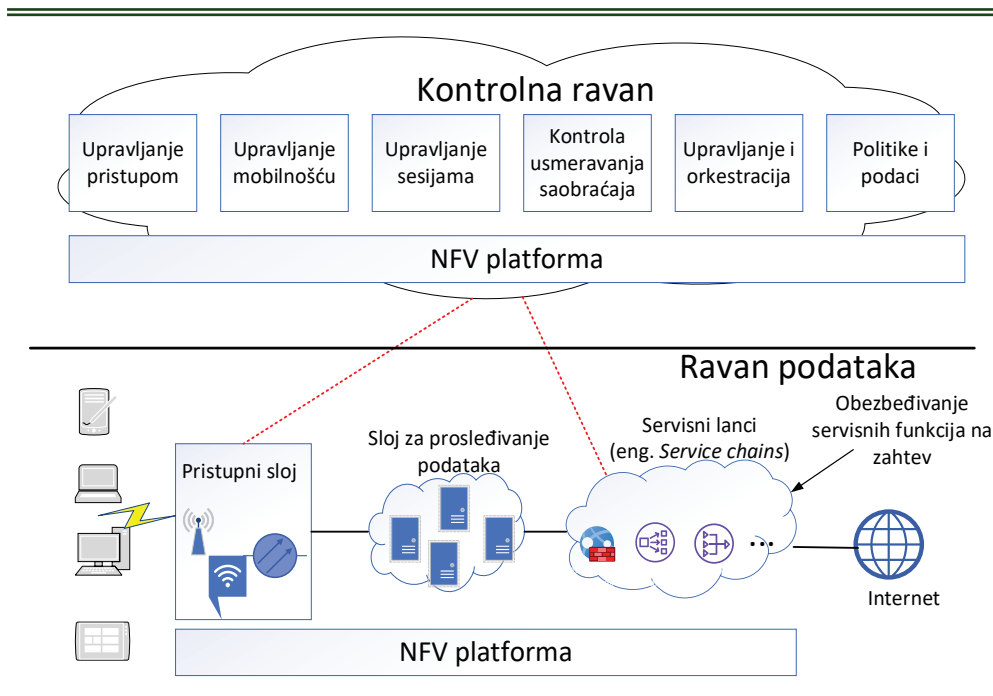
omogućavaju dinamičko prilagođavanje resursa, bržu implementaciju novih servisa i optimizaciju mrežnog saobraćaja. Potrebno je naglasiti, da se softverizacija oslanja na *cloud-native* arhitekturu (mreže postaju deo cloud infrastrukture, omogućavajući dinamičko skaliranje resursa u zavisnosti od potreba korisnika), pružajući fleksibilnost i lakšu integraciju novih tehnologija kao što su IoT i veštačka inteligencija. Upravo ove karakteristike čine 5G i 6G mreže efikasnijim i agilnijim, koje se lako mogu prilagoditi trenutnim zahtevima i biti lakše upravljive.



Slika 66. Softverizacija mreže

Dakle, može se reći, da softverizacija omogućava mrežama postanu inteligentnije, prilagodljivije i ekonomičnije u pružanju mrežnih usluga. Softverski usmerena mreža (Slika 67), gde se upravljanje i kontrola vrše korišćenjem softverskih platformi, eliminiše zavisnost od specifičnih hardverskih uređaja i nudi sledeće prednosti:

- veću fleksibilnost i agilnost – primenom softverskih alata, moguće je brzo uvođenje novih usluga i lakše upravljanje promenama u mreži,
- smanjene troškova – zamenom specijalizovanog hardvera sa standardnim serverima i softverskim rešenjima, troškovi za nabavku, instalaciju i održavanje mreže se značajno smanjuju,
- jednostavnije upravljanje mrežom – centralizovano upravljanje mrežom olakšava monitoring, automatsko skaliranje i optimizaciju resursa,
- veću skalabilnost – softverizacija omogućava mrežama da se lakše skaliraju u skladu sa potrebama, bilo da je u pitanju povećanje kapaciteta ili uvođenje novih funkcionalnosti,
- bržu i efikasniju implementaciju novih servisa – automatizacija procesa za konfiguraciju, upravljanje i orkestraciju mrežnih resursa omogućava lakše uvođenje novih tehnologija i servisa i
- veću bezbednost – primena softverskih rešenja omogućava brže i efikasnije implementiranje bezbednosnih politika i nadzor mreže.



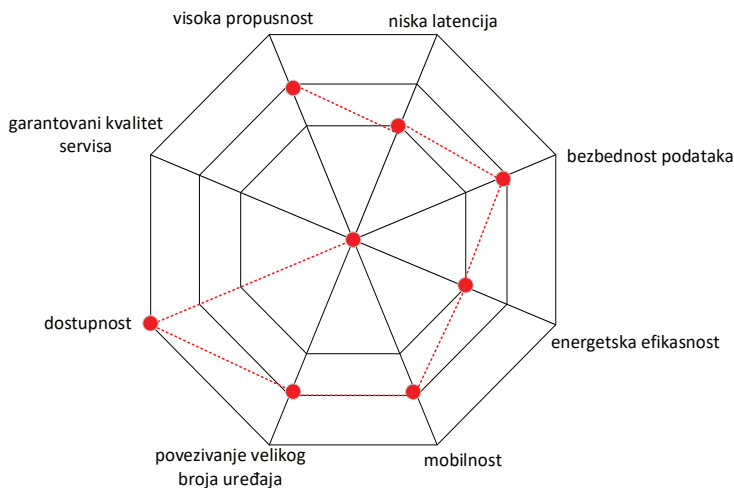
Slika 67. Arhitektura softverizovanih mreža

9.2. DEFINICIJA MREŽNOG SLICING-A U 5G/6G MREŽAMA

Mrežni *slicing* (eng. *Network Slicing* - NS) je tehnika, koja se primenjuje u modernim mrežama, kao što su 5G i 6G mreže, da se na jednoj (zajedničkoj) fizičkoj infrastrukturi omogući kreiranje više virtuelnih mreža tzv. *sliceova*. Svaki od mrežnih *slice*-va se može prilagoditi i optimizovati za potrebe pojedinih korisnika i njihove aplikacije, pružajući različite nivoe usluga i performansi. Dakle, *slice*-ovi su virtuelne mreže, koje mogu biti konfigurisane i upravljane nezavisno, u skladu sa zahtevima korisnika ili aplikacija. Njihove ključne karakteristike su:

- Izolacija, prilagođavanje i optimizacija mrežnih resursa, aplikacija ili funkcionalnosti prema specifičnim potrebama korisnika ili grupe korisnika – svaki mrežni *slice* se kreira u skladu sa specifičnim zahtevima korisnika i njihovih aplikacija (npr. niska latencija, visoka propusnost ili posebne mere bezbednosti),
- fleksibilnost i prilagodljivost – resursi se mogu dinamički dodeljivati, menjati konfiguracije ili prilagođavati performanse svakog *slice*-a u skladu sa trenutnim potrebama korisnika ili specifičnih aplikacija i
- virtualizacija i automatizacija – automatizovano kreiranje, upravljanje i optimizacija različitih *slice*-ova u mreži vrši se korišćenjem NFV i SDN tehnologije.

U realnim okruženjima postoje brojni razlozi zbog kojih se primenjuje mrežni *slicing* (Slika 68). Jedan od njih je podrška za različite usluge i aplikacije, što veoma važno kod mreža u kojima postoje različitim zahtevi aplikacija, kao što su npr. propusnost, latenciju i pouzdanost. Primer za to su 5G i 6G mreže, u kojima ova tehnologija omogućava pružanje specifičnih usluga za različite tipove korisnika i aplikacija, kao što su IoT i industrijski IoT, autonomna vozila i dr. Sve ove aplikacije imaju različite zahteve, a mrežni *slicing* omogućava kreiranje i optimizaciju virtuelnih mreža u skladu sa potrebama svake od njih. Sledeći važan razlog je intencija, da se poboljša efikasnost mreže [106]. U okviru *multitenancy* pristupa, gde svi korisnici dele zajedničku fizičku infrastrukturu, mrežni *slicing* omogućava optimizovano korišćenje resursa za svaki pojedinačni servis. Drugim rečima, svaki *slice* može da koristi mrežne resurse u skladu sa potrebama, čime se smanjuje opterećenje na mreži i povećava efikasnost.



Slika 68. Razlozi za primenu mrežnog *slicing*-a [106]

Mogućnost komercijalizacije novih servisa je dodatni motiv za primenu mrežnog *slicing*-a, jer operatori mogu da kreiraju nove, specijalizovane usluge, prilagođene potrebama različitih industrija ili korisničkih grupa. To znači, da se omogućava stvaranje poslovnih modela, koji operatorima ili pružaocima usluga daju mogućnost da naplate različite aspekte mrežnih usluga, kao što su brzina interneta, kvalitet usluge, dostupnost, i specijalizovane usluge prilagođene potrebama korisnika. Jasno je, da 5G i 6G mreže zahtevaju podršku za različite vrste saobraćaja, koja podrazumeva izuzetno nisku latenciju, visoku propusnost i mogućnost povezivanja milijardi uređaja. U tom smislu, mrežni *slicing* je veoma koristan, jer omogućava efikasno upravljanje i dodelu resursa specifičnih za određene aplikacije, čime se obezbeđuje visok kvalitet usluge i vrši optimizacija performansi.

Sama činjenica, da su mrežni *slice*-vi međusobno izolovani, pruža veći nivo bezbednosti i privatnosti. Aplikacije koje rade sa poverljivim podacima mogu biti smeštene u zasebne *slice*-ove sa specifičnim politikama u pogledu bezbednosti, dok one koje rade sa manje poverljivim podacima koriste druge *slice*-ve, u kojima politike bezbednosti nisu previše rigorozne. Potrebno je naglasiti i činjenicu, da mrežni *slicing* omogućava operatorima da pruže specifične usluge za različite grane industrije i društvenog života u celini. Dakle, mrežni *slicing* se može definisati sa konstatacijom, da je to ključna tehnologija za optimizaciju i skaliranje resursa u modernim 5G i 6G mrežama, koja omogućava da se fizička mreža podeli u virtuelne, nezavisne celine (*slice*-ve), koji se mogu prilagoditi različitim potrebama i time omogućiti veća efikasnost, fleksibilnost i kvalitet servisa, kao i podršku za različite aplikacije i industrije sa specifičnim zahtevima.

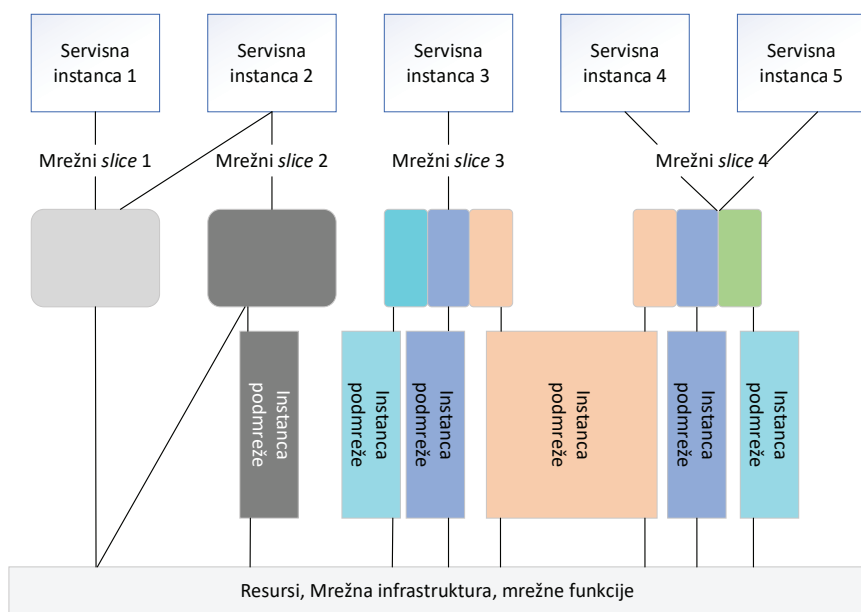
9.3. SDN I DINAMIČKO KREIRANJE I UPRAVLJANJE *SLICE*-VIMA

SDN zajedno sa dinamičkim upravljanjem mrežnim *slice*-vima predstavlja osnovu za modernizaciju mreža. Njihovom preimenom, operatori stiču mogućnost, da efikasno kreiraju, upravljaju i optimizuju upotrebu različitih servisa u realnom vremenu. Korišćenjem SDN tehnologije, proces kreiranja i prilagođavanja mrežnih *slice*-va potrebama korisnika i njihovih aplikacija se izvršava na fleksibilan i skalabilan način i može se automatizovati. Sve to pruža korisnicima bolja iskustva i omogućava efikasnije korišćenje mrežnih resursa. Zato je veoma važno, da se pravilno objasni kako primena SDN tehnologije u mreži omogućava dinamičko kreiranje i upravljanje *slice*-vima. Ključni faktor u ovome je sigurno centralizovano upravljanje svim aspektima mreže implementacijom SDN kontrolera. Njihova instalacija u mrežama olakšava procese kreiranja, konfiguracije i prilagođenja *slice*-va trenutnim potrebama korisnika i njihovih aplikacija. Za razliku od tradicionalnih mreža, gde je upravljanje virtuelnim mrežama tj. *slice*-vima distribuirano, u SDN mrežama se ono vrši na centralizovan način. Primenom kontrolera, različiti *slice*-vi automatski i dinamički dobijaju resurse u skladu sa trenutnim potrebama. Ta fleksibilnost u dodeli resursa, operatorima omogućava, da prilagode mrežu u realnom vremenu. Drugim rečima, operatori dobijaju mogućnost da dodeljuju resurse *slice*-vima prema potrebama njihovih korisnika. Ono što je još važnije, dodela resursa se vrši kroz softver i bez potrebe za fizičkom promenom infrastrukture.

Potrebno je istaći, da primena SDN kontrolera omogućava brzo i automatizovano podešavanje, što znači da administrator može brzo i automatski kreirati i konfigurisati *slice*-ove u skladu sa potrebama. Na primer, za aplikacije sa izuzetno niskom latencijom (npr. autonomna vozila), SDN može automatski kreirati *slice* sa potrebnim resursima i obezbediti niska kašnjenja. Pri tome, moguće je optimizovati performanse u realnom vremenu, uzimajući u obzir trenutno opterećenje i potrebe. Sve ovo je moguće jer SDN kontroler kontinuirano monitoriše saobraćaj i na osnovu prikupljenih podataka vrši njegovo preusmeravanje ili *slice*-u dodeljuje dodatne resurse kako bi održao optimalne performanse.

Primenom SDN kontrolera kao centralne tačke u mreži, moguće je na jednom mestu definisati podršku za različite aplikacije i grupe korisnika i to na način, da se različite vrste saobraćaja i aplikacija izdvoje u različite *slice*-ve. Na primer, u 5G i 6G mrežama, SDN kontroleri mogu automatski kreirati *slice*-ve za uređaje sa zahtevima za manjom propusnošću i latencijom, kao i one druge *slice*-ve koji se moraju optimizovati za aplikacije sa visokom propusnošću poput video *streaming*-a. Automatizacija, koja dolazi sa primenom SDN tehnologije, omogućava orkestraciju *slice*-va u skladu sa zahtevima korisnika i specifičnim scenarijima. Na primer, kada se otkrije povećano opterećenje na određenom *slice*-u, SDN kontroler može automatski proširiti resurse za taj *slice* ili preusmeriti saobraćaj na manje opterećene delove mreže. Štaviše, SDN kontroler je sposoban, da kreira i implementira različite politike u realnom vremenu, koje regulišu međusobnu komunikaciju *slice*-eva, dodelu resursa i kvalitet servisa. To je naročito važno za kritične aplikacije. Dakle, sam proces kreiranja *slice*-va u realnim okruženjima je veoma fleksibilan i zasniva se na različitim principima, sa ključni zahtevom – optimizovati broj *slice*-eva u mreži! U praksi se kod kreiranja *slice*-va primenjuju principi (Slika 69) [107], kao što su:

- svaki *slice* jedan servis,
- svaki *slice* grupa servisa sa istim ili sličnim zahtevima u pogledu resursa,
- svaki *slice* su servisi jednog provajdera servisa i dr.

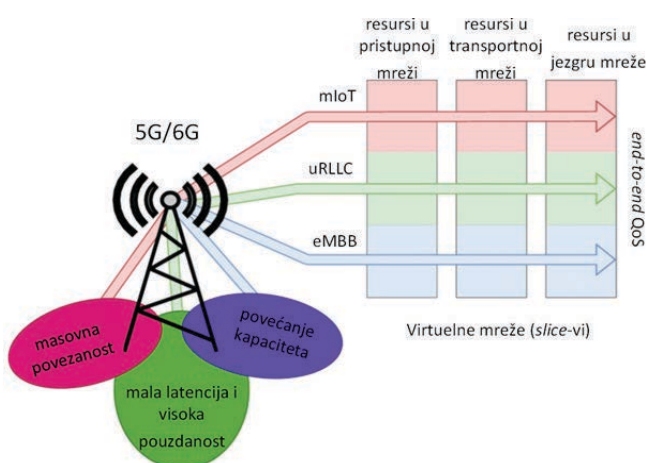


Slika 69. Principi mrežnog slicing-a [107]

9.4. RAZLIČITI TIPOVI *SLICE*-VA I ULOGA SDN-A U NJIHOVOM OPTIMIZOVANJU

Mrežni *slicing* je jedna od važnijih tehnologija u 5G i 6G mrežama. Ona omogućava stvaranje virtuelnih mreža (*slice*-va) na fizičkoj infrastrukturi jedn mreže. Kao što je prethodno napomenuto, ove virtuelne mreže su optimizovane za različite usluge i zahteve korisnika. U realnim 5G i 6G okruženjima se mogu sresti različiti tipovi mrežnih *slice*-va, od kojih su svakako najznačajnija sledeć tri tipa *slice*-va (Slika 70):

- eMBB (eng. *enhanced Mobile Broadband*) *slice* – namenjen za velike brzine prenosa podataka i veliki kapacitet,
- URLLC (eng. *Ultra-Reliable Low Latency Communication*) *slice* – namenjen aplikacijama koje zahtevaju izuzetno nisku latenciju (manju od 1 ms) i visoku pouzdanost i
- mMTC (eng. *massive Machine-Type Communication*) *slice* – namenjen za povezivanje velikog broja uređaja sa malim zahtevima za propusnim opsegom.



Slika 70. Mrežni *slice*-vi u 5G/6G mrežama [108]

eMBB *slice* je namenjen za aplikacije, koje zahtevaju veliki kapacitet i brzine prenosa podataka. Njegovo kreiranje ima za cilj, da se unapredi korisničko iskustvo u mrežnim scenarijima koji zahtevaju širokopojasni prenos podataka, poput *streaming*-a multimedije, cloud servisa i VR/AR tehnologija. Ključne karakteristike ovog tipa mrežnog *slicing*-a su [108]:

- velike brzine prenosa podataka reda veličine nekoliko Gb/s – idealan za aplikacije sa velikim zahtevima za propusnim opsegom, jer omogućava brzo preuzimanje i otpremanje velikih fajlova,
- veliki kapacitet – može podržati veliki broj korisnika, što je značajno u područjima sa visokom gustinom, u kojima veliki broj uređaja koristi mrežu istovremeno i

- pouzdanost i kvalitet servisa – obezbeđuje stabilnu i kvalitetnu vezu čak i u uslovima velike opterećenosti mreže (dinamička alokacija resursa omogućava da aplikacije sa visokim prioritetom imaju dovoljno kapaciteta).

U realnim 5G/6G okruženjima ovaj tip slice-va se koristi za:

- video *streaming* u visokoj rezoluciji – pruža kvalitetno iskustvo gledanja bez kašnjenja i učitavanja,
- realizaciju VR/AR okruženja u realnom vremenu zahvaljujući velikoj brzini i maloj latenciji,
- mobilne cloud servise – omogućava brzo preuzimanje i prenos velikih fajlova, kao i upotrebu cloud aplikacija bez prekida i
- podršku uređajima, kao što su pametni televizori, igračke konzole i drugi multimedijalni uređaji.

URLLC je jedan od ključnih tipova mrežnih *slice*-ova u 5G mrežama i koristi se aplikacije koje zahtevaju ekstremno visoku pouzdanost (99,999%) i veoma malu latenciju (manju od 1 ms) [109]. Posebno je važan tamo gde kašnjenje ili nepouzdana veza mogu izazvati ozbiljne posledice, što se pre svega odnosi na rešenja u obalsti industrijske automatizacije, autonomnih vozila ili daljinskih medicinskih operacija. Njegove ključne karakteristike su:

- ultra niska latencija (manja od 1 ms) – omogućava gotovo trenutnu reakciju mreže, što je ključno za aplikacije u realnom vremenu,
- visoka pouzdanost (99,999% ili veća) – garantuje minimalnu verovatnoću gubitaka podataka ili prekida veze, što je idealno za kritične aplikacije gde greške nisu prihvatljive,
- robusnost u uslovima ometanja – koristi napredne tehnike za izbegavanje smetnji i obezbeđivanje stabilne veze čak i u izazovnim okruženjima, kakva su urbana područja sa velikom gustinom uređaja i
- podrška za male pakete podataka – optimizovan je za prenos manjih količina podataka sa izuzetno kratkim vremenom obrade.

U realnim 5G/6G okruženjima ovaj tip *slice*-va se koristi za [109]:

- Autonomna vozila i brzu komunikacija između vozila i infrastrukture (eng. *Vehicle-to-Infrastructure* – V2I),
- Industrijsku automatizaciju i robotiku – povećava preciznost i smanjuje rizik od grešaka, što ga preporučuje, da se koristi za kontrolu industrijskih robota i mašina, koje zahtevaju brze i pouzdane povratne informacije,
- telemedicinu i daljinska hirurgiju – pouzdana i veza sa izuzetno niskom latencijom je odgovarajuća za uspešnu i bezbednu primenu složenih operacija na daljinu pomoću robotskih sistema i
- pametnim elektro-energetskim mrežama (eng. *Smart Grids*) – omogućava brzu i pouzdanu komunikaciju za upravljanje elektroenergetskim mrežama i detekciju kvarova u realnom vremenu.

mMTC *slice* je dizajniran za podršku velikom broju uređaja koji komuniciraju sa malim zahtevima za prenos podataka. Fokus, prilikom kreiranja ovog tipa *slice*-a je na optimizaciji mreže za aplikacije i usluge koje uključuju IoT uređaje, senzore i druge mašine, koje zahtevaju povezivanje na mrežu, ali sa minimalnim zahtevima za propusnim opsegom i visokom efikasnošću u smislu upravljanja velikim brojem povezanih uređaja. Njegove ključne karakteristike su [110]:

- značajno veliki broj povezanih uređaja – dizajniran je da podrži stotine miliona do milijarde uređaja, što ga čini idealnim za IoT primene (omogućava efikasno upravljanje velikim brojem uređaja koji šalju male količine podataka),
- mala potrošnja energije – optimizuje potrošnju energije kako bi produžio radni vek uređaja sa malim brzinama prenosa podataka (podaci se šalju periodično, što smanjuje potrebu za stalnim aktiviranjem bežične komunikacije),
- mali propusni opseg – uređaji u ovom *slice*-u obično šalju male količine podataka (npr. senzorski podaci), pa nisu potrebne velike brzine prenosa podataka i
- skalabilnost i efikasnost – omogućava efikasno korišćenje mrežnih resursa čak i u scenarijima sa velikim brojem povezanih uređaja (virtuelna mreža se može brzo proširiti kako bi zadovoljila potrebe novih uređaja koji se povezuju, bez značajnog uticaja na kvalitet servisa za druge korisnike).

U realnim 5G/6G okruženjima ovaj tip *slice*-va se koristi za [110]:

- uređaje i saobraćaj aplikacija vezanih za pametne gradove (eng. *Smart Cities*), kao što su pametni ulični svetlosni sistemi, senzori za kvalitet vazduha, uređaji za praćenje saobraćaja i dr., gde se šalju mali podaci, ali u velikim količinama, što zahteva efikasnu i pouzdanu mrežu,
- uređaje i saobraćaj koji su deo okruženja pametne kuće (eng. *Smart Homes*), kao što su pametni termostati, kamere, detektori pokreta i drugi elektronski uređaji sa minimalnim zahtevima za propusnim opsegom i energijom,
- povezivanje senzora za praćenje uslova u poljoprivredi, kao što su stanje zemljišta, temperatura, vlažnost, i drugi parametri, koji mogu komunicirati sa vrlo malim količinama podataka, čime se optimizuje upotreba energije i smanjuje trošak povezivanja.
- povezivanje sajber-fizičkih uređaja (pametne fabrike i automatizovani proizvodni procesi) i senzora koji prate parametre proizvodnje.

9.5. SDN UPRAVLJANJE RESURSIMA IZMEĐU RAZLIČITIH SLICE-VA

Mrežni *slicing* omogućava kreiranje virtuelnih mreža (*slice*-va) na zajedničkoj fizičkoj infrastrukturi, koje su optimizovane u skladu sa specifičnim zahtevima korisnika ili aplikacija. Upravljanje resursima podrazumeva procese i mehanizme, kojima se na efikasan, pouzdan i fleksibilan vrši alokacija mrežnih resursa (resursi mogu biti deo radio dela, transportnog dela ili jezgra mreže) unutar jednog ili više mrežnih *slice*-va. Slobodno se može reći, da alokacija resursa predstavlja ključan aspekt upravljanja resursima. Razlog leži u tome što se mora voditi računa o resursima koji su neophodni za pokretanje virtuelnih mrežnih funkcija (npr. CPU, memorija i sl.), širini propusnog opsega za prenos podataka, a u mobilnim mrežama i o resursima u radio mreži (spektralni resursi spektra moraju biti adekvatno raspodeljeni među *slice*-vima [111]). Pri tome, mora se voditi računa o dinamičkoj prirodi alociranja resursa, jer se oni moraju prilagođavati trenutnim potrebama aplikacija ili korisnika u *slice*-u. Ovo je posebno važno za aplikacije sa promenljivim opterećenjem (npr. tokom perioda maksimalnog opterećenja), ali uostalom i za druge aplikacije pošto je neophodno obezbediti za svaki *slice* vrednosti parametara kvaliteta servisa, koje su deo SLA (eng. *Service Level Agreement*) sporazuma, kao što su propusni opseg, kašnjenje, pouzdanost i dostupnost. Za pravilno upravljanje resursima, važno ulogu ima izolacija *slice*-va. Naime, problemi u jednom *slice*-u (npr. preopterećenje) ne smeju da utiču na performanse drugog *slice*-a i da dovedu do njihove degradacije. Zato je neophodno, da se kontinuirano prate performanse *slice*-va, kako bi se brzo otkrili problemi i pristupilo optimizaciji resursa u realnom vremenu. Ovakav pristup omogućava efikasno upravljanje resursima i smanjuje operativne troškove, čime se utiče na isplativost mrežne infrastrukture.

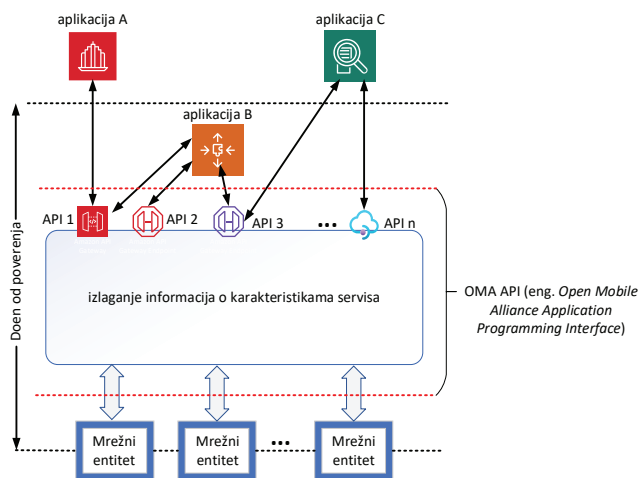
U mrežama sa implementiranim SDN i NFV tehnologijama, SDN kontroleri i orkestratori igraju ključnu ulogu u efikasnom upravljanju resursima. Oni su zaduženi za automatsko dodeljivanje virtuelnih resursa prema unapred definisanim politikama, ali i optimizacija korišćenja fizičkih resursa mreže. SDN kontroler na centralizovan način upravlja resursima svih mrežnih uređaja i koristi globalan uvid u mrežu da optimizuje resurse za svaki *slice*, tako što prati performanse i vrši dinamičku alokaciju resursa (podešava QoS politike). Alokacija resursa se vrši u realnom vremenu i u skladu sa potrebama aplikacija. Pri tome, zadatak kontrolera je, da pri alokaciji resursa prvo zadovolji zahteve aplikacija sa visokim prioritetom, kao i da vodi računa o novim *slice*-vima, kako bi se isti kreirali i skalirali prema zahtevima iz mreže. Važno je istaći i ulogu SDN kontrolera u izolaciji resursa, jer on daje instrukcije za kreiranje VLAN-ova ili VXLANOV-a, kako bi se izvršilo odvajanje saobraćaja i onemogućilo da podaci i resursi jednog *slice*-a utiču na druge *slice*-ve [112]. Još jedan važan zadatak kontrolera je kontinuirano praćenje iskorišćenosti i raspoloživosti resursa, kako bi se na osnovu tih podataka mogao uraditi rebalans resursa između *slice*-ova i aktiviralo skaliranje kada resursi postanu nedovoljni. Drugim rečima, ovo ima za cilj da se prikupe informacije koje su neophodne za promenu konfiguracije i prilagođavanje QoS politika za svaki *slice*. S druge strane, SDN kontroler pruža *Northbound* API koji omogućava SDN orkestratorima (npr. NFV MANO), da donese nove politike za raspodelu resursa, kreiraju, brišu ili prilagođavaju *slice*-ve. U tu svrhu se koristi analitika i veštačka inteligencija, kako bi se predvidele potrebe *slice*-va i proaktivno upravljalo resursima (npr. optimizacija ruta u mreži za minimalno kašnjenje).

Prednosti SDN upravljanja resursima između *slice*-ova su:

- fleksibilnost – brza reakcija na promene u opterećenju,
- efikasnost – maksimalna iskorišćenost fizičkih resursa,
- izolacija – stabilnost i nezavisnost svakog *slice*-a i
- skalabilnost – jednostavno povećanje kapaciteta mreže.

9.6. IZAZOVI IMPLEMENTACIJE MREŽNOG SLICING-A U 5G I 6G MREŽAMA

Mrežni *slicing* u 5G i 6G mrežama omogućava kreiranje virtuelnih mreža na jednoj fizičkoj infrastrukturi, koje moraju da zadovolje specifične zahteve različitih aplikacija. Jedan od glavnih izazova u realizaciji ovakvog okruženja je kompleksnost upravljanja i orkestracije. Dinamička alokacija resursa, praćenje performansi i balansiranje opterećenja između različitih *slice*-ova zahteva sofisticirane sisteme koji mogu obezbediti efikasnost i performanse svih usluga [113]. Takođe, međusobna izolacija između *slice*-ova i zaštita od uticaja problema u jednom *slice*-u na druge predstavlja dodatnu prepreku, naročito kada se uzmu u obzir visoki zahtevi za bezbednost i privatnost. Naime, ovde se mora uzeti u obzir potreba, da između različitih mrežnih domena i usluga mora postojati deljenje informacija. To utiče na bezbednost i privatnost podataka, jer različiti *slice*-vi mogu biti namenjeni za različite poslovne i industrijske aplikacije. Deljenje osetljivih podataka, kada se koriste zajednička infrastruktura ili resursi može ugroziti bezbednost, jer informacije o performansama i deljenim resursima mogu otkriti osetljive detalje o uslugama i dovesti do neovlašćenog pristupa ili napada (Slika 71) [114]. Izlaganje ovih informacija, najčešće dovodi do bezbedonosnih propusta na relaciji provajder infrastrukture-*tenant* (korisnik određenih mrežnih resursa), ali i između samih *tenant*-a. Ovde se uvek mora imati u vidu, da specifične aplikacije iz razloga bezbednosti (npr. kontrola vozila, e-hirurgija i sl.) ne dozvoljavaju deljenje resursa.



Slika 71. Bezbednosni izazov deljenja informacija [114]

Sledeći izazov se odnosi na interoperabilnost i standardizaciju. U sistemu gde različiti *slice*-ovi mogu koristiti različite protokole i tehnologije, deljenje informacija o resursima, performansama i statusima između različitih mrežnih komponenti može biti otežano zbog nedostatka univerzalnih standarda. Nedostatak univerzalnih standarda otežava integraciju različitih uređaja i platformi, dok je kompatibilnost sa starijim sistemima u mreži dodatni izazov za operatore. Ovaj problem je još izraženiji u 6G mrežama, gde se očekuje integracija novih tehnologija poput veštačke inteligencije, *blockchain*-a i kvantnih mreža, koje mogu zahtevati specifične pristupe u deljenju podataka.

Potrebno je istaći, da transparentnost mrežnog *slicing*-a predstavlja značajan izazov u 5G i 6G mrežama. Transparentnost predstavlja sposobnost da različite mreže, aplikacije i servisi, koji koriste različite *slice*-ve međusobno komuniciraju, a da se pri tome korisnici ne moraju baviti složenostima koje dolaze sa segmentacijom mreže. Obezbeđivanje transparentnosti između *slice*-ova može biti komplikovano iz više razloga. Svaki *slice* može imati različite zahteve u pogledu resursa, usluga i kvaliteta usluge, jer mrežni *slicing* uključuje različite tehnologije (5G, IoT, *edge computing* itd.). Da bi se obezbedilo da svi *slice*-vi rade u skladu sa njihovim SLA, potrebno je implementirati napredne sisteme za praćenje i orkestraciju u realnom vremenu. U takvim okolnostima, teško je obezbediti da svi *slice*-vi koriste mrežne resurse efikasno bez međusobnog uticanja. Štaviše, treba imati u vidu i činjenicu, da različiti dobavljači opreme i platforme mogu koristiti različite pristupe implementaciji *slicing*-a, što otežava interoperabilnost između njih. Dakle, pitanje raznih protokola i standarda dodatno komplikuje obezbeđivanje transparentnosti između različitih tehnologija koje čine jednu mrežu.

Na kraju, ekonomičnost i skalabilnost *network slicing*-a u 5G i 6G mrežama može biti značajan problem. Implementacija i održavanje složenih orkestracionih sistema za upravljanje resursima između *slice*-va zahtevaju visoka ulaganja u infrastrukturu i obuku. Osim toga, kako broj *slice*-va raste, potrebna je skalabilna infrastruktura koja može podržati različite aplikacije i garantovati kvalitet usluga. Automatizacija, korišćenje veštačke inteligencije za optimizaciju resursa i predviđanje opterećenja postaju ključni za prevazilaženje ovih izazova.

ZAKLJUČAK

Razvoj tehnologija kao što su SDN, NFV i mrežni *slicing* tehnologija je u velikoj meri uticao na promene u arhitekturi i upravljanju modernim mrežama i to posebno u okviru 5G i nadolazećim 6G mreža. Mogućnost, da se primenom SDN tehnologije na centralizovan način upravlja mrežnom infrastrukturom, korišćenjem softverskih komponenti - kontrolera, značajno utiče na povećanje fleksibilnosti u realizaciji zahteva korisnika i njihovih aplikacija i to pre svega kroz dinamiku u alokaciji mrežnih resursa. Kada se ovoj mogućnosti dodaju novine/prednosti mrežnog *slicing*-a, onda je sasvim jasno, da je kreiranje virtuelnih mreža tj. mrežnih *slice*-ova koji zadovoljavaju različite zahteve korisnika i aplikacija, kao što su nizak latencija, visoka propusnost, ili bezbednost, sasvim realno.

Međutim, implementacija mrežnih *slice*-va i upravljanje njihovim kreiranjem i dodelom resursa preko SDN kontrolera može biti praćeno određenim izazovima. Tu se pre svega odnosi nakompleksnost orkestracije i upravljanja resursima, bezbednost i izolaciju između različitih *slice*-va, kao i na potrebu za interoperabilnošću između različitih tehnologija i dobavljača. Transparentnost u pogledu resursa i performansi, kao i odgovarajuća zaštite podataka, takođe ostaju ključni izazovi. U 5G i 6G mrežama, ove tehnologije predstavljaju osnovu za implementaciju servisa, kao što su IoT, autonomna vozila, industrijski automatizovani sistemi, i AR/VR aplikacije, koji zahtevaju visoke standarde u pogledu kvaliteta servisa, skalabilnosti i upravljanja u realnom vremenu.

Očekuje se, da sa porastom složenosti i dinamike u budućim mrežama, upotreba SDN tehnologije i mrežnog *slicing*-a bude od velikog značaja za obezbeđivanje efikasnog, skalabilnog i bezbednog upravljanja resursima. Razvoj naprednih tehnika kao što su veštačka inteligencija i automatska orkestracija, može značajno da unapredi mogućnosti upravljanja, optimizacije i bezbednosti u ovim mrežnim okruženjima. Jasno je, da je potrebno i dalje posvećivati ozbiljnu pažnju i ulagati u standardizaciju i globalnu interoperabilnost, kako bi se omogućilo jednostavno povezivanje i rad različitih tehnologija i mrežnih komponenti.

Literatura

- [1] Ž. Bojović, J. Šuh, and P. Bojović, *Softverske tehnologije u računarskim mrežama sa velikim podacima*. Fakultet Tehničkih Nauka u Novom Sadu, 2021.
- [2] Ž. Bojović, "Application of Network Function Virtualization in Modern Computer Environments." Boston-Delft: now publishers, 2024.
- [3] S. Lee, K. Levanti, and H. S. Kim, "Network monitoring: Present and future," *Comput Netw*, vol. 65, pp. 84–98, 2014.
- [4] S. H. Haji et al., "Comparison of software defined networking with traditional networking," *Asian J. Res. Comput. Sci.*, vol. 9, no. 2, pp. 1–18, 2021.
- [5] S. Singh and R. K. Jha, "A survey on software defined networking: Architecture for next generation network," *J. Netw. Syst. Manag.*, vol. 25, pp. 321–374, 2017.
- [6] J. Šuh, Ž. Bojović, M. Despotović-Zrakić, Z. Bogdanović, and A. Labus, "Designing a course and infrastructure for teaching software-defined networking," *Comput Appl Eng Educ*, vol. 25, no. 4, pp. 554–567, 2017.
- [7] K. Kirkpatrick, "Software-defined networking," *Commun ACM*, vol. 56, no. 9, pp. 16–19, 2013.
- [8] O. Blial, M. B. Mamoun, and R. Benaini, "An overview on SDN architectures with multiple controllers," *J. Comput. Netw. Commun.*, vol. 2016, no. 1, p. 9396525, 2016.
- [9] M. S. Bonfim, K. L. Dias, and S. F. Fernandes, "Integrated NFV/SDN architectures: A systematic literature review," *ACM Comput. Surv. CSUR*, vol. 51, no. 6, pp. 1–39, 2019.
- [10] F. A. Lopes, M. Santos, R. Fidalgo, and S. Fernandes, "A software engineering perspective on SDN programmability," *IEEE Commun. Surv. Tutor.*, vol. 18, no. 2, pp. 1255–1272, 2015.
- [11] P. Danielis, V. Altmann, J. Skodzik, E. B. Schweissguth, F. Golaowski, and D. Timmermann, "Emulation of SDN-supported automation networks," in *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, IEEE, Sep. 2015, pp. 1–8.
- [12] M. Karakus and A. Durresi, "Quality of service (QoS) in software defined networking (SDN): A survey," *J. Netw. Comput. Appl.*, vol. 80, pp. 200–218, 2017.
- [13] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun Surv Tutor*, vol. 16, no. 3, pp. 1617–1634, 2014.
- [14] Sdxcentral, "What Is Open SDN? A Definition." [Online]. Available: <https://www.sdxcentral.com/networking/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/what-is-openflow/open-sdn/>
- [15] T. Malbašić, P. D. Bojović, Ž. Bojović, J. Šuh, and D. Vujošević, "Hybrid SDN Networks: A multi-parameter server load balancing scheme," *J Netw Syst Manag*, vol. 30, no. 2, p. 30, 2022.
- [16] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *ACM SIGCOMM Comput Commun Rev*, vol. 44, no. 2, pp. 70–75, 2014.

- [17] J. He and W. Song, "Achieving near-optimal traffic engineering in hybrid software defined networks," in *2015 IFIP Networking Conference (IFIP Networking)*, IEEE, May 2015, pp. 1–9.
- [18] GeeksforGeeks, "Types of Software Defined Networks Implementation." [Online]. Available: <https://www.geeksforgeeks.org/types-of-software-defined-networks-implementation>
- [19] N. F. Mir, *Computer and Communication Networks*. Pearson Education, 2015.
- [20] TechTarget, "Virtualization." [Online]. Available: <https://www.techtarget.com/searchitoperations/definition/virtualization>
- [21] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: a performance comparison," in *2015 IEEE International Conference on Cloud Engineering*, IEEE, Mar. 2015, pp. 386–393.
- [22] S. N. T. C. Chiueh and S. Brook, "A survey on virtualization technologies," *Rpe Report*, 142, 2005.
- [23] E. Bugnion, J. Nieh, and D. Tsafirir, *Hardware and software support for virtualization*, vol. 12. 2017.
- [24] R. Chayapathi, S. F. Hassan, and P. Shah, *Network Functions Virtualization (NFV) with a Touch of SDN: Netw Fun Vir (NFV EPub_1)*. Addison-Wesley Professional, 2016.
- [25] D. Stefanović, "I mreža se virtuelizuje," *Svet Kompjut*. 2005.
- [26] B. Goldworm and A. Skamarock, *Blade servers and virtualization: transforming enterprise computing while cutting costs*. John Wiley & Sons, 2007.
- [27] N. C. Fernandes et al., "Virtual networks: Isolation, performance, and trends," *Ann. Telecommun.-Ann. Télécommunications*, vol. 66, pp. 339–355, 2011.
- [28] S. Zhou, "Virtual networking," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 4, pp. 80–85, 2010.
- [29] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Performance characteristics of virtual switching," in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, IEEE, Oct. 2014, pp. 120–125.
- [30] J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby, "Virtual switching in an era of advanced edges." 2010. [Online]. Available: <http://66.246.76.178/papers/adv-edge.pdf>
- [31] N. Varis and J. Manner, "Performance of a software switch," in *2011 IEEE 12th International Conference on High Performance Switching and Routing*, IEEE, Jul. 2011, pp. 256–263.
- [32] Y. Dong, Z. Yu, and G. Rose, "SR-IOV Networking in Xen: Architecture, Design and Implementation," in *Workshop on I/O Virtualization*, Dec. 2008.
- [33] "Open vSwitch." [Online]. Available: <https://www.openvswitch.org/>
- [34] Y. Wang, E. Keller, B. Biskeborn, J. V. D. Merwe, and J. Rexford, "Virtual routers on the move: live router migration as a network-management primitive," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 231–242, 2008.
- [35] "Virtual Routing and Forwarding (VRF)." [Online]. Available: <http://www.patrickdenis.biz/blog/virtual-routing-and-forwarding-vrf/>
- [36] "Quagga Routing Software." [Online]. Available: <https://www.nongnu.org/quagga/>
- [37] "Open Source Routing." [Online]. Available: <https://www.opensourcerouting.org/>
- [38] M. Eder, "Hypervisor-vs. container-based virtualization," *Future Internet FI Innov Internet Technol Mob Commun IITM*, vol. 1, 2016.

- [39] NetApp, "What are containers?" [Online]. Available: <https://www.netapp.com/devops-solutions/what-are-containers/>
- [40] S. Ferry, "Kubernetes Application Containers: Managing Containers and Cluster Resources." [Online]. Available: <https://developer.hpe.com/blog/kubernetes-application-containers-managing-containers-and-cluster-resour/>
- [41] N. Dragoni, I. Lanese, S. T. Larsen, M. Mazzara, R. Mustafin, and L. Safina, "Microservices: How to make your application scale," in *Perspectives of System Informatics: 11th International Andrei P. Ershov Informatics Conference, PSI 2017, Moscow, Russia, June 27-29, 2017, Revised Selected Papers 11*, Springer International Publishing, 2018, pp. 95–104.
- [42] "Sandboxed Containers: Are Virtual Machines Making a Comeback in a New Avatar?" [Online]. Available: <https://itnext.io/sandboxed-containers-are-virtual-machines-making-a-comeback-in-a-new-avatar-cf83e1c4a5fb>
- [43] "Serverless Has Servers." [Online]. Available: <https://blog.bytebytego.com/p/serverless-has-servers>
- [44] C. Pahl and B. Lee, "Containers and clusters for edge cloud architectures—a technology review," in *2015 3rd International Conference on Future Internet of Things and Cloud*, IEEE, Aug. 2015, pp. 379–386.
- [45] I. Hrček, "Praktičan uvod u kontejner terminologiju." [Online]. Available: <https://startit.rs/uvod-kontejner-terminologija/>
- [46] A. Mouat, *Using Docker: Developing and Deploying Software with Containers*. O'Reilly Media, Inc., 2015.
- [47] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, 2014.
- [48] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using Docker technology," in *SoutheastCon 2016*, IEEE, Mar. 2016, pp. 1–5.
- [49] G. Kakivaya *et al.*, "Service fabric: a distributed platform for building microservices in the cloud," in *Proceedings of the Thirteenth EuroSys Conference*, Apr. 2018, pp. 1–15.
- [50] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 50–57, 2016.
- [51] Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–37, 2022.
- [52] Cloud Native Computing Foundation, "Certified Kubernetes Software Conformance."
- [53] A. E. Nocentino and B. Weissman, *SQL Server on Kubernetes: Designing and Building a Modern Data Platform*. Apress, 2021.
- [54] O. Yilmaz, "Extending the Kubernetes API," in *Extending Kubernetes: Elevate Kubernetes with Extension Patterns, Operators, and Plugins*, Berkeley, CA: Apress, 2021, pp. 99–141.
- [55] M. Hausenblas and S. Schimanski, *Programming Kubernetes: Developing Cloud-Native Applications*. O'Reilly Media, 2019.
- [56] Kubernetes, "ReplicaSet." [Online]. Available: <https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>
- [57] B. Burns, J. Beda, K. Hightower, and L. Evenson, *Kubernetes: Up and Running*. O'Reilly Media, Inc., 2022.

- [58] DEV, "When to use Kubernetes deployments, pods, and services." [Online]. Available: <https://dev.to/educative/when-to-use-kubernetes-deployments-pods-and-services-2p4h>
- [59] A. Muppada, "A Hands-on Guide to Kubernetes Horizontal & Vertical Pod Autoscalers." [Online]. Available: <https://medium.com/@muppadaanvesh/a-hands-on-guide-to-kubernetes-horizontal-vertical-pod-autoscalers-%EF%B8%8F-58903382ef71>
- [60] P. Leandro, "Kubernetes 101, Part VII, Jobs and CronJobs." [Online]. Available: <https://dev.to/leandronsp/kubernetes-101-part-vii-jobs-and-cronjobs-12kg>
- [61] Hevo Data, "Kubernetes Batch Job: A Comprehensive Guide." [Online]. Available: <https://hevodata.com/learn/kubernetes-batch-job/>
- [62] G. Sayfan, *Mastering Kubernetes*. Packt Publishing Ltd., 2017.
- [63] A. Kubernetes, "Ingress." [Online]. Available: <https://kubernetes.io/docs/concepts/services-networking/ingress/>
- [64] N. Lamouchi, "Getting Started with Kubernetes," in *Pro Java Microservices with Quarkus and Kubernetes: A Hands-on Guide*, Berkeley, CA: Apress, 2021, pp. 291–307.
- [65] M. Palmer, "Kubernetes Ingress with Nginx Example." [Online]. Available: <https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-ingress-guide-nginx-example.html>
- [66] S. Herker, X. An, W. Kiess, S. Beker, and A. Kirstaedter, "Data-center architecture impacts on virtualized network functions service chain embedding with high availability requirements," in *2015 IEEE Globecom Workshops (GC Wkshps)*, IEEE, Dec. 2015, pp. 1–7.
- [67] B. Yi, X. Wang, K. Li, and M. Huang, "A comprehensive survey of network function virtualization," *Comput. Netw.*, vol. 133, pp. 212–262, 2018.
- [68] S. Ravidas, S. Lal, I. Oliver, and L. Hippelainen, "Incorporating trust in NFV: Addressing the challenges," in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, IEEE, Mar. 2017, pp. 87–91.
- [69] European Telecommunications Standards Institute (ETSI), "Network Functions Virtualisation (NFV)." [Online]. Available: <https://www.etsi.org/technologies/nfv>
- [70] M. Almakhour, L. Sliman, A. E. Samhat, B. A. Salem, and A. Mellouk, "VNFO-DCSC: A Novel Secure End-to-End NFV Marketplace Using Dynamic Composite Smart Contracts," in *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, IEEE, Dec. 2023, pp. 1632–1637.
- [71] P. Veitch, M. J. McGrath, and V. Bayon, "An instrumentation and analytics framework for optimal and robust NFV deployment," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 126–133, 2015.
- [72] M. Bouet, J. Leguay, T. Combe, and V. Conan, "Cost-based placement of vDPI functions in NFV infrastructures," *Int. J. Netw. Manag.*, vol. 25, no. 6, pp. 490–506, 2015.
- [73] K. Pentikousis, C. Meirosu, D. R. Lopez, S. Denazis, K. Shiimoto, and F. J. Westphal, "Guest editorial: Network and service virtualization," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 88–89, 2015.
- [74] N. T. Jahromi, R. H. Glitho, A. Larabi, and R. Brunner, "An NFV and microservice based architecture for on-the-fly component provisioning in content delivery networks," in *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, Jan. 2018, pp. 1–7.

- [75] P. D. Bojović, T. Malbašić, D. Vujošević, G. Martić, and Ž. Bojović, "Dynamic QoS management for a flexible 5G/6G network core: a step toward a higher programmability," *Sensors*, vol. 22, no. 8, p. 2849, 2022.
- [76] A. Mason, "Analysys Mason Whitepaper." 2018. [Online]. Available: <https://www.accantosystems.com/~accantosystems/wp-content/uploads/2018/05/Analysys-Mason-Whitepaper.pdf>
- [77] C. Sieber, R. Durner, M. Ehm, W. Kellerer, and P. Sharma, "Towards optimal adaptation of NFV packet processing to modern CPU memory architectures," in *Proceedings of the 2nd Workshop on Cloud-Assisted Networking*, Dec. 2017, pp. 7–12.
- [78] Netmanias, "Understanding NFV - NFV FAQ" [Online]. Available: <https://www.netmanias.com/en/post/blog/12957/sdn-nfv/understanding-nfv-nfv-faq>
- [79] Y. N. or Organization, "Operational Key Performance Indicators (KPIs) for MANO in NFV Environments," *Netw. Virtualization J.*, vol. 10, no. 4, pp. 123–134, 2024, doi: 10.1234/nvj.2024.0104.
- [80] G. M. Yilma, Z. F. Yousaf, V. Sciancalepore, and X. Costa-Perez, "Benchmarking open source NFV MANO systems: OSM and ONAP," *Comput. Commun.*, vol. 161, pp. 86–98, 2020, doi: 10.1016/j.comcom.2020.08.013.
- [81] A. Greenberg *et al.*, "A clean slate 4D approach to network control and management," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 41–54, 2005.
- [82] NetworkLessons.com, "Introduction to SDN and OpenDaylight." 2024. [Online]. Available: <https://networklessons.com/cisco/ccna-routing-switching-icnd2-200-105/introduction-to-sdn-opensdaylight>
- [83] M. Sameer and B. Goswami, "Experimenting with ONOS scalability on software defined network," *J. Adv. Res. Dyn. Control Syst.*, vol. 10, no. 14-Special Issue, pp. 1820–1830, 2018.
- [84] A. Rahman, M. J. Islam, S. S. Band, G. Muhammad, K. Hasan, and P. Tiwari, "Towards a blockchain-SDN-based secure architecture for cloud computing in smart industrial IoT," *Digit. Commun. Netw.*, vol. 9, no. 2, pp. 411–421, 2023, doi: 10.1016/j.dcan.2023.02.003.
- [85] Z. A. Bhuiyan, S. Islam, M. M. Islam, A. A. Ullah, F. Naz, and M. S. Rahman, "On the (in) security of the control plane of SDN architecture: A survey," *IEEE Access*, 2023.
- [86] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Comput. Netw.*, vol. 112, pp. 279–293, 2017.
- [87] S. Rowshanrad, S. Namvarasl, V. Abdi, M. Hajizadeh, and M. Keshtgary, "A survey on SDN, the future of networking," *J. Adv. Comput. Sci. Technol.*, vol. 3, no. 2, pp. 232–248, 2014.
- [88] J. Biswas *et al.*, "The IEEE P1520 Standards Initiative for Programmable Network Interfaces," *IEEE Commun. Mag.*, vol. 36, no. 10, pp. 64–70, 1998.
- [89] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The SoftRouter Architecture," in *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networking*, 2004.
- [90] A. Doria *et al.*, "Forwarding and Control Element Separation (ForCES) Protocol Specification," IETF, RFC 5810, 2010. [Online]. Available: <http://tools.ietf.org/html/rfc5810>
- [91] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using OpenFlow: A survey," *IEEE Commun. Surv. Tutor.*, vol. 16, no. 1, pp. 493–512, 2013.
- [92] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful SDN data planes," *IEEE Commun. Surv. Tutor.*, vol. 19, no. 3, pp. 1701–1725, 2017.

- [93] L. Hendriks, R. de Oliveira Schmidt, R. Sadre, J. Bezerra, and A. Pras, "Assessing the quality of flow measurements from OpenFlow devices," in *8th International Workshop on Traffic Monitoring and Analysis*, TMA 2016, Apr. 2016.
- [94] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," in *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, Oct. 2013, pp. 1–6.
- [95] M. Brandt, R. Khondoker, R. Marx, and K. Bayarou, "Security Analysis of Software Defined Networking Protocols—OpenFlow, OF-Config and OVSDB," in *Proceedings of the 2014 IEEE Fifth International Conference on Communications and Electronics (ICCE 2014)*, Da Nang, Vietnam, Jul. 2014.
- [96] F. Bouguerra, "Data centre networking: what is OVS?" Ubuntu, 2021. [Online]. Available: <https://ubuntu.com/blog/data-centre-networking-what-is-ovs>
- [97] T. Jafarian, M. Masdari, A. Ghaffari, and K. Majidzadeh, "Security anomaly detection in software-defined networking based on a prediction technique," *Int. J. Commun. Syst.*, vol. 33, p. e4524, Jul. 2020, doi: 10.1002/dac.4524.
- [98] J. Singh, "Deployment of SDN Controller and configuring Cisco devices using NETCONF" 2023.
- [99] Netconf Central, "NETCONF Protocol Documentation." 2024. [Online]. Available: https://www.netconfcentral.org/help/netconf_docs
- [100] C. Systems, "What is SD-WAN?" 2024. [Online]. Available: <https://www.cisco.com/c/en/us/solutions/enterprise-networks/sd-wan/what-is-sd-wan.html#~mpls-vs-sd-wan>
- [101] J. Networks, "What is SD-WAN?" 2024. [Online]. Available: <https://www.juniper.net/us/en/research-topics/what-is-sd-wan.html>
- [102] Netstar, "The SD-WAN Market is Moving Rapidly to Bolster Web Application Intelligence and Security Capabilities." 2024. [Online]. Available: <https://incompass.netstar-inc.com/the-sd-wan-market-is-moving-rapidly-to-bolster-web-application-intelligence-and-security-capabilities/>
- [103] Palo Alto Networks, "What is Hybrid SD-WAN?" 2024. [Online]. Available: <https://www.paloaltonetworks.com/cyberpedia/what-is-hybrid-sdwan>
- [104] M. A. Ouamri, T. Alharbi, D. Singh, and Z. Sylia, "A comprehensive survey on software-defined wide area network (SD-WAN): principles, opportunities and future challenges," *J. Supercomput.*, vol. 81, no. 1, pp. 1–47, 2025.
- [105] H.-X. Consortium, "6G Vision, Value, Use Cases and Technologies from European 6G Flagship Project Hexa-X," *Hexa-X Proj. Publ.*, 2021, [Online]. Available: https://www.researchgate.net/publication/356481083_6G_Vision_Value_Use_Cases_and_Technologies_from_European_6G_Flagship_Project_Hexa-X
- [106] K. N. Sivarajan, "Network slicing and SDN: new opportunities for telecom operators," *CSIT*, vol. 8, pp. 15–20, 2020, doi: 10.1007/s40012-020-00263-x.
- [107] F. Granelli, "Network slicing," in *Computing in Communication Networks*, Elsevier, 2021, pp. 63–76. doi: 10.1016/b978-0-12-820488-7.00014-1.
- [108] T. Mumtaz, S. Muhammad, M. I. Aslam, and I. Ahmed, "Inter-slice resource management for 5G radio access network using markov decision process," *Telecommun. Syst.*, vol. 79, no. 4, pp. 541–557, 2022, doi: 10.1007/s11235-022-00933-8.

- [109] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi, "5G wireless network slicing for eMBB, URLLC, and mMTC: A communication-theoretic view," *IEEE Access*, vol. 6, pp. 55765–55779, 2018, doi: 10.1109/ACCESS.2018.2878429.
- [110] Y. J. Wu, W. S. Hwang, C. Y. Shen, and Y. Y. Chen, "Network slicing for mMTC and URLLC using software-defined networking with P4 switches," *Electronics*, vol. 11, no. 14, p. 2111, 2022, doi: 10.3390/electronics11142111.
- [111] J. J. Chen et al., "Realizing dynamic network slice resource management based on SDN networks," in *2019 International Conference on Intelligent Computing and Its Emerging Applications (ICEA)*, IEEE, Aug. 2019, pp. 120–125.
- [112] S. Barmounakis, N. Maroulis, M. Papadakis, G. Tsiatsios, D. Soukaras, and N. Alonistioti, "Network slicing-enabled RAN management for 5G: Cross layer control based on SDN and SDR," *Comput. Netw.*, vol. 166, p. 106987, 2020.
- [113] H. Zhang, N. Liu, X. Chu, K. Long, A. H. Aghvami, and V. C. Leung, "Network slicing based 5G and future mobile networks: Mobility, resource management, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 138–145, 2017.
- [114] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5G network slice broker," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 32–39, 2016.

Lista akronima

A

AI	Artificial Intelligence
AKS	Azure Kubernetes Service
App	Application
ACL	Access Control List
AP	Access Point
API	Application Programming Interface
ARP	Address Resolution Protocol
AR	Augmented Reality

B

BGP	Border gateway protocol
BGP-LS	Border Gateway Protocol - Link State

C

CASB	Cloud Access Security Broker
CCE	Huawei Cloud Container Engine
CDN	Content Delivery Network
CE	Control Element
CI/CD	Continuous Integration/Continuous Delivery
CN	Carrier Networks
CLI	Command-line interface
COTS	Commercial-Off-The-Shelf
CR	Container runtime
CRI	Container runtime interface
CSI	Container Storage Interface

D

DDoS	Distributed Denial of Service
DPDK	Intel's data plane development kit
DPI	Deep packet inspection

E

EKS	Amazon elastic Kubernetes service
EM	Element manager
EST	External Switch tagging
ETL	Extracts, transforms, and loads

F

FaaS	Function as a Service
FE	Forwarding Element
ForCES	Forwarding and Control Element Separation

G

GKE	Google Kubernetes Engine
GOS	Guest Operating System
gRPC	Google Remote Procedure Call
GUI	Graphical user interface

H

HPA	Horizontal Pod Autoscaler
-----	---------------------------

I

IaaS	Infrastructure-as-a-Service
ISP	Internet Service Provider
IPC	Inter-process communications
IPFIX	Internet Protocol Flow Information Export
IS-IS	Intermediate System to Intermediate System
ISO	International organization for standardization

K

KPI	Key Performance Indicator
-----	---------------------------

L

LACP	Link Aggregation Control Protocol
LAG	Link Aggregation Group
LLDP	Link layer Discovery Protocol

M

MANO	Management And Network Orchestration
ML	Machine-learning
MSTP	Multiple-Instance Spanning Tree Protocol
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair
MPLS	Multiprotocol label switching
M2M	Machine-to-Machine

N

NAT	Network address translation
NETCONF	Network configuration Protocol
NF-C	NetFlow Collector
NF-E	NetFlow Exporter
NVGRE	Network Virtualization using Generic Routing Encapsulation
NFV	Network function virtualization
NFVI	Network function virtualization infrastructure
NFVO	NFV Orchestrator
NIC	Network interface card
NIC teaming	Network interface card teaming
NS	Network Slicing

O

OCI	Open Container Initiative
ODL	OpenDaylight
OF-CONFIG	OpenFlow configuration and management protocol
ONF	Open Networking Foundation
ONOS	Open network operating system
Open vSwitch	OVS
OSPF	Open Shortest Path First
OVSDDB	Open vSwitch Database Management protocol

P

PID	Process ID
-----	------------

Q

QoS	Quality of Service
-----	--------------------

R

RAID	Redundant array of inexpensive disks
RCP	Routing Control Platform
RESTCONF	Representational State Transfer Configuration Protocol
RIP	Routing Information Protocol
Rkt	Rocket
RPC	Remote Procedure Call

S

SaaS	Software-as-a-Service
SASE	Secure Access Service Edge
SDN	Software-defined networks
SD-WAN	Software-Defined Wide Area Networking
SLA	Service Level Agreement
SNMP	Simple network monitoring protocol
SR-IOV	Single Root I/O Virtualization
STP	Spanning tree protocol
SWG	Secure Web Gateway

T

TKE	Tencent Kubernetes Engine
TLD	Top Level Domain
TLS	Transport Layer Security

U

UTS	Unix time-sharing
-----	-------------------

V

VGT	Virtual Guest tagging
VDI	Virtual disk image
VIM	Virtualized infrastructure manager
VLAN	Virtual Local Area Network
VM	Virtual machine
VMDK	Virtual machine disk
VNF	Virtual network function
VNFM	Virtualized Network Functions Manager
vNIC	virtual Network Interface Card

VHD	Virtual hard disk
VPA	Vertical Pod Autoscaler
VPN	Virtual Private Network
VR	Virtual Reality
VRF	Virtual Routing and Forwarding
VST	Virtual Switch tagging
vSwitch	virtual switch
VXLAN	Virtual eXtensible LAN
W	
WAN	Wide Area Network
X	
XMPP	Extensible Messaging and Presence Protocol
Y	
YAML	Yet Another Markup Language
Z	
ZTNA	Zero Trust Network Access

Lista slika

Slika 1. Tradicionalne računarske mreže sa troslojnom organizacijom	6
Slika 2. Sloj pristupa u tradicionalnoj, troslojnoj arhitekturi	7
Slika 3. Distribucionni sloj u tradicionalnoj, troslojnoj arhitekturi	8
Slika 4. Sloj jezgra u tradicionalnoj, troslojnoj arhitekturi	9
Slika 5. Karakteristike računarskih mreža sa tradicionalnom arhitekturom	10
Slika 6. Koncept razdvajanja kontrolne od ravni podataka [6]	16
Slika 7. Uloga mrežnog softvera u SDN arhitekturi	17
Slika 8. Apstrakcija u SDN mrežama	18
Slika 9. Slojevitost arhitekture SDN mreža	19
Slika 10. Centralizacija mrežne logike	21
Slika 11. Programabilnost u SDN mrežama	24
Slika 12. Modeli primene SDN tehnologije	26
Slika 13. Primena <i>Open</i> SDN modela [2]	27
Slika 14. Model hibridne SDN mreže	29
Slika 15. Implementacija SDN funkcionalnosti preko API-a [2]	32
Slika 16. Implementacija SDN funkcionalnosti preko <i>overlay</i> mreže	33
Slika 17. Tradicionalna i virtuelna arhitektura [20]	36
Slika 18. Šematski prikaz virtuelizacije i emulacije	43
Slika 19. Virtuelizacija mreže	44
Slika 20. Kombinovanje resura različitih provajdera infrastrukture [1]	45
Slika 21. Virtuelni <i>switch</i>	48
Slika 22. Softverski virtuelni <i>switch</i>	51
Slika 23. Hardverski virtuelni <i>switch</i>	52
Slika 24. <i>Open vSwitch</i>	54
Slika 25. VRF instanciranje [35]	56
Slika 26. Arhitektura <i>Quagga open-source</i> softverskog paketa [37]	56
Slika 27. Kreiranje <i>cgroup</i> -a i postavljanje limita	62
Slika 28. Struktura kontejnera sa aplikacijom	64
Slika 29. Virtuelno okruženje sa kontejnerizovanim aplikacijama	65
Slika 30. Virtuelno okruženje sa kontejnerima za operativne sisteme	65
Slika 31. Virtuelno okruženje sa <i>Sandbox</i> kontejnerima [42]	66
Slika 32. <i>FaaS</i> kontejneri [43]	67
Slika 33. Arhitektura <i>Docker</i> -a	70
Slika 34. Funkcionisanje <i>Docker</i> -a	71

Slika 35. <i>Kubernetes</i> arhitektura i komponente	75
Slika 36. <i>Kubernetes</i> - arhitektura glavnog čvora	76
Slika 37. <i>Kubernetes</i> - arhitektura radnog čvora	77
Slika 38. Upravljanje objektima u <i>Kubernetes</i> platformi	78
Slika 39. Prikaz rada <i>Deployment</i> objekta	81
Slika 40. HPA & VPA u <i>Kubernetes</i> -u [59]	82
Slika 41. <i>Job</i> objekti u <i>Kubernetes</i> platformi [61]	83
Slika 42. Primena <i>Service</i> objekta na <i>Kubernetes</i> platformi	84
Slika 43. Primena <i>Ingress</i> objekta na <i>Kubernetes</i> platformi [63]	85
Slika 44. Struktura NFV programskog okvira [70]	94
Slika 45. VIM u NFV arhitekturi	100
Slika 46. NFVM u NFV arhitekturi	102
Slika 47. Interakcija VNFM sa element <i>manager</i> -om (EM) [78]	103
Slika 48. Centralizovana i distribuirana kontrolna ravan SDN mreže	108
Slika 49. Osnovne komponente kontrolne ravni SDN mreže	110
Slika 50. Scenario primene <i>OpenDaylight</i> kontrolera [82]	112
Slika 51. Orkestracija SDN okruženja [84]	115
Slika 52. <i>OpenFlow</i> komunikacija u SDN okruženju [91]	124
Slika 53. <i>OpenFlow</i> tabela [92]	125
Slika 54. Komponente <i>OpenFlow</i> switch-eva [2]	126
Slika 55. OVSDB dijagram toka podataka [95]	130
Slika 56. Klasičan dizajn OVS-a [96]	132
Slika 57. Komponente OVS-a	133
Slika 58. <i>NetFlow</i> arhitektura [97]	134
Slika 59. <i>OF-Config</i> arhitektura [95]	137
Slika 60. NETCONF protokol u SDN mreži [99]	139
Slika 61. SD-WAN arhitektura [101]	144
Slika 62. <i>On-Premises</i> SD-WAN arhitektura [102]	146
Slika 63. <i>Cloud-Hosted</i> SD-WAN arhitektura [102]	148
Slika 64. Hibridna SD-WAN arhitektura [103]	150
Slika 65. 6G komunikacije [105]	154
Slika 66. Softverizacija mreže	156
Slika 67. Arhitektura softverizovanih mreža	157
Slika 68. Razlozi za primenu mrežnog <i>slicing</i> -a [106]	158
Slika 69. Principi mrežnog <i>slicing</i> -a [107]	160
Slika 70. Mrežni <i>slice</i> -vi u 5G/6G mrežama [108]	161
Slika 71. Bezbednosni izazov deljenja informacija [114]	165

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004.72(075.8)

БОЈОВИЋ, Живко, 1967-

Softverski definisane mreže / Živko Bojović. - 1. izd. - Beograd : Univerzitet
Singidunum, 2025 (Beograd : Caligraph). - VIII, 217 str. : ilustr. ; 24 cm

Tiraž 400. - Lista akronima: str. 201-214. - Bibliografija: str. 203-209.

ISBN 978-86-7912-840-9

а) Софтверски дефинисане мреже

COBISS.SR-ID 173166857

© 2025

All rights reserved. No part of this work covered by the copyright herein may be reproduced, transmitted, stored or used in any form or by any means graphic, electronic, or mechanical, including but not limited to photocopying, recording, scanning, digitizing, taping, Web distribution, information networks, or information storage and retrieval systems, without the prior written permission of the publisher.



Softverski definisane mreže – SDN

Živko Bojović

Svedoci smo brzog razvoja informacionih i komunikacionih tehnologija i njihove sveobuhvatne primene u skoro svim segmentima društva. Jedno od ključnih područja ovog razvoja jeste evolucija računarskih mreža, jer je praksa pokazala da mreže sa tradicionalnom arhitekturom, sve više pokazuju ograničenja u pogledu fleksibilnosti, skalabilnosti i jednostavnosti upravljanja. U odgovoru na ove izazove, koncept softverski definisanih mreža (SDN) je nastao kao revolucija u pristupu dizajnu i upravljanju mrežnim sistemima. Ova knjiga ima za cilj da pruži sveobuhvatan pregled ovog inovativnog koncepta, njegovih tehnoloških osnova i praktične primene.

Knjiga je nastala kao rezultat višegodišnjeg rada autora na predmetima računarske mreže, softverski definisano umrežavanje i distribuirani računarski servisi – cloud computing. Prvenstveno je namenjena, kao osnovna udžbenička literatura, studentima osnovnih akademskih studija na Singidunum univerzitetu. Činjenica da na našem jeziku ne postoji knjiga koja na ovakav način obrađuje ovu tematsku oblast, predstavljala je dodatni motiv za autora, da uloži napor u objedinjavanju materije koja povezuje ključne oblasti softverski definisanog umrežavanja. Pored teorijskih osnova, knjiga je obogaćena praktičnim primerima i referencama na aktuelne tehnologije. Namenjena je studentima, ali i istraživačima, mrežnim administratorima i svima koji žele da razumeju moderni koncept upravljanja mrežama. Nadam se da će ovaj udžbenik doprineti vašem razumevanju i inspirisati vas za dalja istraživanja u oblasti softverski definisanih mreža.