UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI FAKULTET

ZORAN G. ČIČA

IMPLEMENTACIJA FUNKCIJA PAKETSKOG PROCESIRANJA U INTERNET RUTERIMA VELIKOG KAPACITETA

Doktorska disertacija

Beograd, 2012.

BELGRADE UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ZORAN G. ČIČA

IMPLEMENTATION OF PACKET PROCESSING FUNCTIONS IN HIGH CAPACITY INTERNET ROUTERS

PhD Thesis

Belgrade, 2012.

MENTOR:

dr Aleksandra Smiljanić, vanredni profesor, Beogradski Univerzitet, Elektrotehnički fakultet

ČLANOVI KOMISIJE:

dr Miodrag Popović, redovni profesor, Beogradski Univerzitet, Elektrotehnički fakultet
dr Žarko Markov, naučni savetnik, Institut Iritel
dr Milan Bjelica, docent, Beogradski Univerzitet, Elektrotehnički fakultet

DATUM ODBRANE:

IMPLEMENTACIJA FUNKCIJA PAKETSKOG PROCESIRANJA U INTERNET RUTERIMA VELIKOG KAPACITETA

REZIME:

Internet predstavlja jedan od najvažnijih temelja razvoja modernog društva i učestvuje u svim aspektima svakodnevnog života - poslovnom, socijalnom, zabavnom, edukativnom itd. Internet je postigao globalni uspeh zahvaljujući svojoj robusnosti i mogućnosti da povezuje različite tehnologije u jednu međusobno povezanu mrežu. Osnovu arhitekture Interneta čine ruteri koji omogućavaju globalnu povezanost svih delova Internet mreže. Pošto ruteri čine osnovnu gradivnu jedinicu Interneta, performanse i mogućnosti rutera imaju ogroman uticaj na kvalitet rada Internet mreže.

Broj Internet korisnika neprestano raste. Takođe, razvijaju se i nove aplikacije i servisi koji zahtevaju sve veće protoke, usled čega se u Internet mreži instaliraju linkovi sve većih kapaciteta. Kao posledica, količina saobraćaja na Internetu neprestano raste, pa samim tim Internet ruteri postaju sve opterećeniji, naročito u jezgru Internet mreže gde je saobraćaj najintezivniji. Internet ruteri moraju neprestano da se usavršavaju i unapređuju, da bi mogli veoma brzo obrađivati ogromne količine podataka. Dodatne otežavajuće okolnosti sa stanovišta obrade podataka u ruterima su potreba za uvođenjem mehanizama kvaliteta servisa i multikast saobraćaj koji je sve popularniji.

Mnogi istraživači i naučnici rade na unapređivanju funkcionalnosti rutera i razvoju novih rešenja i algoritama koji treba da omoguće efikasniji rad rutera. Međutim, velik problem u razvoju novih rešenja i unapređenja postojećih funkcija je zatvorenost rutera komercijalnih proizvođača pa samim tim razvijana rešenja se tipično ispituju zasebno bez potpune integracije sa svim funkcijama rutera. Ovakav način ispitivanja je nepotpun jer ne omogućava kompletan uvid u kvalitet rada novog rešenja u realnom okruženju. Da bi se izbegli navedeni problemi, razvojni tim pod vodstvom dr Aleksandre Smiljanić je u okviru projekta „Sistemska integracija Internet rutera“ podržanog od strane Ministarstva za Nauku i tehnološki razvoj Republike Srbije započeo razvoj prototipa Internet rutera. Konačni cilj projekta je bio razvoj komercijalnog proizvoda, međutim, pored ovog cilja namera je bila i da se obezbedi otvorena platforma istraživačima i studentima na kojoj bi mogli da proučavaju internu strukturu i arhitekturu rutera i da razvijaju i testiraju nova rešenja u realnom okruženju.

Internet ruteri se sastoje iz dve celine tj. ravni - ravan podataka i kontrolna ravan. Ravan podataka je zadužena za brzu obradu korisničkih paketa i stoga je implementirana hardverski. Kontrolna ravan je zadužena za komunikaciju sa okruženjem (drugim ruterima, administratorima itd.) i ona se implementira softverski radi veće fleksibilnosti. U okviru ove teze je razvijen deo ravni podataka koji vrši procesiranje korisničkih paketa.

U tezi je objašnjena uloga i funkcije ravni podataka i kontrolne ravni. Takođe je izvršena klasifikacija i analiza postojećih arhitektura rutera. U okviru projekta „Sistemska integracija Internet rutera“ je odlučeno da se razvija jednostepena arhitektura, pa je stoga odabrana arhitektura rutera sa baferima na ulazu jer je ona najskalabilnija i omogućava podršku velikom broju portova velikih brzina. Usvojeno je da prototip rutera radi sa gigabitskim eternet portovima pošto su oni bili najekonomičniji i najdostupniji u trenutku razvoja prototipa, a omogućavali su testiranje velikih kapaciteta.

Ravan podataka se sastoji od paketskih procesora i komutatora (paketskog sviča). U okviru ove teze je izvršena implementacija i analiza paketskih procesora. Prvo su navedene i objašnjene sve funkcije koje paketski procesor mora da izvrši. Paketski procesori su implementirani na FPGA čipovima, pri čemu se u okviru jednog FPGA čipa smeštaju četiri paketska procesora. FPGA čipovi su izabrani za razvoj jer imaju visoke performanse, a njihova konfiguracija se može lako modifikovati što ih čini veoma pogodnim za razvoj. Izvršena je podela funkcija paketskog procesiranja na logičke celine tzv. module. Ulazni modul sloja linka vrši obradu eternet okvira na ulazu u ruter. Ulazni mrežni modul obrađuje IP pakete koji su izdvojeni iz eternet okvira u ulaznom modulu sloja linka. Lukap modul vrši IP lukap tj. pretragu lukap tabele radi određivanja na koji izlaz rutera IP paket treba da se prosledi. SGS modul, na osnovu naprednog SGS algoritma, vrši raspoređivanje ćelija za slanje kroz komutator ka izlazima rutera. Ćelije imaju fiksnu dužinu i dobijaju se segmentacijom IP paketa u ulaznom mrežnom modulu. DDR2 kontroler je modul koji obezbeđuje baferisanje ćelija koje čekaju svoj trenutak za prosleđivanje ka izlazu u DDR2 SDRAM memoriju koja se koristi zbog svog velikog kapaciteta. SERDES modul vrši konverziju paralelnog u serijski prenos i obrnuto radi ostvarivanja brze (multigigabitske) komunikacije sa komutatorom. Izlazni mrežni modul vrši rekonstrukciju IP paketa iz ćelija na izlazu

rutera, a izlazni modul sloja linka vrši formiranje eternet okvira za slanje na izlazni link rutera.

Svi navedeni moduli su kreirani koristeći VHDL programski jezik i implementirani na FPGA čipove. U razvijenom prototipu rutera je napravljena štampana ploča koja sadrži osam paketskih procesora raspoređenih na dva FPGA čipa i komutator raspoređen na jedan FPGA čip. Izvršeno je testiranje razvijenih i implementiranih paketskih procesora, pri čemu je testiranje izvršeno i u realnom okruženju gde se prototip rutera povezivao sa ruterima kompanije Cisco, kao i softverskim ruterima na Linux platformi. Implementirane funkcije paketskog procesiranja predstavljaju najvažniji deo ravni podataka i omogućavaju jasan uvid u način funkcionisanja rutera. Realizovane funkcije se mogu lako modifikovati i unapređivati, a takođe se mogu dodavati i nove funkcije čime razvijeni paketski procesori dobijaju na istraživačkom, ali i edukativnom značaju. Implementirani paketski procesori predstavljaju prvi originalan doprinos ove teze.

Pored same implementacije izvršena je i analiza svih funkcija paketskog procesora da bi se utvrdila potencijalna uska grla u skalabilnosti rutera koje se ogleda u mogućnosti povećanja broja portova i brzine portova, tj. samog kapaciteta rutera. Analizom je utvrđeno da su najkritičnije funkcije baferisanje ćelija i IP lukap. Baferisanje ćelija je kritično sa stanovišta zahtevanih protoka i dimenzije bafera koji rastu sa povećanjem broja i brzine portova. IP lukap postaje kritičan sa porastom brzine portova kao i neizbežnom tranzicijom na duže IPv6 adrese.

U drugome delu teze je izvršena klasifikacija postojećih lukap algoritama koji definišu rad IP lukapa. Postoje tri klase lukap algoritama - algoritmi zasnovani na strukturi stabla, TCAM algoritmi i algoritmi bazirani na heširanju. Sve tri klase su detaljno opisane i analizirane, sa posebnim osvrtom na najpoznatije predstavnike svake klase. Izložena klasifikacija predstavlja značajnu pomoć budućim istraživačima u ovoj oblasti.

U tezi je takođe dat predlog tri nova lukap algoritma koji predstavljaju originalan doprinos ove teze koji je rezultirao objavljivanjem radova u časopisima sa SCI liste, kao i međunarodnim konferencijama iz ove oblasti. Predloženi lukap algoritmi su detaljno opisani i analizirani. Utvrđeno je da su predloženi lukap algoritmi veoma efikasni i pogodni za hardversku implementaciju jer ekonomično troše hardverske resurse.

Izvršena je implementacija predloženih lukap algoritama na FPGA čip, čime je potvrđena njihova hardverska ekonomičnost. Predloženi lukap algoritmi podržavaju velike IPv4 i IPv6 lukap tabele, i ostvaruju visok protok IP lukapa čime su podržani portovi velikih brzina. Veoma je važno naglasiti da se predloženi novi lukap algoritmi mogu integrisati u bilo koji ruter. Takođe, predloženi lukap algoritmi su upoređeni sa već postojećim lukap algoritmima i utvrđeno je da predloženi lukap algoritmi imaju minimalne memorijske zahteve koji predstavljaju hardverski resurs koji implementacije lukap algoritama najviše troše. Predloženi lukap algoritmi imaju znatno bolje performanse u slučaju velikih IPv6 lukap tabela u odnosu na postojeće lukap algoritme, što je od ogromnog značaja s obzirom na neminovnu potpunu tranziciju Interneta na duže IPv6 adrese.

KLJUČNE REČI:

Internet, ruter, paketski svič, ravan podataka, paketsko procesiranje, FPGA, VHDL, IP lukap

NAUČNA OBLAST:

Telekomunikacije i elektronika

UŽA NAUČNA OBLAST:

Arhitektura svičeva i rutera

UDK BROJ:

621.3

IMPLEMENTATION OF PACKET PROCESSING FUNCTIONS IN HIGH CAPACITY INTERNET ROUTERS

ABSTRACT:

Internet is one of the most important parts of the modern society. It participates in all aspects of everyday's life - business, social, entertainment, education etc. Internet achieved global success thanks to its robustness and internetworking between various technologies. Routers enable Internet's global connectivity and thus represent the foundation of the Internet. As routers are the main components of the Internet, their performances and capabilities have great impact on Internet quality performances.

The number of Internet users continuously grows. New applications and services that demand high throughput are constantly developed, and as consequence higher capacity links are installed. The Internet traffic continuously grows, so Internet routers are more and more loaded with traffic, especially in the Internet core, where Internet traffic is most intensive. Therefore, Internet routers must be always upgraded to support high speed processing of large amount of the Internet traffic. QoS mechanisms and multicast traffic represent additional difficulties in the future router development.

Many researchers and scientists are involved in router development process that includes development of new solutions and algorithms that enable more efficient router performances. However, the main problem in the development process is the closed router architecture in routers of commercial companies, thus developed solutions are tested without complete integration with the rest of the router functions. This leads to incomplete development and testing. To avoid aforementioned problems, research team led by Aleksandra Smiljanić started Internet router prototype development in the project „System integration of the Internet router" supported by the Serbian Ministry of Science. The main goal of the project was development of the commercial router. Also, very important goal was development of the open source platform for researchers and students that would be used for the education purposes, as well as the research purposes where new solutions could be tested in the real environment.

Internet routers contain two planes - data plane and control plane. Data plane is implemented in hardware and is responsible for fast IP packet processing. Control plane is implemented in software and is responsible for communication with router's

environment (neighbor routers, administrators and etc.). In this PhD thesis IP packet processors are developed and implemented. IP packet processors represent the most important part of the data plane.

The role and functions of data plane and control plane are given in this PhD thesis. Also, classification and analysis of the existing router architectures is given. As the decision was to implement router prototype based on the single stage architecture, router architecture based on input buffers was chosen as this architecture is the most scalable among the single stage architectures. Router prototype supports gigabit ethernet ports. Gigabit ethernet ports were chosen as they were the most economical choice at the time when the router prototype development started. Gigabit ethernet ports enable the testing of high capacities in the developed router prototype.

Data plane contains packet processors and crossbar. Implementation and analysis of packet processors is given in this thesis. First, the list of all packet processing functions is given and each function is explained. Packet processors are implemented on FPGAs, where four packet processors fit on one FPGA. FPGA chips are selected as they have very high performances and can be reconfigured which makes them very attractive in the research process. Packet processing functions are grouped in modules. Link layer input module processes incoming ethernet frames. Network input module processes IP packets extracted from incoming ethernet frames. Lookup module performs IP lookup where lookup table is examined to determine output port to which the IP packet should be forwarded. SGS module, based on the advanced SGS scheduling algorithm, schedules cells for the transmission to output ports via crossbar. Cells have fixed size and are result of the IP packet segmentation in the network input module. DDR2 controller is the module that stores cells in buffers. The buffers are implemented in DDR2 SDRAM memory as these memories have very large storage capacities. SERDES module performs serial to parallel conversion and vice versa in its communication to crossbar over high capacity internal links. Output network module reassembles IP packets, while output link layer module encapsulates reassembled IP packets in the ethernet frames for the transmission over the output link.

All aforementioned modules are developed using the VHDL programming language and implemented on the FPGA chips. Printed board has been designed and it contains eight packet processors implemented on two FPGA chips and crossbar

implemented on one FPGA chip. Implemented packet processors were tested using the real environment that included Cisco routers and Linux based software routers. As previously mentioned, implemented packet processors are the most important part of the data plane. Implemented packet processors can be modified and upgraded, and new functions can also be added. Thus, developed packet processors can be used for research and education purposes. The developed packet processors represent the first original contribution of this thesis.

All packet processing functions are analyzed to determine which functions are potential bottlenecks in the router scalability. Analysis shows that the most critical functions are cell buffering and IP lookup. Cell buffering becomes critical when the number of ports or capacity of ports is increased because needed buffer throughput or size can become too large for practical implementation. IP lookup becomes critical when the port speed is increased and also when the IP addresses becomes longer which is inevitable due to the transition to longer IPv6 addresses.

Classification of existing lookup algorithms is given in this thesis. There are three classes of the lookup algorithms - tree based lookup algorithms, TCAM lookup algorithms and hash based lookup algorithms. A detailed description and analysis is given for all three classes of the lookup algorithms. Also, the most popular lookup algorithms in every class are thoroughly described and analyzed. The provided classification represents significant help for any researcher in this area.

Three new lookup algorithms are proposed in this thesis. The proposed lookup algorithms represent the original contribution of this thesis that resulted in several publications in renowned magazines and conferences. A detailed description and analysis of the proposed lookup algorithms is given. The proposed lookup algorithms are hardware efficient as they economically use hardware resources. All proposed lookup algorithms are implemented on FPGA chips. The FPGA implementation shows that the proposed lookup algorithms frugally use FPGA resources and thus are suitable for hardware implementation. The proposed lookup algorithms support very large IPv4 and IPv6 lookup tables and achieve very high lookup throughput and thus support high speed ports. It is important to notice that the proposed lookup algorithms can be integrated in any router. The proposed lookup algorithms are also compared with the existing lookup algorithms. The comparison shows that the proposed lookup algorithms

have minimal memory requirements. Memory requirements represent the most used hardware resource by lookup algorithms. The proposed lookup algorithms significantly outperform existing lookup algorithms in the case of very large IPv6 lookup tables, which is important property as the transition to longer IPv6 addresses is inevitable.

KEYWORDS:

Internet, router, packet switch, data plane, packet processing, FPGA, VHDL, IP lookup

SCIENTIFIC AREA:

Communications and electronics

SCIENTIFIC SUBAREA:

Switch and router architecture

UDC NUMBER:

621.3

SADRŽAJ

1. UVOD

Internet je postao nezaobilazan element svakodnevnog života, pa se u razvijenim zemljama čak svrstava u elementarna ljudska prava. Ljudi svakodnevno koriste Internet u različite svrhe: poslovne, zabavne, društvene itd. Moderno poslovanje je nezamislivo bez upotrebe Interneta u cilju pružanja informacija o samoj firmi i njenim delatnostima, ostvarivanju kontakata sa poslovnim partnerima, pružanja usluga korisnicima poput e-kupovine itd. S druge strane Internet značajno učestvuje i u društvenom i zabavnom životu. Socijalne mreže su doživele ogroman napredak i omogućavaju jednostavno povezivanje ljudi. Takođe, Internet omogućava pristup raznovrsnom multimedijalnom sadržaju, pri čemu multimedijalni striming sadržaji dobijaju sve više na popularnosti i značaju poput IPTV. Pored toga, Internet se posmatra i kao 'živa' enciklopedija koja omogućava pronalaženje raznovrsnih informacija od interesa. Na osnovu svega navedenog očigledno je da je Internet nosilac današnjeg informacionog društva i integralni deo svakodnevnog života. Otuda je neophodno raditi na daljem usavršavanju Internet tehnologije i omogućavanju još kvalitetnijih karakteristika i usluga Internet mreže.

Osnovu Internet infrastrukture čine ruteri koji omogućavaju globalnu povezanost i integrisanost mreža različitih tehnologija i karakteristika. Ruteri vrše usmeravanje IP paketa tako da oni stignu do željenog odredišta. Ali, da bi izvršili osnovnu funkciju, ruteri implementiraju niz funkcija koje doprinose kvalitetu i raznovrsnosti usluga Internet mreže. S obzirom da broj Internet korisnika i dalje raste, kao i sam Internet saobraćaj, linkovi sve većih kapaciteta se postavljaju u Internet mrežu. Isto tako pored tradicionalnog saobraćaja poput veb surfovanja, čitanja mejlova, transfera fajlova i sl., danas je sve zastupljeniji i multimedijalni saobraćaj u realnom vremenu koji ima potpuno drugačije zahteve u pogledu kvaliteta servisa koji Internet mreža treba da pruži. Multimedijalne aplikacije zahtevaju veoma male varijacije kašnjenja, ograničena kašnjenja s kraja na kraj, a u slučaju prenosa kvalitetnog video zapisa i visoke protoke. Takođe i multikast saobraćaj postaje sve prisutniji pri čemu on donosi nove probleme u implementaciji rutera jer značajno više opterećuje interne resurse rutera klasičnih

arhitektura koje su uglavnom dizajnirane za podršku unikast saobraćaja nemajući u vidu podršku za multikast saobraćaj. Stoga su moderni ruteri veoma složeni uređaji koji moraju da podržavaju velike kapacitete linkova, a samim tim i veliku količinu saobraćaja koji prolazi kroz njih, pri čemu ne smeju da postanu usko grlo daljeg razvoja Interneta.

U okviru projekta ‘Sistemska integracija Internet rutera’ podržanog od strane Ministarstva za nauku i tehnološki razvoj Srbije, razvijen je prototip Internet rutera velikog kapaciteta. U okviru rada na razvoju prototipa bilo je neophodno implementirati ravan podataka rutera kroz koju bi se usmeravali korisnički IP paketi. Paketski procesori implementiraju funkcije ravni podataka. U njima se vrši prijem i ispitivanje paketa, raspoređivanje paketa za slanje ka izlaznim portovima, kao i samo slanje paketa na izlazni link rutera. Ova teza predstavlja rezultate rada na razvoju paketskog procesora u okviru razvijanog prototipa rutera. U ovaj razvoj su uključena i originalna rešenja koja prevazilaze ograničenja postojećih algoritama.

U okviru ove teze je razvijen i implementiran paketski procesor koji se koristi u okviru prototipa Internet rutera i on predstavlja prvi doprinos ove teze. Implementacija je izvršena na FPGA (*Field Programmable Gate Array*) čipovima pošto oni omogućavaju laku modifikaciju i unapređenje dizajna, kao i dodavanje dodatnih funkcionalnosti paketskom procesoru. Trenutna verzija prototipa rutera ima osam gigabitskih eternet portova. Pored omogućavanja pravilnog rada samog prototipa, implementirani paketski procesor ima još dva bitna aspekta - edukativni i istraživački. Naime, implementirani prototip omogućava studentima lakše razumevanje rada delova rutera, ali i mogućnost implementacije pojedinih funkcionalnosti paketskog procesora i njihovo testiranje u realnom okruženju, odnosno na razvojnoj platformi koju čine paketski procesor i prototip rutera. Takođe je bitan i istraživački aspekt. Razvijeni prototip rutera predstavlja otvorenu platformu koja omogućava domaćim istraživačima razvoj i implementaciju naprednih funkcionalnosti paketskog procesora i njihovo testiranje u realnom okruženju. Pri tome je omogućen fokus samo na pojedine funkcionalnosti paketskog procesora pošto bi razvijeno rešenje trebalo samo integrisati sa (postojećim) ostatkom paketskog procesora, pa nije neophodan razvoj kompletnog paketskog procesora da bi se ispitao samo jedan njegov deo.

Cilj istraživačkog tima koji je radio na razvoju prototipa rutera je i podrška za portove velikih brzina (10Gb/s i većih). Otuda je u okviru teze izvršeno i ispitivanje potencijalnih uskih grla u proširenju kapaciteta rutera tj. njegove skalabilnosti. Utvrđeno je da IP lukap i realizacija paketskih bafera predstavljaju najkritičnije tačke u povećanju brzine paketskog procesora. Prema lukap algoritmu se određuje za svaki IP paket na koji izlazni port treba da se usmeri, i zatim se ti paketi čuvaju u baferima dok čekaju na prosleđivanje ka izlaznim portovima rutera. U drugom delu teze je izvršena klasifikacija postojećih lukap algoritama. Takođe, detaljno su opisani i analizirani najpoznatiji predstavnici svake klase. Izložena klasifikacija i analiza predstavljaće veliku pomoć budućim istraživačima lukap algoritama. Potom je dat predlog novih lukap algoritama koji predstavljaju originalni doprinos ove teze koji je rezultirao objavljivanjem nekoliko naučnih radova u renomiranim časopisima i konferencijama. Predloženi lukap algoritmi su detaljno opisani i matematički analizirani. Utvrđeno je da predloženi lukap algoritmi troše minimalne hardverske resurse što je i potvrđeno njihovom hardverskom implementacijom. Takođe, korišćenjem tehnika paralelizacije i pajplajna je postignut visok protok predloženih lukap algoritama koji u najgorem slučaju iznosi 64Gb/s, pri čemu je važno naglasiti da bi ova brzina u najgorem slučaju bila i veća ako bi se koristili integrisani čipovi boljih performansi. Izvršena je i komparativna analiza predloženih lukap algoritama sa postojećim lukap algoritmima koja je pokazala velike prednosti predloženih lukap algoritama, naročito u pogledu velikih IPv6 lukap tabela, što je od posebnog značaja imajući u vidu neminovan prelaz Interneta na duže IPv6 adrese.

Teza je organizovana u šest poglavlja. Posle uvoda, u drugom poglavlju je dat opis strukture rutera i pregled arhitektura rutera što je neophodno da bi se bolje shvatila pozicija paketskog procesora u ruteru. U trećem poglavlju je izložena implementacija paketskog procesora namenjena i upotrebljena u razvijenom prototipu rutera, pri čemu su opisani i analizirani svi delovi paketskog procesora. U četvrtom poglavlju je objašnjena uloga lukap algoritma i klasifikovani su postojeći lukap algoritmi pri čemu su analizirani najpoznatiji predstavnici svake klase. U petom poglavlju su predloženi novi lukap algoritmi pri čemu je data njihova analiza kao i rezultati implementacije. Takođe su predloženi lukap algoritmi upoređeni sa postojećim lukap algoritmima. Šesto poglavlje predstavlja zaključak teze u kom su data završna razmatranja.

2. ARHITEKTURA RUTERA

Ruteri predstavljaju osnovne mrežne elemente Interneta koji omogućavaju globalnu povezanost Internet mreže. Osnovna funkcija rutera jeste usmeravanje IP paketa kroz Internet do njihovog konačnog odredišta. Međutim, iako naizgled ruteri obavljaju jednostavnu osnovnu funkciju oni su veoma složeni uređaji. Da bi uopšte mogli da vrše svoju osnovnu funkciju usmeravanja IP paketa, neophodno je da poznaju topologiju mreže u kojoj se nalaze. Otuda je neophodno da ruteri implementiraju protokole rutiranja koji omogućavaju automatsko i dinamičko učenje rutera o topologiji mreže u kojoj se nalaze i na osnovu koje mogu da prave svoje lukap tabele tj. tabele usmeravanja koje su neophodne za obavljanje funkcije usmeravanja paketa.

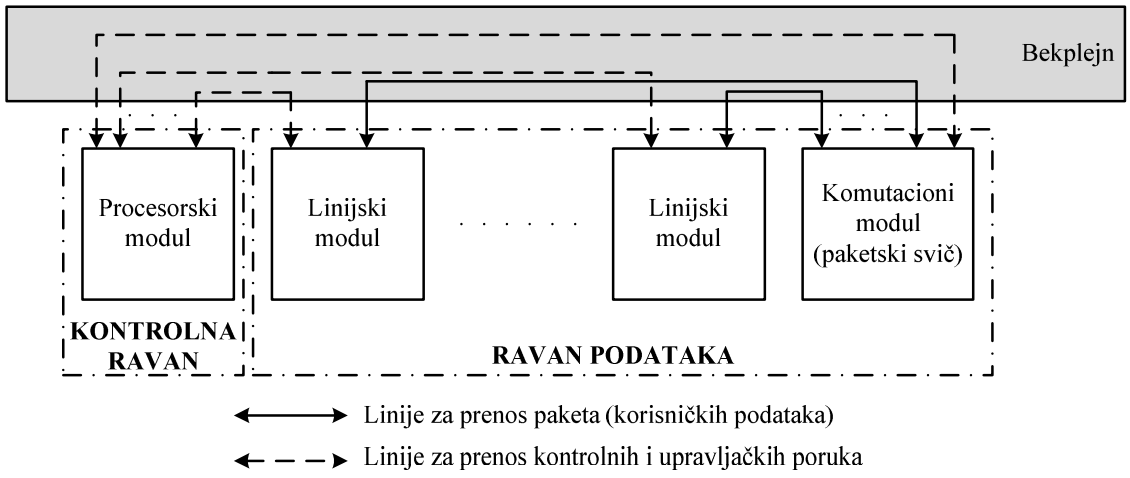
Pošto ruteri predstavljaju osnovnu komponentu Interneta koja je neophodna za njegovo funkcionisanje, neophodno je uspostaviti mehanizme zaštite i bezbednosti rutera od raznih zlonamernih napada koji mogu negativno da utiču na rad rutera, a samim tim i na rad Internet mreže, pa ruteri moraju da implementiraju i mehanizme autentifikacije, autorizacije, šifrovanja podataka, filtriranja paketa i drugih mehanizama zaštite. Važno je i da ruteri uvek budu raspoloživi tj. da vreme ispada iz rada rutera bude što manje, stoga je važno uspostaviti i kvalitetne mehanizme nadgledanja, pri čemu je važno omogućiti i udaljeni nadzor i pristup ruteru. S obzirom na veoma raznovrstan saobraćaj po pitanju zahteva u pogledu kvaliteta servisa, ruter mora da podrži i mehanizme za ostvarivanje različitih nivoa kvaliteta servisa, poput uvođenja prioriteta pri opsluživanju, rezervacije kapaciteta i sl. Kapaciteti linkova takođe rastu, pri čemu ruteri mogu sadržati i velik broj portova, pa je neophodno da ruter ima velik kapacitet tako da može da prosleđuje veoma velike količine paketa u jedinici vremena sa svojih ulaznih na svoje izlazne portove. Pored ovih navedenih osobina i zahteva koje ruter mora da poseduje i podrži, postoji i niz drugih, te je očigledno da je ruter veoma složen uređaj.

Slika 2.1. – Funkcije kontrolne ravni i ravni podataka rutera

S obzirom na složenost rutera, napravljena je podela rutera na dve ravni – ravan podataka i kontrolnu ravan [1]-[7]. Ova podela je načinjena na osnovu poslova koje ruter mora da izvrši. Na slici 2.1 su prikazane funkcije koje izvršavaju ravan podataka i kontrolna ravan. S jedne strane je potrebno da se obavi usmeravanje korisničkih paketa velikog protoka kao osnovna funkcija rutera. S druge strane je neophodan velik skup kontrolnih protokola neophodnih za ispravno i kvalitetno obavljanje osnovne funkcije rutera. Kontrolni protokoli podrazumevaju razmenu kontrolnih poruka čiji je protok znatno manji. Istovremeno rezultat kontrolnih protokola se često aplicira na sve portove. Zbog toga se razdvajaju ravan podataka koja obuhvata obradu korisničkih paketa, i kontrolna ravan koja podrazumeva obradu kontrolnih podataka. Zbog svojih različitih priroda, ravan podataka se implementira u hardveru, a kontrolna ravan u softveru. Naime, protokoli kontrolne ravni se izvršavaju na procesoru opšte namene. Na ovaj način kontrolna ravan postaje laka za održavanje i unapređivanje, a funkcije kontrolne ravni koje ruter treba da podrži se lako dodaju kao novi procesi operativnog sistema ili aplikacije koje se izvršavaju koristeći operativni sistem rutera koji je podržan na centralnom procesoru opšte namene.

Proizvođači rutera, kao i neki autori, često dodatno razlikuju kontrolnu ravan i ravan menadžmenta [8]-[11]. One se izvršavaju u okviru istog operativnog sistema rutera, tako da sa stanovišta fizičke pozicije ove dve ravni u ruteru nema nikakve razlike, a jedina razlika je u skupu funkcija. Kontrolna ravan se odnosi na mehanizme i funkcije kontrole rada ravni podataka poput protokola rutiranja, mehanizama kvaliteta

servisa, rezervacije resursa, multikast podrške i dr. Menadžment ravan se odnosi na funkcije upravljanja ruterom poput konfigurisanja rada rutera, nadgledanja rada rutera, alarmiranja u slučaju neispravnosti, omogućavanja udaljenog pristupa ruteru i dr.



Slika 2.2. – Struktura rutera velikog kapaciteta

Tipična struktura rutera velikog kapaciteta je prikazana na slici 2.2. Sledeći moduli sačinjavaju rutere velikog kapaciteta [1]-[7]:

- linijski moduli
- procesorski moduli
- komutacioni moduli
- bekplejn

Linijski i komutacioni moduli pripadaju ravni podataka, dok procesorski moduli pripadaju kontrolnoj ravni. Linijski moduli sadrže ulazne/izlazne portove i implementiraju prva tri sloja OSI referentnog modela u ravni podataka i izvršavaju ranije navedene funkcije ravni podataka. Linijskih modula tipično ima više pošto ruteri velikih kapaciteta implementiraju veliki broj portova. Dizajn rutera je tipično modularan i skalabilan, da bi se prilagodio različitim saobraćajnim opterećenjima. Komutacioni modul izvršava prosleđivanje paketa sa ulaznih na odgovarajuće izlazne portove. Često se ovaj modul označava i kao krosbar modul, paketski svič ili komutaciona matrica (*switching fabric*). U praksi se najčešće koristi termin paketski svič. Procesorski modul sadrži procesor na kome se postavlja operativni sistem rutera i koji izvršava funkcije kontrolne ravni. Uobičajeno, ruteri sadrže samo jedan procesorski modul, ali ih može biti i više ukoliko kapacitet rutera to zahteva. Svi moduli se hardverski prave u vidu kartica koje se povezuju na bekplejn. Bekplejn ili bekpanel predstavlja štampanu ploču

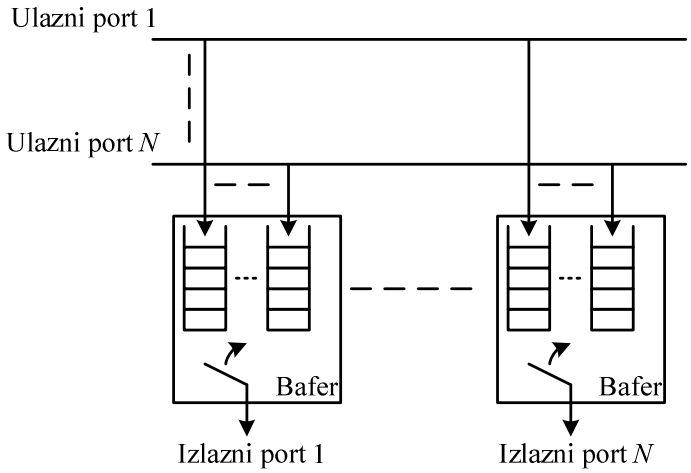
koja definiše pozicije kartica (pozicije gde se utiskuju linijski moduli, a gde procesorski i komutacioni) i koji vrši međusobno povezivanje modula. Na ovaj način je omogućen modularan i skalabilan dizajn rutera, čime je olakšano da operateri koji instaliraju rutere mogu lako da prilagode dotične rutere svojim potrebama i takođe mogu veoma lako da ih nadograđuju i unapređuju.

2.1. Tipovi arhitektura rutera

Ova teza se bavi paketskim procesiranjem u ruteru, koje se implementira u ravni podataka. Paketsko procesiranje se odnosi na obradu korisničkih paketa u ravni podataka u koje između ostalog spadaju provera ispravnosti paketa, ispitivanje IP zaglavlja i njegova modifikacija, filtriranje paketa, odbacivanje paketa, raspoređivanje paketa za slanje i dr. Samo paketsko procesiranje će biti detaljno opisano u sledećem poglavlju. Međutim, sa stanovišta implementacije paketskog procesiranja neophodno je poznavati arhitekturu rutera. Pod arhitekturom rutera se podrazumeva način na koji se paketi prosleđuju sa ulaznih portova na izlazne. Arhitektura rutera dominantno utiče na performanse i kvalitet rada rutera. Loša arhitektura može dovesti do slučajeva blokade. Pod blokadom se podrazumeva da u slučaju određenih saobraćajnih obrazaca dođe do zagušenja u određenim delovima rutera i usled toga odbacivanja paketa iako nijedan izlazni port nije preopterećen. Pod pojmom preopterećenja se podrazumeva da zbirni saobraćaj sa svih ulaznih portova namenjen određenom izlaznom portu prevazilazi njegov kapacitet. Očigledno je da u slučaju preopterećenja izlaznog porta mora doći do odbacivanja paketa jer izlazni kapacitet dotičnog porta ne može da opsluži sve pakete. Međutim, neke arhitekture rutera mogu biti blokirajuće čime i u regularnim uslovima kada nijedan izlazni port rutera nije preopterećen ipak dolazi do odbacivanja. Takođe, veoma je važno da arhitektura rutera bude skalabilna tako da podržava širok raspon broja portova. U suprotnom, ako arhitektura nije skalabilna to predstavlja potencijalni problem za operatera koji bi koristio takve rutere jer bi bio ograničen u proširenju broja portova na takvim ruterima, što bi značilo da bi morali da se koriste dodatni ruteri na lokacijama koje se trebaju proširiti dodatnim portovima što predstavlja skupo i komplikovano rešenje.

Deo u ruteru koji fizički vrši komutaciju paketa sa ulaznih na izlazne portove se naziva paketski svič. Arhitektura rutera je definisana načinom rada paketskog sviča. Predloženo je i razvijeno nekoliko realizacija paketskog sviča [7]:

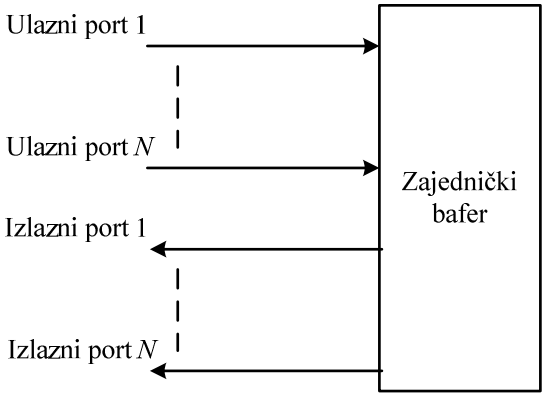
- paketski svičevi sa baferima na izlazu [12]-[15]
- paketski svičevi sa zajedničkim baferom [16]-[18]
- paketski svičevi sa baferima na ulazu [1]-[2], [19]-[24]
- *Birkhoff-Von Neumann* paketski svičevi [25]-[26]
- torusni paketski svičevi [27]
- Klosovi paketski svičevi [28]-[30]



Slika 2.1.1. – Paketski svič sa baferima na izlazu

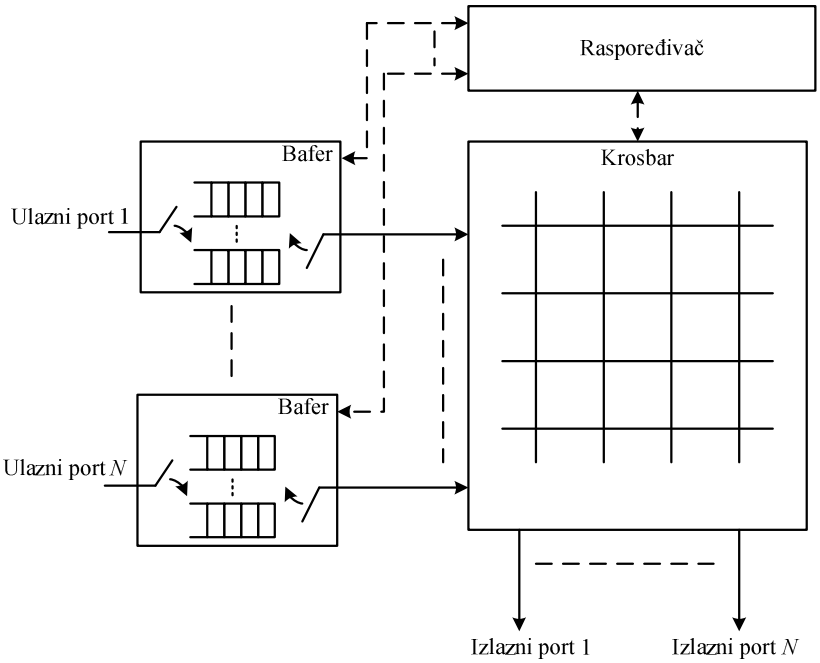
Paketski svič sa baferima na izlazu je prikazan na slici 2.1.1. Paket sa ulaznog porta se upisuje u bafer izlaznog porta kome je namenjen. Međutim, istovremeno može da postoji više ulaznih portova koji šalju pakete ka istom izlazu. Otuda bafer na izlaznom portu mora da bude dovoljno brz tj. da ima dovoljno velik protok da primi sve te pakete. U najgorem slučaju svi ulazni portovi istovremeno šalju paket ka istom izlaznom portu. Otuda bafer na izlaznom portu mora da ima dovoljan protok da bude sposoban da prima pakete sa svih N ulaznih portova istovremeno i da iščitava jedan paket koji se šalje na izlazni link. Očigledno je da protok bafera predstavlja ograničenje u pogledu skalabilnosti ovakvog paketskog sviča, čime ovakvi svičevi nisu pogodni za implementaciju rutera velikih kapaciteta koji sadrže velik broj brzih portova. S druge strane, s obzirom da se svi paketi odmah prosleđuju na izlazni port bez zadrške, veoma je lako implementirati podršku za kvalitet servisa poput rezervacije kapaciteta, klasa

prioriteta u prosleđivanju, fer servisa u slučaju preopterećenja, pošto su samo paketi tokova koji izlaze na dati izlazni port u istom baferu.



Slika 2.1.2. – Paketski svič sa zajedničkim baferom

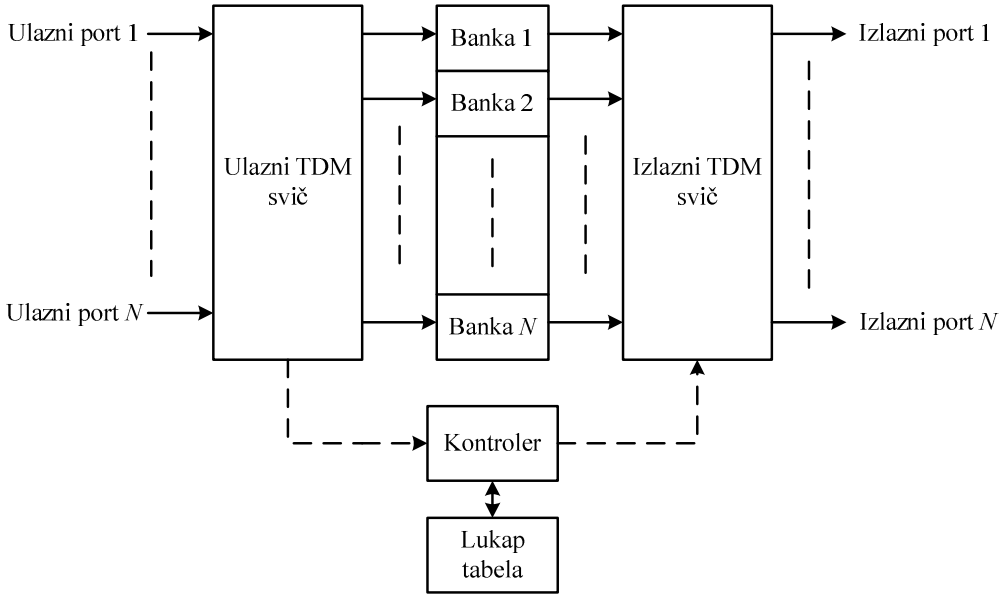
Paketski svič sa zajedničkim baferom prikazan na slici 2.1.2. pokušava da optimizuje memorijske resurse u odnosu na paketski svič sa baferima na izlazu, a da pri tom zadrži iste osobine kao i paketski svič sa baferima na izlazu. Veličina bafera na izlaznom portu u slučaju paketskog sviča sa baferima na izlazu mora da se dimenzioniše da podrži prijem paketa i u slučaju kratkotrajnih preopterećenja na izlaznom portu. Međutim, ne mogu istovremeno svi izlazni portovi biti preopterećeni pa samim tim memorijski resursi nisu optimalno korišćeni jer su neki baferi prepunjeni, a neki su samo delimično popunjeni. Samim tim je efikasnije koristiti memorijske resurse kada svi izlazni portovi dele jedan zajednički bafer. Svi ulazni portovi upisuju svoje pakete u zajednički bafer, a izlazni portovi čitaju pakete koje šalju na izlazni link iz istog tog bafera. To znači da protok zajedničkog bafera mora da podrži istovremeni upis paketa sa svih N ulaznih portova, kao i iščitavanje paketa sa svih N izlaznih portova, tj. potreban je čak i veći protok nego u slučaju paketskog sviča sa izlazima jer postoji veći broj čitanja iz bafera zbog činjenice da svi izlazni portovi koriste isti bafer. To znači da i dalje postoji problem sa skalabilnošću. Takođe, dobre osobine u pogledu podrške za kvalitet servisa važe i za ovu arhitekturu, jer sa stanovišta izlaznih portova ništa nije izmenjeno tj. i dalje su paketi tokova za isti izlazni port izolovani od drugih paketa jer imaju svoj izlaz, a ulazni portovi bez zadrške prosleđuju pakete u zajednički bafer pa su oni odmah na raspolaganju.



Slika 2.1.3. – Paketski svič sa baferima na ulazu

Paketski svič sa baferima na ulazu prikazan na slici 2.1.3 rešava problem skalabilnosti koji imaju paketski svič sa baferima na izlazu i paketski svič sa zajedničkim baferom. Paketi se čuvaju u baferima na ulaznim portovima pre prosleđivanja ka izlaznim portovima. Sami paketi se prosleđuju ka izlaznom portu kroz komutator koji se naziva krosbar. Radom paketskog sviča sa baferima na ulazu upravlja algoritam raspoređivanja tj. raspoređivač. On definiše uparivanja ulaznih portova sa izlaznim portovima tako da ulazni port može da pošalje paket kroz krosbar ka izlaznom portu sa kojim je uparen. Krosbar i raspoređivač je lakše implementirati kada su paketi fiksne dužine [7]. Kod unikast krosbarova, svaki ulaz može biti povezan sa najviše jednim izlazom, i svaki izlaz može biti povezan sa najviše jednim ulazom. Pošto su IP paketi promenljive dužine, onda se na ulaznim portovima vrši segmentacija paketa na jedinice fiksne dužine koje se nazivaju ćelije, pa se zatim ćelije raspoređuju za slanje kroz krosbar, te se prema izračunatom rasporedu ćelije i prosleđuju kroz krosbar. Poznati algoritmi raspoređivanja su PIM (*Parallel Iterative Matching*) [19], iSLIP [20] i SGS (*Sequential Greedy Scheduler*) [22]-[24]. Protok bafera više ne ograničava protok rutera, jer baferi moraju da podrže samo istovremeni upis jedne ćelije i čitanje jedne ćelije. Otuda protok bafera ne zavisi od broja portova rutera, pa je ruter zasnovan na paketskom sviču sa baferima na ulazu skalabilniji od prethodne dve arhitekture. Međutim, pošto paketi svih tokova namenjenih istom izlaznom portu nisu odmah

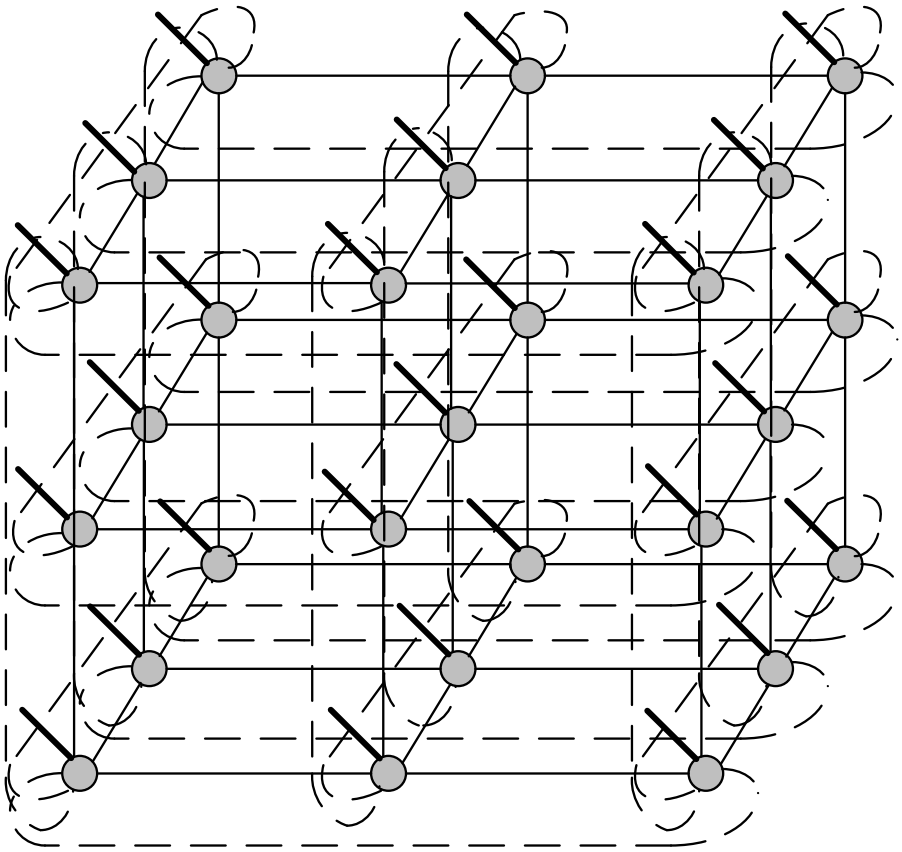
prisutni na izlaznom portu već se nalaze u baferima na ulaznim portovima, podrška kvalitetu servisa je znatno komplikovanija jer su paketi smešteni u različitim baferima. Naime, paketi se bore za resurse i sa paketima koji dolaze na isti ulaz, i sa paketima koji idu na isti izlaz. To je ujedno najveća mana paketskih svičeva sa baferima na ulazu. Sva tri navedena paketska sviča (sa baferima na izlazu, sa zajednički baferom, sa baferima na ulazu) spadaju u tzv. jednostepene paketske svičeve jer se fizički prenos paketa sa ulaznog porta na izlazni port vrši u jednom koraku. Paketski svičevi sa baferima na ulazu se smatraju najboljom jednostepenom arhitekturom jer podržavaju najveće kapacitete komutiranja [7].



Slika 2.1.4. – *Birkhoff-von Neumann* paketski svič

BN (*Birkhoff-von Neumann*) paketski svič je prikazan na slici 2.1.4. BN paketski svič koristi TDM (*Time Division Multiplex*) svič na ulazu i na izlazu. Zbog upotrebe TDM svičeva, na ulaznom portu se vrši segmentacija IP paketa na ćelije fiksne dužine. Kroz TDM svič na ulazu, ulazni portovi prosleđuju ćelije u memoriju (bafer) koja je podeljena na banke da bi se omogućio istovremeni upis više ćelija. Ulazni TDM svičevi omogućuju svakom ulaznom portu u svakom slotu pristup jednoj banci za upis ćelije, pri čemu svaki ulazni port u istom slotu pristupa različitoj banci. Kroz N slotova ulazni port će pristupiti svim bankama. Izlazni portovi odlučuju kad će da šalju neki paket na izlazni link i tada preko TDM sviča na izlazu čitaju ćelije dotičnog paketa. Isto kao i kod ulaznih TDM svičeva, izlazni TDM svičevi omogućuju svakom izlaznom portu u svakom slotu pristup jednoj banci za čitanje ćelije, pri čemu svaki izlazni port u istom

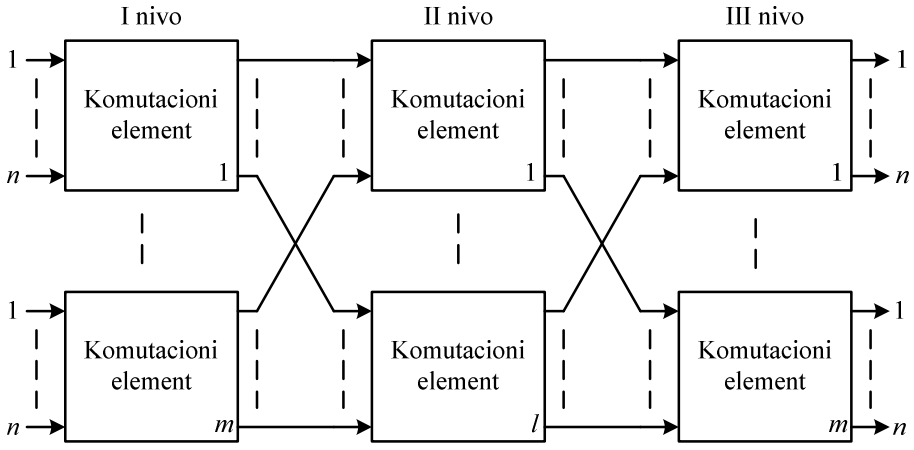
slotu pristupa različitoj banci. Kroz N slotova izlazni port će pristupiti svim bankama. Ideja BN sviča je da TDM svič na ulazu rasporedi ćelije svakog paketa ravnomerno po bankama, tako da izlaz putem izlaznog TDM sviča kojim pristupa svaki put drugoj banci uvek ima ćelije za čitanje i time neprestano prosleđuje pakete na izlazni link (dok ih naravno ima na raspolaganju za prosleđivanje). Međutim, veoma lako može da se konstruiše slučaj gde se sve ćelije sa svih ulaznih portova namenjene istom izlazu rasporede u istu banku, čime protok tog izlaznog porta drastično opada (preciznije N puta) jer izlazni port pristupa istoj banci tek u svakom N-tom slotu izlaznog TDM sviča. Ovo predstavlja ozbiljnu manu BN svičeva.



Slika 2.1.5. – Torusni paketski svič

Torusni paketski svičevi se baziraju na strukturi torusa kao što je prikazano na slici 2.1.5. Ideja je da svaka tačka u torusu predstavlja mali ruter koji ima šest internih portova i jedan eksterni port. Interni portovi omogućavaju konekciju sa susednih šest malih rutera kao što je prikazano na slici 2.1.5, a eksterni port predstavlja ulazno/izlazni port preko koga se ruter povezuje sa eksternom mrežom (Internetom). Na slici 2.1.5. eksterni portovi su prikazani sa debljim linijama, a konekcije između internih portova

tanjim linijama. Kada paket dođe preko eksternog porta u ruter, određuje se pozicija izlaznog porta u torusu i unapred se definiše putanja kojom paket mora ići kroz torus do dotičnog izlaznog porta. Kroz torusni paketski svič se prosleđuju jedinice fiksne dužine tzv. flitovi, pa je po ulazu IP paketa neophodno izvršiti njegovu segmentaciju na flitove. Za svaki par ulazni/izlazni port postoji više potencijalnih putanja kroz torus pa se često radi balansiranje saobraćaja po putanjama da bi se izbeglo potencijalno zagušenje putanja u torusu. U istu svrhu se nekad koristi i tehnika adaptivnog rutiranja koje se prilagođava trenutnom stanju saobraćaja u torusu, ali ono komplikuje implementaciju. U svakom slučaju, mogu se konstruisati slučajevi gde dolazi do zagušenja pojedinih deonica u torusu iako nijedan od izlaznih portova nije preopterećen. Zagušenje se u tim slučajevima može izbeći samo ako se značajno podignu kapaciteti svih deonica u torusu što je tehnički nepraktično, jer su u slučajevima velikog broja ulaznih/izlaznih portova kapaciteti potrebni za izbegavanje zagušenja preveliki. Takođe, problem je što u slučaju velikih torusa tj. torusa sa velikim brojem malih rutera, kašnjenje za neke parove ulazni/izlazni port može biti veoma veliko usled velikog broja deonica u torusu kroz koje paket mora proći. S druge strane, veoma dobro svojstvo rutera baziranih na torusnom paketskom sviču je velika skalabilnost i jednostavna proširivost, pošto je potrebno samo dodati mali ruter u torus kada se želi dodati novi port. U opštem slučaju je potrebno više koraka da paket sa ulaznog porta stigne na odgovarajući izlazni port, pa torusni paketski svič spada u višestepene paketske svičeve.



Slika 2.1.6. – Klosov paketski svič

Klosovi (eng. *Clos*) paketski svičevi se zasnivaju na trostepenoj strukturi. Klosova struktura podrazumeva da svaki komutacioni element bude povezan jednom

linijom sa svakim od komutacionih elemenata iz sledećeg nivoa (stepena). Klosov paketski svič je prikazan na slici 2.1.6. Ukupan broj ulaznih, odnosno izlaznih portova je $m \cdot n$, gde je m broj komutacionih elemenata u prvom nivou, odnosno trećem nivou, a n broj ulaza u jedan komutacioni element prvog nivoa, odnosno broj izlaza iz jednog komutacionog elementa trećeg nivoa. Ideja je da se koristi neblokirajuća Klosova struktura za postizanje neblokirajućeg paketskog sviča velikog kapaciteta. Naravno postoji razlika u odnosu na principe komutacije kola gde je originalno primenjena Klosova struktura. U komutaciji kola se za uspostavu veze nekog ulaza i izlaza, zauzimao put kroz Klosovu strukturu i ti resursi su bili zauzeti dok veza traje, nezavisno od trajanja drugih veza. Međutim, u Klosovoj strukturi sa komutacijom paketa resursi se zauzimaju u trajanju vremenskih slotova jednake dužine. U suprotnom, komutacija paketa promenljive dužine bi komplikovala konfiguraciju i upravljanje Klosovim paketskim svičem. Otuda se i ovde paketi na ulazu u ruter segmentiraju na ćelije fiksne dužine. U svakom slotu se vrši rekonfigurcija komutacionih elemenata Klosove strukture. Pokazano je da je Klosov paketski svič neblokirajući ako se primeni adekvatno balansiranje saobraćaja [28]. Broj komutacionih elemenata srednjeg nivoa l mora da zadovolji nejednačinu $l \geq n$ da bi Klosov paketski svič bio neblokirajući. Klosov paketski svič se koristi u slučaju kad je potrebno ostvariti najveće kapacitete paketskog sviča sa velikim brojem portova.

U ovoj tezi će biti korišćena jednostepena arhitektura jer je njena izrada ekonomičnija. Pri tome će biti korišćena arhitektura bazirana na paketskom sviču sa baferima na ulazu jer je ta arhitektura skalabilna i omogućava ostvarivanje velikih kapaciteta, odnosno podršku velikog broja portova. Za upravljanje radom paketskog sviča koristiće se SGS algoritam raspoređivanja zbog njegovih naprednih karakteristika i dokazane osobine neblokiranja [22]-[24].

3. FPGA IMPLEMENTACIJA PAKETSKIH PROCESORA

Kao što je istaknuto u uvodnom delu ove teze, Internet danas praktično spada u elementarna prava pojedinca u razvijenim društvima. Internet je nezamenjiv segment u svim aspektima današnjeg života, poslovnom, socijalnom, zabavnom, informativnom itd. Otuda dolazi do neprestanog razvoja novih servisa, pri čemu mnogi od njih imaju veoma striktne zahteve po pitanju kvaliteta servisa što zahteva značajne resurse Internet mreže. Usled razvoja novih zahtevnih servisa nastaje potreba da se kapaciteti infrastrukture Interneta unapređuju u cilju obezbeđivanja podrške novim servisima. Unapređenje kapaciteta se postiže povećanjem kapaciteta linkova kao i instalacijom novih linkova i rutera. Međutim, povećanje kapaciteta linkova dovodi do većeg opterećenja rutera koji sada moraju da obrađuju veće količine informacija u jedinici vremena da bi izvršili svoju osnovnu namenu – prosleđivanje informacija (IP paketa) ka krajnjem korisniku. Takođe, i multikast servisi postaju sve prisutniji. Kod multikast servisa, ruter mora da saobraćaj sa jednog svog ulaza distribuira na više svojih izlaza. To dovodi do potencijalnih problema zagušenja u ruteru ako arhitektura rutera nije prilagođena multikast saobraćaju, već samo unikast saobraćaju. Otuda arhitektura rutera mora da pruži kvalitetnu podršku multikast saobraćaju da ne bi dolazilo do zagušenja rutera i pada njegovih performansi [31]-[32].

U okviru projekta ‘Sistemska integracija Internet rutera’ podržanog od strane Ministarstva nauke i tehnološkog razvoja Srbije je razvijen Internet ruter u okviru kojega su implementirana napredna rešenja koja ispunjavaju zahteve modernih rutera velikih kapaciteta [5]-[6]. Jedan od delova projekta je bio implementacija i analiza funkcija paketskog procesora, i rezultati rada na tom delu projekta su prezentovani u okviru ove teze. Realizovani paketski procesori uz krosbar čine osnovu ravni podataka samog rutera. Za implementaciju ravni podataka su korišćeni FPGA čipovi umesto ASIC (*Application-Specific Integrated Circuit*) čipova. ASIC čipovi iako tipično daju bolje performanse i optimalnu implementaciju, nisu fleksibilni i potencijalne

modifikacije implementacije zahtevaju novi ASIC čip. Otuda ASIC čipovi nisu pogodni sa istraživačkog stanovišta gde je cilj implementirati raznovrsne algoritme i upoređivati njihove performanse. S druge strane FPGA čipovi nude visok stepen fleksibilnosti čime je moguće bez promene čipa vršiti promene dizajna implementacije i testirati raznovrsna rešenja.

Krajnji cilj rada na projektovanju i integrisanju Internet rutera je razvoj komercijalnog proizvoda. Međutim, pored komercijalnog aspekta, postoje i veoma značajni edukativni i istraživački aspekt. S jedne strane implementacija rutera omogućava studentima lakše razumevanje unutrašnje arhitekture rutera i njenog funkcionisanja. Takođe, studenti mogu da implementiraju delove rutera i testiraju ih u razvijenom prototipu čime stiču i praktična znanja. Sličan edukativni cilj imaju NetFPGA platforme [33]-[34]. U okviru NetFPGA projekta realizovana je otvorena platforma koja sadrži FPGA čip na kome se mogu implementirati raznovrsne mrežne funkcije i algoritmi, pri čemu se oni mogu testirati, kao i u našem slučaju, u realnom okruženju. Prednost našeg rešenja proističe iz činjenice da naš prototip sadrži krosbar za razliku od NetFPGA čime je omogućeno testiranje svih funkcija rutera sa ulaznim baferima. Pored edukativnog, značajni su i istraživački potencijali našeg prototipa. S obzirom da komercijalni ruteri nisu otvoreni, naročito na nivou hardverske ravni podataka, nemoguće je u okviru njih implementirati nova napredna rešenja i testirati ih. To značajno otežava razvoj novih algoritama i rešenja jer je teško izvršiti njihovo testiranje u realnim uslovima. Otuda naš razvijeni ruter dobija i na istraživačkom značaju, jer postavlja osnove u okviru kojih se nova rešenja lako dodaju i testiraju u realnom okruženju. Nova rešenja mogu da se dodaju kako u ravni podataka, tako i u kontrolnoj ravni. U ravni podataka je to omogućeno upotrebom FPGA čipova, a u kontrolnoj ravni korišćenjem otvorenog softvera koji se može modifikovati i unapređivati. Razvoj ravni podataka je prvi doprinos ove teze i on postavlja osnovu rutera, razvijenog u okviru projekta ‘Sistemska integracija Internet rutera’, koja omogućava prethodno navedene aspekte.

U ovom poglavlju biće prezentovana implementacija funkcija paketskog procesiranja koje uz krosbar čine kompletnu ravan podataka. Najpre će biti objašnjena arhitektura samog rutera i njegovo funkcionisanje da bi se bolje shvatila pozicija paketskog procesora u okviru rutera. Biće izložena opšta arhitektura paketskog

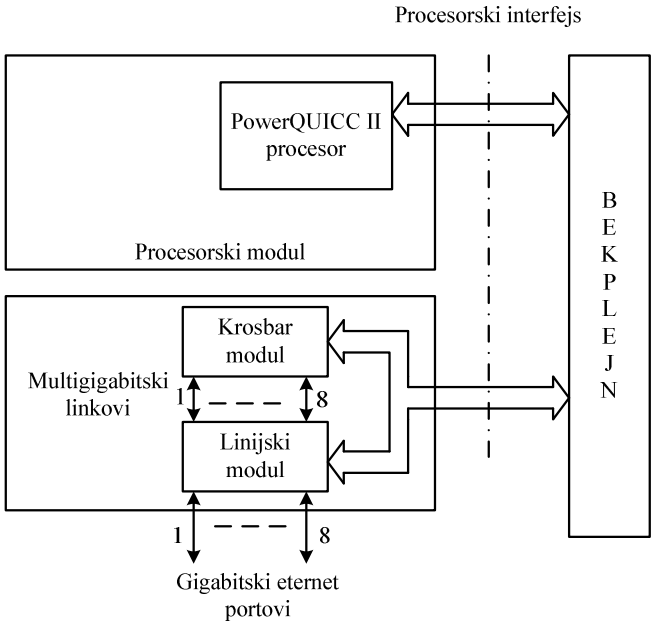
procesora, a zatim će biti objašnjeni i pojedini delovi paketskog procesora pri čemu će biti data i analiza njihove skalabilnosti. Na kraju će biti izloženi tehnički podaci o hardverskoj implementaciji i iskorišćenim resursima.

3.1. Arhitektura prototipa rutera

Ruteri velikih kapaciteta moraju imati modularnu strukturu da bi bili ekonomski isplativi kako proizvođaču tako i korisniku. Modularnom strukturom se omogućava bolja skalabilnost i veći kapacitet rutera, ali isto tako i fleksibilnost u pogledu konfiguracije koja bi zadovoljila specifične potrebe korisnika. Osnovni moduli rutera su linijski modul, procesorski modul i krosbar (komutacioni modul). Linijski moduli su najbrojniji i oni povezuju ruter sa njegovom okolinom (drugim ruterima i korisničkim mrežama), a njihova primarna funkcija je paketsko procesiranje. Tipično, proizvođači nude veći broj različitih linijskih modula radi mogućnosti povezivanja rutera na različite tipove mreže, kao što su eternet, POS (*Packet Over SDH*) i drugi. Na ovaj način korisnik može konfigurisati ruter prema svojim potrebama birajući one linijske module koji mu omogućavaju konekciju sa mrežama sa kojima ruter treba da se poveže. Procesorski modul koncentriše u sebi funkcije kontrolne ravni. On implementira protokole rutiranja, kao što su OSPF (*Open Shortest Path First*), BGP (*Border Gateway Protocol*), RIP (*Routing Information Protocol*), menadžment protokole kao što je SNMP (*Simple Network Management Protocol*), a omogućava i administraciju tj. konfigurisanje rutera od strane administratora mreže. Krosbar predstavlja komutator koji vrši prosleđivanje paketa sa ulaznih na odgovarajuće izlazne portove rutera.

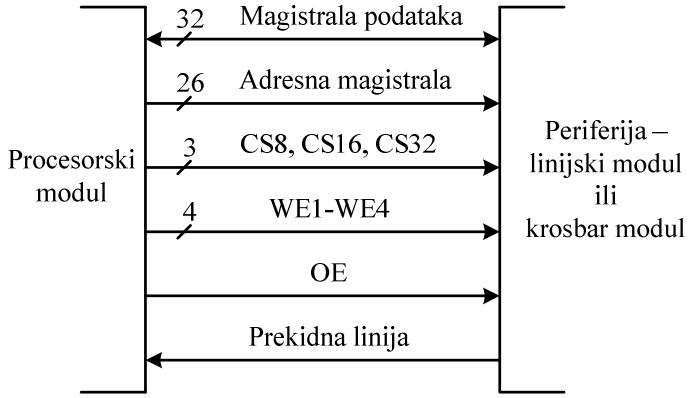
Prototip Internet rutera koga smo razvili sadrži pomenuta tri modula – procesorski modul, krosbar modul i linijski modul. Ova tri modula su raspoređena na dve štampane ploče. Na jednoj štampanoj ploči je procesorski modul. Jedan od učesnika na projektu je bio i Institut Iritel. Otuda je štampana ploča na kojoj je smešten procesorski modul standardna procesorska ploča koju Iritel koristi u svojim SDH (*Synchronous Digital Hierarchy*) sistemima. Pri tome je softverski deo ploče prilagođen funkcionalnostima Internet rutera i implementira protokole kontrolne ravni. Druga štampana ploča je razvijena u okviru projekta i sadrži krosbar modul i linijski modul. Ova varijanta je adekvatna za prvu verziju prototipa rutera jer je ekonomičnija, a sa stanovišta testiranja funkcionalnosti nema značajnih razlika. U budućim verzijama

prototip rutera će sadržati ova dva modula na odvojenim štampanim pločama, a takođe će biti i više linijskih modula radi ostvarivanja i testiranja većih kapaciteta. Obe štampane ploče se postavljaju u standardni rek koji se koristi u Iritelovim SDH sistemima, i povezane su bekplejnom. U okviru linijskog modula vrše se funkcije paketskog procesiranja u koje spadaju: detekcija eternet okvira na ulaznom portu, provera ispravnosti eternet okvira, filtriranje eternet okvira po MAC adresi i enkapsuliranom protokolu, provera ispravnosti IP zaglavlja, modifikacija IP zaglavlja, lukap, segmentacija IP paketa na ćelije fiksne dužine, raspoređivanje ćelija za slanje kroz krosbar, baferisanje ćelija, rekonstrukcija IP paketa na izlaznom portu, formiranje i slanje eternet okvira. Krosbar vrši prosleđivanje ćelija sa ulaznih na odgovarajuće izlazne portove. U okviru procesorskog modula se vrše funkcije kontrolne ravni i kontroliše se rad ravni podataka (npr. u lukap tabelu na ravni podataka prosleđuje nove podatke u slučaju promena u topologiji mreže). Otuda se svi kontrolni paketi (npr. OSPF paketi) namenjeni ruteru prosleđuju direktno sa ulaznih portova na linijskom modulu ka procesorskom modulu koristeći procesorski interfejs. Na isti način kontrolna ravan šalje svoje generisane kontrolne pakete u mrežu. Ona preko procesorskog interfejsa prosleđuje svoje kontrolne pakete ka odgovarajućem izlaznom portu linijskog modula koji ih zatim šalje na izlazni link tj. u mrežu. Uprošćena blok-shema razvijenog prototipa rutera je prikazana na slici 3.1.1.



Slika 3.1.1. – Struktura prototipa rutera razvijenog u okviru projekta 'Sistemska integracija Internet rutera'

Procesorski modul je sa hardverskog stanovišta identičan već postojećoj procesorskoj ploči koju koristi Iritel u svojim sistemima, a razlika je samo sa softverskog stanovišta, pošto je on prilagođen funkcijama Internet rutera. Konekcija sa linijskim i krosbar modulom je ostvarena procesorskim interfejsom, koji omogućava procesoru, upis i čitanje 32-bitnih podataka ka periferiji, odnosno linijskom i krosbar modulu. Na procesorskom modulu se nalazi PowerQUICC II procesor.

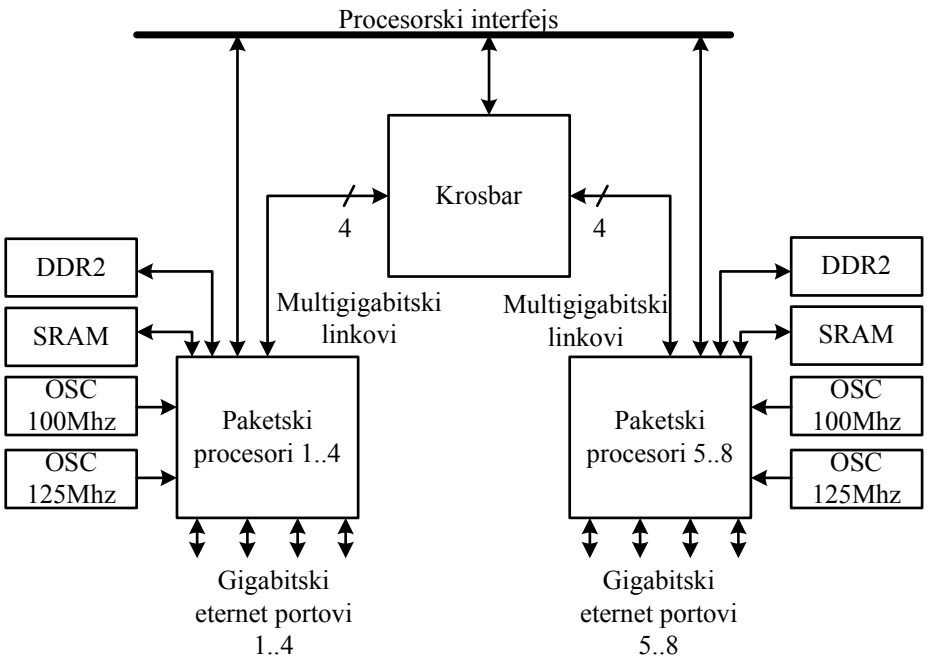


Slika 3.1.2. – Procesorski interfejs

Procesorski interfejs je prikazan na slici 3.1.2. On se sastoji od 32-bitne magistrale podataka, 26-bitne adresne magistrale, kontrolnih signala za selektovanje čipova, kontrolnih signala za aktiviranje upisa i čitanja i prekidne linije za izazivanje prekida u procesoru. Postoje tri kontrolna signala za selektovanje čipova u zavisnosti od karakteristika čipa (*CS8, CS16, CS32*) tj. tipa podataka koje podržava: 8-bitne, 16-bitne ili 32-bitne. Implementirani linijski i krosbar moduli podržavaju 32-bitni prenos. Ukoliko procesor upisuje podatak tada se aktiviraju kontrolni signali (*WE1-WE4*) za upis koji predstavljaju četiri signala za omogućavanje upisa – svaki za po jedan bajt 32-bitnog podatka na magistrali podataka. U slučaju čitanja, aktivira se kontrolni signal (*OE*) koji aktivira trostatičke bafere kojima se omogućava adresiranom modulu da izbaci tražene podatke na magistralu podataka ka procesoru na procesorskom modulu. Da bi se omogućilo krosbar modulu ili linijskom modulu signaliziranje nekog važnog događaja procesorskom modulu, koristi se prekidna linija čijom aktivacijom se izaziva pokretanje prekidne rutine u radu procesora. Pošto procesor pristupa većem broju modula tj. čipova, adresni prostor je podeljen po modulima. Otuda su u modulima kreirani adresni dekoderi koji prepoznaju da li ih je procesor adresirao ili ne. Ukoliko su adresirani onda znaju da procesor želi da ostvari komunikaciju sa njima. Takođe, na

procesorskom modulu je omogućena i funkcija udaljenog pristupa, pa tako administrator rutera može dinamički pristupati ruteru u cilju podešavanja ili modifikovanja konfiguracije rutera ili očitavanja statusa rutera. Ovakav način pristupa je realizovan preko eternet porta na procesorskom modulu.

Na slici 3.1.3 je prikazana blok-shema štampane ploče koja sadrži linijski i krosbar modul. Ona sadrži tri FPGA čipa (Xilinx Virtex5 LX110T čipovi [35]). Jedan FPGA čip se koristi za implementaciju krosbar modula, dok se druga dva koriste za implementaciju paketskih procesora u okviru linijskog modula. Pri tome svaki FPGA čip linijskog modula sadrži po četiri paketska procesora. Dobra osobina korišćenih FPGA čipova je što imaju četiri ugrađena eternet bloka čime je značajno olakšana implementacija funkcija paketskog procesiranja drugog sloja OSI modela. Svaki FPGA čip linijskog modula je povezan sa jednom SRAM (*Static Random Access Memory*) memorijom (Cypress SRAM konfiguracije 512Kx32b [36]) i jednom DDR2 SDRAM (*Double Data Rate Synchronous Dynamic Random Access Memory*) memorijom (Micron SODIMM 1GB DDR2-800 [37]). SRAM memorija sadrži lukap tabelu koja se koristi u okviru lukapa za određivanje izlaznog porta na koji treba usmeriti IP paket. DDR2 SDRAM memorija se koristi za baferisanje ćelija. Obe ove memorije koriste tj. dele sva četiri paketska procesora smeštena na istom FPGA čipu. Svi paketski procesori su povezani preko krosbara, tj. preko krosbara se prosleđuju ćelije sa ulaznih portova na odgovarajuće izlazne portove. Za povezivanje paketskih procesora i krosbara se koriste multigigabitski linkovi pošto su neophodne velike brzine prenosa ćelija između paketskih procesora i krosbara. I linijski i krosbar modul, odnosno FPGA čipovi u okviru kojih su oni implementirani, su povezani sa procesorskim modulom preko procesorskog interfejsa. Pošto je sistem digitalan neophodni su izvori takta tj. oscilatori da bi linijski modul i krosbar mogli ispravno funkcionisati. Na FPGA čipove koji implementiraju funkcije paketskog procesora su povezani oscilatori frekvencija 100MHz i 125MHz na osnovu kojih se generišu svi taktovi neophodni za ispravan rad ravni podataka. Oscilator od 100MHz se koristi za formiranje takta od 200MHz koji se koristi na fizičkom interfejsu ka DDR2 SDRAM memoriji. Oscilator od 125MHz se koristi za formiranje taktova od 125MHz koji se koriste u celokupnom paketskom procesoru izuzimajući već pomenuti fizički interfejs ka DDR2 SDRAM memoriji.



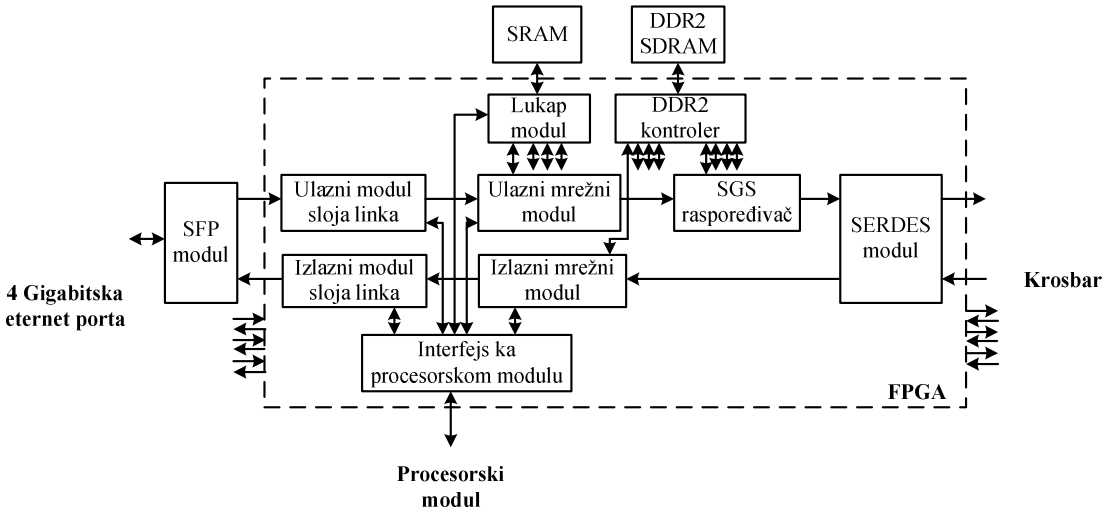
Slika 3.1.3. – Blok-shema štampane ploče koja sadrži linijski i krosbar modul

3.2. Implementacija paketskog procesora

Ovo potpoglavlje sadrži opis implementacije paketskog procesora u linijskom modulu prototipa rutera koga smo razvili. Radi povećanja fleksibilnosti i modularnosti implementacije korišćen je FPGA čip. Na ovaj način je omogućeno jednostavno menjanje dizajna implementacije i probanje različitih rešenja tokom razvoja funkcija paketskog procesora. Isto tako FPGA implementacija omogućava jednostavno dodavanje novih funkcija paketskog procesiranja u slučaju potrebe. Unutar ovog potpoglavlja će biti prezentovana arhitektura implementiranog paketskog procesora i biće dat pregled osnovnih funkcionalnosti koje paketski procesor treba da podrži. Nakon toga će u narednim odeljcima biti dat opis pojedinih delova paketskog procesora, pri čemu će biti izvršena analiza njihove skalabilnosti, odnosno mogućnosti da podrže povećanje kapaciteta linkova i broja portova rutera.

Osnovni zadatak ravni podataka je da prosledi (komutira) pakete sa ulaznih portova na odgovarajuće izlazne portove. Paketsko procesiranje se izvršava u okviru linijskih modula rutera i uključuje obradu paketa na fizičkom sloju, sloju linka podataka i mrežnom sloju, odnosno na prva tri sloja OSI modela. Na slici 3.2.1 je prikazana blok-shema implementacije paketskog procesora, pri čemu se na jedan FPGA čip smeštaju četiri paketska procesora koji odgovaraju gigabitskim eternet portovima. Fizički sloj se

tipično implementira primenom specijalizovanih čipova prilagođenih radu sa tehnologijom odgovarajuće mreže. Kao što se vidi sa slike 3.2.1 u ovoj implementaciji su korišćeni SFP (*Small Form-factor Pluggable Transceiver*) moduli koji vrše funkcije fizičkog sloja gigabitskog eterneta [38]. Takođe, sa slike 3.2.1 se može videti da se funkcije sloja linka podataka i mrežnog sloja vrše u okviru FPGA čipa.



Slika 3.2.1. – Blok-shema implementiranog paketskog procesora

Primljeni eternet okviri se obrađuju u okviru ulaznog modula sloja linka. Ulazni modul sloja linka detektuje okvire, proverava njihovu ispravnost, vrši filtriranje okvira na osnovu odredišne MAC adrese i enkapsuliranog protokola i izdvaja IP pakete iz okvira. Izdvojeni korisnički IP paketi se prosleđuju ulaznom mrežnom modulu. Kontrolni okviri i paketi se prosleđuju procesorskom modulu na obradu.

Mrežni sloj, za razliku od prva dva sloja, je tehnološki nezavisan što je uostalom i omogućilo razvoj i globalizaciju Interneta. Implementacija mrežnog sloja rutera u najvećoj meri utiče na njegove performanse, pa ona unosi najveći stepen slobode prilikom izgradnje rutera, odnosno njegove ravni podataka. Kao što se može videti sa slike 3.2.1 u implementaciju mrežnog sloja spadaju blokovi: ulazni mrežni modul, lukap modul, SGS raspoređivač, DDR2 kontroler, SERDES modul i izlazni mrežni modul.

Ulazni mrežni modul prihavata IP pakete od ulaznog modula sloja linka i obavlja sledeće funkcije: provera ispravnosti IP zaglavlja, ažuriranje TTL (*Time To Live*) polja i polja za proveru ispravnosti u IP zaglavlju, prosleđivanje odredišne IP adrese lukap modulu, segmentacija IP paketa u ćelije fiksne dužine, prosleđivanje ćelija SGS raspoređivaču zajedno sa informacijom kom izlaznom portu su ćelije namenjene.

Lukap modul izvršava lukap, odnosno vrši pretragu lukap tabele da bi odredio izlazni port na koji treba usmeriti IP paket sa odredišnom IP adresom za koju se vrši pretraga. Lukap tabela je smeštena u eksternu SRAM memoriju. Lukap modul koriste sva četiri porta tj. paketska procesora. Konačni rezultat lukapa se vraća ulaznom mrežnom modulu koji tu informaciju zajedno sa ćelijama odgovarajućeg paketa prosleđuje SGS raspoređivaču.

Raspoređivač vrši raspoređivanje ćelija za slanje ka odgovarajućim izlaznim portovima preko krosbara, pri čemu se ćelije smeštaju u ulazni bafer dok čekaju na svoj red za slanje. Raspoređivanje zavisi od arhitekture rutera. U našem slučaju, korišćena je arhitektura rutera sa baferima na ulazu jer ona omogućava najveću skalabilnost u slučaju jednostepenih rutera [7]. Pri tome je korišćen sekvencijalni pohlepni raspoređivač (SGS – *Sequential Greedy Scheduler*) za koji je matematički dokazano da je neblokirajući kad se koristi ubrzanje $u = 2$. Ubrzanje je koeficijent povećanja brzine internih linkova (koji spajaju paketske procesore sa krosbarom) u odnosu na eksterne linkove rutera (linkovi koji spajaju ruter sa mrežom) [7], [22]. Zbog toga SGS raspoređivač omogućava podršku za garanciju protoka i kašnjenja kroz ruter što je od velikog značaja u modernim ruterima s obzirom na povećanje udela saobraćaja koji ima striktne zahteve u pogledu kvaliteta servisa poput multimedijalnih aplikacija. Raspoređivač prima ćelije i informaciju ka kojem izlaznom portu treba proslediti paket, odnosno njegove ćelije, od ulaznog mrežnog modula. Raspoređivač vrši proračun redosleda slanja ćelija. Ćelije se smeštaju u ulazni bafer dok čekaju na svoj red za slanje ka odgovarajućem izlaznom portu preko krosbara. Ulazni bafer je smešten u DDR2 SDRAM memoriju koja ima veliki kapacitet i time mogućnost smeštanja velikog broja ćelija. DDR2 kontroler obezbeđuje pristup DDR2 SDRAM memoriji, odnosno obezbeđuje operacije upisa i čitanja. Pri tome sva četiri paketska procesora koriste, odnosno dele istu DDR2 SDRAM memoriju za svoje ulazne bafere.

Paketski procesori se povezuju preko krosbara koji vrši komutaciju ćelija sa ulaznih na odgovarajuće izlazne portove. Za ostvarivanje velikih brzina multigigabitskih linkova koristi se diferencijalni serijski prenos koji omogućava visoke protoke [39]. Otuda se u paketskim procesorima koriste SERDES (*SERializer/DESerializer*) blokovi koji vrše konverziju paralelnog prenosa ćelija u serijski prenos prilikom slanja, odnosno

obrnutu konverziju prilikom prijema ćelija u komunikaciji sa krosbarom. SERDES modul primljene ćelije prosleđuje izlaznom mrežnom modulu.

Na izlaznom portu se u okviru izlaznog mrežnog modula ćelije prikupljaju i smeštaju privremeno u izlazni bafer dok se ne prikupe sve ćelije jednog paketa. Kompletirani paketi (paketi čije su sve ćelije stigle na izlazni port) se u izlaznom mrežnom modulu rekonstruišu iz primljenih ćelija u originalne IP pakete koji se prosleđuju drugom sloju tj. izlaznom modulu sloja linka. Izlazni baferi sva četiri paketska procesora su kao i ulazni baferi smešteni u istu DDR2 SDRAM memoriju.

U izlaznom modulu sloja linka IP paketi se enkapsuliraju u okvire koji se prosleđuju SFP modulu za slanje na izlazni link.

Pošto je neophodna komunikacija sa kontrolnom ravni, odnosno procesorskim modulom implementiran je interfejs ka procesorskom modulu preko koga se razmenjuju informacije i podaci sa procesorskim modulom. Tako se kontrolni paketi namenjeni procesorskom modulu prosleđuju preko ovog interfejsa procesorskom modulu kao što su ARP (*Address Resolution Protocol*) zahtevi, OSPF paketi itd. U suprotnom smeru, procesorski modul šalje svoje kontrolne pakete u mrežu. Time je omogućen ispravan rad kontrolne ravni, odnosno komunikacija rutera u mreži. Npr. razmena OSPF paketa omogućava uspostavljanje susedstva sa drugim OSPF ruterima u mreži. Preko ovog interfejsa, procesorski modul može da ažurira lukap tabelu u slučaju promena topologije mreže. U okviru ažuriranja lukap tabele procesorski modul vrši brisanje postojećih prefiksa onih mreža koje ruteru više nisu dostupne i dodavanje novih prefiksa onih mreža koje su ruteru postale dostupne. Takođe u slučaju kada nekoj mreži postane povoljnije pristupiti preko nekog drugog izlaznog porta vrši se modifikacija izlaznog porta pridruženog odgovarajućem mrežnom prefiksu u lukap tabeli. Sve navedene promene u lukap tabeli se izvršavaju tako što procesorski modul šalje parove adresa-podatak lukap modulu preko interfejsa ka procesorskom modulu. Adresa označava lokaciju u SRAM memoriji na koju treba upisati dati podatak. Pored toga procesorski modul može slati komande za aktivaciju i deaktivaciju porta. Kad je port deaktiviran, ne šalju se i ne primaju se eternet okviri.

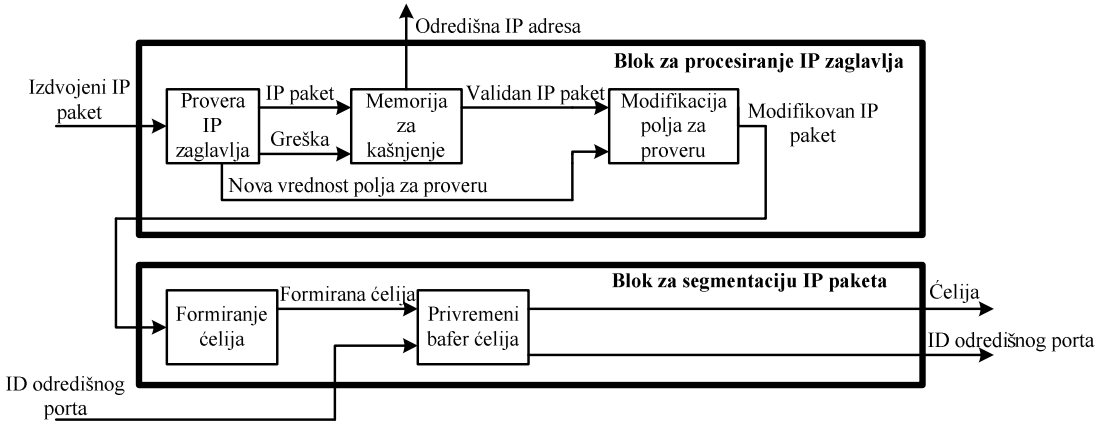
Nakon ovog opisa generalnog rada implementacije paketskog procesora u sledećim odeljcima će biti opisani delovi paketskog procesora. Pri tome će biti data i

analiza funkcija koje predstavljaju potencijalno usko grlo, odnosno mogu da ograniče povećanje broja portova i povećanje kapaciteta tj. brzina portova.

3.2.1. Ulazni modul sloja linka

Sloj linka podataka je implementiran koristeći ugrađeni MAC (*Media Access Control*) blok u okviru korišćenog FPGA čipa (Xilinx Virtex5 LX110T). Postoje ukupno četiri ugrađena MAC bloka, tako da svaki od četiri paketska procesora na FPGA čipu koristi po jedan. Ugrađeni MAC blok vrši funkcije MAC podsloja. U prijemnom (ulaznom) smeru u te funkcije spadaju određivanje početka okvira, odstranjivanje preambule i SFD (*Start Frame Delimiter*) polja iz okvira, provera ispravnosti okvira u prijemnom smeru, filtriranje okvira po odredišnoj MAC adresi. Kao rezultat rada ugrađenog MAC bloka u prijemnom smeru dobija se okvir iz koga su odstranjena polja koja ne nose informacije: preambula, SFD polje, FCS (*Frame Check Sequence*) polje. Ulazni modul sloja linka pored ugrađenog MAC bloka sadrži i blok koji vrši ispitivanje protokola koji je enkapsuliran unutar okvira i filtriranje okvira. U slučaju da protokol koji je enkapsuliran nije prepoznat, okvir se odmah odbacuje. U slučaju da je enkapsuliran IP protokol vrši se filtriranje odredišne IP adrese. Ovo je moguće pošto se unapred zna njena pozicija u eternet okviru. Ako je ova adresa jednaka nekoj od IP adresa u filtru (npr. u pitanju je IP adresa nekog od portova rutera, multikast IP adresa na koju se šalju OSPF '*Hello*' paketi i sl.), okvir se prosleđuje direktno ka procesorskom modulu preko interfejsa ka procesorskom modulu. U trenutnoj verziji prototipa, procesorski modul je konfigurisan za rad sa eternet okvirima. Otuda se procesorskom modulu moraju prosleđivati eternet okviri koji sadrže kontrolne pakete, a ne izdvojeni kontrolni paketi. Iz tog razloga se u ulaznom modulu sloja linka vrši preusmeravanje takvih eternet okvira ka procesorskom modulu. Naravno, filtriranje odredišne IP adrese i preusmeravanje IP paketa ka procesorskom modulu se može implementirati na nivou mrežnog sloja onog trenutka kada se omogući da procesorski modul prima direktno IP pakete. U slučaju da odredišna IP adresa nije u filtru, IP paket se izdvaja i prosleđuje ulaznom mrežnom modulu kao što je prikazano na slici 3.2.1. U slučaju drugih enkapsuliranih protokola, okviri se usmeravaju ka procesorskom modulu, pošto samo IP paketi treba da prolaze kroz ostatak ravni podataka. Ulazni modul sloja linka ne predstavlja potencijalno usko grlo, pošto ne zavisi od broja portova i obuhvata jednostavne funkcije.

3.2.2. Ulazni mrežni modul



Slika 3.2.2.1. – Struktura ulaznog mrežnog modula

Struktura ulaznog mrežnog modula je prikazana na slici 3.2.2.1 Osnovne funkcije ovog modula su provera ispravnosti IP zaglavlja, modifikacija IP zaglavlja (dekrementiranje TTL polja i usled toga upis nove vrednosti u polje za proveru) i segmentacija IP paketa u ćelije fiksne dužine. Modul je podeljen na dva bloka – blok za procesiranje IP zaglavlja i blok za segmentaciju IP paketa. Blok za procesiranje IP zaglavlja vrši deo funkcija ulaznog modula u kome se vrši ispitivanje i modifikacija IP zaglavlja, dok blok za segmentaciju IP paketa vrši formiranje ćelija fiksne dužine.

U bloku za procesiranje IP zaglavlja se IP paket kasni za trajanje zaglavlja, jer se u tom periodu vrši ispitivanje validnosti zaglavlja. Kašnjenje se vrši pomoću memorije za kašnjenje. Ukoliko je IP zaglavlje neispravno, IP paket se odbacuje. Prva provera ispravnosti je verifikacija polja verzije u IP zaglavlju. Pošto je trenutno podržana samo verzija 4 IP protokola, IP zaglavlje je neispravno ukoliko se u polju verzije nalazi bilo koja vrednost različita od 4. Druga provera ispravnosti je provera vrednosti TTL polja. Ukoliko TTL polje ima vrednost 0, IP paketu je isteklo tzv. vreme života i mora se odbaciti da bi se sprečilo njegovo potencijalno kruženje u mreži. Ukoliko TTL polje ima vrednost veću od 0, onda se ta vrednost dekrementira. Poslednja provera ispravnosti IP zaglavlja je provera ispravnosti polja za proveru. Ukoliko proračunata vrednost polja za proveru primljenog IP zaglavlja nije jednaka vrednosti koja se nalazi u polju za proveru IP zaglavlja, IP paket se odbacuje. Paket se odbacuje tako što se sadržaj neispravnog paketa upisanog u memoriju za kašnjenje briše iz nje, dok se ostatak paketa ignoriše. Dok se vrši provera IP zaglavlja, istovremeno se vrši i računanje nove vrednosti polja za proveru zbog izvršenog dekrementiranja TTL polja. Ukoliko je IP

zaglavlje ispravno vrši se njegovo čitanje iz memorije za kašnjenje i prosleđivanje ka modulu za modifikaciju polja za proveru. Ovaj modul vrši prosleđivanje IP paketa ka bloku za segmentaciju IP paketa, pri čemu umesto stare vrednosti polja za proveru upisuje novu. Nova vrednost polja za proveru se upisuje tek ovde jer se polje za proveru u IP zaglavlju ne nalazi na njegovom kraju. Takođe, kada se utvrdi validnost IP zaglavlja, vrši se prosleđivanje odredišne IP adrese lukap modulu koji treba da odredi na koji izlazni port treba usmeriti dotični IP paket. Pored odredišne IP adrese i druga polja IP zaglavlja se mogu izdvojiti na isti način u slučaju potrebe poput izvorišne IP adrese, ToS (*Type of Service*) polja i drugih.

Blok za segmentaciju IP paketa vrši formiranje ćelija fiksne dužine neophodne za efikasan rad SGS algoritma za raspoređivanje, i time omogućava komutiranje velikih kapaciteta [40]-[41]. Deo za formiranje ćelija prima IP paket i od njega formira ćelije pri čemu svakoj ćeliji dodaje zaglavlje. Zaglavlje ćelije sadrži: identifikaciju izvorišnog porta na kom je ćelija formirana, identifikaciju odredišnog porta na koji ćelija treba da se prosledi, bit indikacije prve ćelije paketa, i bit indikacije poslednje ćelije paketa. Identifikacija izvorišnog porta omogućava izlaznom portu da razlikuje tokove ćelija. Identifikacija odredišnog porta se koristi kao informacija za krosbar da zna na koji port treba da usmeri ćeliju. Na ovaj način je izbegnuto korišćenje posebnih linija ka krosbaru kojima bi se on konfigurisao, a koje bi predstavljale problem u slučaju velikog broja portova jer bi tada broj tih linija bio prevelik pa bi bilo problematično njihovo efikasno provođenje kroz bekplejn rutera, pošto bi tada krosbar bio na zasebnoj štampanoj ploči. Bitovi indikacije prve i poslednje ćelije IP paketa olakšavaju lakšu detekciju prve i poslednje ćelije paketa na izlaznom portu, a samim tim i lakše određivanje ćelija koje pripadaju datom paketu. Pored navedenih polja u zaglavlju ćelije potencijalno mogu da se dodaju i druga polja poput tipa servisa ukoliko je potrebno uvesti podršku za kvalitet servisa. U slučaju omogućavanja kvaliteta servisa, polje tip servisa bi se koristilo u kombinaciji sa identifikacijom izvorišnog porta za razlikovanje tokova po prioritetima za opsluživanje [40]-[41]. U trenutnoj verziji prototipa, tip servisa se ne stavlja u zaglavlje.

Zaglavlje je veličine 32 bita da bi se omogućila podrška velikog broja portova, ali i ostavila mogućnost za definisanje dodatnih polja. Veličina same ćelije treba biti pažljivo odabrana. Što je ćelija duža to je iskorišćenost internih linkova veća jer je

procenat udela zaglavlja ćelije manji. Takođe, krosbar ima više vremena za vršenje komutacije tj. periodi između uzastopnih konfigurisanja krosbara su duži pa je implementacija krosbara lakša. Međutim, ako su ćelije preduge, postoji opasnost slabog iskorišćenja ćelije u slučaju kratkih IP paketa jer bi samo mali deo ćelije nosio koristan sadržaj. S druge strane, ako su ćelije prekratke, onda je potencijalni problem komutacija takvih ćelija pošto bi vreme za komutaciju moglo biti prekratko za uspešnu implementaciju komutatora. Takođe bi tada procenat udela zaglavlja ćelije bio prevelik pa bi bila mala iskorišćenost internih linkova koji povezuju paketske procesore i krosbar. Izabrana veličina ćelije je 512b, koja pokušava da balansira navedena dva oprečna zahteva. Pošto se ćelije moraju proslediti SGS raspoređivaču sa informacijom o izlaznom portu kome su one namenjene, neophodno je sačekati rezultat lukap modula. Otuda se ćelije smeštaju u privremeni bafer. Čim se od lukap modula primi identifikacija izlaznog porta kome su ćelije namenjene vrši se prosleđivanje ćelija ka SGS raspoređivaču zajedno sa identifikacijom izlaznog porta kome su dotične ćelije namenjene.

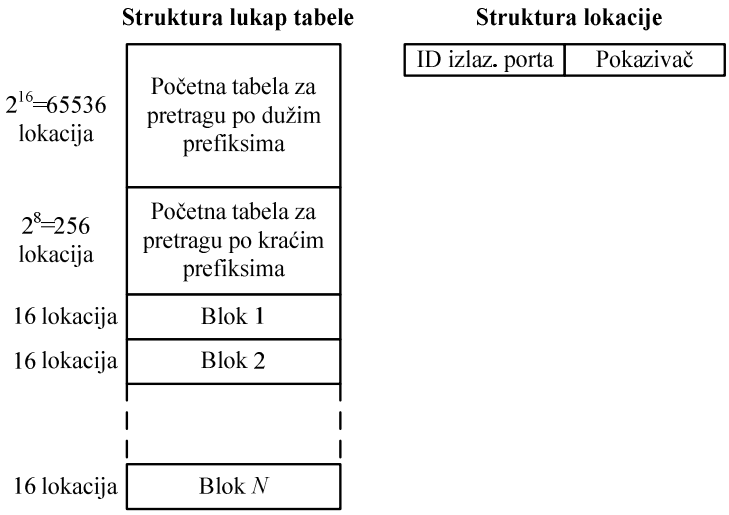
Sve navedene funkcije nisu kompleksne i ne troše mnogo vremena, a pri tome ne zavise od broja portova ili tokova, pa samim tim ulazni mrežni modul ne predstavlja potencijalno usko grlo u pogledu skalabilnosti rutera.

3.2.3. Lukap modul

Lukap jeste određivanje izlaznog porta na koji treba usmeriti IP paket na osnovu njegove odredišne IP adrese. Lukap tabela sadrži mrežne adrese i za svaku mrežnu adresu čuva identifikaciju (ID) izlaznog porta na koji treba treba usmeriti IP paket namenjen dotičnoj mreži. Pored mrežnih adresa u lukap tabeli mogu da se nalaze i agregacije mrežnih adresa koje se koriste radi smanjenja broja zapisa u lukap tabelama. Mrežne adrese i agregacije mrežnih adresa se označavaju terminom prefiks. Prilikom pretrage lukap tabele se može naći više prefiksa koji odgovaraju odredišnoj IP adresi za koju se vrši pretraga i tada se mora primeniti LPM (*Longest Prefix Matching*) pravilo koje određuje da se za konačno rešenje uzima prefiks koji ima najduže poklapanje sa odredišnom IP adresom za koju se vrši pretraga. IP lukap mora da se izvrši u vremenu trajanja IP paketa. Otuda sa stanovišta najgoreg slučaja, IP lukap mora da se kompletira u vremenu trajanja najkraćeg IP paketa. S obzirom da trajanje IP paketa opada sa porastom brzine linkova, odnosno portova, očigledno je da kompleksnost lukapa

značajno zavisi od brzine porta. Takođe, kompleksnost lukapa zavisi i od broja zapisa u lukap tabeli, kao i od dužine IP adrese. S obzirom da su IPv4 adrese potrošene, neminovan je prelazak na duže IPv6 adrese. Samim tim adresni prostor koji treba pretražiti je značajno veći pa je i kompleksnost lukapa takođe veća u slučaju dužih IPv6 adresa. Brzina lukapa se povećava korišćenjem pajplajn tehnike i paralelizacije koje troše značajne hardverske resurse. Otuda je značajno da se lukap algoritam izvršava dovoljno brzo, a da pri tome troši što manje hardverskih resursa.

U slučaju gigabitskog eterneta, brzina linka, odnosno porta je $C = 1\text{Gb/s}$, a dužina najkraćeg okvira je $L = 64\text{B}$. Lukap za najkraći okvir u tom slučaju mora da se izvrši u okviru $L/(B \cdot N_P) = 128\text{ns}$, pri čemu je N_P broj portova koji dele jedan lukap modul. U našoj implementaciji četiri paketska procesora tj. porta dele jedan lukap modul pa je $N_P = 4$. Vreme od 128ns ne zahteva naročito brz lukap algoritam, te će implementacija takvog algoritma trošiti malo resursa FPGA čipa. U okviru radnog prototipa je implementiran algoritam zasnovan na tehnici m-arnog stabla koji je korišćen u Intelovim mrežnim procesorima [42]-[43]. Lukap počinje pretragom po prefiksima dužim od 15 bita, a u slučaju neuspeha te pretrage vrši se pretraga po prefiksima kraćim od 16 bita. U slučaju pretrage po prefiksima dužim od 15 bita koriste se strajdovi dužine 16-4-4-4-4, a u slučaju pretrage po prefiksima kraćim od 16 bita se koriste strajdovi dužine 8-4-4. Strajd predstavlja broj bita IP adrese koji se razmatra u jednom koraku pretrage. Tako npr. 16-4-4-4-4 označava da se u prvom koraku razmatra prvih 16 bita odredišne IP adrese, pa potom se u svakom sledećem koraku razmatra narednih 4 bita dok se ne iskoriste svi biti odredišne IP adrese ili se pretraga okonča pre toga dolaskom do kraja m-arnog stabla. Svaki korak zahteva jedan pristup lukap tabeli koja je smeštena u eksternoj SRAM memoriji, pa je u najgorem slučaju potrebno osam pristupa eksternoj memoriji. To znači da maksimalno vreme pristupa eksternoj memoriji može da iznosi $128\text{ns}/8 = 16\text{ns}$, i imajući u vidu da standardne SRAM memorije imaju tipična vremena pristupa 8-10ns očigledno je da implementirani lukap algoritam zadovoljava zahteve u pogledu brzine izvršavanja lukapa u najgorem slučaju (trajanje najkraćeg okvira).



Slika 3.2.3.1. – Struktura lukap tabele

Struktura lukap tabele je prikazana na slici 3.2.3.1. Dva početna bloka memorije se rezervišu za početne tabele za pretragu po dužim, odnosno kraćim prefiksima. Pretraga po dužim prefiksima počinje od lokacije u početnoj tabeli za pretragu po dužim prefiksima koju određuju prvih 16 bita odredišne IP adrese za koju se vrši pretraga. Slično, pretraga po kraćim prefiksima počinje od lokacije u početnoj tabeli za pretragu po kraćim prefiksima koju određuju prvih 8 bita odredišne IP adrese za koju se vrši pretraga. Ostatak lukap tabele sadrži blokove koji se sastoje od 16 sukcesivnih memorijskih lokacija. Svaka memorijska lokacija u lukap tabeli predstavlja jedan čvor m-arnog stabla i sadrži dva polja – ID izlaznog porta pridruženog dotičnom čvoru i pokazivač na decu čvora. Pošto se koriste, sem u prvom koraku, strajdovi dužine 4 bita, svaki čvor ima šesnaestoro dece i ona se smeštaju u sukcesivne lokacije tj. u jedan blok pa pokazivač na decu pokazuje na početak bloka tj. na prvo dete. Na ovaj način je značajno redukovan broj pokazivača (umesto 16 koristi se samo 1), čime je značajno smanjena veličina jednog čvora i omogućeno je da veličina memorijske lokacije bude prihvatljiva za implementaciju u eksternoj memoriji, što ne bi bio slučaj da je korišćeno svih 16 pokazivača. Naravno, cena koja se plaća je da se zauzimaju resursi za svu decu iako možda sva deca ne postoje.

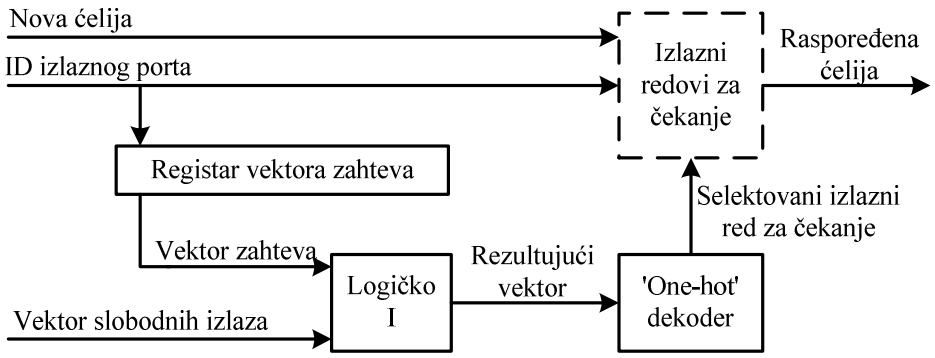
Lukap modul koji vrši pretragu lukap tabele je veoma jednostavan i sastoji se u kretanju kroz m-arno stablo koje zahteva čitanje odgovarajućih lokacija eksterne SRAM memorije. Smer kretanja zavisi od bita odredišne IP adrese. Pri tome se ispita svaki čvor koji se pročita. Ako je njegov ID izlaznog porta validan, on se pamti kao trenutno

najbolje rešenje. Ukoliko je pokazivač validan onda se kreće na sledeći čvor čija adresa ima ofset u odnosu na pokazivač jednak vrednosti sledeća četiri bita odredišne IP adrese za koju se vrši pretraga. Ako pokazivač nije validan onda je pretraga stigla do kraja m-arnog stabla. U slučaju neuspešne pretrage po dužim prefiksima vrši se pretraga po kraćim prefiksima po istom principu kao i po dužim prefiksima. Rezultat pretrage je identifikacija izlaznog porta na koji treba da se usmeri IP paket i ona se prosleđuje ulaznom mrežnom modulu koji je prosleđuje SGS raspoređivaču zajedno sa ćelijama dotičnog paketa. Ako se uzme u obzir da najveći broj prefiksa za slučaj IPv4 adresa pada u opseg 16-24 bita dužine [44], najveći broj pretraga će sadržati 1-3 pristupa eksternoj SRAM memoriji.

Kao što se vidi, implementirani lukap je zadovoljavajući u slučaju gigabitskih portova, pri čemu ga karakteriše mala potrošnja FPGA resursa. Implementirani algoritam nije, međutim, adekvatan za brze portove i IPv6 adresiranje jer bi u tim slučajevima zahtevao prevelik broj pristupa eksternoj memoriji. Npr. u slučaju da lukap modul mora da podrži četiri 10G eternet porta vreme izvršenja lukapa bi bilo 10 puta kraće nego u slučaju gigabitskih eternet portova tj. svega 12.8ns. To znači da bi bio dozvoljen samo jedan pristup eksternoj memoriji tokom jednog izvršenja lukapa, a taj uslov opisani algoritam ne zadovoljava. Dakle mora se dizajnirati složeniji i brži lukap algoritam, koji će biti opisan u poglavlju 5.

Lukap je jedna od najkritičnijih funkcija zbog svoje kompleksnosti, te ograničava brzinu portova. Isto tako lukap se komplikuje sa porastom dužine IP adrese, odnosno sa prelazom na IPv6 adrese koje su četiri puta duže od IPv4 adresa. Kompleksnost lukapa raste sa brojem zapisa u lukap tabeli, i loša organizacija lukap tabele može dovesti do prevelikih memorijskih zahteva koji se ne mogu implementirati. Otuda je neophodan razvoj naprednog lukap algoritma čija će implementacija omogućiti podršku portova velikih brzina kao i velik broj zapisa u lukap tabeli, u slučaju IPv4 i IPv6 adresa.

3.2.4. SGS raspoređivač



Slika 3.2.4.1. – Struktura SGS raspoređivača

Arhitektura implementiranog rutera je zasnovana na arhitekturi sa baferima na ulazu. U slučaju ove arhitekture, ćelije IP paketa se smeštaju u bafere na ulaznom portu, a raspoređivač vrši proračun redosleda slanja ćelija kroz krosbar ka odgovarajućim izlaznim portovima. Algoritam raspoređivanja definiše način rada raspoređivača, tj. način proračunavanja redosleda slanja ćelija. Postoji više algoritama raspoređivanja, a najpoznatiji su PIM [19], iSLIP [20]-[21] i SGS [22]-[24]. Za SGS algoritam je matematički dokazano da komutira saobraćaj bez blokiranja u slučaju kada krosbar ima ubrzanje $u = 2$ [22]-[24]. Odnosno, kada nijedan izlazni port nije preopterećen sav saobraćaj prolazi kroz ruter bez gubitaka ukoliko se paketi raspoređuju prema SGS algoritmu. Zahvaljujući ovoj osobini, SGS obezbeđuje garancije protoka i kašnjenja [22], a takođe se lako modifikuje da podrži i multikast saobraćaj bez blokiranja [31]-[32]. Ove osobine daju dobru podršku multimedijalnom saobraćaju koji ima sve veće učešće u ukupnom Internet saobraćaju. Iz navedenih razloga SGS je izabran kao algoritam raspoređivanja i on je korišćen u okviru implementacije paketskog procesora.

Kao što je rečeno raspoređivač određuje redosled slanja ćelija ka odgovarajućim izlaznim portovima preko krosbara. Struktura SGS raspoređivača je prikazana na slici 3.2.4.1. Na svakom ulaznom portu se formiraju tzv. izlazni redovi čekanja koji sadrže ćelije za odgovarajuće izlazne portove. Vektor zahteva je binarni vektor u kome se smešta binarna indikacija koji izlazni redovi za čekanje imaju ćelije za slanje, a koji su prazni. Vektor zahteva se čuva u registru. Ulazni mrežni modul prosleđuje ćelije ka SGS raspoređivaču zajedno sa identifikacijom izlaznog porta kome su te ćelije namenjene. Po prijemu svake ćelije se utvrđuje kom izlaznom portu je ona namenjena i upisuje se u odgovarajući izlazni red za čekanje koji sadrži ćelije namenjene dotičnom

izlaznom portu. Takođe u slučaju da je dotični izlazni red za čekanje bio prazan vrši se ažuriranje vektora zahteva tako da binarna indikacija, koja odgovara dotičnom izlaznom redu za čekanje, označava da postoje ćelije za slanje u dotičnom izlaznom redu za čekanje. Izlazni redovi za čekanje su smešteni u DDR2 kontroleru i DDR2 SDRAM memoriji i njihova implementacija će biti objašnjena u sledećem odeljku. SGS raspoređivači svih portova su vezani u lanac. Svaki SGS raspoređivač dobija od prethodnog SGS raspoređivača u lancu koji izlazni portovi su slobodni tj. nisu zauzeti od nekog od prethodnih SGS raspoređivača u lancu. Informacija o slobodnim izlaznim portovima se dobija u vidu vektora slobodnih izlaza koji u binarnom vidu daje informaciju o zauzetim i slobodnim izlaznim portovima. Prvi SGS raspoređivač u lancu dobija vektor slobodnih izlaza u okviru kojega su svi izlazni portovi označeni kao slobodni. Između vektora zahteva i vektora slobodnih izlaza se vrši operacija ‘logičko I’ 'i kao rezultat se dobija binarni vektor u kome su označeni samo oni izlazni portovi koji su slobodni i za koje dati SGS raspoređivač ima ćelije. Ovaj rezultujući vektor se vodi na ulaz ‘one-hot’ dekodera koji kao rezultat daje identifikaciju izlaznog reda za čekanje čija je ćelija raspoređena za slanje. SGS raspoređivač prosleđuje sledećem SGS raspoređivaču u lancu ažurirani vektor slobodnih izlaza u kome je izlazni port za koga je raspoređeno slanje ćelije označen kao zauzet.

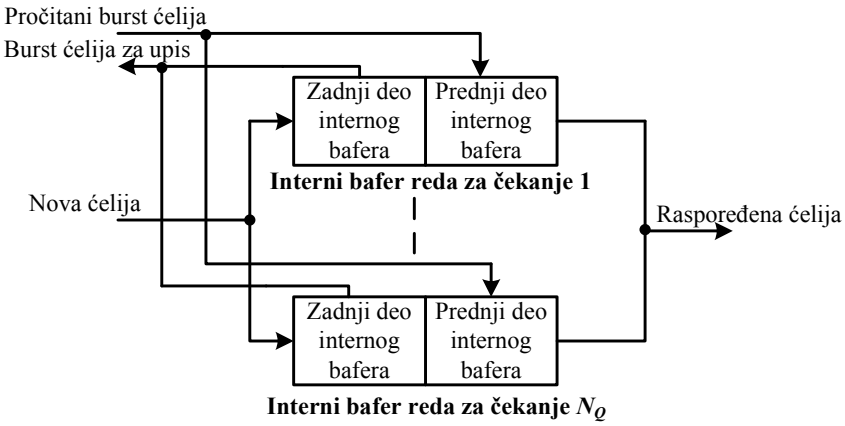
Raspoređene ćelije se šalju iz svog izlaznog reda za čekanje preko krosbara ka svom izlaznom portu u onom vremenskom slotu za koji su raspoređene. Istovremeno SGS raspoređivač mora da pošalje ka krosbaru i informaciju na koji izlazni port ta ćelija treba da se prosledi. Ova informacija je neophodna da bi se krosbar mogao adekvatno konfigurisati. Postoje dva načina za slanje ove konfiguracione informacije. Jedan način je da se one šalju posebnim linijama koje bi povezivale SGS raspoređivač i krosbar, a drugi način je da se ova informacija doda u zaglavlje ćelije. U prvom slučaju bi bile neophodne zasebne linije između svakog SGS raspoređivača i krosbara, što u slučaju velikog broja portova rutera predstavlja problem jer bi postojao velik broj takvih linija kroz bekplejn s obzirom da su SGS raspoređivači na štampanim pločama linijskih modula, a krosbar na svojoj štampanoj ploči. Otuda ovo rešenje limitira skalabilnost rutera jer se kroz bekplejn može postaviti ograničen broj interkonekcionih linija. U drugom slučaju se ovaj problem prevazilazi jer nema dodatnih linija koje bi se koristile za prenos konfiguracionih informacija ka krosbaru, već bi ta informacija bila uneta u

zaglavlje ćelije odakle bi je krosbar mogao izvući. Ova varijanta ima manu što povećava zaglavlje. Na osnovu prethodnog razmatranja, u okviru implementacije je izabrano drugo rešenje jer je skalabilnije.

Kao što je rečeno u prethodnom poglavlju ruteri sa arhitekturom sa ulaznim baferima predstavljaju najskalabilniju jednostepenu arhitekturu. Takođe, u ovom odeljku se moglo videti da je interna struktura SGS raspoređivača jednostavna za implementaciju i lako proširiva. Otuda SGS raspoređivač ne predstavlja potencijalno usko grlo za proširenje kapaciteta Internet rutera.

3.2.5. DDR2 kontroler

Ćelije koje dolaze u ruter se raspoređuju za slanje kroz krosbar prema SGS algoritmu, a dok čekaju na svoj red za slanje se smeštaju u bafere. Isto tako, na izlaznom portu pristigle ćelije se moraju čuvati u baferima dok se čeka na kompletiranje paketa kome dotične ćelije pripadaju, ili da se postojeći paketi pošalju jer krosbar ima ubrzanje u odnosu na izlaz. Da bi se obezbedila mogućnost smeštanja velikog broja ćelija u slučajevima privremenih zagušenja, neophodno je realizovati bafere velikih kapaciteta. Otuda se za realizaciju bafera često koriste eksterne DRAM (*Dynamic Random Access Memory*) memorije koje imaju velike kapacitete [45]-[49]. U realizovanoj implementaciji je korišćena DDR2 SDRAM memorija jer obezbeđuje velike brzine i kapacitete, a cene tih memorija su ekonomične. Međutim, generalni problem DRAM memorija je nedeterministički protok u slučaju kada se pristupa lokacijama po slučajnom principu, usled procesa aktivacije banaka i redova, ali i periodičnog osvežavanja memorijskih blokova koji se moraju vršiti da bi se zadržao integritet upisanih podataka. To znači da u slučaju sukcesivnog upisa (ili čitanja) ćelija u različite banke ili redove DRAM memorije, ostvareni protok može značajno da opadne. Jedini način da se u radu sa DRAM memorijom postigne velik i približno deterministički protok je da se vrši upis dovoljno dugih burstova ćelija u sukcesivne memorijske lokacije i time što više izbegne negativni uticaj procesa aktivacije banaka, odnosno redova.



Slika 3.2.5.1. – Struktura internog bafera

Najefikasniji menadžment memorijskog prostora se ostvaruje primenom virtuelnih redova za čekanje jer se time obezbeđuje dinamičko alociranje memorijskog prostora redovima za čekanje u skladu sa njihovim potrebama [22], [49]. Na taj način se memorijski prostor maksimalno efikasno koristi jer se lako prilagođava trenutnim veličinama redova za čekanje koje mogu biti veoma promenljive tokom vremena. Međutim, tehnika virtuelnih redova za čekanje zahteva memorije sa determinističkim vremenom pristupa bilo kojoj memorijskoj lokaciji, odnosno deterministički protok u slučaju slučajnog pristupa memorijskim lokacijama. Otuda ova tehnika nije pogodna za primenu u DRAM memorijama. Zato se umesto virtuelnih redova za čekanje koristila ideja predložena u [45]-[46]. Ona se zasniva na dodeljivanju jednog internog bafera FPGA čipa svakom redu za čekanje. Oni se koriste za prikupljanje dovoljno velike količine ćelija da oformi burst ćelija koje bi bile upisane u sukcesivne lokacije u DRAM memoriji. Takođe, u suprotnom smeru kad ćelije treba da se pročitaju, čita se burst ćelija koji se smešta takođe u interni bafer.

Struktura internog bafera je prikazana na slici 3.2.5.1. Interni bafer smešta dolazne i odlazne burstove ćelija. Kao što je rečeno baferi na ulaznim portovima čuvaju ćelije koje čekaju svoj redosled na slanje kroz krosbar do odgovarajućeg izlaznog porta. S druge strane, na izlaznim portovima se čuvaju ćelije nekompletiranih paketa koje čekaju na pristizanje svih ćelija paketa da bi mogla da otpočne rekonstrukcija originalnog IP paketa iz ćelija i njegovo slanje na izlazni link. U oba slučaja struktura internog bafera je ista. Interni bafer se sastoji iz dva dela – prednjeg i zadnjeg dela. Oba dela su istog kapaciteta koji je jednak veličini bursta ćelija koji se upisuje u eksternu DRAM memoriju. Na ulaznom portu tokovi su identifikovani po izlaznom portu kome

su ćelije namenjene, a na izlaznom portu po identifikaciji ulaznog porta sa kog su ćelije došle. Kada pristigne nova ćelija ona se upisuje u prednji deo internog bafera ukoliko je ukupan broj ćelija tog toka manji od kapaciteta prednjeg dela internog bafera. U suprotnom se ćelija upisuje u zadnji deo bafera. Onog momenta kad se zadnji deo bafera napuni burst sastavljen od ćelija iz zadnjeg dela internog bafera se upisuje u eksternu DRAM memoriju. Ćelije koje treba da se pročitaju iz internog bafera radi slanja ka krosbaru (ulazni port) ili rekonstrukcije kompletiranog paketa (izlazni port) se jednostavno čitaju po FIFO (*First In First Out*) principu iz prednjeg dela internog bafera. Svaki put kad se isprazni prednji deo bafera u njega se upisuje burst ćelija iz eksterne DRAM memorije ili ako on ne postoji onda ćelije iz zadnjeg dela bafera.

U okviru implementiranog DDR2 kontrolera je realizovan i memorijski interfejs koji omogućava operacije upisa i čitanja burstova ćelija ka eksternoj DDR2 SDRAM memoriji. S obzirom da izabrana DDR2 SDRAM memorija radi na spoljašnjem taktu od 200MHz, a moduli paketskog procesora zbog brzine gigabitskog eterneta na 125MHz (8 bita na 125MHz je ekvivalentno protoku od 1Gb/s) u okviru DDR2 kontrolera su implementirana i dva FIFO bafera – jedan za operacije upisa i jedan za operacije čitanja. Oni se koriste za omogućavanje prelaza između različitih takt oblasti od kojih jedna radi na 200MHz, a druga na 125MHz unutar DDR2 kontrolera. Interni baferi rade na taktu na kom rade i ostali moduli paketskog procesora (125MHz), dok u oblast takta od 200MHz u okviru DDR2 kontrolera spada samo fizički deo memorijskog interfejsa ka eksternoj DDR2 SDRAM memoriji.

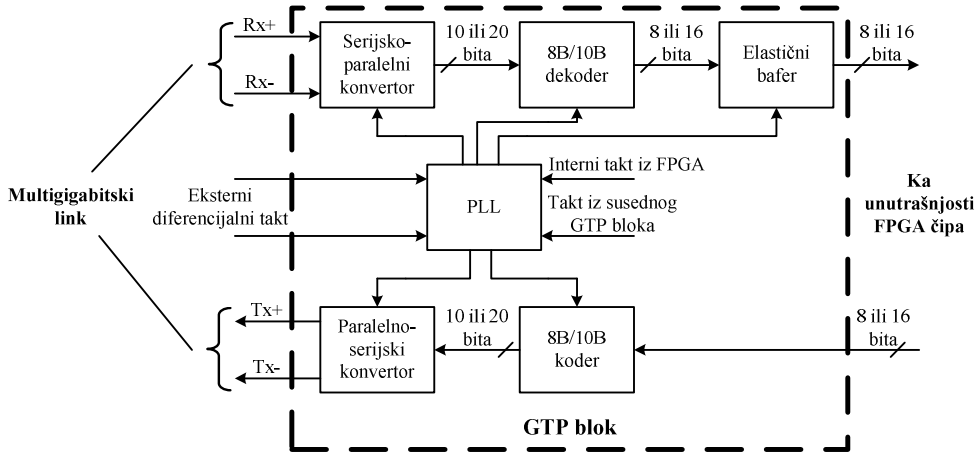
Opisana struktura internih bafera u radu sa eksternom DRAM memorijom omogućava upis i čitanje dovoljno dugih burstova čime se postiže visok protok i približno determinističko vreme pristupa. Naravno, cena koja je plaćena je značajna upotreba internih memorijskih resursa FPGA čipa za implementaciju internih bafera. Veličina jednog internog bafera je $2 \cdot L_B \cdot N_Q$, gde je L_B veličina bursta ćelija u bitima, a N_Q broj portova rutera tj. broj redova za čekanje na ulaznom portu, odnosno izlaznom portu. Kao što se vidi ukupni zahtevani interni memorijski resursi FPGA čipa za realizaciju internih bafera linearno rastu sa porastom broja portova rutera. Isto tako povećanje brzine linkova zahteva i povećanje brzine pristupa memorijskim lokacijama jer su vremenski slotovi kraći. Na osnovu navedenog može se zaključiti da implementacija paketskih bafera predstavlja ozbiljan problem i može biti usko grlo koje

ograničava skalabilnosti rutera. Stoga je važno izvršiti analizu eksternih memorijskih čipova na tržištu i utvrditi koji od njih je najbolji za realizaciju bafera ćelija pre svega imajući u vidu omogućavanje što veće skalabilnosti rutera.

3.2.6. SERDES modul

Za povezivanje paketskih procesora sa krosbarom se koriste multigigabitski linkovi koji koriste serijski diferencijalni prenos. Diferencijalni prenos se koristi zbog povećanja otpornosti na šum. Da bi se ćelije koje se razmenjuju sa krosbarom prilagodile za prenos preko multigigabitskih linkova koristi se SERDES modul koji vrši proces konverzije paralelnog prenosa ćelija u serijski u smeru slanja ćelija ka krosbaru i obrnutu konverziju u suprotnom smeru. Primljene ćelije iz krosbara se prosleđuju izlaznom mrežnom modulu.

SERDES modul je implementiran upotrebom GTP blokova koji postoje u Xilinx FPGA čipovima i koji omogućuju korišćenje multigigabitskih linkova [35]. GTP blokovi su konfigurabilni i omogućavaju implementaciju fizičkog sloja različitih standarda za multigigabitski prenos poput PCI-Express [50], RapidIO [51], SATA [52] i dr. Takođe je moguće konfigurisati GTP blok za implementaciju sopstvenog rešenja fizičkog sloja u slučaju potrebe. Struktura GTP bloka je prikazana na slici 3.2.6.1. GTP blokovi imaju podršku za linijsko kodovanje i dekodovanje, elastični bafer u prijemnom smeru, ekstrakciju takta i dr. PLL (*Phase-Locked Loop*) generiše takt neophodan za rad svih delova GTP bloka. PLL ima izbor između više izvora na koje može da se sinhroniše i u okviru konfigurisanja GTP bloka se definiše koja opcija će se koristiti: eksterni diferencijalni takt, interni takt doveden iz unutrašnjosti FPGA čipa, takt doveden iz susednog GTP bloka. Maksimalna brzina koju podržava GTP blok je 3.125Gb/s. U implementaciji paketskog procesora, GTP blokovi su konfigurisani da rade kao fizički sloj PCI-Express standarda, pri čemu je podešena brzina od 2.5Gb/s zbog trenutnih tehničkih mogućnosti bekplejna. Pošto je prenos diferencijalni postoje četiri linije za podatke na multigigabitskom linku, po dve za svaki smer prenosa (*Tx+* i *Tx-* za predajni smer, i *Rx+* i *Rx-* za prijemni smer). Linijsko kodovanje koje se koristi je 8B/10B linijsko kodovanje, pa je brzina korisničkog saobraćaja na multigigabitskim linkovima 2Gb/s ($\frac{8}{10} \cdot 2.5\text{Gb/s} = 2\text{Gb/s}$).



Slika 3.2.6.1. – Struktura GTP bloka

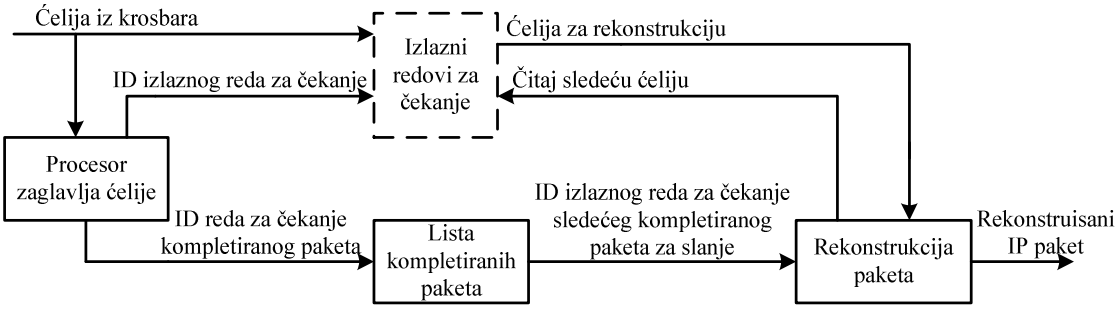
3.2.7. Izlazni mrežni modul

Izlazni mrežni modul prikazan na slici 3.2.7.1 prima ćelije pristigle iz krosbara od SERDES modula. Ove ćelije se smeštaju u izlazne redove za čekanje koji se nalaze u eksternoj DDR2 SDRAM memoriji i internim baferima DDR2 kontrolera, i organizovani su kako je opisano u odeljku 3.2.5. Izlazni redovi za čekanje se razlikuju na osnovu identifikacije ulaznog porta na kom su nastale tj. sa kog su pristigle, pri čemu je ova informacija inkorporirana u zaglavlje svake ćelije pa se lako određuje kom redu za čekanje ćelija pripada. Izlazni redovi za čekanje su neophodni za privremeno smeštanje ćelija dok se ne kompletira paket kome pripadaju (tj. pristignu sve ćelije dotičnog paketa). Takođe, zbog ubrzanja linkova iz krosbara u slučaju privremenog preopterećenja izlaznog linka može da se desi da protok ćelija koje pristižu sa ulaznih portova prevazilazi kapacitet izlaznog linka.

Pošto su sve kontrolne informacije o ćeliji smeštene u njenom zaglavlju, ćelije po ulazu u izlazni mrežni modul prolaze kroz blok za procesiranje zaglavlja ćelije. Ovaj blok ispituje zaglavlje ćelije i na osnovu njega izvršava odgovarajuće akcije. Prvo se određuje kom izlaznom redu za čekanje ćelija pripada i ćelija se upisuje u taj izlazni red za čekanje. Izlazni red za čekanje kom ćelija pripada se određuje na osnovu identifikacije ulaznog porta sa kog je ćelija došla, a koja se nalazi u zaglavlju ćelije. Takođe, na osnovu zaglavlja ćelije se određuje poslednja ćelija paketa na osnovu bita indikacije poslednje ćelije paketa koji se nalazi u zaglavlju ćelije. Kada se detektuje poslednja ćelija paketa, to onda znači da je paket kome ta ćelija pripada kompletiran. Tada se vrši ažuriranje liste kompletiranih paketa koja je smeštena u FIFO memoriju. Lista kompletiranih paketa je uvedena pošto može da se desi da istovremeno postoji

više kompletiranih paketa. Lista kompletiranih paketa sadrži samo identifikaciju izlaznog reda za čekanje kome kompletirani paket pripada. Kompletirani paketi se šalju po FIFO principu. Prilikom slanja kompletiranog paketa vrši se njegova rekonstrukcija tako što se čitaju ćelije dotičnog paketa iz DDR2 kontrolera. Sa ćelija se skida zaglavlje i njihov korisni sadržaj se lepi jedan za drugim. U slučaju poslednje ćelije paketa se uzima samo onaj deo ćelije koji pripada rekonstruisanom IP paketu, pošto korisni deo poslednje ćelije ne mora biti popunjen do kraja bajtovima IP paketa. U prvoj ćeliji se nalazi IP zaglavlje paketa u okviru koga se nalazi 16-bitni podatak o ukupnoj dužini paketa u bajtovima koji se upisuje u brojač pročitanih bajtova paketa. Ovaj brojač se dekrementira za svaki pročitani bajt paketa. Vrednost brojača na početku obrade poslednje ćelije paketa je jednaka broju preostalih bajtova paketa i na osnovu toga se zna koji deo poslednje ćelije sadrži bajtove paketa. Rekonstruisani IP paket se prosleđuje izlaznom modulu sloja linka za formiranje okvira i dalje slanje na izlazni link. Treba napomenuti da se u slučaju podrške saobraćaja različitih nivoa kvaliteta servisa pored identifikacije ulaznog porta sa kog je ćelija došla, koristi i ToS vrednost u zaglavlju ćelije za razlikovanje izlaznih redova za čekanje. Takođe, za svaki nivo kvaliteta servisa, odnosno vrednost ToS polja, se formira zasebna lista kompletiranih paketa čime je omogućeno slanje paketa u skladu sa zahtevanim prioritetima tj. lako je realizovati mehanizam kvaliteta servisa [40]-[41].

Kako se može videti implementacija izlaznog mrežnog modula ne obavlja složene operacije tako da ovaj modul ne predstavlja usko grlo odnosno ne ograničava skalabilnost rutera.



Slika 3.2.7.1. – Struktura izlaznog mrežnog modula

3.2.8. Izlazni modul sloja linka

Izlazni modul sloja linka na primljeni IP paket dodaje eternet okvir i šalje formirani okvir fizičkom sloju (SFP modulu) za slanje na izlazni link. Kao što je već rečeno u odeljku 3.2.1, sloj linka podataka je implementiran koristeći ugrađene MAC blokove korišćenog FPGA čipa. Ugrađeni MAC blok dodaje kontrolne delove zaglavlja (preambulu, SFD i FCS polje) i takođe reguliše IFG (*InterFrame Gap*) koji predstavlja razmak između uzastopnih okvira. Takođe je uz ugrađeni MAC blok implementiran i blok koji vrši dodavanje ostalih delova eternet zaglavlja: izvorišne i odredišne MAC adrese, i informacije o enkapsuliranom protokolu u okviru. Pored okvira u koje su enkapsulirani korisnički IP paketi, izlazni modul sloja linka takođe prosleđuje i okvire koje šalje procesorski modul u mrežu. Pošto procesorski modul kreira zaglavlje okvira sem preambule, SFD i FCS polja, onda se okvir dobijen od procesorskog modula direktno prosleđuje ugrađenom MAC bloku. Kao što je već rečeno u odeljku 3.2.1. izlazni modul sloja linka ne predstavlja ograničenje u pogledu skalabilnosti rutera.

3.2.9. Interfejs ka procesorskom modulu

Interfejs ka procesorskom modulu ne spada u deo paketskog procesiranja, ali će ipak biti ukratko objašnjena njegova uloga, kao i koji se sve podaci razmenjuju preko ovog interfejsa. Interfejs ka procesorskom modulu omogućava povezivanje kontrolne ravni i ravni podataka. Preko ovog interfejsa je omogućeno slanje i prijem kontrolnih paketa kao što su ARP okviri, OSPF paketi i dr. U slučaju kada pristigne kontrolni paket namenjen ruteru vrši se njegovo prosleđivanje iz ulaznog modula sloja linka ka procesorskom modulu preko ovog interfejsa. Isto tako kad procesorski modul treba da pošalje kontrolni paket ili okvir preko nekog izlaznog porta on preko interfejsa ka procesorskom modulu prosleđuje formirani okvir ka izlaznom modulu sloja linka odgovarajućeg izlaznog porta za slanje na izlazni link. Pored kontrolnih paketa, ovaj interfejs se koristi i za slanje komandi procesorskog modula ka paketskom procesoru, kao i za prijem statusnih informacija od paketskog procesora. U komande spada aktivacija i deaktivacija porta, kao i ažuriranje lukap tabele. Ažuriranje lukap tabele predstavlja rezultat rada protokola rutiranja u okviru procesorskog modula i neophodno je za održavanje tačnih informacija u lukap tabeli neophodnih za pravilno usmeravanje paketa [43]. Komande ažuriranja se prosleđuju u parovima adresa-podatak, gde adresa definiše lokaciju u SRAM memoriji na koju se upisuje dati podatak. Na ovaj način je

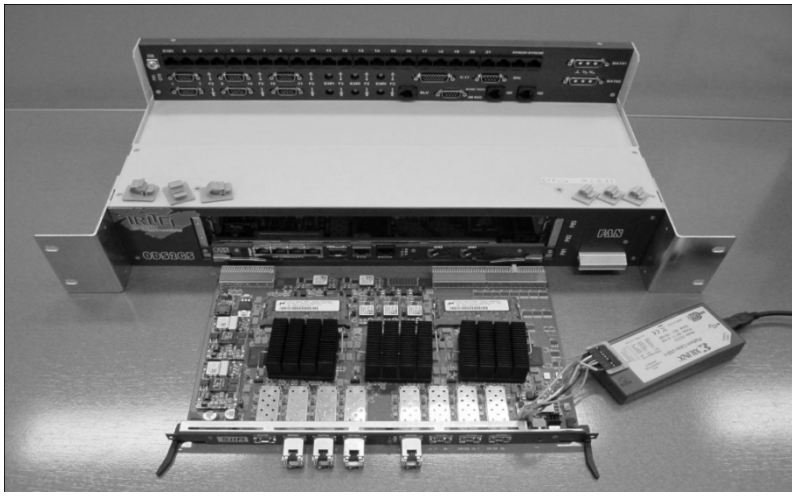
omogućeno brisanje postojećih prefiksa, dodavanje novih prefiksa ili modifikacija ID-a izlaznog porta pridruženog postojećem prefiksu. U očitavanje statusa spada provera prisutnosti konekcije (tj. da li je kabl priključen ili ne). Trenutno se koristi paralelni procesorski interfejs prikazan na slici 3.1.2, pošto je taj interfejs korišćen u Iritelovim sistemima, a procesorski modul koristi procesorsku ploču iz Iritelovih sistema kao što je već rečeno [5]-[6]. U sledećoj verziji prototipa rutera, namera je da se pređe na neki od brzih serijskih interfejsa ka centralnom procesoru poput PCI-Express ili RapidIO koji omogućavaju veće brzine komunikacije. Interfejs ka procesorskom modulu tipično ne predstavlja potencijalno usko grlo u skalabilnosti rutera jer je intezitet komunikacije znatno manji u odnosu na protok paketa kroz ravan podataka. Međutim, sa razvojem novih servisa i funkcionalnosti, ovaj interfejs može ograničiti skalabilnost rutera.

3.2.10. Performanse implementacije paketskog procesora

Kao što je navedeno u potpoglavlju 3.1, u okviru jednog FPGA čipa je implementirano četiri paketska procesora. Korišćen je Xilinx-ov Virtex5 LX110T čip [35]. Sva četiri paketska procesora koriste isti lukap modul i lukap tabelu za izvršenje lukapa. Lukap tabela je smeštena u SRAM memoriju kapaciteta 2MB, pri čemu je konfiguracija memorije 512Kx32b, a korišćen je čip firme Cypress [36]. Baferi u koji se smeštaju ćelije su realizovani u okviru internih bafera FPGA čipa koji su alocirani DDR2 kontroleru, kao i u eksternoj DDR2 SDRAM memoriji kapaciteta 1GB. Micron SODIMM DDR2-800 memorija je korišćena u okviru implementacije [37]. DDR2 kontroler i DDR2 SDRAM memoriju koriste sva četiri paketska procesora.

Svi delovi paketskog procesora rade na taktu od 125MHz, sem dela DDR2 kontrolera koji radi na 200MHz kako je i navedeno u odeljku 3.2.5. Iskorišćeni resursi FPGA čipa za implementaciju paketskih procesora su dati u tabeli 3.2.10.1. Svih modula ima po četiri koliko ima i paketskih procesora, samo modula koje dele paketski procesori ima po jedan (lukap modul i DDR2 kontroler). Resursi za ulazne i izlazne module linka podataka su dati zajedno pošto njih nije bilo moguće razdvojiti zbog upotrebe ugrađenih MAC blokova. Iz tabele 3.2.10.1 se može videti da je memorija najkritičniji interni resurs FPGA čipa. Najviše memorije zauzimaju baferi u FPGA memoriji koji su deo DDR2 kontrolera, pri čemu je bitno naglasiti da bi oni dodatno rasli za slučaj da broj portova raste (trenutno su veličine internih bafera dimenzionisane

za podršku 16 portova). Takođe su implementirani i FIFO baferi koji služe kao most između različitih domena takta.



Slika 3.2.10.1. – Realizovani prototip rutera

Opisana implementacija predstavlja osnovu paketskog procesiranja unutar ravni podataka prototipa Internet rutera, i predstavlja prvi doprinos ove teze. Pošto je korišćen FPGA čip moguće je na jednostavan način dodavati nove funkcije paketskog procesiranja, ali i modifikovati i unapređivati postojeće funkcije paketskog procesiranja. Na ovaj način je obezbeđena platforma za dalja istraživanja kako funkcija ravni podataka rutera, tako i kontrolne ravni. Prototip rutera prikazan na slici 3.2.10.1 je testiran povezivanjem sa softverskim ruterima, kao i Cisco ruterima [53]. Štampana ploča (izvučena iz prototipa rutera) prikazana na slici 3.2.10.1 predstavlja linijski i krosbar modul jer, kako je i navedeno u potpoglavlju 3.1, u prvoj verziji prototipa su ova dva modula smeštena na istoj štampanoj ploči iz ekonomskih razloga.

Tabela 3.2.10.1. – Iskorišćeni resursi FPGA čipa za implementaciju paketskih procesora

Modul	Broj modula	Logički elementi	Registri	Memorija
Ulazni i izlazni modul sloja linka	4	970	6020	288Kb
Ulazni mrežni modul	4	572	1015	144Kb
Izlazni mrežni modul	4	976	4753	144Kb
Lukap modul	1	103	171	0Kb
DDR2 kontroler	1	451	4118	1404Kb
SGS raspoređivač	4	740	3862	0Kb
SERDES modul	4	182	1463	144Kb
Ukupno	**-**	**3994 (6%)**	**21402 (31%)**	**2124Kb (40%)**

S obzirom na planove našeg istraživačkog tima da se implementiraju i veće brzine portova, kao i da se poveća ukupan broj portova rutera, u okviru ovog poglavlja izvršena je analiza u cilju prepoznavanja potencijalnih uskih grla pri povećanju kapaciteta rutera. Prvo potencijalno usko grlo je lukap koji značajno zavisi od brzine portova, ali i od dužine IP adrese i broja zapisa u lukap tabeli. Drugo potencijalno usko grlo su baferi koji čuvaju ćelije. Njihova implementacija zavisi i od broja portova i od brzine portova, pa je potrebno izvršiti adekvatnu analizu uzimajući u obzir tipove postojećih memorija i njihove performanse. Drugi deo teze je posvećen rešavanju problema lukapa. Cilj drugog dela teze je da predloži rešenja koja bi se implementirala u okviru prototipa rutera sa portovima velikih brzina (10Gb/s i više). U drugom delu teze će biti predloženi novi lukap algoritmi koji su namenjeni za primenu u slučaju veoma brzih portova i lukap tabela sa velikim brojem zapisa, pri čemu će biti podržane i IPv4 i IPv6 adrese. Predloženi novi lukap algoritmi predstavljaju drugi doprinos teze.

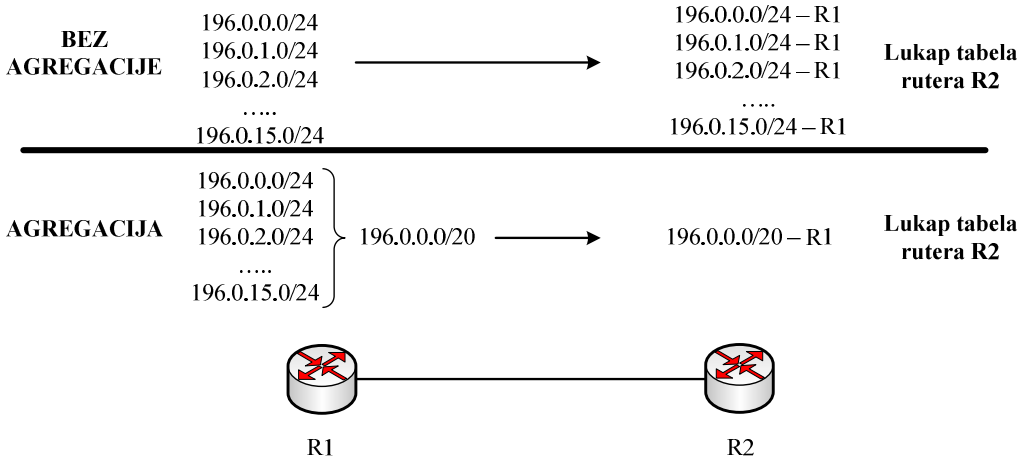
4. PREGLED LUKAP ALGORITAMA

Kao što je već navedeno u prethodnom poglavlju, lukap predstavlja jedno od potencijalnih uskih grla u daljem razvoju Internet rutera te je stoga potrebno obratiti posebnu pažnju na ovu funkciju. Prvo će biti objašnjen sam lukap i aspekti koji ga čine veoma komplikovanim i zahtevnim za realizaciju. Takođe će biti navedeni zahtevi koje moderni lukap algoritmi moraju da ispune. Potom će biti data klasifikacija lukap algoritama, gde će biti navedene i prezentovane klase lukap algoritama. Pri tome će za svaku klasu biti navedeni najpoznatiji lukap algoritmi iz dotične klase, kao i analiza njihovih performansi.

4.1. Lukap

Ruteri predstavljaju osnovnu gradivnu jedinicu Internet mreže. Njihova osnovna funkcija je rutiranje, tj. usmeravanje paketa ka njihovom odredištu na osnovu njihove IP adrese. Proces određivanja izlaznog porta rutera na koji treba usmeriti pristigli paket se naziva lukap. Da bi lukap mogao da se izvrši neophodno je da ruter zna topologiju mreže tj. da zna gde se koje odredište nalazi. IP adresa se sastoji iz dva dela – mrežnog dela i host dela. Ruter vrši usmeravanje na osnovu mrežnog dela IP adrese čime usmerava paket ka odredišnoj mreži gde se nalazi host kome je paket namenjen. U samoj odredišnoj mreži usmeravanje se dalje vrši koristeći host deo adrese. Otuda je neophodno da ruter poznaje samo mrežne delove IP adresa, tj. da za svaku mrežu zna gde se ona nalazi i preko kog porta mu je dostupna. Da bi prikupio ove informacije neophodno je da ruteri međusobno komuniciraju i razmenjuju informacije, čime ruter uči topologiju mreže u kojoj se nalazi. Protokoli rutiranja poput OSPF [54], BGP [55], RIP [56] i dr. se koriste za razmenu tih informacija. Kao rezultat rada protokola rutiranja, ruter formira tzv. lukap tabelu koja za svaku mrežnu adresu definiše koji izlazni port rutera predstavlja najkraću putanju do mreže koja je definisana tom mrežnom adresom. Za svaki pristigli paket se prema definisanom lukap algoritmu, vrši

pretraga lukap tabele na osnovu IP adrese paketa, i određuje se na koji izlazni port rutera paket treba da se usmeri.

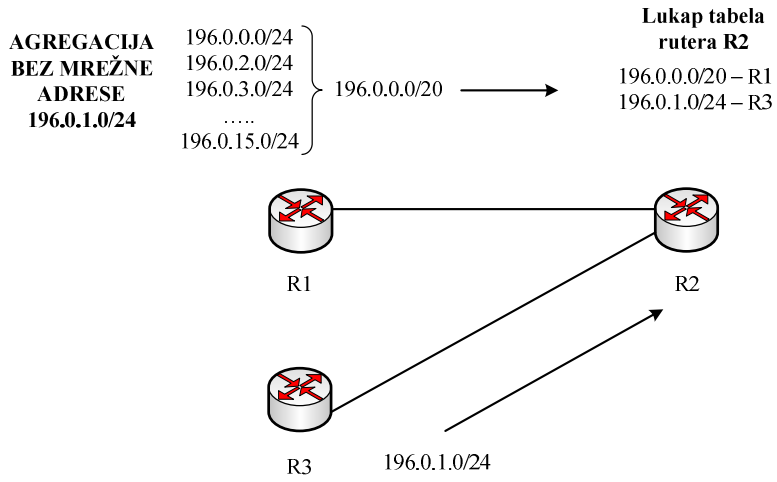


Slika 4.1.1. – Primer agregacije mrežnih adresa

U početku se na Internetu koristilo klasno adresiranje gde su za unikast IP adrese definisane tri klase – A, B i C, u kojima su mrežni delovi IP adrese sadržali prvih 8, 16 i 24 bita, respektivno. Klasno adresiranje nije racionalno koristilo adresni prostor pa se ubrzo prešlo na besklasno adresiranje gde je granica između mrežnog i host dela adrese bila proizvoljna, čime je omogućena racionalnija upotreba adresnog prostora. Takođe, besklasno adresiranje je omogućilo ruterima da vrše agregaciju više mrežnih adresa u jedan zapis koji bi oglašavali drugim ruterima putem protokola rutiranja, čime se smanjuje ukupan broj unosa u lukap tabelama rutera. Jedan takav primer je pokazan na slici 4.1.1 gde ruter R1 agregira 16 mrežnih adresa u jedan zapis koji oglašava ruteru R2. Na ovaj način ruter R2 u lukap tabelu smešta samo jedan zapis umesto 16 zapisa da nije korišćena agregacija. U primeru sa slike 4.1.1 agregacija je idealna jer se opseg agregiranog zapisa u potpunosti poklapa sa kompletnim opsegom mrežnih adresa koje su agregirane.

Međutim, agregacija ne mora da bude idealna. U tom slučaju agregirani zapis može da obuhvati i neke mrežne adrese koje nisu ušle u proces agregacije. Na slici 4.1.2 je prikazan takav primer, gde ruter R1 formira agregirani zapis od 15 mrežnih adresa i prosleđuje ga ruteru R2, a ruter R3 prosleđuje ruteru R2 informaciju o mrežnoj adresi 196.0.1.0/24. Mrežna adresa 196.0.1.0/24 je obuhvaćena agregiranim zapisom iako nije bila uključena u proces agregacije u ruteru R1. Otuda će lukap u ruteru R2 za pakete namenjene hostovima u mreži 196.0.1.0/24 da nađe dva rešenja – jedno koje odgovara

agregiranom zapisu dobijenom od rutera R1 i koje kazuje da ti paketi treba da se usmere ka ruteru R1 i drugo koje odgovara zapisu dobijenom od rutera R3 i koje kazuje da te pakete treba usmeriti ka ruteru R3. Pošto lukap može da nađe više rešenja, usvojeno je pravilo najdužeg poklapanja koje određuje da se za konačno rešenje usvoji zapis koji se najduže poklapa sa IP adresom za koju se vrši lukap. Ovo pravilo se naziva LPM pravilo. Pošto se u lukap tabeli mogu nalaziti mrežne adrese, ali i agregacije mrežnih adresa, usvojen je termin prefiks koji označava oba slučaja i u nastavku rada će se koristiti ovaj termin. Primenom LPM pravila, ruter R2 će sve pakete namenjene hostovima iz mreže 196.0.1.0/24 pravilno usmeravati ka ruteru R3.



Slika 4.1.2. – Primer agregacije mrežnih adresa koja nije idealna

Kao što se moglo videti iz prethodnih primera agregacija značajno smanjuje broj zapisa u lukap tabelama i samim tim njihovu veličinu, ali dovodi do pojave da za jednu IP adresu može da postoji više rešenja. Ova pojava značajno doprinosi kompleksnosti lukapa jer prilikom nalaženja rešenja neophodno je biti siguran i da je ono najbolje rešenje jer u suprotnom može doći do grešaka u usmeravanju. Tako u primeru sa slike 4.1.2. u procesu lukapa u ruteru R2 za paket koji je namenjen hostu iz mreže 196.0.1.0/24, ako je prvo nađeno rešenje ono koje odgovara agregiranom zapisu 196.0.0.0/20 i ako se pretraga okonča odmah po nalaženju prvog rešenja došlo bi do greške u usmeravanju paketa jer umesto ka ruteru R3 paketi bi se usmeravali ka ruteru R1. Otuda lukap koji na prvi pogled izgleda jednostavan postaje veoma komplikovan jer nije dovoljno samo naći rešenje, nego je i potrebno biti siguran da je ono zaista najbolje.

Dodatni problemi prilikom realizacije lukapa su tehnički zahtevi. Linkovi koji se instaliraju u mreži dostižu i 100Gb/s. Lukap mora da nađe rešenje za vreme trajanja IP paketa jer u suprotnom bi dolazilo do gomilanja paketa na ulazu, usled čekanja na rezultat lukapa koji treba da odredi na koji izlaz rutera paketi treba da se usmere, i samim tim i gubitaka paketa. Prilikom dizajna lukapa uzima se u obzir najgori slučaj. Najgori slučaj je da u ruter pristižu paketi minimalne dužine jer oni najkraće traju i lukap ima najmanje vremena na raspolaganju u tom slučaju. Najmanja dužina IP paketa je 40B, u kome se prenosi samo osnovno zaglavlje IP paketa. U tabeli 4.1.1 je dat prikaz vremena za koji lukap treba da se izvrši za različite brzine linkova. Prilikom proračuna se mora uzeti u obzir i činjenica da drugi sloj (sloj linka podataka) dodaje svoja polja na IP paket, pa se za minimalnu dužinu paketa u proračunu u stvari uzima minimalna dužina okvira na drugom sloju, a ne dužina najkraćeg IP paketa bez polja drugog sloja. U proračunu prikazanom u tabeli 4.1.1 je smatrano da se koristi eternet tehnologija i da je najkraće trajanje okvira 64B. Može se videti da, pri velikim brzinama linkova, lukap mora da se izvrši za veoma kratko vreme – svega 5.12ns za slučaj 100Gb/s linka.

Tabela 4.1.1. – Vreme izvršavanja lukapa za različite brzine linkova

Brzina linka	Trajanje najkraćeg okvira (vreme za koje lukap mora da se izvrši)
100Mb/s	5120ns
1Gb/s	512ns
10Gb/s	51.2ns
40Gb/s	12.8ns
100Gb/s	5.12ns

Imajući u vidu da lukap tabela mora da se smesti u neku memoriju i da prilikom izvršavanja lukapa mora da se pristupa tim podacima jasno je da je broj pristupa lukap tabeli veoma ograničen pri velikim brzinama linkova. Očigledno je da zahtevane (i velike) brzine lukap algoritama zahtevaju hardversku realizaciju lukapa pri čemu mora da se posebno vodi računa o organizaciji podataka (zapisa) u lukap tabeli i samoj pretrazi lukap tabele da bi se postigle zahtevane performanse, a da se ne potroše veliki hardverski resursi. Zahtevi za visokom brzinom izvršavanja lukapa dodatno doprinose kompleksnosti realizacije lukapa. Usled toga se često koriste dodatni protokoli poput MPLS (*Multi-Protocol Label Switching*) [57] i GMPLS (*Generalized Multi-Protocol Label Switching*) [58] protokola, koji omogućavaju efikasnije usmeravanje na bazi labela umesto IP adresa u jezgru Internet mreže, gde su i najveće brzine linkova.

Međutim, MPLS i GMPLS protokoli zahtevaju dodatnu kompleksnost u ravni podataka, i na kontrolnoj ravni. Takođe, IP lukap se ne može u potpunosti izbeći jer ruteri na ulazu u MPLS mrežu moraju odrediti prvu MPLS labelu i izlazni port na osnovu IP adrese.

Topologija mreže se neprestano menja usled dodavanja novih linkova, rutera, ali isto tako i usled otkaza linkova, rutera itd. Protokoli rutiranja obaveštavaju sve rutere u mreži o nastalim promenama. Na osnovu ovih informacija, centralni procesor rutera izračunava novu lukap tabelu i spušta je na paketske procesore rutera. Proces promene sadržaja lukap tabele se naziva proces ažuriranja u okviru koga se dodaju novi prefiksi, menja informacija o izlaznom portu za postojeće prefikse ili brišu postojeći prefiksi. Proces ažuriranja je znatno sporiji od procesa pretrage tj. lukapa, jer se lukap vrši za svaki paket, a ažuriranje značajno ređe, samo kada dođe do promene u mreži. Međutim, neophodno je da se promene brzo unose u lukap tabele da bi što ređe dolazilo do grešaka u usmeravanju podataka usled zastarelosti i netačnosti podataka u lukap tabeli, pa je neophodno da se proces ažuriranja lukap tabele izvršava u realnom vremenu. S obzirom na već pomenute zahteve za brzim lukapom, lukap algoritmi često koriste razne vidove komprimovanja podataka u lukap tabelama što s druge strane otežava efikasno ažuriranje takvih struktura. Otuda je prilikom implementacije lukapa potrebno voditi računa o ovim oprečnim zahtevima, s jedne strane treba što bolje komprimovati lukap tabelu da bi se postigla velika brzina lukapa, a sa druge strane kompresija ne sme da dovede do toga da proces ažuriranja ne može da se izvrši u realnom vremenu. Ovi oprečni zahtevi dodatno doprinose složenosti problema IP lukapa.

Dodatni problem u realizaciji lukapa predstavlja i velik broj prefiksa koje lukap tabela može da sadrži. Najveće lukap tabele sadrže i do 400000 prefiksa [44] pa samim tim lukap tabele postaju veoma velike čime je lukap dodatno otežan jer je neophodno efikasno smestiti i pretražiti veliku količinu podataka. Prvobitno su se koristile IPv4 adrese, ali s obzirom da je IPv4 adresni prostor potrošen, neminovan je prelaz na duže IPv6 adrese, što dodatno komplikuje lukap jer je sada prostor pretrage znatno povećan [59]-[60].

Rezimirajmo zahteve koje IP lukap treba da zadovolji. Lukap mora da podržava efikasnu pretragu i za slučaj IPv4 i IPv6 adresa. Mora da podrži rad sa velikim lukap tabelama koje sadrže nekoliko stotina hiljada prefiksa. Lukap algoritam treba da se

izvršava veoma brzo – u vremenu nekoliko nanosekundi za slučaj veoma brzih linkova. Pored toga ažuriranje lukap tabele mora da bude dovoljno efikasno da može da se izvršava u realnom vremenu i time uspešno prati promene topologije mreže.

4.2. Klasifikacija lukap algoritama

Lukap algoritmi su došli u fokus istraživača krajem devedesetih godina prošlog veka, kad je besklasno adresiranje bilo široko u upotrebi i kad su lukap tabele dobile značajnije dimenzije, a pri tome brzine linkova bile u porastu [7]. Pre toga, lukap je u ruteru za sve portove bio izvršavan u centralnom procesoru rutera. Međutim, kako su rasle brzina portova, veličina lukap tabela i broj samih portova, postalo je nemoguće izvršavati lukap u centralnom procesoru koji je pored toga morao izvršavati i druge zadatke kao npr. izvršavanje protokola rutiranja (OSPF, BGP), obavljanje funkcija menadžmenta i dr. Stoga je lukap izmešten na same portove pa je svaki port dobio svoju lukap tabelu i na svakom portu se izvršavao lukap algoritam prema kome se pretraživala odgovarajuća kopija lukap tabele. S takvim razvojem situacije, postalo je neophodno razviti efikasne hardverske implementacije lukapa koje bi koristile što manje hardverske resurse i time bile ekonomičnije za implementaciju. Razvijen je velik broj veoma različitih lukap algoritama, pa se često vrše klasifikacije tih realizacija radi lakšeg pregleda postojećih rešenja.

Lukap algoritam definiše strukturu lukap tabele i određuje kako se vrši pretraga lukap tabele. Takođe definiše i način na koji se vrši ažuriranje lukap tabele. Samim tim lukap algoritam definiše u potpunosti način izvršavanja lukapa i njegovu implementaciju. Postoje razne klasifikacije lukap algoritama. Tako se po jednoj klasifikaciji razlikuju softverski i hardverski lukap algoritmi. Međutim, ova klasifikacija danas nije adekvatna usled tehničkih zahteva u modernim ruterima, jer lukap mora da se implementira u hardveru. Po drugoj klasifikaciji razlikujemo da li se proces ažuriranja vrši u kontrolnoj ravni (centralnom procesoru) ili ravni podataka. Češća varijanta je da se proces ažuriranja vrši u kontrolnoj ravni iz više razloga. Prvi razlog je taj što se u centralnom procesoru izvršavaju protokoli rutiranja koji daju podatke za ažuriranje lukap tabele. Drugi razlog je što svi portovi koriste istu lukap tabelu, pa je pogodno da centralni procesor izračuna tabelu i pošalje je paketskim procesorima u vidu parova (adresa, podatak) koji definišu na koju lokaciju u lukap tabeli treba upisati dati podatak.

Na taj način se izbegava angažovanje dodatnih resursa na svakom portu u ravni podataka za ažuriranje [43]. Time se postiže ekonomičnija izrada i cena rutera. Međutim, ni ova klasifikacija ne daje nikakve podatke o samoj strukturi lukap algoritama i načinu njihovog rada. Otuda je najbitnija klasifikacija koja se bazira na principima rada lukap algoritama i grupiše ih u odgovarajuće klase. Postoje tri glavne klase lukap algoritama koje obuhvataju sve postojeće lukap algoritme. Prva klasa obuhvata lukap algoritme bazirane na strukturi stabla [59]-[78]. Druga klasa obuhvata lukap algoritme bazirane na specijalnim TCAM (*Ternary Content Addressable Memory*) memorijama [79]-[85]. Treća klasa obuhvata algoritme bazirane na principu heširanja [86]-[90]. Veoma često se dešava da lukap algoritmi kombinuju tehnike i dobre osobine iz više različitih klasa radi postizanja što boljih performansi. Prve dve navedene klase obuhvataju najveći broj lukap algoritama. U sledećim potpoglavljima će biti dat opis i analiza svake klase, pri čemu će se u svakoj klasi analizirati karakteristični i poznati predstavnici lukap algoritama iz dotične klase.

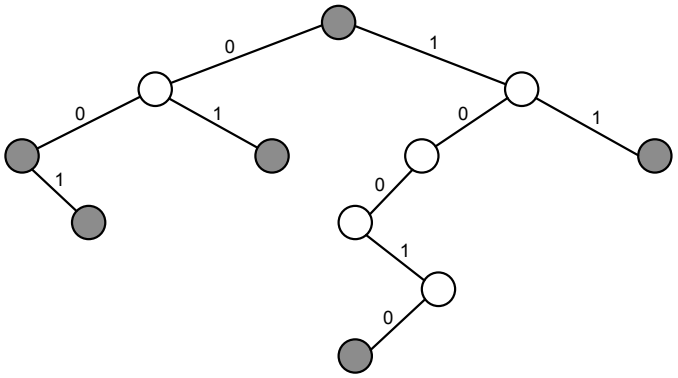
Prilikom analize lukap algoritama, analiziraće se pre svega memorijski resursi jer se oni najviše troše radi smeštanja velikog broja zapisa u slučaju velikih lukap tabela. Pri tome će se prvo analizirati memorijski resursi pojedinih delova lukap algoritama, a zatim će biti data analiza internih memorijskih zahteva. Interni memorijski zahtevi se moraju razmatrati pošto oni značajno utiču na protok lukap algoritama. Interne memorije (memorije koje se nalaze u okviru integrisanog čipa) su brže od eksternih, ali imaju manji kapacitet jer dele prostor čipa sa logikom koja implementira algoritme. Pošto su interni memorijski resursi ograničeni neophodno je da interni memorijski zahtevi lukap algoritama budu mali da bi mogli da stanu unutar jednog integrisanog čipa. Eksterne memorije imaju veće kapacitete od internih memorija, ali su u slučaju brzih eksternih memorija ti kapaciteti takođe ograničeni (tipično nekoliko MB), pa ukupni memorijski zahtevi lukap algoritama takođe ne smeju da budu preveliki. Interni memorijski zahtevi će biti analizirani za tri slučaja kada je na raspolaganju jedna, dve i tri eksterne memorije pri čemu je za eksterne memorije stavljeno ograničenje od 144 bita na dužinu magistrale podataka kao u [79]. Veći broj eksternih memorija neće biti razmatran pošto on ne bi bio praktičan za implementaciju zbog prostora na štampanoj ploči.

4.3. Algoritmi zasnovani na strukturi stabla

Binarno stablo predstavlja prirodnu strukturu kojom može da se predstavi lukap tabela jer se svaki prefiks može predstaviti čvorom stabla pri čemu vrednost prefiksa određuje poziciju čvora u stablu tako što definiše put od korena stabla do dotičnog čvora [67], [73]. Uzimajući bit po bit IP adrese, pretraga lukap tabele se vrši kretanjem kroz stablo tako što vrednost ‘0’ uzetog bita određuje smer levo u stablu, a vrednost ‘1’ smer desno. Pri tome oni čvorovi stabla koji odgovaraju unetim prefiksima u lukap tabelu sadrže i informaciju o izlaznom portu na koji treba usmeriti pakete čija se odredišna IP adresa poklapa sa datim prefiksom (preciznije čija odredišna IP adresa ima najduže poklapanje u skladu sa LPM pravilom). Informacija o izlaznom portu predstavlja u stvari identifikaciju (ID) izlaznog porta. Prikaz jedne male lukap tabele u vidu binarnog stabla je dat na slici 4.3.1. Svaki čvor u stablu mora da ima polja za pokazivače levo i desno neophodna za kretanje kroz stablo i polje za smeštanje ID-a izlaznog porta jer on predstavlja konačni rezultat pretrage. Na slici 4.3.1 neosenčeni čvorovi ne sadrže validan ID izlaznog porta i oni su prisutni da bi se omogućilo kretanje kroz stablo do čvorova koji sadrže validni ID izlaznog porta. Osenčeni čvorovi sadrže validan ID izlaznog porta.

Binarno stablo predstavlja veoma jednostavnu strukturu. Pretraživanje takve strukture se svodi na kretanje kroz stablo na bazi bita IP adrese dok se ne dođe do kraja stabla, a kao konačno rešenje pretrage se uzima poslednji čvor sa validnim ID-em izlaznog porta koji je posećen. Ažuriranje strukture stabla je takođe jednostavno. Proces dodavanja novog prefiksa se svodi na kretanje kroz stablo i upis ID-a izlaznog porta u odgovarajući čvor, pri čemu se u slučaju da neki čvorovi na putu ne postoje, oni tada kreiraju. Brisanje prefiksa je takođe jednostavno. Ono se svodi na brisanje ID-a izlaznog porta u dotičnom čvoru i eventualno brisanje tog čvora u slučaju da taj čvor ne poseduje naslednike. U slučaju da se dotični čvor briše, onda se brišu i njegovi prethodnici dok se ne dođe do prvog čvora koji ima validan ID izlaznog porta ili se grana. Ova jednostavnost je doprinela popularnosti strukture stabla za primenu u lukapu. Međutim, kako su zahtevi za brzinom lukapa rasli, binarno stablo nije moglo da ispuni te nove zahteve. Naime, ukupna dubina binarnog stabla je jednaka dužini IP adrese u najgorem slučaju, 32 bita za IPv4, odnosno 128 bita za IPv6. To znači da ako se svi čvorovi smeste u istu memoriju u najgorem slučaju potrebno je 32 pristupa za slučaj IPv4

adresa, a još više za IPv6 slučaj. Sa stanovišta današnjih zahteva da se lukap izvrši u nekoliko desetina nanosekundi ili čak svega nekoliko nanosekundi, tako velik broj pristupa memoriji je neprihvatljiv. Ovo ograničenje može da se reši tako da se svaki nivo stabla smesti u zasebnu memoriju, pa da se primeni pajplajn tehnika tako što se pretraga za narednu IP adresu vrši na nekom nivou kada se pretraga za prethodnu IP adresu preseli na sledeći nivo. Međutim, ni to nije praktično rešenje jer je zahtevani broj memorija jednak broju nivoa stabla, odnosno dužini najdužeg prefiksa, i samim tim prevelik za praktičnu implementaciju. Iz tog razloga razvijene su različite tehnike kojima se unapređuje lukap algoritam zasnovan na binarnom stablu, istovremeno zadržavajući dobre osobine binarnog stabla u što je većoj meri moguće. Ove tehnike će biti opisane u narednim odeljcima.



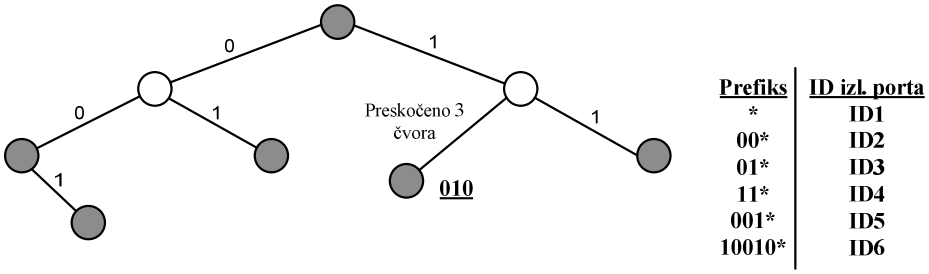
Prefiks	ID izl. porta
*	ID1
00*	ID2
01*	ID3
11*	ID4
001*	ID5
10010*	ID6

Slika 4.3.1. – Lukap tabela u vidu binarnog stabla

4.3.1. Kompresija putanje

Prva tehnika koja se koristi je kompresija putanje u binarnom stablu [71], [73]. Ova tehnika se zasniva kompresiji putanja bez grananja i validnih ID-eva izlaznih portova. Na takvim putanjama nema korisnih odluka – nema validnih ID-eva izlaznih portova koje bi trebalo zapamtiti kao trenutno najbolja rešenja u toku pretrage, a takođe nema ni grananja koje bi moglo da utiče na smer pretrage. Otuda je najbolje takve čvorove komprimovati, odnosno izbaciti ih iz stabla čime se pretraga koja ide tom putanjom skraćuje za broj komprimovanih čvorova. Međutim, ova tehnika zahteva uvođenje dodatnih polja u okviru čvorova stabla. Jedno dodatno polje mora da definiše broj preskočenih (komprimovanih) čvorova – u slučaju da nema preskočenih čvorova ovo polje će imati vrednost 0. Drugo dodatno polje mora da sadrži vrednost komprimovanog puta i ta vrednost se upoređuje sa odgovarajućim bitima iz IP adrese za koju se vrši pretraga. U primeru sa slike 4.3.1.1 je korišćena lukap tabela prikazana na

slici 4.3.1 i komprimovana je putanja za prefiks 10010*. Tako, ako IP adresa za koju se vrši pretraga počinje bitima '10abc', pretraga će od korena krenuti do njegovog desnog deteta, a potom će od tog čvora da skrene ulevo. Pošto je ova putanja komprimovana u ovom čvoru će biti utvrđeno da su preskočena tri čvora i poredиće se vrednost 'abc' iz IP adrese sa vrednošću '010' koja odgovara komprimovanoj putanji. Ako se poklapaju onda će se uzeti ID izlaznog porta iz tog čvora kao rezultat pretrage. U suprotnom će biti uzet ID izlaznog porta iz korena stabla, pošto jedino taj čvor sadrži validan ID izlaznog porta na putanji definisanoj IP adresom koja počinje bitima '10abc'. Ruteri su devedesetih godina koristili lukap algoritam PATRICIA koji se zasnivao na strukturi binarnog stabla sa komprimovanim putanjama [71], [73]. Međutim, ova tehnika ne predstavlja dovoljno unapređenje. U opštem slučaju maksimalna dubina stabla ostaje ista kao i u slučaju binarnog stabla bez kompresije. Takođe, kod veoma velikih tabela, koje mogu sadržati i do 400K prefiksa u IPv4 slučaju [44], broj komprimovanih putanja nije značajan i samim tim ova tehnika ne doprinosi značajnom unapređenju originalnog binarnog stabla.



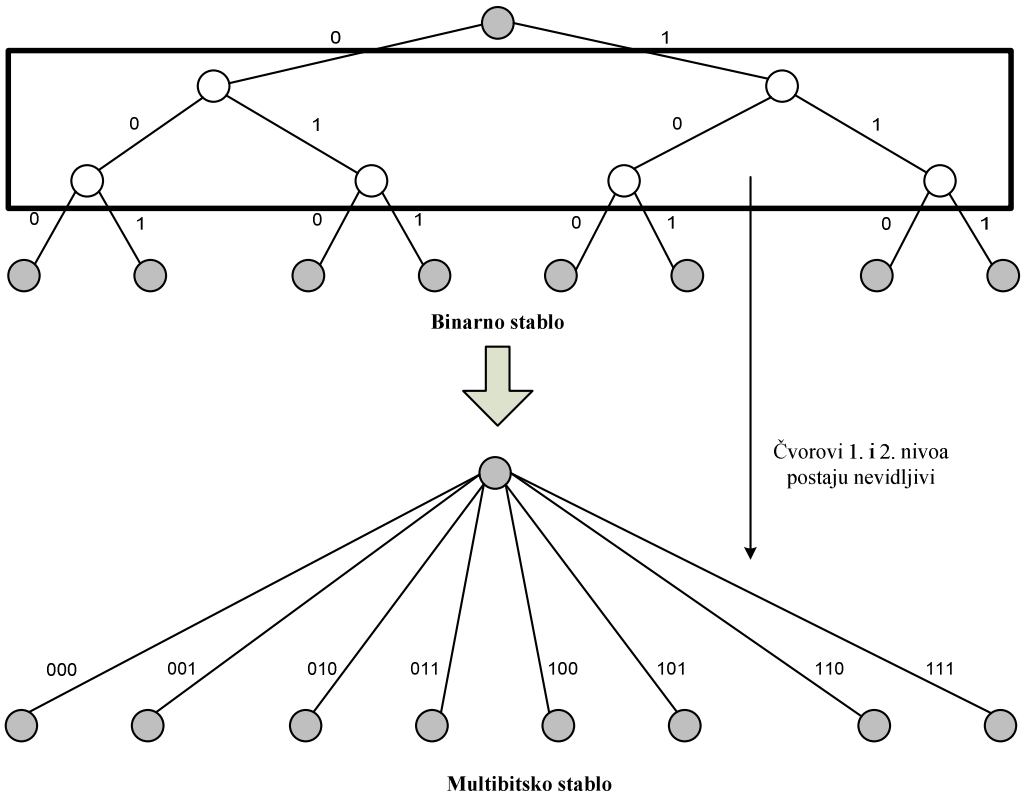
Prefiks	ID izl. porta
*	ID1
00*	ID2
01*	ID3
11*	ID4
001*	ID5
10010*	ID6

Slika 4.3.1.1. – Primer kompresije putanje

4.3.2. Guranje prefiksa u listove stabla

Druga tehnika je guranje prefiksa tj. njihovih ID-eva izlaznih portova u listove stabla [66], [74]. List stabla je onaj čvor koji nema dece. Na ovaj način se izbegava pamćenje ID-a izlaznog porta prilikom kretanja kroz stabla i uzima se ID izlaznog porta iz lista stabla u kom je pretraga okončana, pošto je ovom tehnikom obezbeđeno da se sve pretrage moraju okončati u listovima stabla. Pored toga, postoji i modifikacija ove tehnike gde se prefiksi guraju na određene nivoe stabla i koja se koristi u slučaju kada želimo da smanjimo ukupan broj nivoa stabla koji sadrže ID-eve izlaznih portova [61], [65].

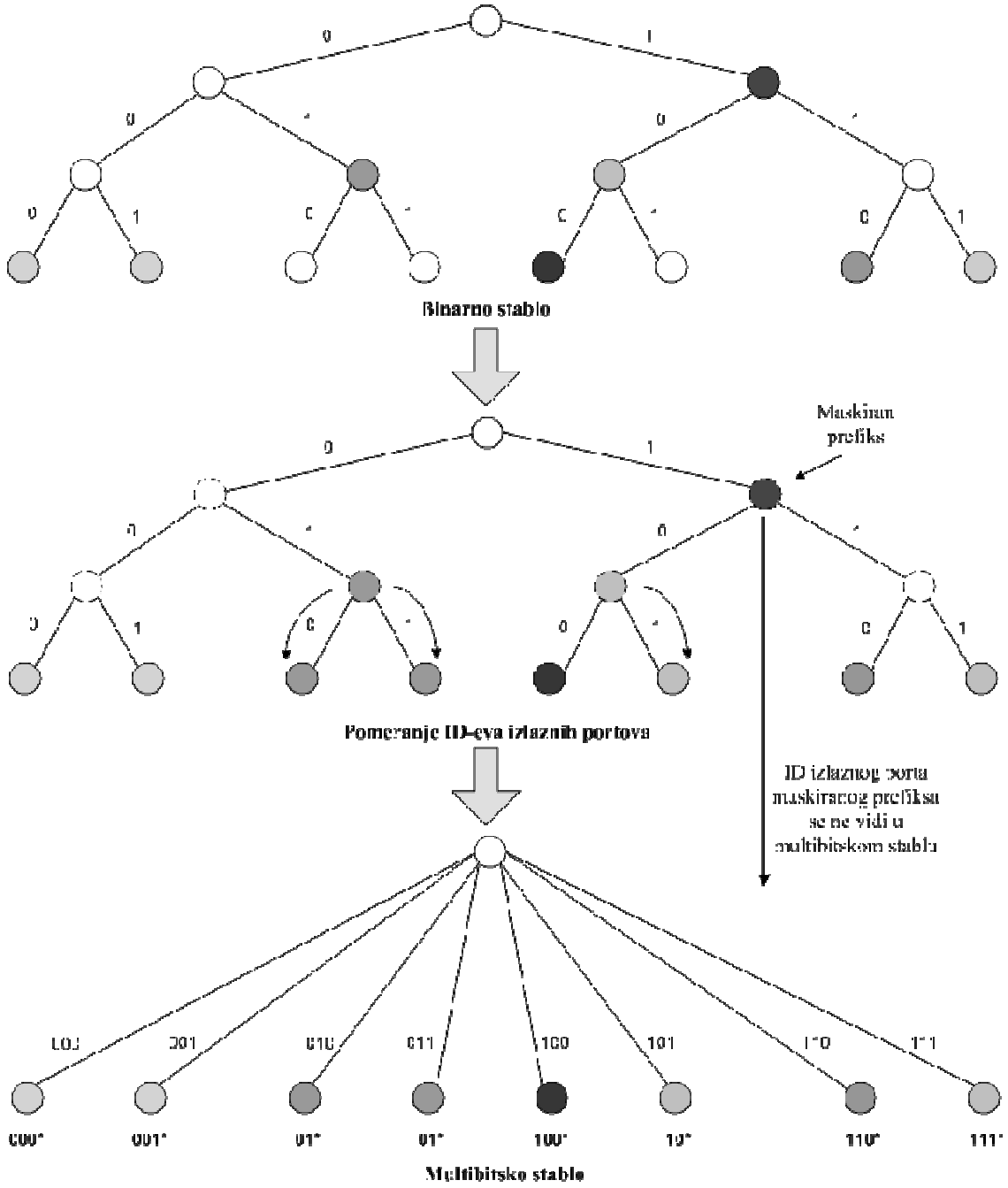
4.3.3. Multibitska stabla



Slika 4.3.3.1. – Formiranje multibitskog stabla sa strajdom dužine 3

Pošto je osnovni problem binarnog stabla velik broj pristupa memoriji usled velikog broja nivoa stabla, može se izvršiti transformacija binarnog stabla u multibitsko stablo tj. m-arno stablo [63], [66], [69]-[70], [73], [75], [77]. Ova tehnika podrazumeva da se pri kretanju kroz stablo ne koristi samo jedan bit, već m bita, pri čemu se tih m bita označava terminom strajd, a parametar m predstavlja dužinu strajda. Na ovaj način se maksimalan broj nivoa stabla smanjuje. Preciznije, multibitsko stablo sa strajdom dužine m ima m puta manje nivoa u odnosu na binarno stablo. Multibitsko stablo ne mora da ima konstantnu vrednost strajda, već ona može da bude promenljiva duž nivoa stabla. Na slici 4.3.3.1 je prikazano formiranje jednog nivoa multibitskog stabla od binarnog stabla pri čemu je strajd dužine 3. Kao što se sa slike može videti, umesto tri koraka da se dođe do čvorova iz poslednjeg nivoa prikazanog binarnog stabla, potreban je samo jedan korak u prikazanom multibitskom stablu. Na ovaj način se prevazilazi problem broja nivoa binarnog stabla, odnosno broja pristupa memoriji, pa je multibitsko stablo u osnovi svih lukap algoritama baziranih na strukturi stabla. Ali, postoje i novi problemi koje uvodi multibitsko stablo. Prvi problem je smanjena granularnost stabla.

Kao što se može videti i sa slike 4.3.3.1, čvorovi iz prvog i drugog nivoa (neosenčeni čvorovi) originalnog binarnog stabla se ne vide, pa se samim tim ne vide ni ID-evi izlaznih portova koji su sadržani u njima. Otuda je neophodno pomeriti ID-eve izlaznih portova iz tih, sada nevidljivih, čvorova u čvorove multibitskog stabla već navedenom tehnikom guranja prefiksa tj. ID-eva izlaznih portova.



Slika 4.3.3.2. – Primer maskiranja prefiksa i pomeranja prefiksa iz čvorova binarnog stabla u čvorove multibitskog stabla

Primer pomeranja ID-eva izlaznih portova je prikazan na slici 4.3.3.2. Guranje prefiksa potencijalno može da dovede i do povećanja memorijskih zahteva u slučaju ređe popunjenih stabala. Razlog je formiranje velikog broja novih čvorova u vidljivim nivoima multibitskog stabla koji nisu postojali u binarnom stablu. Kod ređe popunjenih stabala broj novoformiranih čvorova u multibitskom stablu može da bude značajno veći u odnosu na broj nevidljivih čvorova naročito u slučaju dugih strajdova.

Međutim, kod m-arnih stabala, ažuriranje postaje komplikovanije zbog maskiranja pojedinih ID-eva izlaznih portova usled toga što su čvorovi na koje ti ID-evi treba da se pomere zauzeti dužim prefiksima tj. njihovim ID-evima izlaznih portova. Tako je u primeru sa slike 4.3.3.2 maskiran prefiks 1*. Naime, usled formiranja multibitskog stabla u primeru sa slike 4.3.3.2 ID-evi izlaznih portova prefiksa 01*, 1* i 10* se moraju pomeriti u čvorove koji će postojati i u multibitskom stablu. Otuda se ID izlaznog porta prefiksa 01* pomera u čvorove koji odgovaraju prefiksima 010* i 011* kao na slici 4.3.3.2. ID izlaznog porta prefiksa 10* se pomera samo u čvor koji odgovara prefiksu 101*, jer se u čvoru koji odgovara prefiksu 100* nalazi ID izlaznog porta dužeg prefiksa - 100*. ID izlaznog porta prefiksa 1* bi trebao da se pomeri u čvorove koji odgovaraju prefiksima 100*, 101*, 110* i 111*. Međutim, u svim tim čvorovima su smešteni ID-evi dužih prefiksa pa se ID izlaznog porta prefiksa 1* ne vidi u rezultujućem multibitskom stablu. Ta pojava se označava maskiranjem prefiksa 1*, pošto je on maskiran dužim prefiksima 100*, 10*, 110* i 111*.

U slučaju brisanja nekih ID-eva izlaznih portova iz čvorova koji odgovaraju maskiranom prefiksu, ID izlaznog porta koji odgovara dotičnom maskiranom prefiksu mora da se smesti u te čvorove koji su se ispraznili. Isto tako prilikom brisanja prefiksa mora da se vodi računa u kojim čvorovima se nalazi ID izlaznog porta koji odgovara datom prefiksu da bi se izvršilo pravilno brisanje.

Drugi problem je što se dužina strajda ne može neograničeno povećavati. Razlog je što je broj dece jednog čvora jednak 2^m u slučaju multibitskog stabla sa strajdom dužine m. To znači da su povećani memorijski resursi za čuvanje jednog čvora, jer on sad mora da čuva 2^m pokazivača umesto dva pokazivača kao u slučaju binarnog stabla. Tako npr. ako bi u slučaju IPv4 hteli smanjiti stablo na samo jedan nivo potrebno je postaviti $m = 32$, ali je tada broj dece korena jednak 2^{32} što zahteva prevelike memorijske resurse za čuvanje pokazivača na tu decu. Takođe treba uzeti u

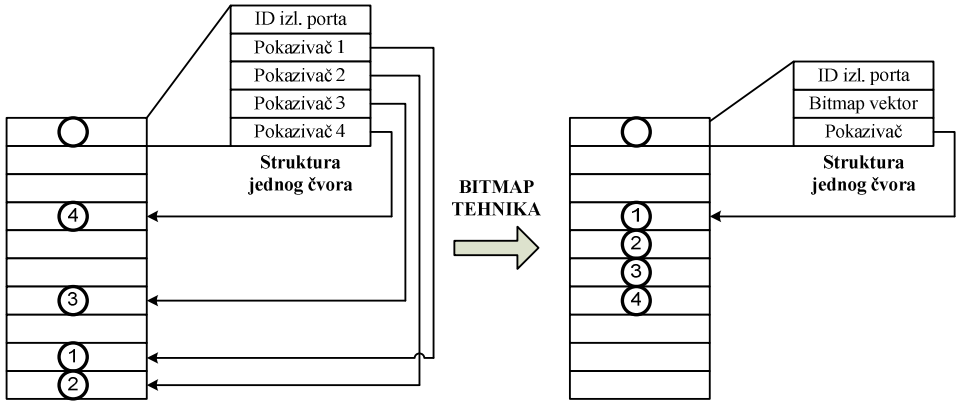
obzir da jedan nevidljivi čvor u najgorem slučaju može da postavi svoj ID izlaznog porta u 2^{m-1} čvorova. Tako da u slučaju kada treba postaviti/promeniti/obrisati ID izlaznog porta u tim čvorovima, potreban je velik broj memorijskih pristupa pa je i ažuriranje nepraktično u tom slučaju.

4.3.4. Bitmap tehnika

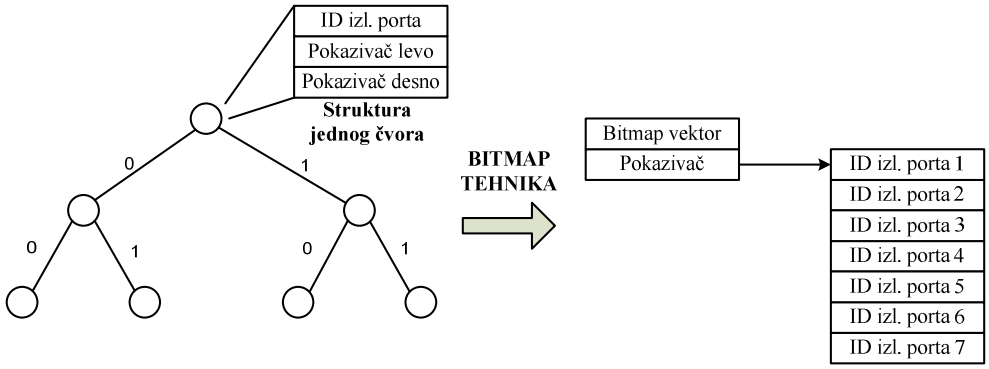
Da bi se prevazišli problemi multibitskih stabala navedenih u prethodnom odeljku i postigle optimalne performanse lukapa uvedene su dodatne tehnike koje unapređuju lukap algoritme bazirane na multibitskim stablima. Jedna od najvažnijih tehnika je bitmap tehnika [59]-[61], [63]-[66], [69], [72]. Bitmap tehnika podrazumeva da se informacije mapiraju u određeni memorijski prostor, a njihova validnost, odnosno postojanje mapira binarnim vektorom, gde '0' na i-toj poziciji vektora označava nevalidnost/nepostojanje i-te informacije, dok '1' označava validnost/postojanje i-te informacije. Pored binarnog vektora, koji se naziva bitmap vektor, čuva se i jedan pokazivač koji pokazuje na početak memorijskog bloka gde su smeštene informacije na koje se odnosi vektor. Pozicija i-te informacije se računa kao i-ti ofset, odnosno, i-ta lokacija od početka memorijskog bloka. Tipična primena bitmap tehnike u multibitskim stablima je čuvanje pokazivača na decu [63], [66], [69], [72] – slika 4.3.4.1.a. Umesto čuvanja 2^m pokazivača, čuva se jedan bitmap vektor dužine 2^m bita i samo jedan pokazivač. Bitmap vektor se koristi za određivanje da li traženo dete postoji ili ne, dok pokazivač pokazuje na početak memorijskog bloka gde se sukcesivno čuvaju deca dotičnog čvora. Ako dete postoji onda se ide na lokaciju dotičnog deteta, a ta lokacija se računa kao ofset u odnosu pokazivač, gde je ofset jednak poziciji deteta u bitmap vektoru. Na ovaj način se vrši značajna ušteda memorijskih resursa za stablo koje se pretražuje, jer je memorijski resurs jednog čvora u stablu značajno manji. Dodatna optimizacija je da se čuvaju samo deca koja postoje, a da u računanje ofseta ulaze samo postojeća deca, a ne pozicija deteta u vektoru (kada u ofset ulaze i nepostojeća deca). Ova dodatna optimizacija obezbeđuje minimalne memorijske resurse, ali po cenu komplikovanijeg menadžmenta memorije jer memorijski blokovi u kojima se čuvaju deca jednog čvora nisu konstantne veličine.

Pored toga bitmap tehnika se može koristiti i za mapiranje delova stabala – tzv. podstabala (slika 4.2.1.4.b) [59]-[61], [64]. Tada se u bloku memorije čuvaju ID-evi izlaznih portova čvorova iz dotičnog podstabla, a bitmap vektor se koristi za

označavanje koji čvorovi sadrže validan ID izlaznog porta, a koji ne, dok jedan pokazivač pokazuje na početak memorijskog bloka. I ovde mogu da se čuvaju memorijski blokovi konstantne dužine koji obuhvataju sve čvorove iz bitmapa ili promenljive dužine koji obuhvataju samo čvorove sa validnim ID-em izlaznog porta. Na ovaj način se takođe štede memorijski resursi, ali se i omogućava da se u jednom koraku proveri podstablo u cilju određivanja da li postoji čvor od interesa koji sadrži ID izlaznog porta, umesto da se vrši kretanje kroz podstablo koje zahteva više koraka. U multibitskom stablu se na ovaj način mogu mapirati nevidljivi čvorovi iz binarnog stabla [63], [66], [69]. Time se mogu izbeći problemi koji se javljaju sa guranjem prefiksa, poput vođenja računa o maskiranim prefiksima, problema pri ažuriranju kada treba modifikovati veći broj čvorova koji odgovaraju nevidljivom prefiksu koji je gurnut i dr.



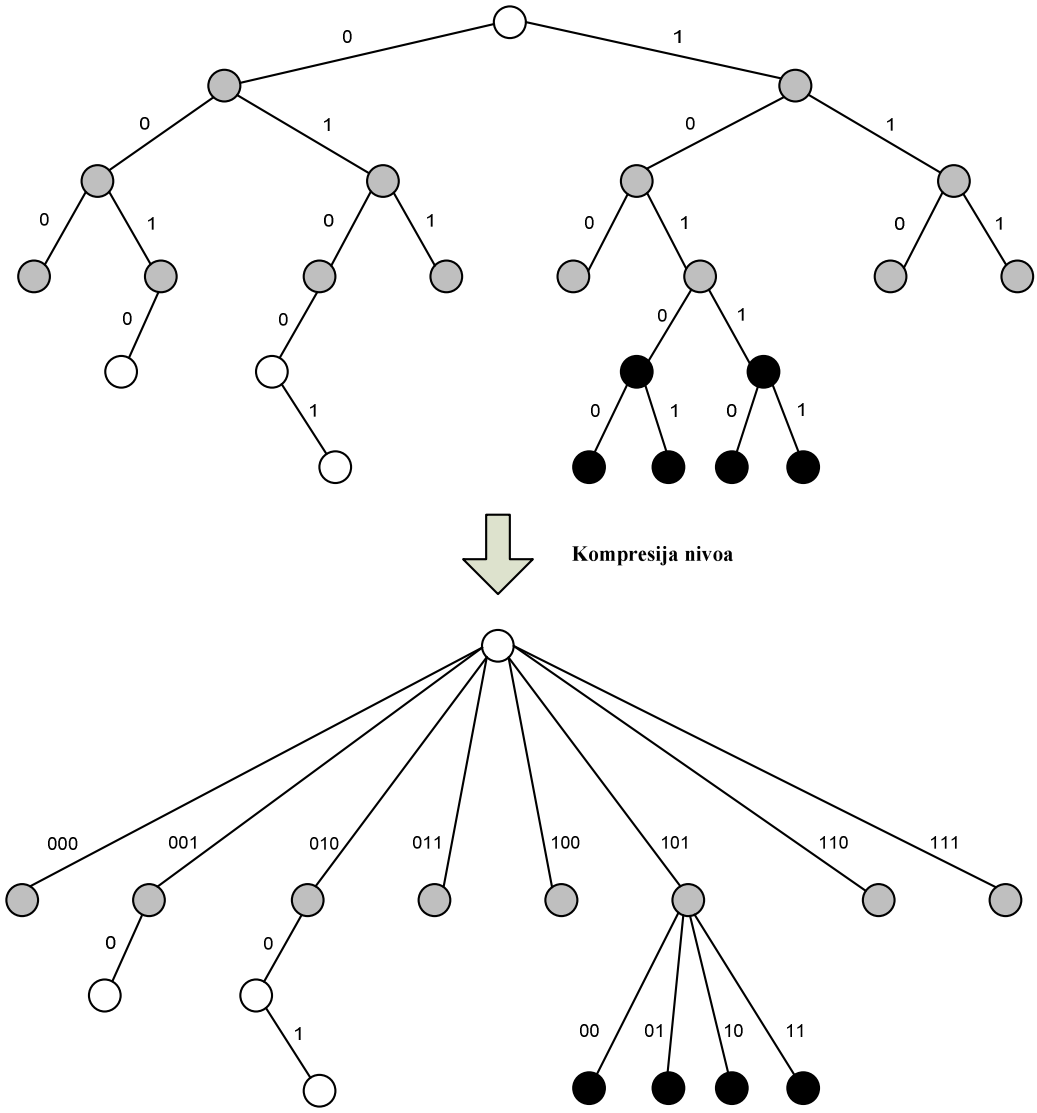
a)



b)

Slika 4.3.4.1. – a) Primena bitmap tehnike u multibitskom stablu; b) Primena bitmap tehnike za delove stabla

4.3.5. Kompresija nivoa



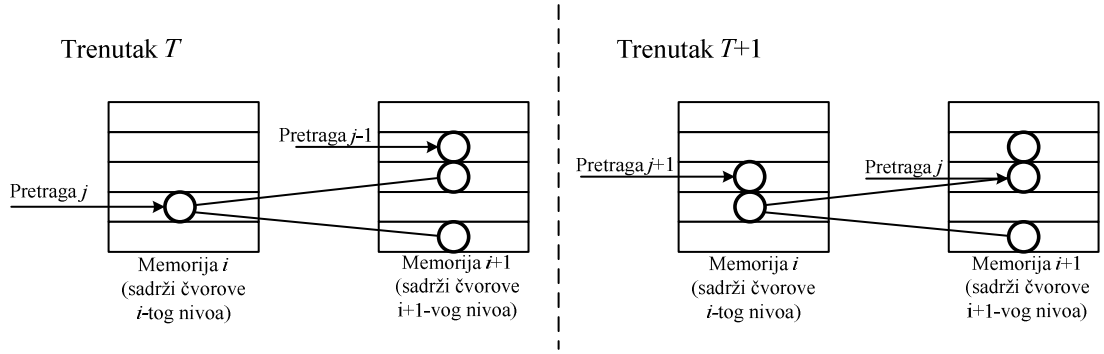
Slika 4.3.5.1. – Primer tehnike kompresije nivoa

Tehnika kompresije nivoa se koristi za optimizaciju strukture binarnog stabla tako što se popunjeni delovi stabla zamene multibitskom strukturom [75]. Na ovaj način se vrši kompresija veoma popunjenih delova stabla. Pri tome, da bi ova tehnika proizvela bolje efekte kompresije, uvodi se prag popunjenosti dela stabla umesto da se zahteva kompletna popunjenost dela stabla. Pa ako je stablo popunjeno iznad praga zamenjuje se multibitskom strukturom. Ova tehnika proizvodi multibitsku strukturu gde je vrednost strajda promenljiva i prilagođava se trenutnoj strukturi binarnog stabla. Problem ove tehnike je u ažuriranju takve strukture, pošto je neophodno neprestano pratiti popunjenost delova stabla i u skladu s tim vršiti kompresije/dekompresije delova stabla što je vremenski zahtevan proces. Na slici 4.3.5.1 je prikazan primer kompresije

nivoa, gde su osenčenim čvorovima označeni delovi stabla koji su zamenjeni multibitskom strukturom. U prikazanom primeru prva tri nivoa su zamenjena jednim multibitskim pri čemu je dužina strajda tri bita (svetlije osenčeni čvorovi), a takođe je deo stabla ispod čvora koji odgovara prefiksu 101* zamenjen multibitskom strukturom dužine strajda od dva bita (tamnije osenčeni čvorovi).

4.3.6. Paralelizacija i pajplajn

Tehnike paralelizacije i pajplajna se koriste za povećanje protoka lukapa koji se meri brojem obavljenih lukapa u jedinici vremena [59]-[62]. One se mogu primeniti u svim klasama lukap algoritama. Tehnika paralelizacije podrazumeva pronalaženje delova algoritma koji mogu da se izvršavaju istovremeno, i oni se implementiraju u odvojenim hardverskim celinama da bi se algoritam ubrzao. U slučaju algoritama zasnovanim na stablima, paralelizacija se sastoji iz razdvajanja delova stabla u posebne hardverske celine čime se omogućava paralelno pretraživanje više delova stabla odjednom i samim tim brže nalaženje rešenja. Pajplajn tehnika se zasniva na tehnici pokretne trake. Ideja je da se lukap modul podeli u hardverske celine kroz koje se lukap obavlja etapno. Cilj je da se iskorišćenje hardverskih resursa maksimizuje tako što će svaka hardverska celina biti zauzeta u svakom trenutku izvršavanjem odgovarajuće etape lukapa za jednu IP adresu. Na taj način hardverske resurse istovremeno koristi više IP paketa čime se značajno povećava protok lukap modula. Tipičan primer paralelizacije i pajplajna je da se svaki nivo stabla smesti u zasebnu memoriju, tako da se pretraga vrši krećući se iz memorije u memoriju [62]. Time čim jedna pretraga za neki paket pređe na sledeću memoriju, sledeća pretraga, za naredni paket, može da koristi trenutnu memoriju kao što je prikazano na slici 4.3.6.1.



Slika 4.3.6.1. – Primer pajplajn tehnike

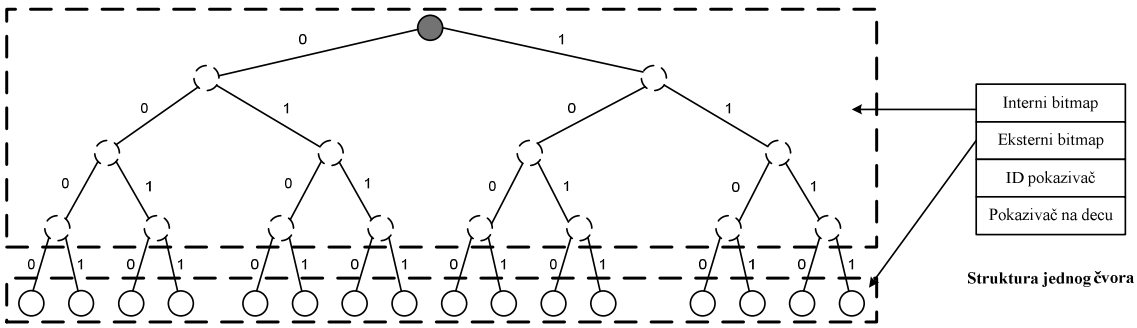
4.3.7. Predstavnici lukap algoritama zasnovanih na strukturi stabla

Kao što je već napomenuto postoji velik broj lukap algoritama koji se zasnivaju na strukturi stabla i u okviru ovog odeljka će biti navedeni i ukratko opisani neki od karakterističnih i poznatih lukap algoritama iz ove klase. U sledećem poglavlju ovi algoritmi će biti poređeni po performansama sa lukap algoritmima koji su predloženi i implementirani u okviru ove teze. Stoga će u nastavku ovog odeljka biti ukratko opisan rad izabranih lukap algoritama, a biće i date opšte formule kojima se karakterišu njihove performanse i koje će se koristiti u sledećem poglavlju za poređenje performansi više lukap algoritama.

i) FIPL algoritam

FIPL (*Fast Internet Protocol Lookup*) algoritam je zasnovan na tzv. TB (*Tree Bitmap*) tehnici [63], [66]. Ideja TB tehnike je da prevaziđe problem nevidljivih internih čvorova tj. čvorova iz nivoa koji se ne vide u m-arnom stablu tako što će u svakom čvoru m-arnog stabla čuvati i tzv. interni bitmap koji odgovara tim čvorovima. Time se izbegava i guranje tih čvorova u vidljive nivoe m-arnog stabla. Na slici 4.3.7.1.1 je prikazana struktura jednog čvora. Čvor sadrži dva bitmapa – interni i eksterni. Interni bitmap definiše koji od internih (nevidljivih) čvorova sadrži validan ID izlaznog porta. Osim internih čvorova u interni bitmap se uključuje i sam posmatrani čvor. Uz interni bitmap se čuva i ID pokazivač koji pokazuje na početak bloka memorijskih lokacija u izlaznoj memoriji koja čuva ID-eve izlaznih portova. Taj blok sadrži onoliko memorijskih lokacija koliko ima internih čvorova, pri čemu je svakom internom čvoru dodeljena jedna lokacija u koju se smešta eventualan validan ID izlaznog porta tog čvora. Eksterni bitmap čuva informaciju o postojanju dece dotičnog čvora i uz njega je vezan pokazivač na decu. Deca se čuvaju u sukcesivnim lokacijama, pri čemu pokazivač pokazuje na prvo dete tj. prvu lokaciju memorijskog bloka gde su smeštena deca. Ukoliko nijedno dete ne postoji pokazivač ima tzv. NULL vrednost kojom se označava da deca ne postoje. Ukoliko postoji bar jedno dete rezervišu se resursi za svu decu, tj. veličina rezervisanog memorijskog bloka je jednaka 2^m čvorova. FIPL koristi strajd dužine $m = 4$ bita, i za svaki podatak u čvoru (interni bitmap, eksterni bitmap, ID pokazivač, pokazivač na decu) koristi $2^m = 16$ bita. Pretraga se sastoji u kretanju kroz m-arno stablo i u svakom čvoru se ispituje interni bitmap u cilju nalaženja čvora sa validnim ID-em izlaznog porta koji se najduže poklapa sa IP adresom za koju se vrši

pretraga. Kada se završi pretraga m-arnog stabla, pristupa se lokaciji u izlaznoj memoriji koja odgovara čvoru sa najdužim poklapanjem da bi se pročitao ID izlaznog porta koja predstavlja konačni rezultat lukap funkcije.



Slika 4.3.7.1.1. – Struktura jednog čvora u FIPL algoritmu

Protok lukap funkcije zavisi od načina implementacije. Ukoliko se svi čvorovi m-arnog stabla nalaze u istoj memoriji onda je protok lukap modula m/L lukapa po ciklusu takta, gde je m dužina strajda, a L dužina IP adrese, jer je u najgorem slučaju neophodno pristupiti L/m puta memoriji koja sadrži čvorove m-arnog stabla. Ukoliko se koristi više memorija, tako da svaka od njih čuva jedan nivo m-arnog stabla, onda je moguće implementirati optimalni pajplajn i tada je garantovani protok lukap funkcije jedan lukap po ciklusu takta.

Memorijski resursi neophodni za smeštanje čvorova m-arnog stabla (M_{ms}) iznose:

$$M_{ms} = N_n \cdot 2^m \cdot (4 \cdot 2^m) \qquad (4.3.7.1.1)$$

gde je N_n broj postojećih nizova od 2^m čvorova jer ako bar jedno dete nekog čvora postoji rezerviše se prostor za svu decu. Kao što je rečeno svi podaci čvora (interni bitmap, eksterni bitmap, ID pokazivač, pokazivač na decu) su iste dužine (2^m) i ima ih četiri, pa je broj bita koje koristi jedan čvor $4 \cdot 2^m$.

Svaki čvor čiji interni bitmap ukazuje na postojanje bar jednog čvora sa validnim ID-em izlaznog porta rezerviše blok od $2^m - 1$ lokacija u izlaznoj memoriji, pošto toliko ima internih čvorova kao što se vidi sa slike 4.3.7.1.1. Otuda su memorijski zahtevi izlazne memorije (M_{iz}):

$$M_{iz} = N_c \cdot (2^m - 1) \cdot H \qquad (4.3.7.1.2)$$

gde je N_c broj čvorova čiji interni bitmap ukazuje na postojanje bar jednog čvora sa validnim ID-em izlaznog porta, a H predstavlja dužinu ID-a izlaznog porta.

U slučaju da postoje eksterni memorijski čipovi na raspolaganju, u prvi eksterni memorijski čip se smešta izlazna memorija jer je ona najveća. U eventualne preostale eksterne memorijske čipove se smeštaju redom smeštaju nivoi m-arnog stabla koji zauzimaju najveće memorijske resurse. Otuda su interni memorijski zahtevi (M_{int}^{i}):

$$M_{int}^{i} = \begin{cases} M_{tot} - M_{iz}, & i = 1 \\ M_{tot} - M_{iz} - \sum_{j=1}^{i-1} M_n^j, & i > 1 \end{cases} \quad (4.3.7.1.3)$$

pri čemu i predstavlja broj eksternih memorijskih čipova na raspolaganju, M_{tot} predstavlja ukupne memorijske resurse, M_n^j je veličina memorije nivoa j pri čemu su memorije nivoa sortirane u opadajućem redosledu po veličini, pa tako indeks $j = 1$ odgovara nivou sa najvećom memorijom. Ukupni memorijski resursi M_{tot} iznose:

$$M_{tot} = M_{ms} + M_{iz} \quad (4.3.7.1.4)$$

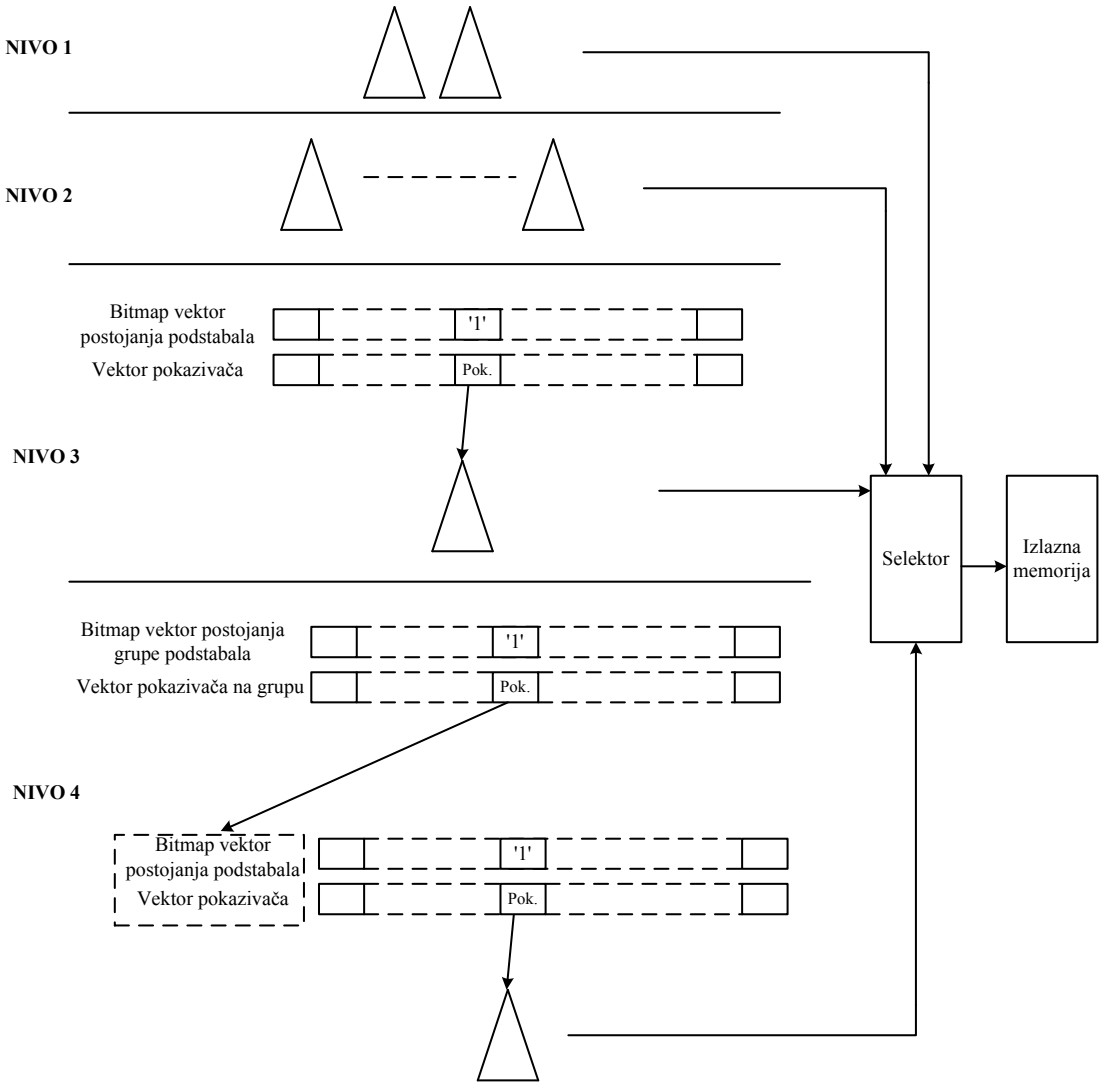
Iz (4.3.7.1.1) se vidi da je veličina memorije jednog nivoa M_n^j:

$$M_n^j = N_n^j \cdot 2^m \cdot (4 \cdot 2^m) \quad (4.3.7.1.5)$$

gde je N_n^j broj postojećih nizova od 2^m čvorova nivoa j.

ii) EHAF algoritam

EHAF (*Efficient Hardware Architecture for Fast IP Lookup*) algoritam je prilagođen za IPv4 adrese i zasnovan je na podeli binarnog stabla na nivoe koji se pretražuju u paraleli [64]. Pri tome je dubina svih nivoa jednaka i iznosi $D = 8$. pa je ukupan broj nivoa četiri. Svaki nivo sadrži podstabla čiji bitmap vektori definišu koji od čvorova sadrže validan ID izlaznog porta. ID-evi izlaznih portova se čuvaju u izlaznoj memoriji. Arhitektura EHAF algoritma je prikazana na slici 4.3.7.2.1. Prvi nivo sadrži dva podstabla dubine $D_p = 7$, pri čemu se na osnovu vrednosti prvog bita IP adrese određuje koje će se od njih pretraživati. Drugi nivo sadrži 2^{D+1} podstabala dubine $D_p = 7$ pri čemu se na osnovu prvih $D + 1$ bita IP adrese određuje koje će se pretraživati. U slučaju prvog i drugog nivoa čuvaju se sva podstabla bez obzira da li je neko od njih prazno, tj. ne sadrži nijedan čvor sa validnim ID-em izlaznog porta.



Slika 4.3.7.2.1. – EHAF arhitektura

U trećem i četvrtom nivou nije ekonomično čuvati sva podstabla (koja su, isto kao i u prva dva nivoa, dubine $D_p = 7$) s obzirom da je njihov broj isuviše velik, pa se onda čuvaju samo neprazna podstabla. Otuda se za podstabla čuva bitmap vektor dužine 2^{2D+1} koji definiše postojanje/nepostojanje svih podstabala na trećem nivou. Uz ovaj vektor se čuva i vektor pokazivača koji čuva pokazivače na podstabla trećeg nivoa, u slučaju da podstablo ne postoji onda pokazivač ima NULL vrednost. U slučaju trećeg nivoa se na osnovu prvih $2D + 1$ bita određuje pozicija u oba vektora i čitaju se podaci na tim pozicijama. Ukoliko se na osnovu pročitanog bita bitmap vektora utvrdi postojanje podstabla onda se ide na lokaciju koja je određena pročitanim pokazivačem iz vektora pokazivača na kojoj se nalazi interni bitmap dotičnog podstabla. U slučaju četvrtog nivoa se čuva bitmap vektor dužine 2^{2D+1} koji određuje postojanje grupe

podstabala četvrtog nivoa koja sadrže čvorove sa istih početnih $2D + 1$ bita. Takođe se čuva i vektor pokazivača koji pokazuju na lokacije koje čuvaju bitmape podstabala u okviru grupe. Uz bitmap koji definiše postojanje/nepostojanje podstabala u okviru grupe se čuva i vektor pokazivača na lokacije koje čuvaju neprazna podstabla (ona koja imaju bar jedan čvor sa validnim ID-em izlaznog porta). U slučaju četvrtog nivoa se pretraga vrši tako što se prvo ispita, na osnovu prvih $2D + 1$ IP adrese, postojanje grupe podstabala u koju spada podstablo od interesa, pa onda ako grupa postoji se ispita njen bitmap postojanja/nepostojanja podstabala. Ako podstablo postoji onda se preko pokazivača pristupa lokaciji koja sadrži bitmap dotičnog podstabla koje se potom ispituje. Treba primetiti da u slučaju prvog i drugog nivoa nisu korišćeni pokazivači pošto se sva podstabla čuvaju pa su resursi neophodni za smeštanje bitmapa podstabala i ID-eva izlaznih portova unapred rezervisani. Rezultati pretraga svih nivoa se vode na selektor koji selektuje pozitivan rezultat najdubljeg nivoa. Potom se pristupa izlaznoj memoriji da bi se pročitao ID izlaznog porta koji predstavlja konačni rezultat lukapa.

Maksimalni i garantovani protok lukapa je jedan rezultat lukapa po jednom ciklusu takta pošto se pajplajn lako implementira u EHAF arhitekturi. Zahtevani memorijski resursi po nivoima (M_n^i) iznose:

$$M_n^1 = 2 \cdot (2^D - 1) \tag{4.3.7.2.1}$$

$$M_n^2 = 2^{D+1} \cdot (2^D - 1) \tag{4.3.7.2.2}$$

$$M_n^3 = 2^{2D+1} \cdot \left(1 + \left\lfloor \log_2 N_{pod}^3 \right\rfloor\right) + N_{pod}^3 \cdot (2^D - 1) \tag{4.3.7.2.3}$$

$$M_n^4 = 2^{2D+1} \cdot \left(1 + \left\lfloor \log_2 N_{gpod}^4 \right\rfloor\right) + N_{gpod}^4 \cdot 2^D \cdot \left(1 + \left\lfloor \log_2 N_{pod}^4 \right\rfloor\right) + \\ + N_{pod}^4 \cdot (2^D - 1) \tag{4.3.7.2.4}$$

gde su N_{pod}^3 i N_{pod}^4 broj nepraznih podstabala na nivoima 3 i 4, respektivno, N_{gpod}^4 predstavlja broj grupa podstabala četvrtog nivoa koja sadrže bar jedno neprazno podstablo. Zahtevani memorijski resursi za izlaznu memoriju (M_{iz}) iznose:

$$M_{iz} = \left[2 + 2^{D+1} + N_{pod}^3 + N_{pod}^4\right] \cdot (2^D - 1) \cdot H \tag{4.3.7.2.5}$$

gde je H dužina ID-a izlaznog porta.

U slučaju da postoje eksterni memorijski čipovi na raspolaganju, u prvi eksterni memorijski čip se smešta izlazna memorija jer je ona najveća. U eventualne preostale eksterne memorijske čipove se redom smeštaju vektori pokazivača na podstabla trećeg

nivoa i grupe podstabala četvrtog nivoa, respektivno. Memorije koje sadrže bitmape podstabala zahtevaju prevelike magistrale podataka pa se zato one ne izmeštaju u eksterne memorije. Otuda su interni memorijski zahtevi (M_{int}^{i}):

$$M_{int}^{i} = \begin{cases} M_{tot} - M_{iz}, & i = 1 \\ M_{tot} - M_{iz} - M_{pk}^{3}, & i = 2 \\ M_{tot} - M_{iz} - M_{pk}^{3} - M_{pk}^{4}, & i = 3 \end{cases} \quad (4.3.7.2.6)$$

pri čemu i predstavlja broj eksternih memorijskih čipova na raspolaganju, M_{tot} predstavlja ukupne memorijske resurse, M_{pk}^{3} predstavlja memorijske resurse za pokazivače na podstabla trećeg nivoa, a M_{pk}^{4} predstavlja memorijske resurse za pokazivače na grupe podstabala četvrtog nivoa. Ukupni memorijski resursi M_{tot} iznose:

$$M_{tot} = M_{n}^{1} + M_{n}^{2} + M_{n}^{3} + M_{n}^{4} + M_{iz} \quad (4.3.7.2.7)$$

Iz (4.3.7.2.3) se vidi da je veličina memorije pokazivača na podstabla trećeg nivoa M_{pk}^{3}:

$$M_{pk}^{3} = 2^{2D+1} \cdot \left(1 + \left\lfloor \log_2 N_{pod}^{3} \right\rfloor\right) \quad (4.3.7.2.8)$$

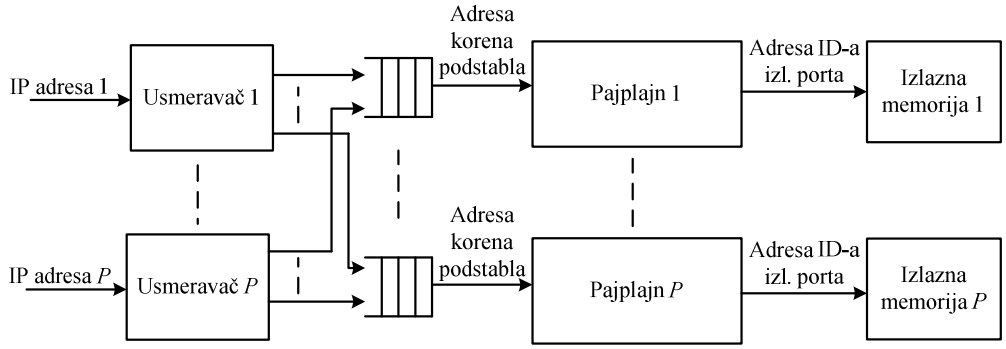
Iz (4.3.7.2.4) se vidi da je veličina memorije pokazivača na grupe podstabala četvrtog nivoa M_{pk}^{4}:

$$M_{pk}^{4} = 2^{2D+1} \cdot \left(1 + \left\lfloor \log_2 N_{gpod}^{4} \right\rfloor\right). \quad (4.3.7.2.9)$$

iii) POLP algoritam

POLP (*Parallelized Optimal Linear Pipeline*) algoritam se zasniva na upotrebi tehnika pajplajna i paralelizacije kao što se i vidi iz njegovog naziva [62]. Arhitektura POLP algoritma je prikazana na slici 4.3.7.3.1. Ideja POLP algoritma je da čvorovi binarnog stabla na dubini D predstavljaju korene podstabala originalnog binarnog stabla. Zatim se tako definisana podstabla rasporede po pajplajnovima kojih ima P, tako da svaki pajplajn sadrži podjednak broj čvorova. Svaki pajplajn se sastoji od F faza kroz koje prolazi pretraga u okviru pajplajna. Svaka faza sadrži memoriju koja čuva čvorove podstabala. Da bi memorije po fazama imale jednake kapacitete, što olakšava implementaciju, vrši se raspoređivanje čvorova podstabala raspoređenih u dotični pajplajn tako da svaka faza sadrži podjednak broj čvorova. Pri tome je jedino važno da deca jednog čvora mogu da budu smeštena samo u fazama posle faze u kojoj je smešten dotični čvor. Pri tome deca ne moraju da budu u prvoj sledećoj fazi, već mogu da budu smeštena i u neku udaljeniju fazu i ta osobina omogućava ravnomerno raspoređivanje

čvorova po fazama. Jedna memorijska lokacija predstavlja jedan čvor podstabla. U okviru memorijske lokacije tj. čvora se čuvaju pokazivač na decu i binarni indikator da li taj čvor sadrži validan ID izlaznog porta ili ne. Pokazivač se sastoji od dve informacije – faza u kojoj se nalaze deca i adresa lokacije u memoriji te faze gde se nalazi prvo dete (drugo dete je smešteno na sukcesivnu lokaciju). Svakom pajplajnu je pridružena jedna izlazna memorija kao što je prikazano na slici 4.3.7.3.1 radi ostvarivanja maksimalnog protoka lukapa. Izlazne memorije čuvaju ID-eve izlaznih portova čvorova podstabala smeštenih u odgovarajućim pajplajnovima. Prefiksi kraći od D bita se guraju u čvorove nivoa D, tj. korene odgovarajućih podstabala.



Slika 4.3.7.3.1. – Arhitektura POLP algoritma

Pretraga se vrši na sledeći način. Odredišna IP adresa se propušta kroz jedan od usmeravača koji na osnovu prvih D bita IP adrese određuje u kom pajplajnu se nalazi podstablo od interesa. Zatim se preostali biti IP adrese prosleđuju ka redu za čekanje odgovarajućeg pajplajna. Red za čekanje je neophodan pošto može doći do kolizije tj. da dva ili više usmeravača istovremeno usmere pretragu u isti pajplajn. Otuda je broj usmeravača jednak P jer veći broj bi sigurno doveo do kolizija, a manji bi ostvarivao manji maksimalan protok lukap funkcije. Pretraga u pajplajnu se zasniva na kretanju kroz binarno podstablo koje se odvija prolaskom kroz faze. Kroz faze se prolazi besposleno ukoliko one ne sadrže čvor od interesa, a u suprotnom se čvor od interesa ispituje i određuje da li sadrži validan ID izlaznog porta i da li postoje deca čvora. Na kraju se u izlaznoj memoriji koja čuva ID-eve izlaznih portova pristupa lokaciji koja odgovara zadnjem ispitanom čvoru, koji sadrži validan ID izlaznog porta, iz pajplajna. Kada pretraga pređe iz jedne faze u narednu, sledeća pretraga dolazi u datu fazu pa je pajplajn optimalan, a pri tome svi pajplajnovi mogu da se pretražuju istovremeno u paraleli, što znači da P pretraga može da otpočne u isto vreme. Maksimalan protok

lukap funkcije je P rezultata pretrage po jednom ciklusu takta. Ali, u najgorem slučaju svi usmeravači mogu da prosleđuju pretrage u isti pajplajn pa je u najgorem slučaju protok lukap modula jednak jedan lukap po jednom ciklusu takta.

Što se tiče memorijskih resursa svi blokovi u POLP arhitekturi sadrže memorije. U usmeravačima, njihove memorije čuvaju za svako podstablo informaciju u koji pajplajn je ono raspoređeno. Broj podstabala je 2^D, a broj bita neophodan za identifikaciju pajplajna je $\lfloor \log_2 P \rfloor$, gde $\lfloor\ \rfloor$ označava prvi veći ceo broj. Tako su zahtevani memorijski resursi u usmeravačima (M_u):

$$M_u = P \cdot 2^D \cdot \lfloor \log_2 P \rfloor \qquad (4.3.7.3.1)$$

Memorija u okviru jedne faze pajplajna se dimenzioniše prema najgorem slučaju tj. najvećem broju čvorova N_c raspoređenih u okviru jedne faze pri čemu se u obzir uzimaju sve faze svih pajplajnova. Kao što je rečeno jedna memorijska lokacija sadrži pokazivač i binarnu indikaciju o tome da li je čvoru pridružen validan ID izlaznog porta. Pokazivač se sastoji od dva dela pa dužina pokazivača iznosi $\lfloor \log_2 (F-1) \rfloor + \lfloor \log_2 N_c \rfloor$, pri čemu prvi član zbira adresira fazu (pošto prva faza ne mora da se adresira jer se od nje kreće, onda je potrebno adresirati samo sledeće faze), a drugi deo zbira adresira lokaciju u memoriji dotične faze. Otuda su zahtevani memorijski resursi neophodni za realizaciju pajplajnova (M_p):

$$M_p = P \cdot F \cdot N_c \cdot (\lfloor \log_2 (F-1) \rfloor + \lfloor \log_2 N_c \rfloor + 1) \qquad (4.3.7.3.2)$$

Memorije na izlazu pajplajna (izlazne memorije) čuvaju ID-eve izlaznih portova, pri čemu broj lokacija jedne izlazne memorije mora biti jednak broju čvorova u okviru pajplajna za koji je vezana. Pošto bilo koja lokacija u memoriji faze može biti zauzeta nekim čvorom podstabla i pošto svaki čvor podstabla može da sadrži validan ID izlaznog porta, memorijski resursi izlaznih memorija (M_{iz}) iznose:

$$M_{iz} = P \cdot F \cdot N_c \cdot H \qquad (4.3.7.3.3)$$

gde je H dužina ID-a izlaznog porta.

Prilikom razmatranja internih memorijskih resursa, neophodno je prvo utvrditi koje je memorijske blokove najbolje izmestiti u raspoložive eksterne memorijske čipove. U slučaju POLP-a najbolje je izmestiti izlazne memorije u eksterne čipove pošto su one najveće. U slučaju da je broj eksternih memorijskih čipova veći od broja izlaznih memorija onda se sledeće izmeštaju memorije unutar pojedinih faza, pri čemu treba

napomenuti da to izmeštanje neće značajno uštediti interne memorijske resurse. Razlog je što su memorije unutar faza znatno manjih kapaciteta u odnosu na izlazne memorije. Interni memorijski zahtevi (M_{int}^{i}) iznose:

$$M_{int}^{i} = \begin{cases} M_{tot} - i \cdot M_{iz}^{1}, & i \leq P \\ M_{tot} - M_{iz} - (i - P)M_{f}, & i > P \end{cases} \quad (4.3.7.3.4)$$

pri čemu i predstavlja broj eksternih memorijskih čipova na raspolaganju, M_{tot} predstavlja ukupne memorijske resurse, M_{iz}^{1} je veličina jedne izlazne memorije, a M_f predstavlja veličinu memorije jedne faze. Ukupni memorijski resursi M_{tot} iznose:

$$M_{tot} = M_u + M_p + M_{iz} \quad (4.3.7.3.5)$$

Iz (4.3.7.3.3) se vidi da je:

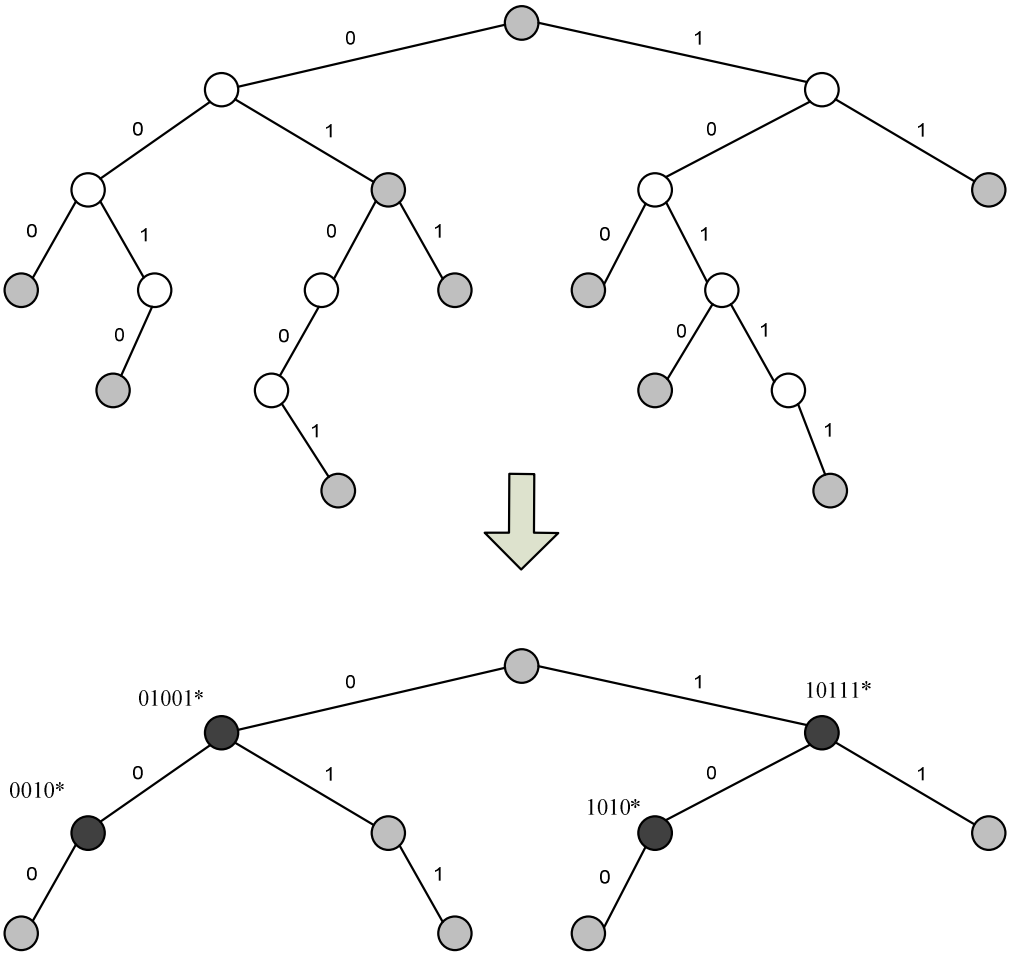
$$M_{iz}^{1} = F \cdot N_c \cdot H \quad (4.3.7.3.6)$$

Iz (4.3.7.3.2) se vidi da je:

$$M_f = N_c \cdot (\lfloor \log_2(F - 1) \rfloor + \lfloor \log_2 N_c \rfloor + 1) \quad (4.3.7.3.7)$$

iv) Stablo sa prioritetima

Stablo sa prioritetima je zasnovano na ideji da se prazni čvorovi u podstablima popune sa nekim od čvorova sa validnim ID-em izlaznog porta koji predstavljaju njihove potomke [76]. Pri tome, selektuje se onaj potomak sa validnim ID-em izlaznog porta koji se nalazi na najvećoj dubini stabla jer on ima najveći prioritet tj. ako se IP adresa poklopi sa njim pretraga se može okončati. Na ovaj način se smanjuje ukupna dubina binarnog stabla i izbegava se problem praznih čvorova koji se koriste samo za prolaz kroz stablo tj. formiranje puta kroz stablo. Međutim, sada čvorovi pored ID-a izlaznog porta i dva pokazivača (na levo i desno dete) moraju da sadrže i dodatne podatke. Jedan podatak je binarni indikator da li je u pitanju pomereni čvor (tzv. čvor sa prioritetom) ili ne, a drugi podatak je vrednost samog prefiksa. Pošto se može desiti da se pomeri i čvor sa najveće dubine binarnog stabla mora se rezervisati L bita za čuvanje prefiksa, gde je L dužina prefiksa. Primer formiranja stabla sa prioritetima je prikazan na slici 4.3.7.4.1, pri čemu su sa tamnijom bojom prikazani čvorovi sa prioritetom (tj. premešteni čvorovi).



Slika 4.3.7.4.1. – Formiranje stabla sa prioritetima

Stablo sa prioritetima se formira na jednostavan način. Ide se redom po nivoima binarnog stabla. Za svaki prazan čvor na koji se naiđe traži se njegov potomak sa validnim ID-em izlaznog porta koji se nalazi na najvećoj dubini, a takav potomak sigurno mora da postoji jer u suprotnom dotični prazni čvor ne bi ni postojao u stablu. Tako npr. za prazan čvor koji odgovara poziciji prefiksa 0* potomak sa validnim ID-em izlaznog porta koji je na najvećoj dubini je čvor koji odgovara prefiksu 01001* i on se pomera u prazan čvor. Na sličan način se popunjavaju i ostali prazni čvorovi. Prazni čvorovi koji ostanu da vise tj. nemaju potomaka sa validnim ID-em izlaznog porta zbog premeštanja se takođe brišu iz stabla. Na ovaj način se optimizuje ukupan broj čvorova u binarnom stablu, a i smanjuje se broj nivoa. Pretraga se vrši kretanjem kao po binarnom stablu pri čemu ukoliko se utvrdi da je trenutni čvor onaj sa prioritetom vrši se poređenje prefiksa sa IP adresom. Ako se nađe poklapanje onda je to kraj pretrage pošto boljeg poklapanja ne može da bude u stablu (jer bi se u suprotnom taj čvor sa

boljim poklapanjem nalazio na mestu ispitivanog čvora sa prioritetom) – otuda i naziv čvor sa prioritetom. Na ovaj način se takođe smanjuje prosečno vreme pretrage.

U slučaju kada su svi čvorovi u istoj memoriji protok lukap funkcije u najgorem slučaju iznosi $1/L$ rezultat lukap funkcije po jednom ciklusu takta, pošto u slučaju da postoji jedna putanja do najveće dubine binarnog stabla na kojoj su svi čvorovi sa validnim ID-em izlaznog porta onda nijedan od tih čvorova ne može biti čvor sa prioritetom pa otuda svi ostaju na svom mestu i mora se proći kroz sve njih ako se vrši pretraga za IP adresu koja odgovara tom putu. Naravno, verovatnoća ovog događaja je zanemarljivo mala, ali je sa teoretskog stanovišta garantovani protok $1/L$. Optimalni pajplajn može da se izvrši u slučaju da se čvorovi svakog nivoa smeštaju u zasebnu memoriju i tada on iznosi jedan rezultat lukapa po jednom ciklusu takta (ali je tada u najgorem slučaju potrebno L memorija).

Memorijski resursi neophodni za smeštanje svih čvorova M_{ps} (jedan čvor sadrži ID izlaznog porta, dva pokazivača (jedan na levo i jedan na desno dete), vrednost prefiksa i binarnu indikaciju da li je u pitanju čvor sa prioritetom) iznose:

$$M_{ps} = N_c \cdot (H + 2 \cdot \lfloor \log_2 N_c \rfloor + L + 1) \qquad (4.3.7.4.1)$$

gde je N_c ukupan broj čvorova prioritetnog stabla (tj. broj prefiksa u lukap tabeli), a H dužina ID-a izlaznog porta. Broj nivoa stabla sa prioritetom koji utiče na protok lukap funkcije zavisi od rasporeda čvorova originalnog binarnog stabla i njihovog ravnomernog rasporeda po stablu jer npr. videli smo da je dovoljna samo jedna putanja čiji svi čvorovi imaju validan ID izlaznog porta pa da se nijedan čvor sa te putanje ne može premestiti.

Da bi se ostvario optimalan pajplajn neophodno je svaki nivo prioritetnog stabla smestiti u zasebnu memoriju, pa u slučaju da postoje eksterni memorijski čipovi na raspolaganju u njih se redom smeštaju nivoi stabla koji zauzimaju najveće memorije. Otuda su interni memorijski zahtevi (M_{int}^{i}):

$$M_{int}^{i} = M_{ps} - \sum_{j=1}^{i} M_n^j \qquad (4.3.7.4.2)$$

pri čemu i predstavlja broj eksternih memorijskih čipova na raspolaganju, M_n^j je veličina memorije nivoa j pri čemu su memorije nivoa sortirane u opadajućem

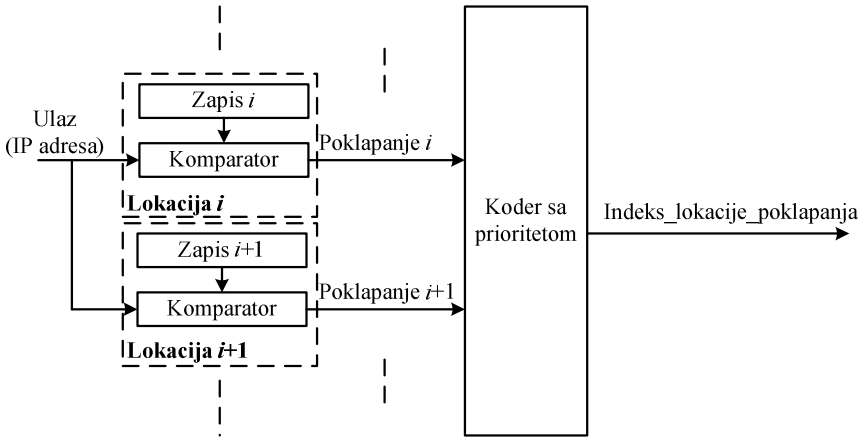
redosledu po veličini, pa tako indeks $j = 1$ odgovara nivou sa najvećom memorijom. Iz (4.3.7.3.1) se vidi da je veličina memorije jednog nivoa M_n^j:

$$M_n^j = N_c^j \cdot (H + 2 \cdot \lfloor \log_2 N_c \rfloor + L + 1) \qquad (4.3.7.4.3)$$

gde je N_c^j broj čvorova nivoa j.

4.4. TCAM algoritmi

U ovu klasu algoritama spadaju svi oni lukap algoritmi koji koriste TCAM memorije [79]-[85]. TCAM memorije su specijalizovani čipovi razvijeni upravo za primenu u lukapu, kao i inspekciji paketa koja je veoma slična lukapu. Logička šema TCAM memorije je prikazana na slici 4.4.1.



Slika 4.4.1. – Logička šema strukture TCAM memorije

TCAM memorija se sastoji od niza lokacija, pri čemu se svaka lokacija sastoji od zapisa i komparatora. Ulazni signal se poredi istovremeno na svakoj lokaciji tako što se u komparatoru porede ulazni signal i zapis. Izlaz lokacije predstavlja rezultat komparacije u vidu binarnog signala čija vrednost '0' kazuje da nema poklapanja, odnosno '1' da ima poklapanja. Pošto više zapisa može da se poklapa sa ulaznim signalima, rezultati komparacije sa svih lokacija se vode na koder sa prioritetom koji na svoj izlaz izbacuje indeks lokacije sa poklapanjem najvišeg prioriteta. U slučaju lukapa ulazni signal je ustvari IP adresa, a zapis je ustvari prefiks. TCAM memorija je specifična po tome što bit u zapisu može da ima jednu od tri vrednosti: '0', '1' i 'X', pri čemu 'X' vrednost predstavlja tzv. '*don't care*' vrednost koja u komparaciji uvek daje poklapanje sa ulaznim signalom na poziciji na kojoj se nalazi. Pošto TCAM memorija ima veoma usko polje primene često proizvođači ne nude samo TCAM memoriju

proizvođačima rutera i svičeva, već složene integrisane čipove koji u sebi, pored TCAM memorije, sadrže i kompletnu logiku za izvršavanje lukapa tj. čipove koji predstavljaju kompletno rešenje.

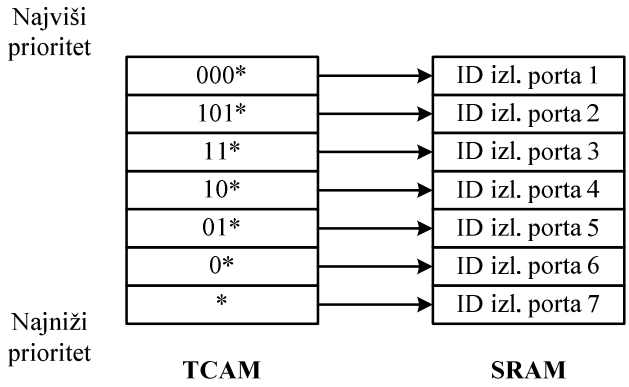
Osnovna implementacija lukapa baziranog na TCAM memoriji je prikazana na slici 4.4.2. U ovoj osnovnoj varijanti u TCAM memoriji se čuvaju svi prefiksi lukap tabele, dok se ID-evi izlaznih portova čuvaju u običnoj memoriji – tipično SRAM memoriji. Pri tome, broj lokacija obe memorije je identičan pa ako je prefiks na lokaciji i TCAM memorije, onda je njegov ID izlaznog porta na lokaciji i SRAM memorije. Lukap se obavlja tako što se IP adresa dovodi na ulaz TCAM memorije, koja nalazi prefiks sa najdužim poklapanjem i vraća indeks lokacije gde je smešten ID izlaznog porta koji odgovara tom prefiksu. Čitanjem dotične lokacije se nalazi ID izlaznog porta kao konačan rezultat pretrage tj. lukapa. Treba napomenuti da su biti prefiksa predstavljeni * upisani sa vrednostima ‘X’ u TCAM memoriju.

Kao što se vidi upotreba TCAM memorije je veoma jednostavna i veoma brzo se nalazi rezultat lukapa. Međutim, postoje i određeni problemi u upotrebi TCAM memorija. Prvi problem je cena. TCAM memorije su značajno skuplje od SRAM memorija koje se inače koriste u drugim vrstama lukap algoritama. Takođe, kapacitet TCAM memorija je mali tako da je problematično smeštanje velikih lukap tabela pošto u osnovnoj varijanti ona treba da smesti sve prefikse, a pri tome širina lokacije mora biti minimalno jednaka dužini IP adrese. Pošto upotreba TCAM memorije podrazumeva istovremenu proveru svih lokacija i njihovo poređenje sa IP adresom, potrošnja ovakvih čipova je velika i može da ide i preko 10W što predstavlja veliko opterećenje za rutere [82]. Zelene tehnologije su savremeni trend koje imaju za cilj smanjenje potrošnje svih elektronskih uređaja, samim tim i mrežnih uređaja. Na primer, u Cisco ruterima velik deo ukupne potrošnje ide na lukap upravo zbog upotrebe TCAM memorija [80]. S druge strane SRAM memorije troše znatno manje snage (manje od 1W), pa su samim tim poželjnije za upotrebu u odnosu na TCAM memorije sa stanovišta potrošnje.

Dodatni problem TCAM memorija je sortiranje prefiksa u TCAM memoriji tako da duži prefiksi idu na lokacije višeg prioriteta. U suprotnom, da se ne vrši sortiranje moglo bi doći do grešaka u rezultatima TCAM memorije. Na primer, da je na slici 4.4.2 prefiks 10* stavljen na lokaciju višeg prioriteta u odnosu na prefiks 101*, došlo bi do greške u rezultatu TCAM memorije za sve IP adrese koje počinju sa 101 jer bi se vraćao

ID izlaznog porta koji odgovara kraćem prefiksu 10* pošto bi koder sa prioritetom selektovao lokaciju 10* jer je višeg prioriteta. Problem sortiranja može da utiče na efikasno ažuriranje prilikom upisa novih prefiksa ako prefiksi u TCAM memoriji nisu dobro raspoređeni.

Razvijena su brojna unapređenja osnovne implementacije lukap algoritama baziranih na TCAM memoriji. Unapređenja za osnovni cilj imaju rešavanje dva osnovna problema TCAM memorije na koja se može uticati – mali kapacitet i velika potrošnja. Otuda lukap algoritmi bazirani na TCAM memoriji imaju za cilj unapređenje u vidu smanjenja potrošnje ili smanjenja potrebne veličine TCAM memorije.

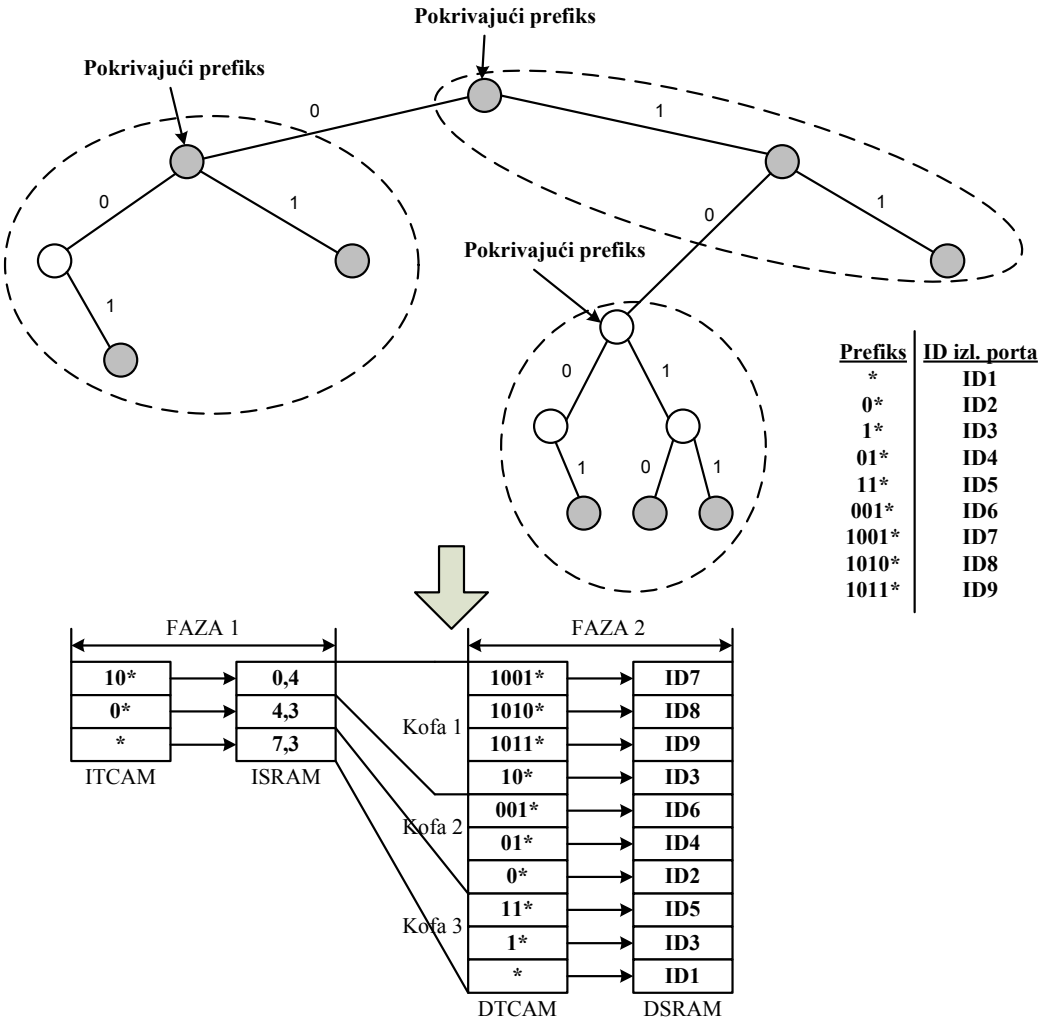


Slika 4.4.2. – Osnovna šema lukap modula baziranog na TCAM memoriji

4.4.1. Optimizacija potrošnje

Potrošnja se može smanjiti tako što se ne porede svi prefiksi u lukap tabeli sa IP adresom, već samo deo prefiksa čime se smanjuje ukupna potrošnja jer je samo deo lukap tabele (tj. TCAM memorije) aktiviran prilikom pretrage [79], [81]-[85]. Ovo se postiže podelom prefiksa u tzv. kofe, a kriterijum podele je takav da je svaka kofa definisana sa jednim ili više tzv. pokrivajućih prefiksa. Pokrivajući prefiks je onaj najduži prefiks koji je zajednički grupi prefiksa u kofi koji su predstavljeni dotičnim pokrivajućim prefiksom. Drugim rečima, ako bismo tu grupu prefiksa predstavili u vidu binarnog stabla, njihov pokrivajući prefiks bi bio zajednički predak te grupe prefiksa koji se nalazi na najvećoj dubini stabla. Grupa prefiksa predstavlja deo podstabla originalnog binarnog stabla. Lukap tabela je u ovom slučaju organizovana kao na slici 4.4.1.1. U prvoj fazi se nalazi par TCAM-SRAM memorija, gde se u TCAM memoriji nalaze pokrivajući prefiksi, a u SRAM se nalaze identifikacije kofa koje sadrže grupe prefiksa koje su pokrivene tim pokrivajućim prefiksima. Cilj pretrage u prvoj fazi je da

se odredi kofa od interesa koja sadrži prefikse koji predstavljaju kandidate za poklapanje sa datom IP adresom. Druga faza takođe sadrži par TCAM-SRAM memorija, ali se sada u TCAM memoriji nalaze kofe sa prefiksima, dok SRAM memorija sadrži ID-eve izlaznih portova. Rezultat druge faze predstavlja konačno rešenje lukapa – ID izlaznog porta. Pošto se prva faza koristi za određivanje indeksa kofe koja treba da se pretraži, onda se memorije prve faze označavaju sa ITCAM (*Index* TCAM) i ISRAM (*Index* SRAM) [79]. Druga faza sadrži krajnji rezultat lukapa pa se memorije druge faze označavaju sa DTCAM (*Data* TCAM) i DSRAM (*Data* SRAM) [79]. Navedeni nazivi memorija će se koristiti u nastavku teze.



Slika 4.4.1.1. – Primer organizacije lukap tabele sa TCAM memorijom podeljenom na kofe promenljive veličine

DTCAM memorija se može implementirati na dva načina. Prvi način je da se koristi jedna TCAM memorija koja ima osobinu da mogu da se aktiviraju samo pojedini

njeni delovi koji ustvari predstavljaju kofe. Drugi način je da se koristi više TCAM memorija manjeg kapaciteta pri čemu svaka ta mala TCAM memorija predstavlja jednu kofu. Kofe se mogu definisati da budu fiksne veličine ili promenljive veličine. U slučaju kada su kofe fiksne veličine, kofe se redom pune prefiksima tako da su u tom slučaju sve kofe sem eventualno poslednje potpuno pune. Iz tog razloga u ovom slučaju jedna kofa može da sadrži više grupa prefiksa tj. da u prvoj fazi postoji više pokrivajućih prefiksa koji pokazuju na istu kofu. Naravno, forsiranje da sve kofe budu potpuno pune može da utiče na povećanje broja pokrivajućih prefiksa u prvoj fazi jer podela na grupe najčešće nije idealna zbog tako strogog kriterijuma, pa neki od predloženih algoritama često ublaže kriterijum popunjavanja kofe do kraja pre prelaska na sledeću. U slučaju kofa promenljive veličine one se projektuju tako da se u svakoj kofi nalazi samo jedna grupa prefiksa tj. u prvoj fazi svaki pokrivajući prefiks pokazuje na različitu kofu. Pri tome se definiše maksimalna veličina kofe kao ograničenje, da bi se unapred definisalo koliko prefiksa može maksimalno ići u jednu kofu.

Prilikom rada TCAM algoritama sa kofama postoji mogućnost da ako se pretraga usmeri na neku grupu prefiksa, da se ne nađe rešenje tj. poklapanje, ali da se poklapanje moglo naći u nekoj drugoj grupi prefiksa koja sadrži kraće prefikse i koja se nalazi u nekoj drugoj kofi. Ukoliko grupi prefiksa pripada i pokrivajući prefiks, onda tako nešto ne može da se desi. Otuda se svakoj grupi kojoj ne pripada pokrivajući prefiks (tj. pokrivajući prefiks ne sadrži validan ID izlaznog porta) veštački dodaje pokrivajući prefiks, a njemu se pridružuje ID izlaznog porta prvog njegovog pretka sa validnim ID-em izlaznog porta na putu od dotičnog pokrivajućeg prefiksa do korena stabla u originalnom binarnom stablu.

U primeru sa slike 4.4.1.1 su formirane tri kofe, pri čemu je uzeto da su kofe promenljive veličine tako da svakoj kofi odgovara samo jedan pokrivajući prefiks. ISRAM memorija sadrži podatak o početnoj lokaciji kofe i veličini kofe. Pokrivajući prefiks 10* nema validan ID izlaznog porta tako da se njemu veštački pridružuje ID izlaznog porta njegovog pretka 1*, jer se, u suprotnom, za slučaj IP adrese čiji su početni biti 1000 u kofi čiji je pokrivajući prefiks 10* ne bi našlo rešenje, a ono postoji u drugoj kofi koja sadrži prefiks 1*. Može se videti da se u najgorem slučaju aktiviraju maksimalno 4 lokacije u DTCAM memoriji. Imajući u vidu da u primeru postoji 9 prefiksa, osnovna TCAM varijanta bi imala aktivaciju 9 lokacija, pa je ušteda očigledna

i u ovom malom primeru. Naravno, ako uzmemo u obzir da se u toku jedne pretrage aktiviraju i 3 lokacije u ITCAM memoriji onda imamo ukupnu aktivaciju od 7 lokacija u toku jedne pretrage u najgorem slučaju. Vidimo da je ostvarena ušteda u odnosu na osnovno rešenje oko 22%. Naravno, u slučaju većih tabela odnos veličina ITCAM memorije i kofa je znatno povoljniji u odnosu na osnovnu varijantu pa su i uštede značajnije i idu i do nekoliko puta manje potrošnje snage u odnosu na osnovnu varijantu.

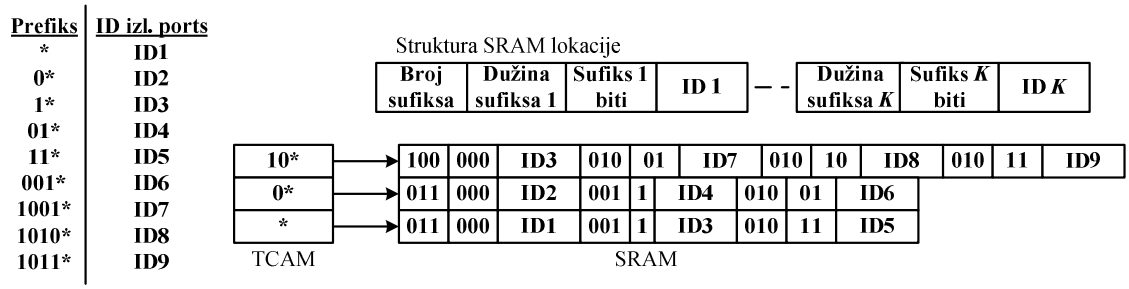
4.4.2. Optimizacija kapaciteta

Pored smanjenja potrošnje lukap algoritama zasnovanih na TCAM memoriji, neophodno je smanjiti i njihov zahtevani kapacitet, pošto je kapacitet TCAM memorije ograničen. Kao što se moglo videti u primeru iz prethodnog odeljka, smanjenje potrošnje ima za cilj smanjenje aktiviranih delova TCAM memorije, ali ukupna veličina je i dalje ista kao i u osnovnoj varijanti. Štaviše, može čak i da bude veća zbog dodavanja pokrivajućih prefiksa ukoliko je to potrebno kao i u primeru sa slike 4.4.4.1 gde je dodat prefiks 10* čime je DTCAM memorija veća nego što bi bila da je korišćena osnovna varijanta. Otuda je bitno voditi računa i o smanjenju traženog kapaciteta TCAM memorije pošto je u pitanju skup resurs. Smanjenje potrebnog kapaciteta TCAM memorije je jedino moguće ako se deo prefiksa smesti van TCAM memorije npr. u SRAM memoriju [79]. Dakle, SRAM memorija bi pored ID-eva izlaznih portova trebalo da čuva i druge informacije. Samim tim neophodne su SRAM memorije koje sadrže duže lokacije. Iz lokacija se podaci čitaju jednim pristupom, pa one omogućavaju brzo čitanje tih dodatnih informacija. Današnje SRAM memorije mogu da imaju i 144b dugačke lokacije [79]. U tom slučaju bi se u TCAM memoriji čuvali pokrivajući prefiksi, a u SRAM memoriji bi se čuvali sami prefiksi zajedno sa svojim ID-evima izlaznih portova. Prefiksi se ne čuvaju u kompletnom formatu, već samo u vidu sufiksa u odnosu na pokrivajući prefiks. Tako da bi svaki prefiks bio predstavljen u vidu: broj bita u sufiksu, konkretna vrednost sufiksa i ID izlaznog porta koji odgovara dotičnom prefiksu. Naravno, pored ovih informacija morala bi da se čuva i informacija o ukupnom broju prefiksa koji su smešteni u lokaciji.

Na ovaj način se može značajno smanjiti ukupan kapacitet TCAM memorije. Ako kao primer uzmemo da može biti maksimalno 5 sufiks bita i da ID izlaznog porta sadrži 16 bita, onda je za predstavu jednog prefiksa u SRAM potrebno 16+5+3=24 bita,

gde je 3 bita neophodno za kodiranje broja sufiks bita. U tom slučaju za 5 prefiksa je neophodno 120 bita, i pri tome je još neophodno 3 bita za kodiranje ukupnog broja prefiksa u lokaciji. To znači da sa ukupno 123b možemo smestiti 5 prefiksa po lokaciji, što je tehnički ostvarljivo. Tako bi u idealnom slučaju ukupan kapacitet TCAM memorije mogao da se smanji i do 5 puta. Primer ovog metoda je prikazan na slici 4.4.2.1, pri čemu je korišćena ista lukap tabela kao u primeru sa slike 4.4.1.1. U primeru je uzeto da se može smestiti do 4 prefiksa po jednoj lokaciji tako da su pokrivajući prefiksi isti kao i u primeru sa slike 4.4.1.1. Takođe je uzeto 3 bita za kodiranje dužine sufiksa. U SRAM lokacijama su prikazane samo informacije od značaja, neiskorišćeni biti na krajevima lokacija nisu prikazani. Važna napomena je da i ovde kao u slučaju algoritama zasnovanih na TCAM, a koji se bave uštedom potrošnje snage, ukoliko pokrivajući prefiks nema validan ID izlaznog porta onda mora da se doda u SRAM lokaciju pri čemu mu se pridružuje ID izlaznog porta od njegovog najbližeg pretka sa validnim ID-em izlaznog porta.

Kombinacijom prethodno izloženih tehnika za uštedu potrošnje snage i smanjenja potrebnog kapaciteta TCAM memorija se unapređuju njihove performanse u oba aspekta - potrošnji snage i zahtevanim memorijskim resursima.



Slika 4.4.2.1. – Primer organizacije lukap tabele zasnovane na TCAM gde su prefiksi smešteni u SRAM memoriju

4.4.3. Predstavnici TCAM lukap algoritama

U okviru ovog odeljka će biti predstavljeni i analizirani najpoznatiji predstavnici TCAM lukap algoritama. Pri tome treba napomenuti da su TCAM memorijski resursi tretirani kao interni memorijski resursi jer TCAM memorije u stvari predstavljaju specijalizovane integrisane čipove poput ASIC-a.

i) Osnovni TCAM algoritam

Implementacija osnovnog TCAM algoritma je bila prikazana na slici 4.4.2. Najpre se TCAM memorija koristi za određivanje prefiksa koji se najduže poklapa sa

zadatom IP adresom, a onda se u SRAM memoriji pročita ID izlaznog porta koji je pridružen dotičnom prefiksu. Pri tome je adresa lokacije u SRAM memoriji kojoj se pristupa jednaka adresi lokacije u TCAM gde je nađeno najduže poklapanje. Protok lukap modula je jedan lukap po jednom ciklusu takta jer se može implementirati optimalni pajplajn.

Broj zapisa u TCAM memoriji je jednak broju prefiksa u lukap tabeli. Pri tome se širina jedne lokacije u TCAM memoriji mora dimenzionisati prema najgorem slučaju, odnosno mora biti jednaka dužini IP adrese koja iznosi L bita. SRAM sadrži samo ID-eve izlaznih portova pa je širina lokacije u SRAM memoriji jednaka dužini ID-a izlaznog porta H. Na osnovu toga zahtevani memorijski resursi TCAM memorije (M_t) iznose:

$$M_t = N_p \cdot L \tag{4.4.3.1.1}$$

gde je N_p broj postojećih prefiksa u lukap tabeli. Zahtevani memorijski resursi SRAM memorije (M_s) iznose:

$$M_s = N_p \cdot H \tag{4.4.3.1.2}$$

Sa slike 4.4.2 se može videti da je potreban samo jedan eksterni memorijski čip koji bi smeštao ID-eve izlaznih portova i tada su interni memorijski zahtevi M_{int}^{i} jednaki zahtevanom kapacitetu TCAM memorije M_t:

$$M_{int}^{i} = M_t, \;\; i \geq 1 \tag{4.4.3.1.3}$$

gde je i broj eksternih memorijskih čipova na raspolaganju. Očigledno je iz (4.4.3.1.3), odnosno slike 4.4.2, da u osnovnom TCAM algoritmu nema smisla uvoditi više od jednog eksternog memorijskog čipa, pošto se TCAM memorija zbog svoje složenosti tretira kao interna memorija, kao što je rečeno u uvodnom delu odeljka 4.4.3.

ii) TCAM algoritam sa kofama promenljive veličine

Primer rada ovog algoritma je prikazan na slici 4.4.1.1. Cilj algoritma je smanjenje potrošnje koju izaziva upotreba TCAM memorije tako što će se broj lokacija koje se aktiviraju u TCAM memoriji smanjiti [79], [82], [84]. Taj efekat se postiže smeštanjem prefiksa u kofe sa prefiksima koji se zajedno aktiviraju, a koje su u ovom slučaju promenljive veličine. Pri tome se definiše maksimalna veličina kofe K koja definiše maksimalan broj prefiksa po kofi. U ovom algoritmu postoje dve faze pretrage, pri čemu svaka faza sadrži po jednu TCAM i SRAM memoriju. ITCAM memorija čuva

tzv. pokrivajuće prefikse koji identifikuju kofe, a ISRAM memorija čuva pokazivače na kofe koji se sastoje od dve informacije – pokazivač na početak kofe i veličina kofe. Rezultat pretrage u prvoj fazi daje pokazivač na kofu koju treba pretražiti. U drugoj fazi, DTCAM memorija sadrži kofe sa prefiksima, i pretražuje se samo ona kofa koja je određena rezultatom pretrage u prvoj fazi. Kao rezultat pretrage kofe dobija se adresa u DSRAM memoriji koja sadrži ID izlaznog porta koji odgovara najdužem poklapajućem prefiksu i koja predstavlja konačni rezultat lukapa.

Maksimalna moguća dužina pokrivajućeg prefiksa L_{pp} je određena maksimalnom veličinom kofe. Najgori slučaj je kada se na dnu binarnog stabla nalazi potpuno popunjeno podstablo dubine D. Tada to podstablo ima ukupno (uključujući i koren) $2^{D+1} - 1$ čvorova. Da bi ovo podstablo stalo u kofu čiji bi pokrivajući prefiks bio koren dotičnog podstabla neophodno je da bude ispunjen uslov $K \geq 2^{D+1} - 1$, odnosno $D \leq \log_2(K+1) - 1$, pri čemu treba uzeti u obzir da je D ceo broj. To znači da čvorovi iz poslednjih $D - 1$ nivoa originalnog binarnog stabla nikako ne mogu biti pokrivajući prefiksi. Otuda je maksimalna dužina pokrivajućeg prefiksa L_{pp}:

$$L_{pp} = L - \lceil \log_2(K+1) - 1 \rceil + 1 \qquad (4.4.3.2.1)$$

pri čemu $\lceil\ \rceil$ označava prvi ceo broj manji ili jednak od proračunate vrednosti. Iz toga sledi da je širina lokacije u ITCAM jednaka L_{pp}, dok je širina jedne lokacije u DTCAM i dalje L jer ona čuva prefikse unete u lukap tabelu. Otuda zahtevani memorijski resursi za ITCAM i DTCAM memorije (M_{it} i M_{dt}, respektivno) iznose:

$$\begin{aligned} M_{it} &= N_k \cdot L_{pp} \\ M_{dt} &= L \cdot \sum_{i=1}^{N_k} N_p^i \end{aligned} \qquad (4.4.3.2.2)$$

gde je N_k broj kofa (odnosno pokrivajućih prefiksa), a N_p^i broj prefiksa u kofi i. Pri tome treba imati na umu da broj prefiksa u DTCAM memoriji može da bude veći od ukupnog broja prefiksa unetih u lukap tabelu zbog toga što se mogu u DTCAM memoriju dodati i neki pokrivajući prefiksi koji ne pripadaju skupu prefiksa unetih u lukap tabelu. SRAM memorije imaju isti broj lokacija kao i TCAM memorije s kojima su upareni u okviru odgovarajuće faze. Zahtevani memorijski resursi za ISRAM i DSRAM memorije (M_{is} i M_{ds}, respektivno) iznose:

$$M_{is} = N_k \cdot \left(\left\lfloor \log_2 \sum_{i=1}^{N_k} N_p^i \right\rfloor + \lfloor \log_2 K \rfloor \right)$$
$$M_{ds} = H \cdot \sum_{i=1}^{N_k} N_p^i \qquad (4.4.3.2.3)$$

gde je H dužina ID-a izlaznog porta. Jedna memorijska lokacije ISRAM memorije sadrži pokazivač na početak kofe u DTCAM memoriji i veličinu same kofe, a veličine ovih delova su redom članovi sume u zagradi izraza za M_{is}. DSRAM čuva samo ID-eve izlaznih portova pa je širina jedne njene lokacije H.

Iz date analize se vidi da su potrebna dva eksterna memorijska čipa za izmeštanje svih memorijskih resursa koji nisu obuhvaćeni TCAM memorijama. U slučaju da je na raspolaganju samo jedan eksterni memorijski čip tada se u njega izmešta DSRAM pošto on zahteva veći kapacitet, a ISRAM bi se morao implementirati u okviru integrisanog čipa u kom je smeštena kontrolna logika algoritma. Na osnovu navedenog interni memorijski resursi (M_{int}^i) su:

$$M_{int}^i = \begin{cases} M_{it} + M_{dt} + M_{is}, & i = 1 \\ M_{it} + M_{dt}, & i \geq 2 \end{cases} \qquad (4.4.3.2.4)$$

iii) M-12Wb algoritam

Cilj M-12Wb (*Many-1 2-Level TCAM with Wide Memory Lookup*) algoritma je da se minimizuju TCAM memorijski resursi, a da pri tome i potrošnja bude smanjena u odnosu na osnovnu TCAM arhitekturu [79]. Kao što smo videli u prethodno opisana dva algoritma, svi postojeći prefiksi se smeštaju u TCAM memoriju. Štaviše, u TCAM algoritmu sa kofama promenljive veličine su ti resursi čak i veći jer se pored postojećih prefiksa u TCAM memorije smeštaju i pokrivajući prefiksi. Ovde se primenjuje koncept prikazan na slici 4.4.2.1 prema kome se deo prefiksa smešta u SRAM memoriju sa dugačkim lokacijama.

Arhitektura M-12Wb algoritma je prikazana na slici 4.4.3.3.1. Ideja je da se originalno binarno stablo podeli na podstabla, pri čemu se formiraju grupe podstabala. Svaka grupa podstabala je predstavljena pokrivajućim prefiksom koji predstavlja zajedničkog pretka na najvećoj dubini svih podstabala iz grupe. Pokrivajući prefiksi se smeštaju u ITCAM memoriju, a koreni podstabala se smeštaju u ISRAM memoriju. U ISRAM memoriji je svakom korenu podstabla pridružena i identifikacija kofe u

DTCAM memoriji. Cilj pretrage prve faze je određivanje kofe u DTCAM memoriji koja sadrži prefikse podstabla u kom treba da se izvrši pretraga za prefiksom koji se najduže poklapa sa datom IP adresom. DTCAM memorija je podeljena na kofe fiksne veličine. Podstabla se mapiraju na kofe, pri čemu više podstabala može da bude mapirano na istu kofu. U kofi se čuvaju korenovi delova podstabala koji su mapirani na kofu, a u odgovarajućim DSRAM lokacijama se čuvaju sami prefiksi u vidu sufiksa u odnosu na odgovarajuće korene delova podstabala. Pri tome se uz prefikse u DSRAM čuvaju i ID-evi izlaznih portova koji i predstavljaju konačni rezultat lukapa. Prilikom pretrage odgovarajuće kofe u DTCAM memoriji se određuje deo podstabla u kom se nalazi najduži poklapajući prefiks, a zatim se u odgovarajućoj lokaciji DSRAM memorije nalazi najduže poklapajući prefiks i njegov ID izlaznog porta koji predstavlja konačni rezultat lukapa.

Današnje SRAM memorije mogu da imaju i do 144b dugačke lokacije [79]. Dugačke lokacije omogućavaju smeštanje veće količine informacija. ISRAM u svakoj svojoj lokaciji čuva nekoliko korena podstabala, čime se smanjuje kapacitet ITCAM memorije. DSRAM u svakoj svojoj lokaciji čuva nekoliko čvorova, pa je samim tim smanjen kapacitet DTCAM memorije. Očigledno, upotrebom SRAM memorija dugačkih lokacija je značajno smanjena veličina TCAM memorija, pri čemu je izvršena i ušteda po pitanju potrošnje.

Širina lokacije ITCAM memorije je jednaka maksimalnoj dužini pokrivajućeg prefiksa L_{pp} i može se koristiti izraz (4.4.3.2.1) izveden u prethodnom pododeljku 4.4.3.2. Za širinu lokacije DTCAM memorije L_{pp2} se može koristiti sličan izraz:

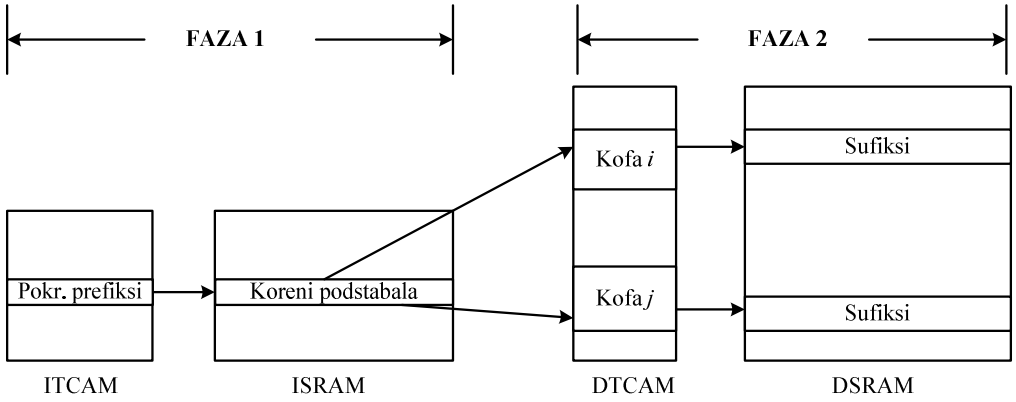
$$L_{pp2} = L - \lceil \log_2(N_c + 1) - 1 \rceil + 1 \qquad (4.4.3.3.1)$$

gde je N_c minimalan broj čvorova koji stanu u jednu lokaciju DSRAM memorije. Sa stanovišta dužine SRAM lokacije L_s, što je duža to će više korena podstabala stati u nju u prvoj fazi, odnosno prefiksa u drugoj fazi pa će samim tim biti i manji memorijski resursi TCAM memorija. Naravno, ograničenje širine lokacije SRAM memorije je tehničke prirode. Zahtevani memorijski resursi za ITCAM i DTCAM memorije (M_{it} i M_{dt}, respektivno) iznose:

$$\begin{aligned} M_{it} &= N_{pp} \cdot L_{pp} \\ M_{dt} &= N_k \cdot K \cdot L_{pp2} \end{aligned} \qquad (4.4.3.3.2)$$

gde je N_{pp} broj pokrivajućih prefiksa, a N_k broj kofa u TCAM memoriji druge faze. Zahtevani memorijski resursi za ISRAM i DSRAM memorije (M_{is} i M_{ds}, respektivno) iznose:

$$M_{is} = N_{pp} \cdot L_s$$
$$M_{ds} = N_k \cdot K \cdot L_s \qquad (4.4.3.3.3)$$



Slika 4.4.3.3.1. – Arhitektura M-12Wb algoritma

Iz date analize se vidi da su potrebna dva eksterna memorijska čipa za izmeštanje svih memorijskih resursa koji nisu obuhvaćeni TCAM memorijama. U slučaju da je na raspolaganju samo jedan eksterni memorijski čip tada se u njega izmešta DSRAM pošto on zahteva veći kapacitet. Na osnovu navedenog interni memorijski resursi (M_{int}^{i}) su:

$$M_{int}^{i} = \begin{cases} M_{it} + M_{dt} + M_{is}, & i = 1 \\ M_{it} + M_{dt}, & i \geq 2 \end{cases} \qquad (4.4.3.3.4)$$

pri čemu i predstavlja broj eksternih memorijskih čipova na raspolaganju. Očigledno je iz (4.4.3.3.4), odnosno slike 4.4.3.3.1, da u slučaju M-12Wb algoritma nema smisla uvoditi više od dva eksterna memorijska čipa, pošto se TCAM memorije zbog svoje složenosti tretiraju kao interne memorije, kao što je rečeno u uvodnom delu odeljka 4.4.3.

4.5. Algoritmi bazirani na heširanju

Heš funkcije vrše kompresiju podataka. One se koriste za preslikavanje podataka iz prostora veće dimenzije u prostor manje dimenzije. Spektar primene heš funkcija je veoma širok. One se koriste npr. u bazama podataka, za mapiranje rečnika, u kriptografiji, itd. U zavisnosti od primene varira i njihova kompleksnost. Kompleksnost

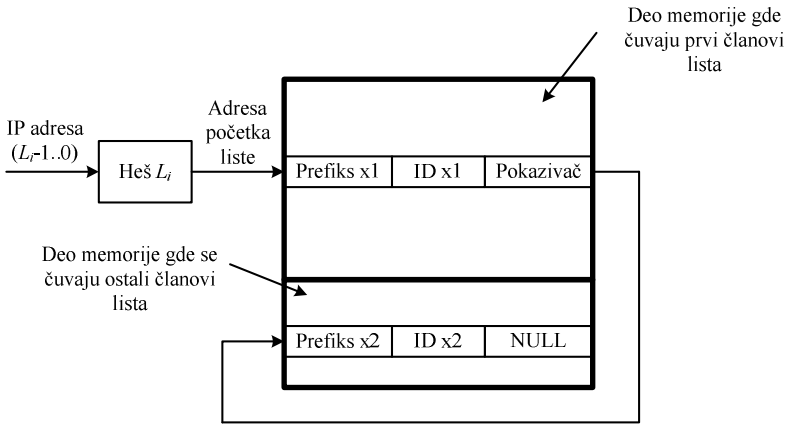
heš algoritama je najveća u slučaju kriptografske primene. S druge strane, u slučaju lukap algoritama mogu da se koriste i jednostavne heš funkcije, po cenu njihovih lošijih performansi. Zahtevi koje heš funkcija mora da zadovoljava u slučaju lukap algoritama su:

- svi biti ulaznog podatka treba da se koriste unutar heš funkcije
- sve izlazne vrednosti moraju da imaju podjednaku verovatnoću pojavljivanja
- ulazni podaci bliskih vrednosti ne smeju da daju izlazne rezultate bliskih vrednosti

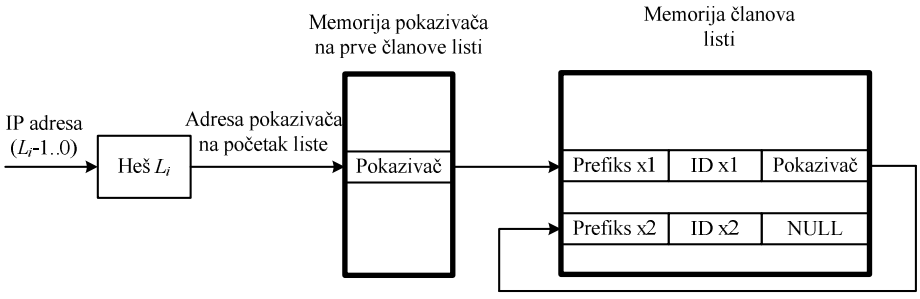
Prvi uslov sprečava da prefiksi koji se razlikuju samo u pozicijama bita koji se ne koriste, ne mapiraju na isti izlazni rezultat heš funkcije. Pri tome je važno napomenuti da ulazni podatak mora da bude konstantne dužine. U suprotnom, ako su podaci različite (promenljive) dužine prvi uslov ne bi bio ispunjen ili heš funkcija ne bi mogla da se izračuna. U slučaju ulaznog podatka čiji je broj bita veći od broja bita koji ulaze u proračun heš funkcije onda svi njegovi biti ne bi bili iskorišćeni pa prvi uslov ne bi bio ispunjen. S druge strane, ako bi se heš funkcija dimenzionisala po ulaznom podatku sa najvećim brojem bita, onda se javlja problem da u slučaju kraćih ulaznih podataka pojedini biti nisu definisani pa heš funkcija ne može da se izračuna. Drugi uslov je bitan da se prefiksi pravilno rasporede po izlaznom prostoru. S obzirom da se ulazni prostor komprimuje na manji izlazni prostor, očigledno je da postoji verovatnoća da se više ulaznih podataka mapira na isti izlazni rezultat. Ovaj događaj da za dva različita ulazna podatka heš funkcija generiše isti izlazni rezultat se naziva kolizija. Drugi uslov minimizuje verovatnoću kolizije. Treći uslov je bitan pošto su često ulazni podaci međusobno bliski pa je bitno da u izlaznom prostoru budu međusobno udaljeniji da bi se smanjila verovatnoća kolizije.

Osnovna varijanta lukap algoritma baziranom na heš funkciji je data na slici 4.5.1 [86]. Pošto postoji zahtev da ulazni podatak bude konstantne dužine, mora da se definiše heš funkcija za svaku moguću dužinu prefiksa. U primeru sa sike 4.5.1 je prikazan deo lukap tabele koja sadrži prefikse dužine L_i. Heš funkcija kao ulaz uzima odgovarajući broj početnih bita IP adrese (u primeru sa slike 4.5.1 prvih L_i bita) i računa lokaciju početka liste prefiksa koji su mapirani na isti heš. Jedna memorijska lokacija može da sadrži jedan ili više mapiranih prefiksa, u cilju povećanja brzine pretraživanja liste. Na slici 4.5.1.a je prikazan slučaj gde je u jednoj lokaciji smešten samo jedan

prefiks sa svojim ID-em izlaznog porta uz pokazivač na sledećeg člana liste. Memorija je podeljena na dva dela, prvi deo je rezervisan za prve članove liste koji su indeksirani rezultatom heš funkcije, a drugi deo sadrži ostale članove listi na koje pokazuju pokazivači članova liste koji im prethode.



a)



b)

Slika 4.5.1. – a) Osnovni lukap heš algoritam; b) Lukap heš algoritam kod kojeg su izdvojeni pokazivači na prve članove liste

Da bi se smanjio broj kolizija, neophodno je da prostor koji adresira heš funkcija bude značajno veći od broja prefiksa. U tom slučaju velik broj lokacija u delu memorije rezervisanom za početne članove listi neće biti iskorišćen. Stoga je efikasnije sa stanovišta ukupnih memorijskih resursa koristiti dve memorije. Prvu memoriju adresira heš funkcija i u njoj se čuvaju samo pokazivači na prve članove listi. Druga memorija se koristi za čuvanje samih članova liste. Ovaj slučaj sa dve memorije je prikazan na slici 4.5.1.b. Na ovaj način je izbegnuto nepotrebno trošenje memorijskih resursa tako što je smanjena širina lokacija koje se ne bi uopšte koristile, a pri tome njihov broj ostaje isti kao u primeru sa slike 4.5.1.a, pa je ušteda očigledna. U primeru sa slike 4.5.1 se podrazumeva da se sve dužine prefiksa ispituju u paraleli. To podrazumeva da za svaku

dužinu prefiksa postoji posebna memorija sa listama mapiranih prefiksa odgovarajućih dužina. Taj broj memorija može da predstavlja problem, pa se u nekim varijantama sve liste svih dužina prefiksa čuvaju u istoj memoriji. Tada se po principu binarne pretrage pretražuju dužine pojedinačno dok se ne dođe do rešenja (u najgorem slučaju posle $\log_2 L$ pokušaja, gde je L dužina IP adrese tj. maksimalna moguća dužina prefiksa). U ovom slučaju se uz postojeće prefikse moraju zapisivati i pomoćne informacije koje usmeravaju pretragu na veću ili manju dužinu, što uz usporavanje pretrage predstavlja dodatan problem jer se povećavaju memorijski zahtevi. Pomoćne informacije su neophodne jer u slučaju neuspeha pretrage na nekoj dužini L_i treba znati da li da se pretraga usmeri ka većim ili manjim dužinama prefiksa.

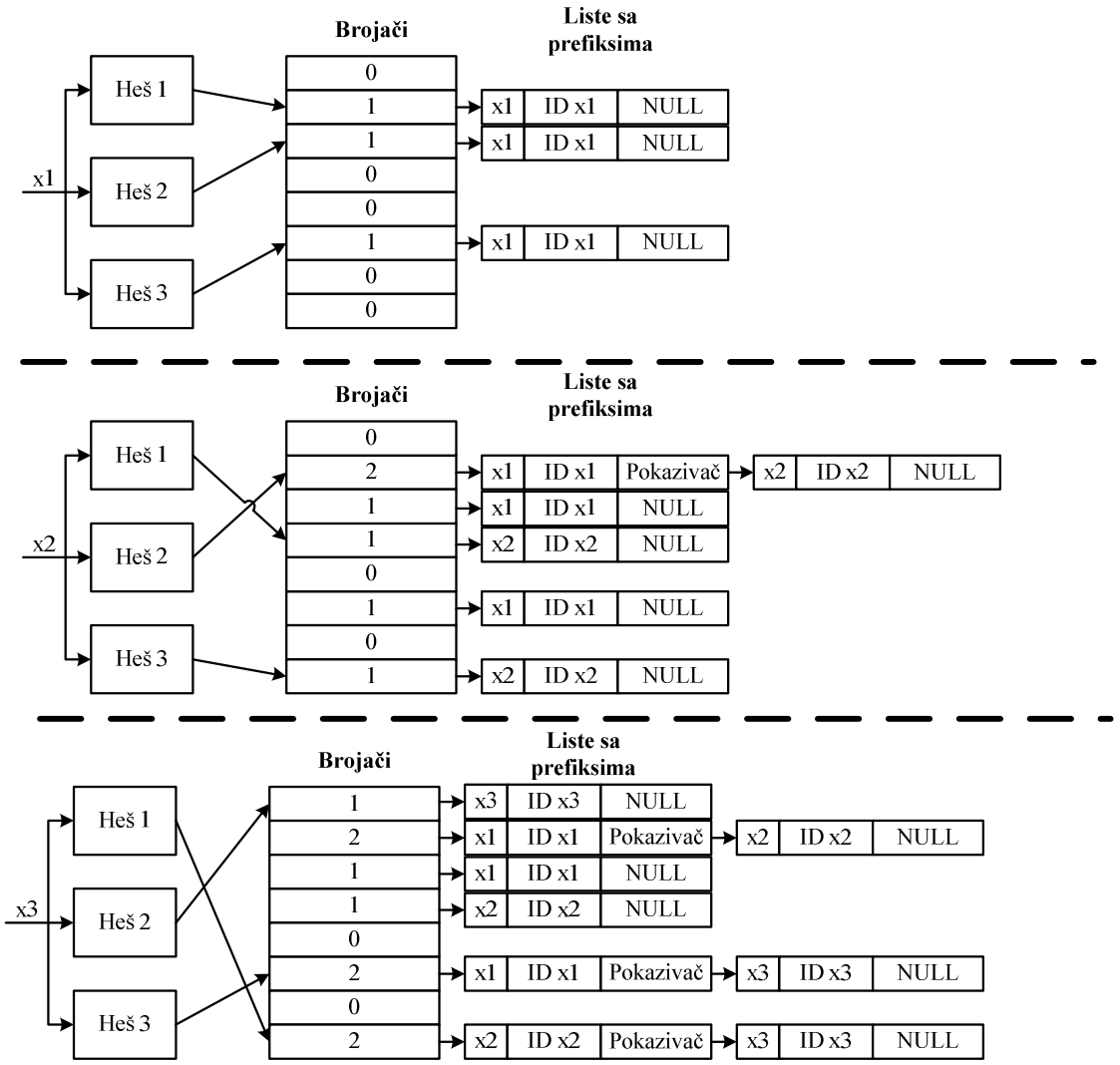
Dobra osobina heš funkcija je što se u jednom koraku dobija rezultat heš funkcije, ali mana je što postoji mogućnost kolizije pa je potrebno čuvati kompletnu vrednost prefiksa da bi bili sigurni da traženi prefiks zaista i postoji, što povećava memorijske zahteve. Takođe, zbog mogućnosti kolizije potrebno je u najgorem slučaju i više memorijskih pristupa da bi ispitali sve mapirane prefikse koji su u koliziji što usporava pretragu, i otežava uvođenje tehnika poput pajplajna za ubrzanje lukapa. Veliki problem je i što se mora formirati heš funkcija za svaku dužinu prefiksa, a njih ima mnogo, naročito u IPv6 slučaju. Otuda se često koristi tehnika guranja prefiksa na određeni manji skup dužina da bi se smanjio broj potrebnih heš funkcija i memorija [88]. Međutim, posledica guranja je povećanje broja prefiksa koje treba mapirati čime se značajno povećavaju i memorijski zahtevi, ali i verovatnoća kolizija. Otuda se heš funkcije često koriste u kombinaciji sa drugim klasama lukap algoritama, da bi se s jedne strane iskoristila dobra osobina jednostavnog i brzog računanja heš funkcije, a s druge strane da bi se osobinama drugih klasa smanjile mane heš funkcija [89].

4.5.1. Blum filtri

Da bi se prevazišli problemi kolizije u osnovnoj varijanti lukap algoritma baziranog na heš funkciji, uvedeni su lukap algoritmi bazirani na tzv. Blum filtrima [86], [90]. Blum filtre je prvi predložio *Burton Howard Bloom*, pa su po njemu i dobili naziv [91]. Blum filtri su memorijski efikasna struktura koja se koristi za ispitivanje da li je neki element član skupa [92]. Osnovna ideja Blum filtara je da se koristi N_{bf} heš funkcija koje treba da daju N_{bf} indeksa. Kada se upisuje podatak u takvu strukturu na

svaku od N_{bf} proračunatih pozicija (indeksa) se upisuje binarna 1. Otuda Blum filtar daje strukturu koja veoma brzo proverava postojanje nekog podatka u tabeli, što je osnovna prednost Blum filtara. Naime, kad se želi proveriti postojanje nekog podatka, samo se proračuna N_{bf} indeksa i ukoliko se na svakoj od pozicija nalazi binarna jedinica onda je velika verovatnoća da dotični podatak postoji u tabeli. U slučaju kada je unet velik broj podataka u tabelu može da dođe do generisanja 'lažno pozitivno' rezultata, koji kaže da podatak postoji u tabeli, a u suštini on se ne nalazi u tabeli. Ovo se dešava zato što drugi podaci koji su uneti u tabelu generišu binarne jedinice na pozicijama koje odgovaraju dotičnom podatku za koji je dobijen 'lažno pozitivan' rezultat. S druge strane, ukoliko se bar na jednoj poziciji nalazi binarna 0, onda podatak sigurno ne postoji. Blum filtri su popularni jer daju efikasnu strukturu po pitanju memorijskih zahteva. Pri tome u zavisnosti od očekivanog broja podataka koji će se uneti u tabelu treba projektovati broj heš funkcija i broj lokacija (indeksa) koje predstavljaju rezultat tih heš funkcija. Kvalitetnim izborom tih parametara se verovatnoća generisanja 'lažno pozitivnog' rezultata može smanjiti na prihvatljivu vrednost, a da pri tome memorijski zahtevi ostanu prihvatljivi. Međutim, problem predstavlja brisanje podataka koje je praktično nemoguće, sem ako se ne čuvaju dodatne informacije. Iz tog razloga se u lukap algoritmima koriste tzv. Blum filtri sa brojanjem [86].

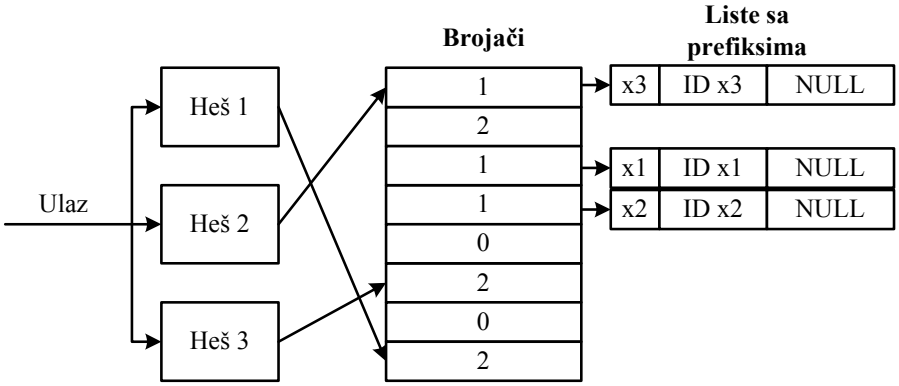
U slučaju Blum filtara sa brojanjem se umesto binarnih vrednosti koriste brojači za određivanje postojanja podataka u tabeli. Svakom indeksu je dodeljen brojač umesto binarne vrednosti. Svaki put kad se upisuje novi podatak brojači na proračunatim lokacijama se inkrementiraju, a sam podatak se dodaje u listu svake lokacije. Na slici 4.5.1.1 je prikazan primer dodavanja tri prefiksa, pri čemu je u primeru uzeto da je $N_{bf} = 3$. Kada se vrši pretraga, prvo se ispituje da li su brojači na svim proračunatim lokacijama različiti od 0. Ako jesu onda se vrši pretraga u listi koja odgovara lokaciji sa najmanjom vrednošću brojača, jer ta lista ima minimalan broj elemenata. Problem ovakve realizacije je što svakom prefiksu odgovara N_{bf} kopija u tabeli, pri čemu se za svaki element u suštini vrši pretraga samo jedne liste u kojoj je smešten (ona sa najmanjom vrednošću brojača), što predstavlja neefikasno korišćenje memorijskih resursa. Otuda je uvedena modifikacija gde se samo jedan primerak prefiksa smešta i to u onu listu kojoj odgovara najmanja vrednost brojača od proračunatih lokacija, kao što



Slika 4.5.1.1. – Primer upisa prefiksa u lukap tabelu zasnovanu na Blum filtru sa brojanjem

je prikazano u primeru sa slike 4.5.1.2. Međutim, ova metoda, iako optimizuje memorijske resurse, ne optimizuje i dubinu listi, pa je zbog toga predloženo menjanje vrednosti brojača u cilju minimizovanja veličina listi čime se ubrzava pretraga tj. lukap. Naravno, posledica je otežano ažuriranje takve strukture pošto se mora voditi računa koji brojači su menjani. Kod korišćenja brojača je takođe nezgodno što treba da se unapred predvidi koliko prefiksa može maksimalno da se smesti po jednoj listi da bi se znalo koliko bita treba da se koristi za formiranje brojača. Takođe, i dalje ostaje problem da strukture sa Blum filtrima treba implementirati i pretraživati na svim dužinama. Otuda se i u ovom slučaju često koristi tehnika guranja prefiksa, sa sličnim problemom povećanja broja prefiksa kod originalnog lukap algoritma baziranog na heširanju.

Osnovna prednost heširanja sa Blum filtrima je što se pomoću Blum filtara odmah utvrdi da li prefiks date dužine postoji ili ne, bez ispitivanja samih članova liste, te se značajno ubrzava određivanje dužine prefiksa. Naravno, postoji i verovatnoća 'lažno pozitivno' događaja da utvrdimo da postoji prefiks putem ispitivanja brojača, a da se taj prefiks ne nalazi u listi. Međutim, kako je već rečeno, pravilnim izborom parametara Blum filtra ovu verovatnoću je moguće svesti na malu vrednost. Takođe, dobrim izborom parametara je moguće optimizovati i veličine listi tako da se pretrage listi skrate. Međutim, ostaje da je, zbog problema 'lažno pozitivno' događaja i zbog toga što liste ipak mogu da sadrže različit broj elemenata, teško uvesti tehnike poput pajplajna za ubrzavanje lukapa. Stoga se često Blum filtri koriste na dužinama koje sadrže najveći procenat prefiksa u lukap tabeli, a za ostale dužine se koriste neke druge metode pretrage [90].



Slika 4.5.1.2. – Primer čuvanja samo jednog zapisa po prefiksu u lukap tabeli zasnovanoj na Blum filtru

4.5.2. Predstavnici lukap algoritama baziranih na heširanju

U narednim pododeljcima će biti analizirani osnovni lukap heš algoritam, kao i jedan karakterističan predstavnik lukap algoritama baziranih na Blum filtrima.

i) Osnovni lukap heš algoritam

Osnovni lukap heš algoritam je prikazan na slici 4.5.1. pri čemu će u okviru ovog pododeljka biti analizirana varijanta prikazana na 4.5.1.b. Da bi se smanjila verovatnoća kolizije prefiksa neophodno je da rezultujući prostor heš funkcije bude dovoljno veći od broja prefiksa koji se heširа odnosno unosi u lukap tabelu. Otuda je varijanta prikazana pod 4.5.1.b memorijski efikasnija jer se u celokupnom prostoru koji adresira heš funkcija čuvaju samo pokazivači na potrebne informacije (prefiks, ID izlaznog porta, pokazivač na sledećeg člana liste) koje se nalaze u posebnoj memoriji. Ta druga

memorija čuva podatke samo za heširane prefikse tj. samo za one prefikse koji su uneti u lukap tabelu. Zato se koriste dve memorije. Jedan član liste sadrži tri podatka: vrednost prefiksa, ID izlaznog porta i pokazivač na sledećeg člana liste. Pri tome treba imati na umu da struktura prikazana na slici 4.5.1.b postoji za svaku moguću dužinu prefiksa tj. ima L takvih struktura koje rade u paraleli.

Memorijski zahtevi memorije pokazivača (M_p) i memorije listi (M_l) iznose:

$$M_p = \sum_{i=1}^{L} (N_m^i \cdot \lfloor \log_2 N_p^i \rfloor)$$
$$M_l = \sum_{i=1}^{L} \left[N_p^i \cdot \left(i + H + \lfloor \log_2 N_p^i \rfloor \right) \right] \quad (4.5.2.1.1)$$

gde je N_p^i broj prefiksa dužine i unetih u lukap tabelu, N_m^i ukupan broj različitih memorijskih lokacija koje mogu da se dobiju heš funkcijom koja se vrši nad prefiksima dužine i, H dužina ID-a izlaznog porta. Kao što je već rečeno primena heš funkcija ne omogućava laku implementaciju pajplajna zbog varijabilne dužine listi. Otuda je garantovani protok lukap modula obrnuto proporcionalan maksimalnoj dužini liste gledano po svim dužinama prefiksa.

U slučaju i IPv4 i IPv6 adresa je tipično da je jedna dužina prefiksa dominantna u odnosu na druge u pogledu broja prefiksa. Tako npr. kod IPv4 adresa najveći broj prefiksa ima dužinu 24, a kod IPv6 najeveći broj prefiksa ima dužinu 48. Samim tim u slučaju upotrebe eksternih memorijskih čipova prvo se izmeštaju memorije za dužinu prefiksa koja sadrži najveći broj prefiksa. Pri tome se u zavisnosti od odnosa N_p^i i N_m^i selektuje koja od memorija se prvo izmešta - memorija pokazivača ili memorija listi. Tipično je memorija pokazivača veća jer da bi se značajno smanjio broj kolizija neophodno je da važi $N_m^i \gg N_p^i$. Na osnovu navedenog interni memorijski resursi (M_{int}^i) su:

$$M_{int}^i = \begin{cases} M_{tot} - \max\left(M_p^{max}, M_l^{max}\right), & i = 1 \\ M_{tot} - M_p^{max} - M_l^{max}, & i = 2 \end{cases} \quad (4.5.2.1.2)$$

pri čemu i predstavlja broj eksternih memorijskih čipova na raspolaganju, M_{tot} ukupne memorijske resurse, M_p^{max} i M_l^{max} veličinu memorije pokazivača i liste na dužini prefiksa koja sadrži najveći broj prefiksa. Ukupni memorijski resursi M_{tot} iznose:

$$M_{tot} = M_p + M_l \quad (4.5.2.1.3)$$

U slučaju da su tri eksterna memorijska čipa na raspolaganju, u treći eksterni memorijski čip se izmešta, u zavisnosti koja je veća, memorija pokazivača ili memorija liste sledeće najpopunjenije dužine prefiksa. Interni memorijski resursi u slučaju tri memorijska čipa na raspolaganju iznose:

$$M_{int}^3 = M_{int}^2 - \max\left(M_p^{max2}, M_l^{max2}\right) \quad (4.5.2.1.4)$$

ii) PFHT algoritam

PFHT (*Prunned Fast Hash Table Lookup*) algoritam je zasnovan na primeni Blum filtra prikazanom na slici 4.5.1.2, pri čemu se prikazana struktura mora implementirati za svaku dužinu prefiksa [86]. Kao što je već rečeno Blum filtri vrše heširanje prefiksa sa N_{bf} heš funkcija. Rezultati heš funkcija izazivaju inkrementiranje brojača koji se nalaze na lokacijama koje predstavljaju rezultat tih heš funkcija. Prefiks koji se dodaje u tabelu se smešta u listu na koju ukazuje brojač sa najmanjom vrednošću uzimajući u obzir samo one brojače koji se nalaze na lokacijama koje su dobijene kao rezultat heš funkcija. Na ovaj način se za svaki dodati prefiks čuva samo jedna njegova kopija čime se optimizuju memorijski resursi. Pretraga se vrši u paraleli po svim dužinama. Na svakoj dužini se odgovarajući deo IP adrese propusti kroz N_{bf} heš funkcija i odrede se lokacije brojača koje treba ispitati. Odgovarajući brojači se ispituju i selektuje se lista onog brojača koji ima najmanju vrednost, pri tome ako je ta vrednost 0 onda to znači da na dotičnoj dužini sigurno ne postoji odgovarajući prefiks i u tom slučaju će pretraga po dotičnoj dužini prefiksa sigurno biti neuspešna. Uz brojače se u memoriji pokazivača, koja ima isti broj lokacija koliko ima i brojača, čuvaju pokazivači na početke odgovarajućih listi radi smanjenja potrebnih memorijskih resursa. Zatim se vrši pretraga liste da bi se ispitalo da li postoji odgovarajući prefiks i ako postoji uzima se njegov ID izlaznog porta kao rešenje pretrage za datu dužinu prefiksa. Kao konačni rezultat se uzima rešenje koje odgovara najvećoj dužini prefiksa. Kao i u osnovnoj varijanti sa heširanjem i ovde je teško implementirati optimalni pajplajn zbog varijabilne dužine listi, pa je garantovani protok lukap modula obrnuto proporcionalan maksimalnoj dužini liste gledano po svim dužinama prefiksa.

Zahtevani memorijski resursi memorije pokazivača (M_p) i memorije listi (M_l) iznose:

$$M_p = \sum_{i=1}^{L} (N_{br}^i \cdot \lfloor \log_2 N_p^i \rfloor)$$
$$M_l = \sum_{i=1}^{L} \left[N_p^i \cdot \left(i + H + \lfloor \log_2 N_p^i \rfloor \right) \right] \quad (4.5.2.2.1)$$

gde je N_p^i broj prefiksa dužine i unetih u lukap tabelu, N_{br}^i ukupan broj brojača koji se koristi na dužini prefiksa i, odnosno broj različitih memorijskih lokacija koje mogu da se dobiju heš funkcijama koja se vrši nad prefiksima dužine i, H dužina ID-a izlaznog porta. Da bi se brojači mogli istovremeno ispitati oni se moraju čuvati u registrima, a ne u memoriji, pa je broj neophodnih registara (u bitima) jednak:

$$R = \sum_{i=1}^{L} (N_{br}^i \cdot b^i) \quad (4.5.2.2.2)$$

gde je b^i broj bita korišćenih za predstavu brojača na dužini prefiksa i.

Sa stanovišta analize internih memorijskih resursa važe identični zaključci kao u slučaju osnovnog lukap heš algoritma, pri čemu je u slučaju PFHT algoritma memorija pokazivača sigurno veća od memorije listi jer N_{br}^i mora biti značajno veći od N_p^i da bi Blum filtar bio kvalitetan. Otuda se za PFHT mogu primeniti izrazi (4.5.2.1.2) i (4.5.2.1.3), uz napomenu da će u okviru funkcije *max* biti selektovana memorija pokazivača.

5. PREDLOG NOVIH LUKAP ALGORITAMA

U prethodnom poglavlju su izložene osnovne klase lukap algoritama, a takođe je data i analiza osnovnih predstavnika lukap algoritama iz svake klase. Ovo poglavlje sadrži ključne doprinose ove teze u vidu originalnih lukap algoritama sa naprednim karakteristikama. Osnovna prednost predloženih algoritama je da imaju male memorijske zahteve čak i u slučaju velikih IPv4 i IPv6 tabela. Na ovaj način omogućen je najviši stepen paralelizacije i pajplajna što omogućava visok protok lukap modula. Ove osobine čine predložene lukap algoritme atraktivnim za slučaj brzih linkova, i lukap tabela sa velikim broj prefiksa.

U ovoj tezi su predloženi originalni lukap algoritam BPFL (*Balanced Parallelized Frugal Lookup*), kao i dve njegove modifikacije BPFLSM (*Balanced Parallelized Frugal Lookup with Shared Memory*) i BPFLSS (*Balanced Parallelized Frugal Lookup with Subtree Splitting*). U ovim algoritmima, osnovno binarno stablo se deli na nivoe. Iz svakog nivoa se čuvaju samo podstabla koja nisu prazna tj. sadrže bar jedan čvor sa validnim ID-em izlaznog porta. Ta neprazna podstabla se identifikuju pomoću tzv. prefiksa podstabala koja predstavljaju put kroz binarno stablo do korena dotičnog podstabla. Pretraga svih nivoa se vrši istovremeno (u paraleli), pri čemu se pretraga svodi prvo na pretragu prefiksa podstabala da se utvrdi da li podstablo od interesa postoji, i ukoliko postoji da se potom izvrši pretraga interne strukture tog podstabla i nađe čvor sa najdužim poklapanjem. Zatim se selektuje rešenje najdubljeg nivoa sa pozitivnim rezultatom zbog LPM pravila. Modifikacije BPFLSM i BPFLSS optimizuju strukturu memorijskih resursa koji se koriste za čuvanje interne strukture podstabala tako da se ti resursi mogu delimično ili kompletno izmestiti u eksterne memorije. Time se u značajnoj meri mogu smanjiti interni memorijski resursi koji su skupi. U nastavku ovog poglavlja će biti dat detaljan opis BPFL-a i njegovih modifikacija zajedno sa analizom.

Ostatak poglavlja je organizovan na sledeći način. Prvo će biti detaljno opisani BPFL i njegove modifikacije, pri čemu će biti data i njihova analiza. Zatim će biti

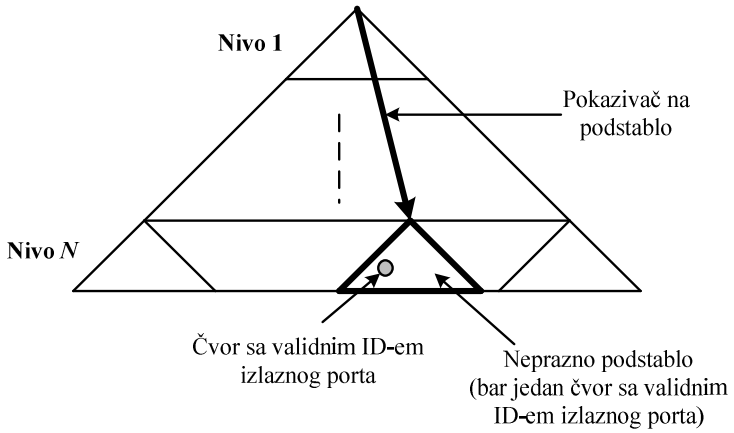
izvršeno poređenje sa lukap algoritmima analiziranim u prethodnom poglavlju. Pošto su algoritmi realizovani upotrebom FPGA čipa, na kraju poglavlja će biti dati detalji implementacije u vidu zauzetih resursa na FPGA čipu.

5.1. BPFL

Prvi predloženi lukap algoritam u ovoj tezi je BPFL. U ovom potpoglavlju će biti izložena ideja na kojoj se zasniva BPFL, a potom će biti dat opis njegove implementacije. Takođe, biće data matematička analiza performansi BPFL-a, koja će biti korišćena pri numeričkom poređenju performansi sa drugim lukap algoritmima. Na kraju će biti izloženi BPFLSS i BPFLSM koji predstavljaju modifikacije BPFL-a, i uz koje će takođe biti data matematička analiza njihovih performansi.

Osnovna ideja BPFL-a je da se prevaziđe problem koji imaju lukap algoritmi zasnovani na m-arnim stablima. Osnovni problem kod m-arnih stabala je što je potreban veći broj pristupa memoriji ukoliko se svi čvorovi stabla smeštaju u jednu memoriju. S druge strane ako se želi koristiti tehnika pajplajna za ubrzanje rada neophodno je svaki nivo stabla smestiti u posebnu memoriju. Ako se koriste dugački strajdovi tj. parametar m se postavi na veliku vrednost, onda se broj nivoa u stablu smanjuje čime se prividno rešavaju navedeni problemi. Ali, posledica je što tada velik broj čvorova postaje nevidljiv. Samim tim se mora koristiti tehnika guranja prefiksa u vidljive nivoe m-arnog stabla što može značajno povećati memorijske zahteve. Takođe je otežano održavanje takvih stabala, pošto jedan gurnut čvor u najgorem slučaju može da utiče na 2^{m-1} čvorova u m-arnom stablu. U slučaju kada se iz lukap tabele briše ili se u lukap tabelu dodaje novi prefiks on može da zahteva ažuriranje čak 2^{m-1} čvorova. Dodatni problem je što svaki čvor mora čuvati ili pokazivače na svoju decu, ili jedan pokazivač zajedno sa bitmapom postojanja dece koja bi se tada čuvala u sukcesivnim memorijskim lokacijama. Sa dugačkim strajdom se povećavaju memorijski resursi u oba slučaja. Ukoliko se koristi bitmap tehnika za čuvanje nevidljivih čvorova mogu se ublažiti problemi nastali tehnikom guranja prefiksa, ali je problem što su bitmapi nevidljivih čvorova takođe preveliki u slučaju dugačkih strajdova. Dodatni problem postoji u IPv6 lukap tabelama gde su prefiksi koncentrisani u opsegu dužina 32-48b, pa prvi nivoi m-arnih stabala u tom slučaju ne sadrže korisne informacije, već služe samo za tranziciju

ka dubljim nivoima stabla. Otuda se memorijski resursi ne koriste efikasno u IPv6 slučaju, pošto postoji dosta praznih čvorova na manjim dubinama m-arnog stabla.

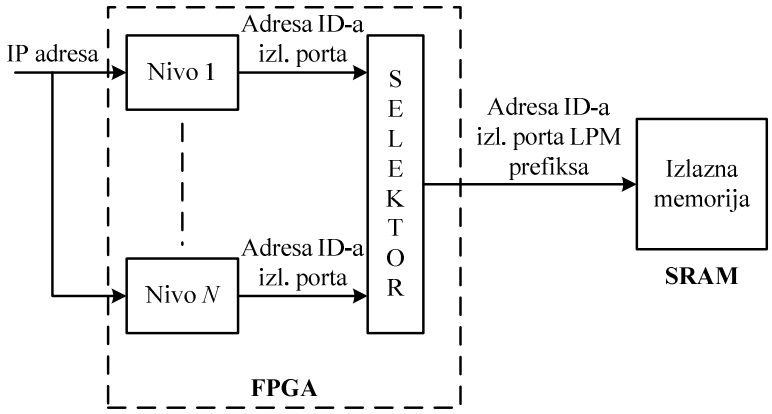


Slika 5.1.1. – Podela binarnog stabla na nivoe

Da bi se prevazišli navedeni problemi, u BPFL-u se osnovno binarno stablo deli na nivoe kao na slici 5.1.1. U svakom nivou se čuvaju samo neprazna podstabla, tj. ona podstabla koja sadrže bar jedan čvor sa validnim ID-em izlaznog porta. Pri tome, podstablo se identifikuje putanjom do korena dotičnog podstabla tj. prefiksom koji odgovara čvoru koji predstavlja koren dotičnog podstabla (slika 5.1.1). Svaki nivo sadrži podstabla jednake dubine D. U BPFL-u svi nivoi se pretražuju u paraleli. U svakom nivou se prvo pretražuju prefiksi podstabala da se utvrdi postojanje podstabla koje odgovara IP adresi za koju se vrši pretraga. Ukoliko se pronađe traženi prefiks podstabla tj. ako podstablo postoji, onda se vrši pretraga interne strukture podstabla u cilju nalaženja najdužeg prefiksa sa validnim ID-em izlaznog porta, a koji se poklapa sa zadatom IP adresom. Na kraju se selektuje pozitivan rezultat pretrage najdubljeg nivoa.

Na slici 5.1.2 je prikazan najviši nivo implementacije BPFL-a. BPFL implementacija koristi dva čipa – FPGA čip i eksternu SRAM memoriju. FPGA čip se koristi za smeštanje prefiksa podstabala, kao i za smeštanje interne strukture podstabala, ali bez samih ID-eva izlaznih portova. Takođe, FPGA čip implementira i kontrolnu logiku za pretraživanje prefiksa podstabala, kao i za ispitivanje interne strukture podstabla. Svaki nivo je implementiran i pretraživan zasebno. Ovaj visok stepen paralelizacije je omogućen zahvaljujući upotrebi FPGA čipa. ID-evi izlaznih portova se dohvataju u poslednjem koraku lukapa, pa su stoga oni izmeštene u sporiju eksternu SRAM memoriju koja je nazvana izlazna memorija. S druge strane, prefiksi podstabala

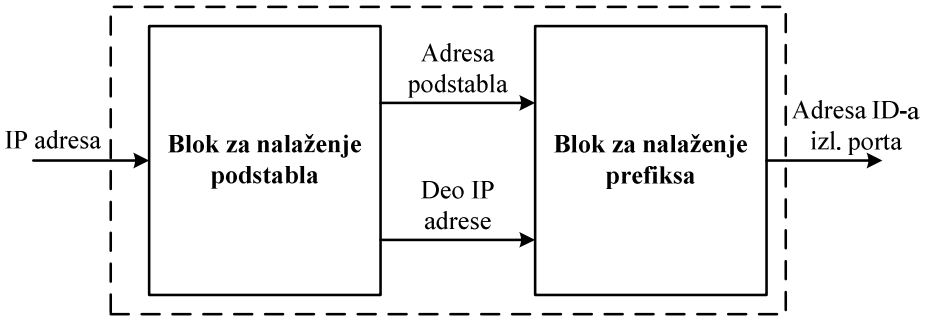
i interne strukture podstabala (bez ID-eva izlaznih portova) su smeštene u interne FPGA memorije koje su i brže. Otuda se kao rezultat pretrage na svakom nivou generiše adresa u izlaznoj memoriji gde se nalazi ID izlaznog porta prefiksa sa najdužim poklapanjem na dotičnom nivou. Ukoliko se poklapajući prefiks ne nađe na nekom nivou onda taj nivo signalizira neuspeh pretrage. Pošto više nivoa može da nađe rešenje implementira se i selektor koji primenjuje LPM pravilo i selektuje rezultat uspešne pretrage najdubljeg nivoa. Selektovano rešenje predstavlja adresu u izlaznoj memoriji gde je smešten ID izlaznog porta najdužeg poklapajućeg prefiksa. Pristupom na izračunatu lokaciju se čita ID izlaznog porta koji predstavlja krajnji rezultat lukapa.



Slika 5.1.2. – Najviši nivo implementacije BPFL-a

Struktura jednog nivoa je prikazana na slici 5.1.3. Svaki nivo se sastoji iz dva bloka: bloka za nalaženje podstabla i bloka za nalaženje prefiksa. U prvom bloku se čuvaju prefiksi podstabala. Za IP adresu sa ulaza se vrši pretraga prefiksa podstabala da bi se utvrdilo da li postoji podstablo od interesa (podstablo koje sadrži čvorove koji odgovaraju datoj IP adresi za koju se vrši pretraga). Ukoliko podstablo postoji, blok za nalaženje podstabla prosleđuje adresu dotičnog podstabla bloku za nalaženje prefiksa zajedno sa relevantnim bitima IP adrese koji odgovaraju podstablu. Blok za nalaženje prefiksa sadrži neprazna podstabla. U ovom bloku se pristupa adresiranom podstablu i vrši se njegovo ispitivanje za zadate relevantne bite IP adrese. Ispitivanjem interne strukture podstabla se utvrđuje da li postoje čvorovi u podstablu sa validnim ID-em izlaznog porta, a koji se poklapaju sa zadatim IP adresnim bitima. U slučaju da postoji više takvih čvorova, selektuje se onaj sa najdužim poklapanjem u skladu sa LPM pravilom. Kao rezultat pretrage u ovom bloku se dobija adresa ID-a izlaznog porta u izlaznoj memoriji koja odgovara čvoru sa validnim ID-em izlaznog porta koji je imao

najduže poklapanje sa zadatim IP adresnim bitima. Ova adresa se prosleđuje na izlaz nivoa kao konačno rešenje pretrage na nivou. U slučaju da blok za nalaženje podstabla nije našao traženo podstablo, onda se bloku za nalaženje prefiksa signalizira da podstablo od interesa nije pronađeno, pa će i blok za nalaženje prefiksa signalizirati da na dotičnom nivou nema poklapanja za zadatu IP adresu. U slučaju da podstablo postoji, ali u okviru njega nema čvorova sa validnim ID-em izlaznog porta koji se poklapaju sa zadatom IP adresom, blok za nalaženje prefiksa će nakon neuspešne pretrage interne strukture podstabla takođe signalizirati da je pretraga bila neuspešna na dotičnom nivou. Ovakva organizacija strukture nivoa omogućava primenu pajplajn tehnike jer kad pretraga za neku IP adresu pređe na blok za nalaženje prefiksa, nova pretraga već može da otpočne u bloku za nalaženje podstabla. Na ovaj način se postiže visok protok lukap modula od jednog lukapa po jednom ciklusu takta, jer nova pretraga može da otpočne pre nego što se prethodna završi.

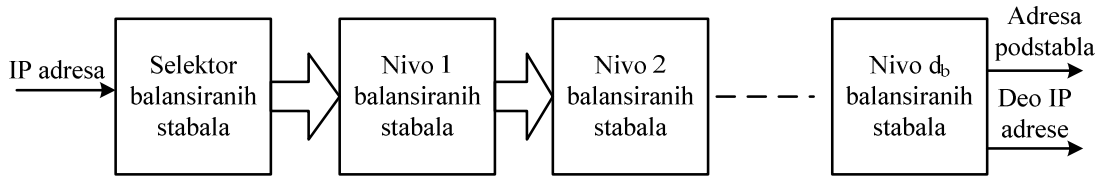


Slika 5.1.3. – Struktura jednog nivoa u BPFL-u

Osnovni cilj bloka za nalaženje podstabla je da utvrdi da li u dotičnom nivou postoji podstablo koje sadrži čvorove koji odgovaraju IP adresi za koju se vrši pretraga. Podstablo na nivou i koje odgovara IP adresi je određeno sa prvih $(i-1)\cdot D$ bita date IP adrese za koju se vrši pretraga. Prvih $(i-1)\cdot D$ bita ćemo zvati prefiksom podstabla. Pri tome redni broj nivoa se kreće u intervalu $i = 1..N$, gde je N ukupan broj nivoa. Ukupan broj nivoa se izračunava kao $N = L/D$, gde je L dužina IP adrese, a D je dubina podstabla na bilo kom nivou, pošto je već rečeno da svi nivoi imaju iste dimenzije podstabala. Prefiksi podstabala treba da se smeste u efikasnu strukturu koja omogućava brzu pretragu.

U BPFL-u je usvojeno da se prefiksi podstabala smeštaju u balansirana stabla. Balansirano stablo je ono binarno stablo u kojemu svaki čvor ima jednak broj potomaka na obe svoje strane (leve i desne). Naravno, u slučaju neparnog broja potomaka broj

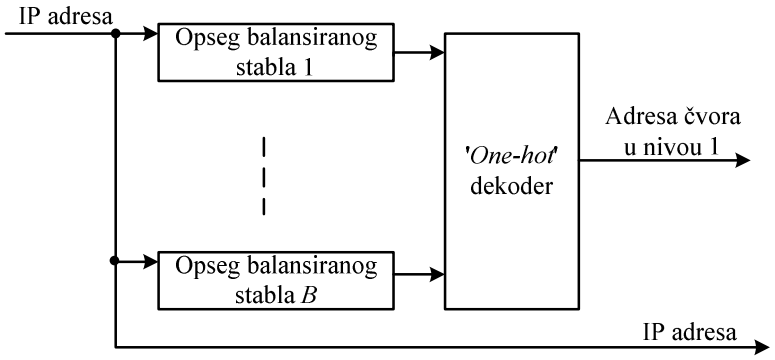
potomaka na jednoj strani je za jedan veći od broja potomaka na drugoj strani. Razlog za izbor balansiranih stabala je njihova osobina da se kreiraju čvorovi na novom nivou stabla tek kad se svi prethodni nivoi u potpunosti popune. Na ovaj način, znajući koliko prefiksa podstabala je potrebno, znamo i koliko čvorova treba da sadrži balansirano stablo pa možemo unapred da znamo njegovu dubinu. Poznavanje dubine balansiranog stabla unapred je neophodno, pošto se prostor za sve čvorove balansiranog stabla mora unapred rezervisati. Naime, prilikom izrade FPGA dizajna sve memorije koje čuvaju čvorove balansiranog stabla moraju biti dimenzionisane i instancirane. Ali, ukoliko bi se koristilo samo jedno balansirano stablo ukupan broj njegovih nivoa bi mogao da bude prevelik. To bi otežalo njegovu efikasnu implementaciju jer zbog pajplajna svaki nivo balansiranog stabla mora da bude smešten u zasebnu memoriju. Pri tome prvi nivoi balansiranog stabla imaju veoma mali broj čvorova pa nisu pogodni za smeštanje u memoriju. Da bi se to prevazišlo usvojeno je da se prefiksi podstabala čuvaju u više balansiranih stabala čiji se opsezi ne preklapaju. Opseg jednog balansiranog stabla je definisan najmanjom i najvećom vrednošću prefiksa podstabala koji se nalaze u dotičnom balansiranom stablu. Na ovaj način se smanjuje potrebna dubina balansiranog stabla jer ih ima više, a isto tako svi nivoi balansiranih stabala popunjavaju memorijske blokove, jer se u svakom bloku čuvaju čvorovi istih nivoa balansiranih stabala pa nema problema da se u memoriji čuva premali broj čvorova. Prefiksi podstabala su razmešteni u balansiranom stablu tako da se u potomcima sa desne strane nekog čvora nalaze prefiksi podstabala čija je vrednost veća od one u tom čvoru, a u potomcima sa leve strane prefiksi podstabala čija je vrednost manja od one u tom čvoru. Ovo omogućava jednostavno kretanje kroz balansirano stablo jednostavnim poređenjem vrednosti za koju se vrši pretraga sa onim vrednostima koje se nalaze u čvorovima balansiranog stabla. Slika 5.1.4 prikazuje najviši nivo implementacije bloka za nalaženje podstabla.



Slika 5.1.4. – Najviši nivo implementacije bloka za nalaženje podstabla

Pošto postoji više balansiranih stabala, neophodno je prvo utvrditi u kom balansiranom stablu treba vršiti pretragu za prefiksom podstabla od interesa. Selekcija

balansiranog stabla se vrši u bloku selektor balansiranih stabala, koji nalazi opseg u koji upada data IP adresa i selektuje ono balansirano stablo čiji opseg obuhvata datu IP adresu. Kao rezultat rada ovog bloka se dobija lokacija korena nađenog balansiranog stabla u memoriji koja čuva korene svih balansiranih stabala (tj. čuva čvorove nivoa 1 svih balansiranih stabala). Potom se pretraga kreće kroz nivoe selektovanog balansiranog stabla. Čvorovi balansiranog stabla sadrže prefikse podstabala, tako da se u svakom čvoru ispituje da li se njegov sadržaj poklapa sa IP adresom. Ukoliko se poklapa onda se preostali nivoi balansiranog stabla prolaze bez procesiranja radi održavanja vremenskog rasporeda pajplajna, a ako se ne poklapa prelazi se na čvor u sledećem nivou balansiranog stabla. Ukoliko je vrednost sadržaja ispitivanog čvora veća od IP adrese skreće se levo, u suprotnom se skreće desno. Kao konačni rezultat blok za nalaženje podstabla vraća adresu podstabla čiju internu strukturu treba pretražiti i deo IP adrese relevantan za ispitavanje interne strukture dotičnog podstabla. Ako traženi prefiks podstabla ne postoji, adresa podstabla će biti nepostojeća čime se signalizira neuspeh pretrage.



Slika 5.1.5. – Selektor balansiranih stabala

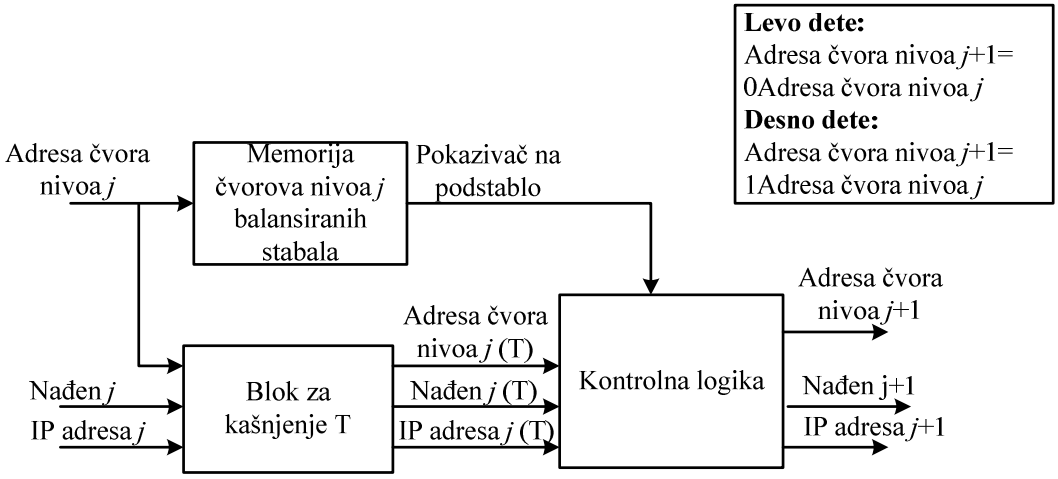
Na slici 5.1.5 je prikazana struktura selektora balansiranih stabala. IP adresa se poredi sa granicama opsega svakog balansiranog stabla. Ukoliko IP adresa upada u opseg balansiranog stabla generiše se '1', a ako ne upada generiše se '0'. Pošto su opsezi nepreklapajući onda IP adresa može da upadne samo u jedan opseg. Na ulaz '*one-hot*' dekodera dolazi vektor čiji biti predstavljaju rezultate provera pripadnosti IP adrese opsezima balansiranih stabala, pri čemu vektor može da sadrži maksimalno jednu '1' jer IP adresa može da pripada samo jednom opsegu. Dekoder dekoduje poziciju '1' u vektoru i na osnovu toga proračunava adresu korena (tj. čvora iz nivoa 1) izabranog balansiranog stabla kome pripada data IP adresa. Pošto se može desiti da IP adresa ne

upada ni u jedan opseg, onda se u takvom slučaju generiše nepostojeća adresa koja signalizira takav slučaj i tada se neće vršiti pretraga balansiranih stabala već će se bez procesiranja proći kroz nivoe balansiranih stabala radi održavanja pajplajna. Tada će blok za nalaženje podstabla na svom izlazu signalizirati nepostojanje podstabla od interesa.

Na slici 5.1.6 je prikazana struktura nivoa *j* balansiranih stabala. Od prethodnog nivoa balansiranog stabla dobija IP adresu za koju se vrši pretraga, kao i adresu čvora u nivou *j* koji se treba ispitati. S obzirom da postoji latencija koju unosi čitanje čvora iz memorije, IP adresa se propušta kroz blok za kašnjenje. U kontrolnoj logici se vrši poređenje IP adrese sa prefiksom podstabla pročitanog čvora. Ukoliko nema poklapanja, pretraga se seli na sledeći nivo, a adresa čvora u sledećem nivou se dobija tako što se na adresu ispitivanog čvora lepi binarna '0' u slučaju da je IP adresa manja od pokazivača na podstablo u ispitanom čvoru, odnosno '1' u suprotnom slučaju. Ova adresa se prosleđuje zajedno sa IP adresom sledećem nivou balansiranog stabla. Kao što se vidi ne koriste se pokazivači za kretanje kroz nivoe balansiranog stabla, već se koristi statičko adresiranje jer su svi čvorovi balansiranog stabla unapred kreirani i potom popunjavani prefiksima podstabala. Na ovaj način se značajno štede memorijski resursi jer čvorovi čuvaju samo prefikse podstabala i nijednu drugu informaciju. Ukoliko se prefiks podstabla poklapa sa prefiksom IP adrese, onda podstablo od interesa postoji.

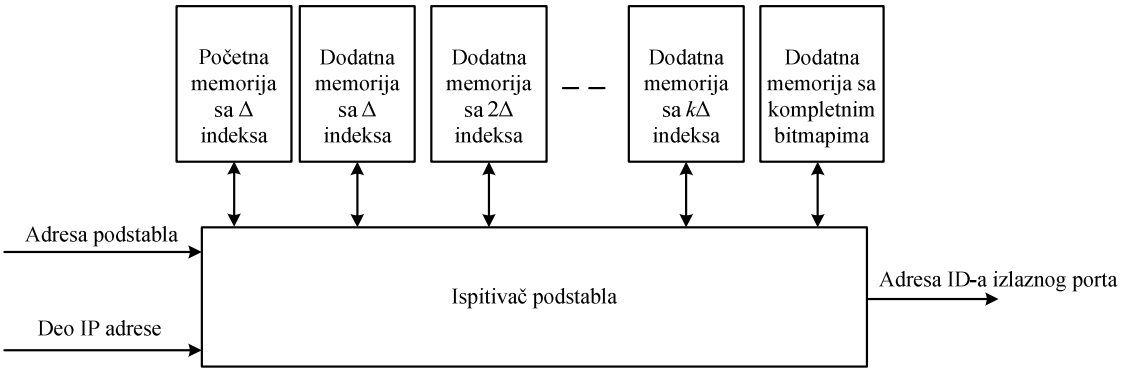
Sama adresa podstabla u bloku za nalaženje prefiksa je takođe statički određena tj. svakom čvoru balansiranog stabla je unapred dodeljena memorijska lokacija za internu strukturu podstabla čiji se prefiks nalazi u dotičnom čvoru. Ovakvim statičkim adresiranjem se izbegava čuvanje pokazivača na lokacije u bloku za nalaženje prefiksa čime se značajno štede memorijski resursi. U slučaju pronalaženja traženog podstabla aktivira se signal *Nađen j+1* ka sledećem nivou balansiranog stabla, a u adresu čvora sledećeg nivoa se stavlja adresa podstabla u bloku za nalaženje prefiksa, gde se nalazi interna struktura traženog podstabla. Na ovaj način kad nivo *j* balansiranog stabla primi aktivan signal *Nađen j*, on zna da je nađena adresa traženog podstabla u nekom od prethodnih nivoa i da ne treba da se vrši dalja pretraga pa on samo prosleđuje rezultate pretrage sledećem nivou bez ikakvih ispitivanja. Poslednji nivo balansiranog stabla prosleđuje adresu traženog podstabla bloku za nalaženje prefiksa, zajedno sa relevantnim bitima IP adrese. U slučaju da prefiks podstabla nije nađen, onda se

prosleđuje nepostojeća adresa kojom se signalizira neuspeh pretrage. U slučaju da je selektor balansiranih stabala signalizirao neuspeh u vidu nepostojeće adrese čvora nivoa 1 balansiranih stabala, onda se ta nepostojeća adresa od selektora prosleđuje kroz sve nivoe balansiranog stabla čime se signalizira neuspeh pretrage, a poslednji nivo balansiranog stabla onda generiše nepostojeću adresu bloku za nalaženje prefiksa.



Slika 5.1.6. – Nivo *j* balansiranih stabala

Blok za nalaženje prefiksa treba da nađe najduže poklapajući prefiks u odgovarajućem podstablu. Otuda se u ovom bloku pretražuje interna struktura podstabla da bi se pronašao čvor sa validnim ID-em izlaznog porta koji se najduže poklapa sa zadatom IP adresom. U ispitivanju se koristi samo deo IP adrese koji odgovara datom nivou. Preciznije, na nivou i se u bloku za nalaženje prefiksa koriste biti IP adrese na pozicijama od $(i-1)\cdot D+1$ do $i\cdot D$. Pošto se sam ID izlaznog porta čuva u eksternoj izlaznoj memoriji, a ne u bloku za nalaženje prefiksa, rezultat pretrage interne strukture podstabla vraća adresu lokacije u izlaznoj memoriji gde se nalazi ID izlaznog porta koji odgovara najdužem poklapanju u tom podstablu. Struktura bloka za nalaženje prefiksa je prikazana na slici 5.1.7.



Slika 5.1.7. – Struktura bloka za nalaženje prefiksa

S obzirom da u realnim tabelama rutiranja velik broj podstabala nije gusto popunjen onda nije ekonomično čuvati kompletan bitmap, već je efikasnije za retko popunjena podstabla čuvati indekse postojećih čvorova iz dva razloga. Prvi je razlog što se interni memorijski resursi čipa tada efikasnije koriste. Takođe, time se značajno štede i memorijski resursi izlazne memorije jer se u slučaju slabo popunjenog podstabla ne rezervišu lokacije za potencijalne ID-eve izlaznih portova svih čvorova iz podstabla, već za manji broj čvorova. Otuda se svakom nepraznom podstablu dodeljuje memorija sa Δ čvorova koji se čuvaju u vidu indeksa pozicije čvora u okviru podstabla. U lokacijama ove memorije se čuvaju indeksi postojećih čvorova i pokazivač na dodatnu memoriju. Dodatne memorije čuvaju dodatnih Δ, 2Δ, …, $k\Delta$ čvorova, pored onih Δ čvorova iz početne memorije. Na ovaj način se postiže fina granularnost potrošnje memorije pri čuvanju podstabala različitih gustina. U slučaju kad broj dodatnih čvorova postane prevelik tj. podstablo postane gusto popunjeno onda se čuva kompletan bitmap tog podstabla. Podstablo je gusto popunjeno ako ima više od $(k+1)\cdot\Delta$ čvorova sa validnim ID-em izlaznog porta. Važno je napomenuti da se resursi u dodatnim memorijama dodeljuju podstablima dinamički u zavisnosti od njihove popunjenosti. Pri tome svakom podstablu može da bude dodeljena samo jedna lokacija u samo jednoj dodatnoj memoriji i na nju ukazuje pokazivač smešten u početnoj memoriji.

Princip rada bloka za nalaženje prefiksa je veoma jednostavan. Ispitivač podstabla pristupa lokaciji u početnoj memoriji čiju adresu je dobio od bloka za nalaženje podstabla i ispituje indekse čvorova tako što utvrđuje da li oni indeksiraju prefikse date IP adrese. U slučaju da postoji više indeksa prefiksa date IP adrese bira se onaj koji ima najduže poklapanje tj. indeksira čvor na najvećoj dubini podstabla.

Istovremeno se ispituje i pokazivač na dodatnu memoriju. Ako pokazivač ne

postoji onda je pretraga okončana ispitivanjem indeksa čvorova iz početne memorije. Tada se nađeno rešenje prosleđuje sa kašnjenjem na izlaz bloka za nalaženje prefiksa radi očuvanja pajplajna. Rešenje pretrage se prosleđuje u vidu adrese u eksternoj izlaznoj memoriji gde je smešten traženi ID izlaznog porta. Ako pokazivač na dodatnu memoriju postoji, onda se pristupa i lokaciji u odgovarajućoj dodatnoj memoriji na koju ukazuje pokazivač. Vrši se ispitivanje indeksa čvorova na identičan način kao kod indeksa pročitanih iz početne memorije. Ako se u dodatnoj memoriji nalazi indeks sa najdužim poklapanjem, on se selektuje, u suprotnom je konačno rešenje ono koje je pronađeno među indeksima smeštenim u početnoj memoriji. U slučaju da se pristupa dodatnoj memoriji koja čuva kompletan bitmap ispitivanje podstabla se razlikuje, pošto se ispituje kompletan bitmap, a ne skup indeksiranih čvorova. Tada se konačno rešenje sigurno nalazi u dodatnoj memoriji jer bitmap mapira sve čvorove podstabla. Nakon ispitivanja dodatne memorije, konačno rešenje se prosleđuje na izlaz bloka za nalaženje podstabla.

Bitno je napomenuti da su ID-evi izlaznih portova u izlaznoj memoriji statički adresirani radi uštede memorijskih resursa. To znači da je svakom indeksu čvora iz početne memorije i dodatne memorije unapred dodeljena jedna lokacija u izlaznoj memoriji. Isto tako u slučaju dodatne memorije koja čuva kompletne bitmape je svakom indeksu bitmapa dodeljena jedna lokacija.

5.1.1. Analiza performansi BPFL-a

U performanse jednog lukap algoritma spadaju, kako je već naglašeno u okviru prethodnog poglavlja u analizama postojećih lukap algoritama, brzina lukap algoritma, odnosno protok lukap modula, zahtevani memorijski resursi i kompleksnost ažuriranja podataka u lukap tabeli. Protok lukap modula se određuje brojem IP adresa za koje se može izvršiti lukap u jedinici vremena, ili u toku jednog ciklusa takta. Memorijski resursi su bitni pošto oni utiču na izbor memorijskih čipova, a i na brzinu lukap algoritama. Kompleksnost ažuriranja treba da bude što manja pošto se lukap tabela mora ažurirati u realnom vremenu da bi se smanjile greške u prosleđivanju IP paketa usled netačnosti u lukap tabeli jer ažuriranje nije izvršeno na vreme.

U BPFL-u se, kao što je pomenuto, koristi pajplajn tehnika. Takođe se koristi i tehnika paralelizacije pošto se svi nivoi pretražuju u paraleli. Ovakva arhitektura omogućava da se postigne generisanje jednog lukap rezultata po jednom ciklusu takta,

pošto, kad pretraga otpočne, ona kreće paralelno po svim nivoima uz upotrebu pajplajn tehnike na svakom od nivoa. Otuda, sledeća pretraga može da krene već u sledećem ciklusu takta, te se tako izračunava jedan rezultat lukapa u jednom ciklusu takta.

U bloku za nalaženje podstabla memorijski resursi se troše na čuvanje čvorova balansiranih stabala, a registri za čuvanje opsega balansiranih stabala. U čvorove balansiranih stabala se stavljaju prefiksi podstabala širine $(i-1)\cdot D$, gde je i redni broj nivoa kome pripadaju ta balansirana stabla. Registri čuvaju gornje i donje granice balansiranih stabala pa je njihova dužina duplo veća od dužine jedne memorijske lokacije koja čuva prefikse podstabala. Otuda su zahtevani memorijski resursi neophodni za balansirana stabla blokova za nalaženje podstabla svih nivoa (M_{bs}):

$$M_{bs} = \sum_{i=1}^{L/D} \left(N_{bs}^{i} \cdot N_{c}^{i} \cdot (i-1) \cdot D \right) \tag{5.1.1.1}$$

gde je N_{bs}^{i} broj balansiranih stabala u nivou i, a N_{c}^{i} broj čvorova u jednom balansiranom stablu u nivou i. Očigledno je da mora da važi $N_{bs}^{i} \cdot N_{c}^{i} \geq N_{pod}^{i}$, gde je N_{pod}^{i} broj nepraznih podstabala nivoa i. Registarski resursi bloka za pretragu pokazivača na podstabla (R_{bs}) koji se koriste za čuvanje opsega balansiranih stabala iznose:

$$R_{bs} = \sum_{i=1}^{L/D} \left(N_{bs}^{i} \cdot 2 \cdot (i-1) \cdot D \right) \tag{5.1.1.2}$$

U bloku za nalaženje prefiksa razlikujemo tri vrste internih memorija – početnu memoriju sa indeksima prvih Δ nepraznih čvorova, dodatne memorije koje čuvaju indekse narednih popunjenih čvorova i dodatnu memoriju koja čuva kompletne bitmape gusto popunjenih podstabala. Broj lokacija početnih memorija nekog nivoa je jednak broju nepraznih podstabala tog nivoa, pri čemu svaka lokacija čuva Δ indeksa čvorova i pokazivač na dodatnu memoriju. Broj bita potreban za čuvanje jednog indeksa iznosi $D+1$. Stoga su zahtevani memorijski resursi za početne memorije svih nivoa (M_{pm}):

$$M_{pm} = \sum_{i=1}^{L/D} \left(N_{bs}^{i} \cdot N_{c}^{i} \cdot \left(\Delta \cdot (D+1) + L_{dm}^{i} \right) \right) \tag{5.1.1.3}$$

gde je L_{dm}^{i} dužina pokazivača na dodatnu memoriju u nivou i. Pokazivač na dodatnu memoriju se sastoji iz dva dela – jedan deo određuje koja od $k+1$ dodatne memorije je

u pitanju, a drugi deo određuje lokaciju u okviru odgovarajuće dodatne memorije. Otuda se dužina pokazivača na dodatnu memoriju L_{dm}^{i} može predstaviti kao:

$$L_{dm}^{i} = \lfloor \log_2(k+1) \rfloor + \left\lfloor \log_2 \left(\max_{j=1..k+1} N_m^{i,j} \right) \right\rfloor \tag{5.1.1.4}$$

gde je $\max_{j=1..k+1} N_m^{i,j}$ broj lokacija najveće od dodatnih memorija u nivou i (najveća u smislu najvećeg broja lokacija). U lokacijama dodatne memorije koja može da čuva $j \cdot \Delta$ indeksa dodatnih popunjenih čvorova čuvaju se samo ti indeksi pa je dužina jedne lokacije $j \cdot \Delta \cdot (D+1)$. Zahtevani memorijski resursi dodatnih memorija koje čuvaju indekse popunjenih čvorova svih nivoa ($M_{dm}^{1..k}$) iznose:

$$M_{dm}^{1..k} = \sum_{i=1}^{L/D} \sum_{j=1}^{k} N_m^{i,j} \cdot j \cdot \Delta \cdot (D+1) \tag{5.1.1.5}$$

gde je $N_m^{i,j}$ broj lokacija u dodatnoj memoriji j nivoa i. Zahtevani memorijski resursi (M_{dm}^{k+1}) dodatnih memorija koje čuvaju kompletne bitmap vektore, dužine $2^{D+1} - 2$, gusto popunjenih podstabala svih nivoa iznose:

$$M_{dm}^{k+1} = \sum_{i=1}^{L/D} N_m^{i,k+1} \cdot (2^{D+1} - 2) \tag{5.1.1.6}$$

gde je $N_m^{i,k+1}$ broj lokacija u dodatnoj memoriji koja čuva kompletne bitmape gusto popunjenih podstabala nivoa i.

Izlazna memorija čuva ID-eve izlaznih portova. Pri tome početne memorije i dodatne memorije svih nivoa imaju unapred dodeljen prostor u okviru izlazne memorije tj. koristi se statičko adresiranje. Tako da su zahtevani memorijski resursi za izlaznu memoriju (M_{iz}):

$$M_{iz} = H \cdot \sum_{i=1}^{L/D} \left[\left(N_{bs}^{i} \cdot N_{c}^{i} \cdot \Delta \right) + \sum_{j=1}^{k} \left(N_m^{i,j} \cdot j \cdot \Delta \right) + N_m^{i,k+1} \cdot (2^{D+1} - 2) \right] \tag{5.1.1.7}$$

gde je H dužina ID-a izlaznog porta.

Kada se koriste eksterni memorijski čipovi prvo se izmešta izlazna memorija jer je najveća. Potom se izmeštaju početna memorija i dodatna memorija sa Δ prefiksa bloka za nalaženje prefiksa najgušće popunjenog nivoa jer su one sledeće najveće memorije. Otuda su interni memorijski resursi (M_{int}^{i}):

$$M_{int}^{i} = \begin{cases} M_{tot} - M_{iz}, & i = 1 \\ M_{tot} - M_{iz} - M_{pm}^{max}, & i = 2 \\ M_{tot} - M_{iz} - M_{pm}^{max} - M_{dm}^{max,1}, & i = 3 \end{cases} \quad (5.1.1.8)$$

pri čemu i predstavlja broj eksternih memorijskih čipova na raspolaganju, M_{tot} predstavlja ukupne memorijske resurse, M_{pm}^{max} predstavlja kapacitet početne memorije najgušće popunjenog nivoa, $M_{dm}^{max,1}$ predstavlja kapacitet dodatne memorije sa Δ prefiksa najgušće popunjenog nivoa.

Sa stanovišta ažuriranja razlikuju se tri događaja: modifikacija ID-a izlaznog porta postojećeg prefiksa u lukap tabeli, dodavanje novog prefiksa sa njegovim ID-em izlaznog porta u lukap tabelu, brisanje postojećeg prefiksa i njegovog ID-a izlaznog porta iz lukap tabele. Modifikacija je veoma jednostavna i zahteva samo pristup ID-u izlaznog porta u izlaznoj memoriji i upis novog ID-a izlaznog porta na mesto starog.

Kada se dodaje prefiks, razlikuje se više slučajeva sa stanovišta kompleksnosti. Prvi slučaj je dodavanje novog prefiksa u postojeće podstablo, pri čemu novi ukupan broj prefiksa ne menja nivo popunjenosti podstabla. To znači da kad se novi prefiks doda, novi broj prefiksa u podstablu ne prelazi neku od granica $i\Delta, (i = 1..k)$. Dodavanje prefiksa se svodi na dopisivanje njegovog indeksa u odgovarajuću lokaciju početne ili dodatne memorije u zavisnosti od ukupnog broja postojećih čvorova podstabla, jedino u slučaju da se čuva kompletan bitmap se upisuje ‘1’ na odgovarajuću poziciju bitmapa. Takođe se u odgovarajuću lokaciju izlazne memorije upisuje ID izlaznog porta dodatog prefiksa.

U slučaju da se pređe granica $i\Delta, (i = 1..k)$, tada se u početnoj memoriji upisuje nova vrednost pokazivača na dodatnu memoriju koja smešta veći broj čvorova podstabla. Pri tome se prefiksi iz odgovarajuće lokacije prethodno korišćene dodatne memorije premeštaju u novu dodatnu memoriju zajedno sa novim prefiksom. U slučaju da broj prefiksa u podstablu postane veći od $(k + 1)\Delta$, onda se formira kompletan bitmap za to podstablo. Uporedo sa premeštanjem prefiksa u internoj memoriji, u eksternoj memoriji se moraju premestiti ID-evi izlaznih portova, jer se koristi statičko adresiranje, pa pozicija prefiksa u internoj memoriji određuje poziciju ID-a izlaznog porta u eksternoj (izlaznoj) memoriji. Najgori slučaj je onda kad se vrši prelaz na gusto popunjeno podstablo za koje se čuva kompletan bitmap, jer se tada u izlaznoj memoriji

moraju premestiti ID-evi izlaznih portova svih postojećih čvorova (kojih ima $(k+1)\cdot \Delta$) uz upis i ID-a izlaznog porta novog prefiksa.

Sledeći slučaj dodavanja prefiksa je kada ne postoji podstablo koje odgovara dodatom prefiksu. U tom slučaju je potrebno dodati i prefiks podstabla u bloku za nalaženje podstabla. Tada se u balansirano stablo, čijem opsegu pripada podstablo dodavanog prefiksa, dodaje prefiks tog podstabla. U ovom slučaju može da dođe do pomeranja prefiksa podstabala u čvorovima balansiranog stabla da bi se mogao uneti novi prefiks podstabla. Usled statičkog adresiranja, svako pomeranje prefiksa podstabala u balansiranom stablu za posledicu ima i pomeranje odgovarajućih podstabala u početnoj memoriji bloka za nalaženje prefiksa, a samim tim i odgovarajućih ID-eva izlaznih portova u izlaznoj memoriji. Sadržaji lokacija u dodatnim memorijama u bloku za nalaženje prefiksa se ne pomeraju jer one nisu statički adresirane. Kada se prefiks podstabla doda u balansirano stablo, onda se upisuje indeks dodatog mrežnog prefiksa u odgovarajuću lokaciju početne memorije u bloku za nalaženje prefiksa, a upisuje se i ID izlaznog porta dodatog prefiksa u odgovarajuću lokaciju izlazne memorije.

Najgori slučaj koji može da se desi jeste kada su sva balansirana stabla u nivou potpuno popunjena i samo jedno balansirano stablo ima jedno prazno mesto. Tada u slučaju dodavanja novog prefiksa i njegovog podstabla može doći do pomeranja svih čvorova u svim balansiranim stablima da bi se novo podstablo i njegov prefiks mogli dodati. Naravno, usput će morati biti izvršene i korekcije opsega balansiranih stabala. Ali, verovatnoća ovog događaja je mala.

U slučaju brisanja nekog prefiksa iz tabele, slučajevi su prilično slični slučajevima dodavanja. Ako se briše prefiks i pri tome novi ukupan broj prefiksa ne prelazi neku od granica $i\Delta, (i = 1..k)$, tada se samo briše indeks prefiksa iz odgovarajuće liste ako je podstablo retko popunjeno, odnosno na odgovarajuću poziciju bitmap vektora se upisuje ‘0’ kada je podstablo gusto popunjeno. ID izlaznog porta u izlaznoj memoriji se ne mora brisati jer bez postojanja prefiksa u internoj memoriji, toj informaciji se ne može pristupiti. U slučaju prelaza granice se mora izvršiti premeštanje indeksa u dodatnu memoriju manjeg kapaciteta, pri čemu se mora uporedo izvršiti i pomeranje odgovarajućih ID-eva izlaznih portova u izlaznoj memoriji zbog korišćenog statičkog adresiranja. Takođe, mora se promeniti vrednost pokazivača u početnoj

memoriji. Kao i u slučaju dodavanja prefiksa i ovde je najgori slučaj kada se prelaz vrši iz gustog popunjenog stabla u retko popunjeno stablo. U slučaju kada se briše poslednji prefiks iz podstabla, onda se mora izbrisati i prefiks podstabla iz odgovarajućeg balansiranog stabla. Kao posledica brisanja može doći do korekcije opsega balansiranog stabla kome je pripadalo obrisano podstablo (ako je prefiks podstabla imao najveću ili najmanju vrednost u balansiranom stablu). Takođe, zbog održavanja balansiranosti može doći do pomeranja prefiksa podstabala u balansiranom stablu, ali se ta pomeranja zadržavaju na nivou jednog balansiranog stabla pa je kompleksnost manja nego u najgorem slučaju koji je opisan kod dodavanja prefiksa.

Kao što se može videti postoje slučajevi koji zahtevaju ažuriranje velikog broja lokacija, što čini ažuriranje veoma kompleksnim u najgorim slučajevima. To je cena koja se plaća zbog korišćenja statičkog adresiranja. S jedne strane su značajno smanjeni memorijski zahtevi, ali je s druge strane ažuriranje značajno usporeno u nekim slučajevima. Međutim, interne memorije koje se koriste kod hardverskih implementacija zasnovanih na FPGA ili ASIC čipovima mogu da se konfigurišu sa dualnim portovima čime se omogućava istovremeno čitanje i upis u takve memorije. Time ažuriranja u internim memorijama ne usporavaju lukap tj. mogu da se odvijaju istovremeno i lukap i ažuriranje lukap tabele. Naravno, bitno je napomenuti da tada može doći do grešaka u prosleđivanju ako lukap pristupi u međuvremenu netačnim (zastarelim) informacijama. Bitno je napomenuti da danas postoje i eksterne memorije koje omogućavaju istovremeno čitanje i upis, pa tako ni ažuriranje u izlaznoj memoriji ne mora da izazove zastoj lukapa [36].

Lukap tabele se izračunavaju na kontrolnoj ravni u cilju uštede hardverskih resursa [43]. U slučaju da se proces ažuriranja vrši u ravni podataka, onda bi bili potrebni dodatni resursi na svakom portu. S obzirom da broj portova rutera može biti velik onda bi bili potrebni veliki dodatni hardverski resursi što bi značajno podiglo cenu rutera. Otuda je znatno efikasnije vršiti ažuriranje lukap tabela u kontrolnoj ravni (tj. na jednom mestu), a potom u vidu parova adresa/podatak proslediti konačne rezultate memorijama koje čuvaju lukap tabele u ravni podataka.

5.1.2. BPFLSM

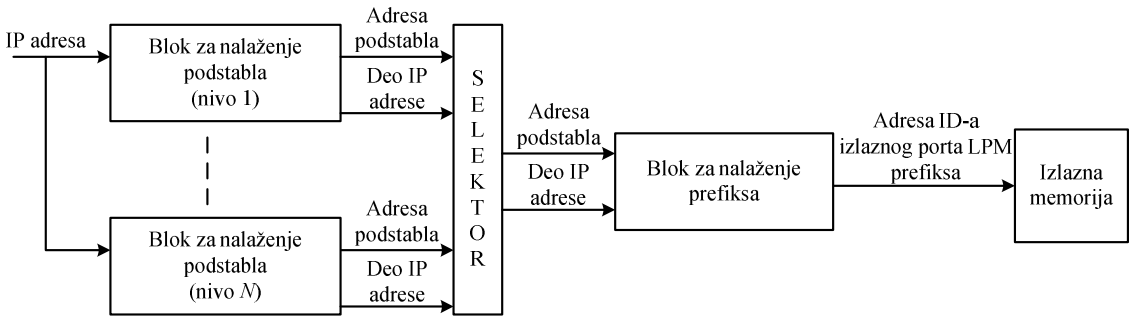
BPFL, zahvaljujući statičkom adresiranju i finom prilagođavanju različitim gustinama popunjenosti nepraznih podstabala, ima ekonomične memorijske zahteve.

Takođe, naš algoritam omogućava visok protok obrađenih IP adresa. Međutim, usled svoje arhitekture koja podrazumeva paralelnu pretragu na više nivoa istovremeno, samo izlazna memorija, koja sadrži ID-eve izlaznih portova kojima se pristupa u poslednjem koraku lukapa, može da se implementira u vidu eksterne memorije. Zbog toga, su u slučaju velikih lukap tabela neophodni integrisani čipovi (FPGA, ASIC) sa većim internim memorijskim resursima. U ovom odeljku je predložen BPFLSM koji uvodi minimalne promene u BPFL arhitekturu, a koje omogućavaju da se dominantni deo memorijskih resursa namenjenih čuvanju interne strukture podstabala smesti u eksterne memorijske čipove.

U BPFL-u se vrši pretraga svih nivoa u paraleli pošto se ne zna unapred na kom nivou će se naći najbolje rešenje. Zbog toga i memorijski resursi namenjeni za čuvanje internih struktura podstabala moraju biti distribuirani po nivoima i svaki nivo ima svoje zasebne memorije za njihovo čuvanje. Na ovaj način se omogućava najviši stepen paralelizacije, a time i velike brzine lukap algoritma. BPFLSM ima za cilj da uvođenjem dodatne informacije u neprazno podstablo omogući da se memorije za čuvanje interne strukture podstabala različitih nivoa udruže. Kao što je već prethodno u tezi istaknuto, broj eksternih memorija koji se može upotrebiti je ograničen brojem pinova integrisanog čipa (FPGA, ASIC) na koje se one vezuju, a takođe i samim prostorom na štampanoj ploči, pošto više memorija zahteva i veći prostor. Kao što se videlo u analizi BPFL-a u slučaju upotrebe 2 ili 3 eksterne memorije, mogu se izmestiti samo 1 ili 2 memorije koje se koriste za smeštanje interne strukture podstabala, a kao što se može videti sa slike 5.1.7, samo na jednom nivou tih memorija ima više. Udruživanje više memorija iz modula za nalaženje prefiksa različitih nivoa u jednu zajedničku memoriju omogućava efikasniju upotrebu eksterne memorije i njenog većeg kapaciteta pošto bi se zajednička memorija koju koriste svi nivoi izmestila u eksternu memoriju.

Ukoliko postoji više nivoa koji sadrže podstablo od interesa bilo bi idealno da se pretraži samo podstablo koje se nalazi na najdubljem nivou. Međutim, ako bi se samo ono pretražilo postoji rizik da dotično podstablo ne sadrži poklapajući prefiks i da je pretragu trebalo izvršiti na nekom drugom (plićem) nivou. Da bi izbegli ovakvu situaciju, neophodno je dodati uz svako podstablo najbolji validan ID izlaznog porta koji se nalazi na putu od korena originalnog binarnog stabla do korena samog podstabla

(uključujući i koren samog podstabla). Na ovaj način, ukoliko u podstablu nema poklapajućeg prefiksa, uzima se ovaj dodati ID izlaznog porta kao konačno rešenje čime se izbegava pretraga nekog od plićih nivoa. Otuda, kad se u svakom nivou u bloku za nalaženje podstabla utvrdi postojanje podstabla od interesa, može se selektovati pretraga prefiksa u podstablu najdubljeg nivoa koji je imao uspešnu pretragu. Time je omogućeno da se memorijski resursi namenjeni za čuvanje interne strukture podstabala svih nivoa udruže, jer će pretraga interne strukture podstabla da se izvrši samo na jednom nivou. Arhitektura BPFLSM je prikazana na slici 5.1.2.1.

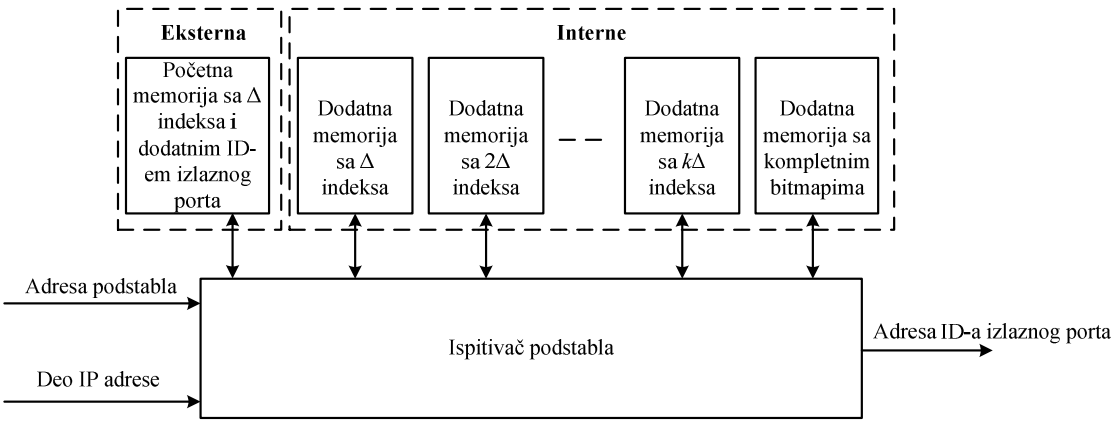


Slika 5.1.2.1. – BPFLSM arhitektura

Kao što se vidi sada se u paraleli pretražuju samo balansirana stabla na svim nivoima. Selektor se u BPFLSM nalazi iza blokova za nalaženje podstabla i selektuje uspešan rezultat bloka za nalaženje podstabla najdubljeg nivoa, te prosleđuje adresu podstabla u početnoj memoriji bloka za nalaženje prefiksa, zajedno sa odgovarajućim delom IP adrese koji je neophodan za pretragu interne strukture podstabla. I dalje se koristi statičko adresiranje, a svaki nivo dobija memorijski blok u početnoj memoriji bloka za nalaženje prefiksa. Blok za nalaženje prefiksa je dat na slici 5.1.2.2 i gotovo je identičan odgovarajućem bloku u okviru BPFL-a prikazanom na slici 5.1.7. Osnovna razlika je što početna memorija sadrži dodatni ID izlaznog porta, i što se memorije koriste za čuvanje podstabala sa svih nivoa. Takođe, kao što je prikazano na slici 5.1.2.2 početna memorija se može izmestiti u eksterni memorijski čip, čime se smanjuju interni memorijski zahtevi integrisanog čipa pa se može koristiti i neki jeftiniji integrisani čip sa manje resursa. Razlog za izmeštanje početne memorije, a ne neke od dodatnih memorija leži u činjenici da ova memorija ima najveće memorijske zahteve jer je koriste sva podstabla, za razliku od dodatnih memorija. U slučaju da se koriste tri eksterne memorije, tada pored izlazne i početne memorije, može da se izmesti i neka od

dodatnih memorija čija magistrala podataka nije prevelika (veća od 144b kako je rečeno u okviru četvrtog poglavlja).

Što se tiče proračuna memorijskih resursa osnovna razlika u odnosu na originalni BPFL je udruživanje memorijskih blokova koji nose informacije o internoj strukturi podstabala kao što se i vidi sa slika 5.1.2.1 i 5.1.2.2. Otuda za memorijske i registarske resurse blokova za nalaženje podstabla i dalje važe formule (5.1.1.1) i (5.1.1.2).



Slika 5.1.2.2. – Blok za nalaženje prefiksa u BPFLSM

U početnoj memoriji su dodeljene lokacije za sva neprazna podstabla svih nivoa, pri čemu su lokacije u odnosu na originalni BPFL proširene sa dodatnim ID-em izlaznog porta. Stoga, u BPFLSM zahtevani memorijski resursi početne memorije (M_{pm}) iznose:

$$M_{pm} = \left(\sum_{i=1}^{L/D} \left(N_{bs}^{i} \cdot N_{c}^{i} \right) \right) \cdot (\Delta \cdot (D+1) + L_{dm} + H) \qquad (5.1.2.1)$$

gde je L_{dm} dužina pokazivača na dodatne memorije. Pri tome važi sličan proračun za L_{dm} kao i u slučaju BPFL-a samo što treba uzeti u obzir da su dodatne memorije sada deljene između nepraznih podstabala svih nivoa. Otuda se dužina pokazivača na dodatnu memoriju L_{dm} može predstaviti kao:

$$L_{dm} = \lfloor \log_2(k+1) \rfloor + \left\lceil \log_2 \left(\max_{j=1..k+1} N_m^j \right) \right\rceil \qquad (5.1.2.2)$$

gde je $\max_{j=1..k+1} N_m^j$ broj lokacija najveće od dodatnih memorija u bloku za nalaženje prefiksa (najveća u smislu najvećeg broja lokacija).

Dodatne memorije imaju istu strukturu kao i u originalnom BPFL-u, samo što sada sadrže popunjene čvorove podstabala različitih nivoa. Zahtevani memorijski resursi dodatnih memorija koje čuvaju indekse popunjenih čvorova ($M_{dm}^{1..k}$) iznose:

$$M_{dm}^{1..k} = \sum_{j=1}^{k} N_m^j \cdot j \cdot \Delta \cdot (D+1) \tag{5.1.2.3}$$

gde je N_m^j broj lokacija u dodatnoj memoriji j. U dodatnoj memoriji koja čuva kompletan bitmap čuvaju se kompletni bitmapi gusto popunjenih podstabala, pri čemu je dužina bitmapa $2^{D+1} - 2$. Zahtevani memorijski resursi dodatne memorije koja čuva kompletne bitmape gusto popunjenih podstabala (M_{dm}^{k+1}) iznose:

$$M_{dm}^{k+1} = N_m^{k+1} \cdot (2^{D+1} - 2) \tag{5.1.2.4}$$

gde je N_m^{k+1} broj lokacija u dodatnoj memoriji koja čuva kompletne bitmape gusto popunjenih podstabala.

Izlazna memorija čuva ID-eve izlaznih portova, pri čemu početne memorije i dodatne memorije imaju unapred dodeljen prostor u okviru izlazne memorije tj. koristi se statičko adresiranje. Tako da su zahtevani memorijski resursi za izlaznu memoriju (M_{iz}):

$$M_{iz} = H \cdot \left[\sum_{i=1}^{L/D} \left(N_{bs}^i \cdot N_c^i \cdot \Delta \right) + \sum_{j=1}^{k} \left(N_m^j \cdot j \cdot \Delta \right) + N_m^{k+1} \cdot (2^{D+1} - 2) \right] \tag{5.1.2.5}$$

gde je H dužina ID-a izlaznog porta.

Kada se koriste eksterni memorijski čipovi prvo se izmešta izlazna memorija jer je najveća. Potom se izmeštaju početna memorija i dodatna memorija sa Δ prefiksa bloka za nalaženje prefiksa jer su one sledeće najveće memorije. Otuda su interni memorijski resursi (M_{int}^i):

$$M_{int}^i = \begin{cases} M_{bs} + M_{dm}^{1..k+1} + M_{pm}, & i = 1 \\ M_{bs} + M_{dm}^{1..k+1}, & i = 2 \\ M_{bs} + M_{dm}^{2..k+1}, & i = 3 \end{cases} \tag{5.1.2.6}$$

pri čemu i predstavlja broj eksternih memorijskih čipova na raspolaganju. $M_{dm}^{1..k+1}$ predstavlja ukupne memorijske resurse za dodatne memorije:

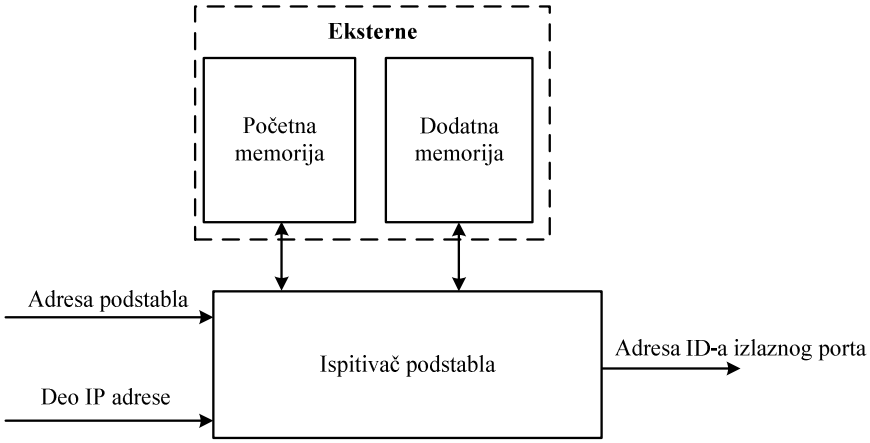
$$M_{dm}^{1..k+1} = M_{dm}^{1..k} + M_{dm}^{k+1} \tag{5.1.2.7}$$

Sa $M_{dm}^{2..k+1}$ su predstavljeni memorijski resursi koje zauzimaju sve dodatne memorije sem dodatne memorije sa Δ prefiksa.

Sa stanovišta ažuriranja nema velikih razlika u odnosu na BPFL. Osnovna razlika je u dodatnom slučaju koji može da se javi. U slučaju brisanja/dodavanja prefiksa, dotični prefiks može da utiče na brisanje/dodavanje dodatog ID-a izlaznog porta u podstablima na dubljim nivoima. Isto važi i za slučaj modifikacije ID-a izlaznog porta postojećeg prefiksa. U najgorem slučaju broj ovih ažuriranja može biti veoma velik. Najgori slučaj bi bio kad bi postojala podstabla samo na najdubljem nivou i još jedan prefiks koji odgovara korenu osnovnog binarnog stabla. Tada bi promena u korenu osnovnog binarnog stabla zahtevala promenu u dodatom ID-u izlaznog porta svih podstabala na najdubljem nivou i ukoliko bi sva ta podstabla postojala onda bi broj promena bio 2^{L-D}, gde je L, dužina IP adrese, a D veličina jednog nivoa tj. dubina podstabla na jednom nivou. Dakle, kompleksnost ažuriranja u najgorem slučaju se pogoršala za BPFLSM algoritam u odnosu na BPFL algoritam. Ali, u praksi kompleksnost ovakvog slučaja je znatno manja, jer obično postojeći prefiksi koji imaju decu predstavljaju agregaciju drugih postojećih prefiksa, pa je broj podstabala na koja neki prefiks može da utiče tipično znatno manji od broja čvorova svih balansiranih stabala nekog nivoa.

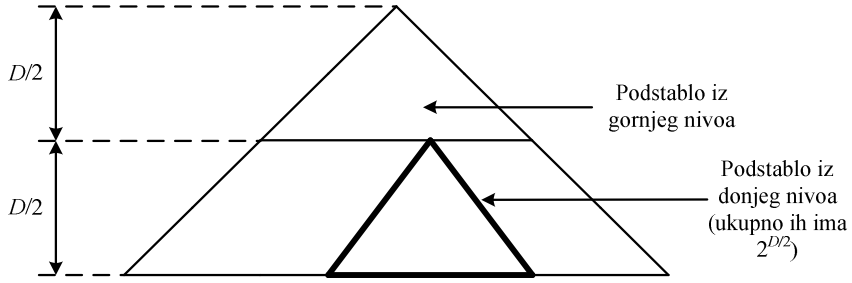
5.1.3. BPFLSS

Kao što smo videli u prethodnom odeljku, BPFLSM je omogućio udruživanje blokova za nalaženje prefiksa, a samim tim i njihovih memorijskih resursa. Udruživanjem memorijskih resursa je omogućeno da se najveća memorija u bloku za nalaženje prefiksa - početna memorija izmesti u eksterni memorijski čip i time značajno smanje interni memorijski zahtevi. BPFLSS dodatno unapređuje memorijsku organizaciju bloka za nalaženje prefiksa BPFLSM-a. Umesto višestrukih dodatnih memorija koristi se samo jedna čime je omogućeno da se kompletni interni memorijski resursi bloka za nalaženje prefiksa izmeste u dva eksterna memorijska čipa i time dodatno smanje interni memorijski zahtevi. Upotreba samo jedne dodatne memorije je omogućena podelom podstabala na dva nivoa.



Slika 5.1.3.1. – Blok za nalaženje prefiksa u BPFLSS

Opšta arhitektura BPFLSS-a je identična arhitekturi BPFLSM-a prikazanoj na slici 5.1.2.1. Suštinska razlika između BPFLSS-a i BPFLSM-a je u organizaciji memorijskih resursa za čuvanje podstabala u bloku za nalaženje prefiksa. Struktura bloka za nalaženje prefiksa BPLFSS-a je prikazana na slici 5.1.3.1. Kao što se može videti postoje samo dve memorije i obe mogu da se izmeste u eksterne memorijske čipove. Važno je napomenuti da se i u BPFLSS uz svako neprazno podstablo čuva i dodatni ID izlaznog porta kao i kod BPFLSM-a, čime je omogućeno formiranje zajedničkog bloka za nalaženje prefiksa.



Slika 5.1.3.2. – Podela podstabla na gornji i donji nivo

U BPFLSS-u se čuvaju samo indeksi popunjenih čvorova za veoma retko popunjena podstabla (ona koja sadrže maksimalno Δ prefiksa). Indeksi popunjenih čvorova podstabla se, uz dodatni ID izlaznog porta, čuvaju u odgovarajućoj lokaciji početne memorije. Svakom indeksu popunjenog čvora je statički dodeljena jedna lokacija u izlaznoj memoriji gde se čuva ID izlaznog porta tog čvora. Struktura lokacije početne memorije u slučaju veoma retko popunjenih podstabala je prikazana na slici 5.1.3.3. U slučaju podstabala sa više od Δ prefiksa, podstabla su podeljena na dva nivoa - gornji i donji. Gornji nivo sadrži samo jedno podstablo, a donji nivo sadrži $2^{D/2}$

podstabala, kao na slici 5.1.3.2. Bitmap podstabla gornjeg nivoa se čuva u početnoj memoriji, a bitmapi podstabala donjeg nivoa koja sadrže bar jedan popunjen čvor se čuvaju u dodatnoj memoriji. Bitmapi podstabala donjeg nivoa koja ne sadrže nijedan popunjen čvor se ne čuvaju. Struktura lokacije početne memorije za slučaj podstabala sa više od Δ prefiksa je prikazana na slici 5.1.3.3. Dodatni ID izlaznog porta se čuva i u ovom slučaju jer i dalje postoji mogućnost da se ne nađe prefiks u pretraživanom podstablu.

Struktura lokacije za podstablo sa ≤Δ prefiksa

0	Dodati ID izlaznog porta	Δ indeksa čvorova sa ID-em izlaznog porta

Struktura lokacije za podstablo sa >Δ prefiksa

1	Dodati ID izlaznog porta	Bitmap gornje polovine	Bitmap postojanja podstabala donjeg nivoa	Pokazivač na dodatnu memoriju	Pokazivač na izlaznu memoriju	Brojač max. broja popunjenih čvorova po podstablu

Slika 5.1.3.3. – Struktura lokacije u početnoj memoriji

Bitmap postojanja podstabala definiše koja podstabla donjeg nivoa su neprazna i na osnovu njega se utvrđuje da li uopšte treba pristupiti dodatnoj memoriji za datu IP adresu. Pokazivač na dodatnu memoriju daje lokaciju prvog nepraznog podstabla donjeg nivoa, a lokacije ostalih nepraznih podstabala se određuju kao ofset u odnosu na prvo neprazno podstablo. Pri računanju ofseta se u obzir uzimaju samo neprazna podstabla donjeg nivoa. U izlaznoj memoriji je dinamički dodeljen memorijski blok za ID-eve izlaznih portova popunjenih čvorova. Na početak memorijskog bloka ukazuje pokazivač na izlaznu memoriju. Brojač maksimalnog broja popunjenih čvorova po podstablu definiše broj lokacija u izlaznoj memoriji koje su dodeljene jednom nepraznom podstablu gornjeg/donjeg nivoa. Brojač dostiže maksimalnu vrednost u slučaju kad je bar jedno podstablo gornjeg ili donjeg nivoa maksimalno popunjeno i tada brojač ima vrednost $2^{D/2+1} - 2$ (broj čvorova u jednom podstablu gornjeg/donjeg nivoa). Početak bloka dodeljen jednom nepraznom podstablu se računa kao ofset u odnosu na pokazivač na izlaznu memoriju, pri čemu u proračun ulaze samo neprazna podstabla tj. njihov broj lokacija. Unutar tako određenog bloka lokacija, pozicija odgovarajućeg ID-a izlaznog porta se računa kao redni broj popunjenog čvora u podstablu uzimajući u obzir samo popunjene čvorove tog podstabla. Tip strukture lokacije u početnoj memoriji je određen vrednošću prvog bita kao što se može videti sa slike 5.1.3.3.

Sa stanovišta ažuriranja postoje dodatne razlike u odnosu na BPFLSM. U BPFLSS može doći do potencijalnih premeštanja u izlaznoj memoriji usled promene broja nepraznih podstabala ili usled promene maksimalnog broja popunjenih čvorova po jednom podstablu, pri čemu se pod podstablom misli na podstablo gornjeg/donjeg nivoa. U najgorem slučaju broj premeštanja je $2^{D+1} - 2$ što predstavlja ukupan broj čvorova jednog podstabla. Takođe, promena broja nepraznih podstabala može da izazove i premeštanje u dodatnoj memoriji. U najgorem slučaju broj premeštanja je jednak broju podstabala donjeg nivoa $2^{D/2}$. Oba navedena slučaja su povoljnija od najgoreg slučaja koji se javljao u BPFL i BPFLSM, a to je pomeranje svih čvorova svih balansiranih stabala u bloku za nalaženje podstabla u nivou sa najvećim brojem podstabala. Pošto je struktura bloka za nalaženje podstabla ista u sva tri algoritma (BPFL, BPFLSM i BPFLSS), najgori slučaj ažuriranja je identičan u sva tri predložena lukap algoritma.

Što se tiče proračuna memorijskih resursa za BPFLSS, osnovne razlike u odnosu na originalni BPFL su udruživanje blokova za nalaženje prefiksa i promena načina čuvanja informacija o internoj strukturi podstabala. Otuda za memorijske i registarske resurse blokova za nalaženje podstabla i dalje važe (5.1.1.1) i (5.1.1.2). U početnoj memoriji su dodeljene lokacije za sva neprazna podstabla svih nivoa, pri čemu se dužina lokacije dimenzioniše za slučaj kada podstablo ima više od Δ prefiksa (prema toj dužini se proračunava vrednost Δ tj. koliko indeksa stane u lokaciju). Bitmap podstabla gornjeg nivoa mora da bude dužine $2^{D/2+1} - 2$ da bi indeksirao sve čvorove. Na osnovu dužine tog bitmapa se vidi koliko ima čvorova po podstablu gornjeg/donjeg nivoa, pa je lako zaključiti da je za brojač (slika 5.1.3.3) neophodno $D/2 + 1$ bita. Bitmap postojanja podstabala donjeg nivoa mora da bude dužine $2^{D/2}$ da bi ih sve indeksirao. Stoga, u BPFLSS zahtevani memorijski resursi početne memorije (M_{pm}) iznose:

$$M_{pm} = \left(\sum_{i=1}^{\frac{L}{D}} \left(N_{bs}^{i} \cdot N_{c}^{i} \right) \right) \cdot \tag{5.1.3.1}$$

$$\cdot \left(1 + H + (2^{D/2+1} - 2) + 2^{D/2} + L_{dm} + L_{im} + (D/2 + 1) \right)$$

gde su L_{dm} i L_{im} dužine pokazivača na dodatnu memoriju i izlaznu memoriju, respektivno. Dodatna memorija se deli na blokove veličine $2^{D/2}$ lokacija radi lakšeg menadžmenta memorije i ažuriranja, pa je:

$$L_{dm} = \left\lfloor \log_2\left(N_{D/2} \cdot 2^{D/2}\right) \right\rfloor \tag{5.1.3.2}$$

gde je $N_{D/2}$ potreban broj blokova veličine $2^{D/2}$ u dodatnoj memoriji za smeštanje bitmapa nepraznih podstabala donjeg nivoa gušće popunjenih podstabala (onih podstabala koji imaju više od Δ popunjenih čvorova). Dinamički alociran deo izlazne memorije je podeljen na blokove veličine 2^{D+1} radi lakšeg menadžmenta memorije i ažuriranja, pa je:

$$L_{im} = \lfloor \log_2(N_{D+1} \cdot 2^{D+1}) \rfloor \tag{5.1.3.3}$$

gde je N_{D+1} broj blokova veličine 2^{D+1} u dinamički alociranom delu izlazne memorije. U skladu sa datom definicijom za $N_{D/2}$ zahtevani memorijski resursi za dodatnu memoriju (M_{dm}) iznose:

$$M_{dm} = N_{D/2} \cdot 2^{D/2} \cdot \left(2^{D/2+1} - 2\right) \tag{5.1.3.4}$$

Imajući u vidu statički dodeljeni prostor u izlaznoj memoriji od Δ lokacija za svaku lokaciju u početnoj memoriji i definiciju za N_{D+1}, zahtevani memorijski resursi za izlaznu memoriju (M_{iz}) iznose:

$$M_{iz} = H \cdot \left[\left(\sum_{i=1}^{L/D} \left(N_{bs}^{i} \cdot N_{c}^{i} \right) \right) \cdot \Delta + N_{D+1} \cdot 2^{D+1} \right] \tag{5.1.3.5}$$

gde je H dužina ID-a izlaznog porta.

Kada se koriste eksterni memorijski čipovi prvo se izmešta izlazna memorija jer je najveća, pa potom početna memorija i dodatna memorija bloka za nalaženje prefiksa. Otuda su interni memorijski resursi (M_{int}^{i}):

$$M_{int}^{i} = \begin{cases} M_{bs} + M_{dm} + M_{pm}, & i = 1 \\ M_{bs} + M_{dm}, & i = 2 \\ M_{bs}, & i = 3 \end{cases} \tag{5.1.3.6}$$

pri čemu i predstavlja broj eksternih memorijskih čipova na raspolaganju. Kao što se vidi upotrebom tri eksterna memorijska čipa, interni memorijski resursi postaju veoma mali.

5.2. Poređenje predloženih lukap algoritama sa postojećim rešenjima

Osnovne osobine po kojima se meri kvalitet lukap algoritama su protok lukap modula, zahtevani memorijski resursi i kompleksnost ažuriranja lukap tabele. Ukoliko su memorijski resursi dovoljno mali onda se mogu koristiti brže memorije, i samim tim omogućiti veći protok lukap modula. Takođe, manji memorijski resursi znače upotrebu manjeg broja eksternih memorijskih čipova čime se smanjuje potreban prostor na štampanoj ploči za smeštanje implementacije, i omogućava lakša sistemska integracija jer veći broj eksternih čipova nije jednostavno efikasno povezati. Kompleksnost ažuriranja je važna jer se usled promena topologije mreže sadržaj lukap tabele mora menjati i neophodno je da ta promena bude dovoljno brza jer u suprotnom bi lukap algoritam bio previše inertan i ne bi mogao da prati promene u mreži.

Predloženi lukap algoritmi će biti poređeni sa postojećim lukap algoritmima opisanim u prethodnom poglavlju. Predložena rešenja će biti poređena sa postojećim imajući u vidu memorijske zahteve, pošto oni određuju kompleksnost implementacije i protok lukap modula. Poređenje će biti urađeno za deset realnih lukap tabela IPv4 adresa koje se kreću u opsegu od 104779 prefiksa do 365309 prefiksa (što odgovara trenutno najvećim lukap tabelama). Korišćene su lukap tabele koje se mogu preuzeti sa [93]. Preuzete lukap tabele su u MRT (*Multi-threaded Routing Toolkit*) formatu [94]. Stoga je napisana aplikacija koja omogućava čitanje zapisa lukap tabela iz MRT formata u vidu parova prefiks i dužina prefiksa. U prilogu B je dat opis dela MRT formata koji se odnosi na zapise lukap tabela rutera. Poređenje je vršeno i za IPv4 i za IPv6 lukap tabele. Međutim, pošto su IPv6 lukap tabele još uvek male (do 10000 prefiksa), izvršeno je generisanje IPv6 lukap tabela na osnovu strukture korišćenih IPv4 lukap tabela [95].

U IPv6 lukap tabelama najzastupljeniji su prefiksi u opsegu 32-48 bita, a u IPv4 lukap tabelama prefiksi u opsegu 16-32 bita. Stoga je uzeto da se svaki prefiks iz IPv4 lukap tabele preslikava u duplo duži prefiks IPv6 tabele, pri čemu se prefiksi koji su kraći od 21 bita produžavaju na 21 bit, s obzirom da je to najkraća dužina mrežnog prefiksa u IPv6 tabelama [95]. Pri tome, da ne bi došlo do toga da postoje samo prefiksi parne dužine, u proseku se svaki četvrti prefiks seli na susedni neparni prefiks [95]. Na ovaj način su IPv6 mrežne adrese u suštini duge 64 bita, a ne 128 bita, što je u skladu sa planiranom strukturom IPv6 adrese, gde je predviđeno da se viših 64 bita koristi kao

mrežni deo adrese, a nižih 64 bita kao host deo [95]. Gustine podstabala u IPv6 su podešene na osnovu gustina podstabala u IPv4, pri čemu su korišćeni faktori smanjenja gustine pošto se očekuje da će prosečna gustina podstabala u IPv6 da bude manja zbog većeg prostora. Prefiksi nivoa 1 i 2 IPv4 tabele generišu prefikse nivoa 3 i 4 IPv6 tabele, pri čemu je faktor smanjenja gustine postavljen na 2. Prefiksi nivoa 3 IPv4 tabele generišu prefikse nivoa 5 i 6 IPv6 tabele, pri čemu je faktor smanjenja gustine postavljen na 3. Prefiksi nivoa 4 IPv4 tabele generišu prefikse nivoa 7 i 8 IPv6 tabele, pri čemu je faktor smanjenja gustine postavljen na 4.

U svim analiziranim slučajevima je uzeto da dužina NH informacije iznosi $H = 16$ bita, što omogućava adresiranje do 65536 portova rutera, a to je više nego dovoljno čak i za rutere velikih kapaciteta sa velikim brojem portova. U prilogu A su za postojeće lukap algoritme date tabele memorijskih zahteva sa vrednostima karakterističnih parametara lukap algoritama u skladu sa matematičkim formulacijama memorijskih zahteva datim u prethodnom poglavlju. U prilogu A su takođe date tabele memorijskih zahteva sa vrednostima karakteričnih parametara predloženih lukap algoritama u skladu sa matematičkim formulacijama njihovih memorijskih zahteva koje su date u okviru ovog poglavlja.

U okviru komparacije lukap algoritama poređeni su ukupni memorijski zahtevi, kao i interni memorijski zahtevi zbog njihovog značaja za efikasnost implementacije lukap algoritama kao što je objašnjeno u četvrtom poglavlju. Interni memorijski zahtevi su analizirani u tri slučaja kada je na raspolaganju jedna, dve ili tri eksterne memorije. Slučajevi, kada su više od tri eksterne memorije na raspolaganju, nisu razmatrani jer oni nisu praktični za implementaciju zbog prostora na štampanoj ploči. Za dužinu magistrale podataka je postavljeno ograničenje od 144b kao u [79]. Memorijski zahtevi nekih od poređenih lukap algoritama zavise od izbora vrednosti karakterističnih parametara algoritma, npr. maksimalna veličina kofe u slučaju TCAM algoritma sa promenljivim kofama. U takvim situacijama su usvajane vrednosti parametara kojima se postižu minimalni memorijski zahtevi. Tabela 5.2.1 prikazuje rezultate za analizirane IPv4 tabele, a tabela 5.2.2 za analizirane IPv6 tabele.

Tabela 5.2.1. – Poređenje lukap algoritama za analizirane IPv4 tabele

Veličina tabele	Algoritam	$M_{tot}[MB]$	$M_{int1}[MB]$	$M_{int2}[MB]$	$M_{int3}[MB]$
104779	FIPL	2.3616	1.2069	0.2347	0.0395
	EHAF	6.4539	0.6851	0.4663	0.3726
	POLP	1.3156	1.0267	0.7378	0.7166
	Prior. stablo	1.0367	0.8948	0.7567	0.6235
	TCAM	0.5996	0.3997	0.3997	0.3997
	TCAM prom. kofe	0.6037	0.4032	0.4021	0.4021
	M-12Wb	0.9171	0.3485	0.1668	0.1668
	HASH	1.1118	0.8618	0.8071	0.7817
	PFHT	2.4647	1.4647	1.0696	0.8508
	BPFL	0.9429	0.1596	0.0931	0.0821
	BPFLSM	0.9559	0.1726	0.0911	0.0801
	BPFLSS	0.6525	0.1726	0.0921	0.0127
130287	FIPL	2.8626	1.4862	0.2882	0.0549
	EHAF	7.9020	0.8185	0.5997	0.4747
	POLP	1.5928	1.2423	0.8918	0.8660
	Prior. stablo	1.2891	1.1037	0.9270	0.7600
	TCAM	0.7455	0.4970	0.4970	0.4970
	TCAM prom. kofe	0.7501	0.5009	0.4997	0.4997
	M-12Wb	1.1456	0.4382	0.2083	0.2083
	HASH	1.6535	1.1222	1.0676	1.0129
	PFHT	4.1175	1.9925	1.5098	1.2910
	BPFL	1.2115	0.1998	0.1184	0.1045
	BPFLSM	1.2274	0.2157	0.1156	0.1016
	BPFLSS	0.8045	0.2091	0.1099	0.0158
153126	FIPL	3.1879	1.6278	0.2996	0.0419
	EHAF	8.6740	0.8142	0.5955	0.5017
	POLP	1.8311	1.4276	1.0242	0.9945
	Prior. stablo	1.5516	1.3353	1.1242	0.9218
	TCAM	0.8762	0.5841	0.5841	0.5841
	TCAM prom. kofe	0.8811	0.5883	0.5869	0.5869
	M-12Wb	1.3477	0.5139	0.2445	0.2445
	HASH	1.8379	1.3066	1.2520	1.1973
	PFHT	4.4118	2.2868	1.7194	1.5006
	BPFL	1.4411	0.2283	0.1372	0.1221
	BPFLSM	1.4585	0.2457	0.1354	0.1202
	BPFLSS	0.9296	0.2364	0.1260	0.0173
175399	FIPL	3.5400	1.8160	0.3475	0.0606
	EHAF	9.8570	0.9563	0.7220	0.5970
	POLP	2.0620	1.6072	1.1525	1.1191
	Prior. stablo	1.7773	1.5310	1.2877	1.0498
	TCAM	1.0036	0.6691	0.6691	0.6691
	TCAM prom. kofe	1.0090	0.6736	0.6721	0.6721
	M-12Wb	1.5378	0.5898	0.2796	0.2796
	HASH	2.0257	1.4944	1.4398	1.3851
	PFHT	4.7301	2.6051	1.9553	1.7365
	BPFL	1.6933	0.2614	0.1587	0.1414
	BPFLSM	1.7134	0.2815	0.1553	0.1379
	BPFLSS	1.0650	0.2680	0.1406	0.0199
210039	FIPL	4.2418	2.2300	0.5652	0.2420
	EHAF	12.4142	1.3390	1.1046	0.8485
	POLP	2.5127	1.9728	1.4329	1.3912
	Prior. stablo	2.1283	1.8294	1.5358	1.2466
	TCAM	1.2018	0.8012	0.8012	0.8012

	TCAM prom. kofe	1.2077	0.8062	0.8046	0.8046
	M-12Wb	1.8561	0.7223	0.3375	0.3375
	HASH	2.3841	1.8528	1.7357	1.6185
	PFHT	5.4883	3.3633	2.6054	2.1366
	BPFL	2.0710	0.3216	0.2041	0.1850
	BPFLSM	2.0982	0.3488	0.1867	0.1676
	BPFLSS	1.2946	0.3305	0.1669	0.0269
240958	FIPL	4.5232	2.2949	0.4791	0.1188
	EHAF	12.7649	1.1595	0.9251	0.8001
	POLP	2.8070	2.2034	1.5999	1.5533
	Prior. stablo	2.4416	2.0946	1.7486	1.4222
	TCAM	1.3788	0.9192	0.9192	0.9192
	TCAM prom. kofe	1.3852	0.9246	0.9228	0.9228
	M-12Wb	2.1109	0.8221	0.3838	0.3838
	HASH	2.6572	2.1259	2.0088	1.8916
	PFHT	5.9755	3.8525	2.9920	2.5233
	BPFL	2.4276	0.3489	0.2178	0.1970
	BPFLSM	2.4539	0.3752	0.2110	0.1900
	BPFLSS	1.4332	0.3501	0.1842	0.0256
273791	FIPL	5.0140	2.5354	0.5507	0.1655
	EHAF	13.9637	1.2796	1.0452	0.9046
	POLP	3.1626	2.4821	1.8017	1.7492
	Prior. stablo	2.8395	2.4279	2.0224	1.6486
	TCAM	1.5665	1.0444	1.0444	1.0444
	TCAM prom. kofe	1.5736	1.0503	1.0483	1.0483
	M-12Wb	2.3945	0.9308	0.4354	0.4354
	HASH	3.5561	2.4311	2.3139	2.1967
	PFHT	8.8762	4.3762	3.3879	2.9192
	BPFL	2.8139	0.3932	0.2501	0.2285
	BPFLSM	2.8428	0.4221	0.2405	0.2183
	BPFLSS	1.6134	0.3878	0.2062	0.0288
307915	FIPL	5.4219	2.7140	0.5774	0.1680
	EHAF	15.1172	1.3402	1.1058	0.9652
	POLP	3.4934	2.7414	1.9895	1.9314
	Prior. stablo	3.1934	2.7271	2.2732	1.8552
	TCAM	1.7619	1.1746	1.1746	1.1746
	TCAM prom. kofe	1.7693	1.1808	1.1787	1.1787
	M-12Wb	2.6649	1.0365	0.4846	0.4846
	HASH	3.8741	2.7491	2.6319	2.5147
	PFHT	9.4710	4.9710	3.8640	3.3953
	BPFL	3.1791	0.4331	0.2782	0.2551
	BPFLSM	3.2103	0.4643	0.2681	0.2443
	BPFLSS	1.7928	0.4239	0.2258	0.0311
348866	FIPL	6.2085	3.1534	0.7602	0.3128
	EHAF	17.7521	1.7820	1.4044	1.1700
	POLP	3.9545	3.1028	2.2512	2.1854
	Prior. stablo	3.6182	3.0758	2.5471	2.0615
	TCAM	1.9962	1.3308	1.3308	1.3308
	TCAM prom. kofe	2.0045	1.3375	1.3352	1.3352
	M-12Wb	3.0211	1.1743	0.5493	0.5493
	HASH	4.2837	3.1587	2.9087	2.7915
	PFHT	10.2966	5.7966	4.5442	3.5442
	BPFL	3.6437	0.5018	0.2782	0.2551
	BPFLSM	3.6828	0.5409	0.3068	0.2806
	BPFLSS	2.0628	0.4925	0.2561	0.0385

365309	FIPL	6.3830	3.2180	0.7544	0.2958
	EHAF	18.2888	1.8235	1.4353	1.2010
	POLP	4.1129	3.2270	2.3411	2.2727
	Prior. stablo	3.7887	3.2154	2.6590	2.1462
	TCAM	2.0903	1.3935	1.3935	1.3935
	TCAM prom. kofe	2.0983	1.4003	1.3980	1.3980
	M-12Wb	3.1675	1.2295	0.5759	0.5759
	HASH	4.5272	3.4022	3.1522	2.9022
	PFHT	10.9358	6.4358	5.1263	4.1263
	BPFL	3.8396	0.5217	0.3313	0.3052
	BPFLSM	3.8795	0.5616	0.3209	0.2945
	BPFLSS	2.1515	0.5089	0.2661	0.0395

U slučaju IPv4 tabela predloženi lukap algoritmi (BPFL, BPFLSM, BPFLSS) po pitanju ukupnih memorijskih resursa spadaju u najbolje. Jedino osnovni TCAM algoritam i TCAM algoritam sa kofama promenljive veličine zahtevaju manju ukupnu memoriju. Od predloženih lukap algoritama najmanje ukupne memorijske resurse zahteva BPFLSS. BPFL i BPFLSM algoritmi zajedno sa M-12Wb algoritmom, stablom sa prioritetima i POLP-om zatim slede po ukupnim memorijskim zahtevima. . Međutim, interni memorijski zahtevi su važniji od eksternih memorijskih zahteva. Najmanje interne memorije zahtevaju predloženi algoritmi (BPFL, BPFLSM, BPFLSS), a slede ih M-12Wb algoritam i FIPL algoritam. Algoritam baziran na stablu sa prioritetima i POLP algoritam nisu veoma dobri, pošto su interni memorijski zahtevi veliki čak i u slučaju tri eksterne memorije. M-12Wb algoritmu su potrebne dve eksterne memorije za dobre performanse. FIPL algoritam postiže veoma dobre rezultate u slučaju tri eksterne memorije koji su slični memorijskim zahtevima predloženih lukap algoritama. Ovi dobri rezultati FIPL-a su posledica toga što dva nivoa m-arnog stabla u FIPL-u sadrže najveći broj čvorova stabla pa se njihovim izmeštanjem u eksterne memorije (zajedno sa izlaznom memorijom) interni memorijski zahtevi smanjuju na veoma malu vrednost. Treba primetiti da BPFLSS u slučaju tri eksterne memorije ostvaruje ubedljivo najbolje rezultate jer zahvaljujući svojoj strukturi omogućava da se kompletni memorijski resursi bloka za nalaženje prefiksa izmeste u eksterne memorije tako da su preostali interni memorijski zahtevi veoma mali čak i u slučaju velikih lukap tabela. Pri tome je važno naglasiti da kod BPFLSS-a samo jedna od tri eksterne memorije ima široku magistralu podataka (ona u kojoj je smeštena početna memorija), dok druge dve imaju uske magistrale (tačnije 16 i 30 bita).

Tabela 5.2.2. – Poređenje lukap algoritama za analizirane IPv6 tabele

Veličina tabele	Algoritam	$M_{tot}[MB]$	$M_{int1}[MB]$	$M_{int2}[MB]$	$M_{int3}[MB]$
104779	FIPL	26.4665	24.5952	18.7434	15.2740
	EHAF	-	-	-	-
	POLP	3.8553	3.0251	2.1949	2.1727
	Prior. stablo	1.4364	1.3200	1.2042	1.0919
	TCAM	0.9993	0.7994	0.7994	0.7994
	TCAM prom. kofe	1.0048	0.8043	0.8032	0.8032
	M-12Wb	1.3773	0.7043	0.4238	0.4238
	HASH	1.4267	1.1767	1.1220	1.0966
	PFHT	2.9229	1.9229	1.4988	1.2800
	BPFL	0.7747	0.3370	0.1704	0.1512
	BPFLSM	0.8144	0.3767	0.1316	0.1079
	BPFLSS	0.9602	0.4094	0.1721	0.0918
130287	FIPL	34.9463	32.2694	23.2967	19.0730
	EHAF	-	-	-	-
	POLP	5.1351	4.0281	2.9211	2.8915
	Prior. stablo	1.7861	1.6414	1.5018	1.3629
	TCAM	1.2425	0.9940	0.9940	0.9940
	TCAM prom. kofe	1.2487	0.9994	0.9982	0.9982
	M-12Wb	1.5831	0.8116	0.4871	0.4871
	HASH	1.7570	1.5070	1.3898	1.3644
	PFHT	3.5603	2.5603	2.0502	1.5814
	BPFL	0.9477	0.4129	0.2146	0.1911
	BPFLSM	0.9960	0.4612	0.1618	0.1323
	BPFLSS	1.1528	0.5297	0.2332	0.1121
153126	FIPL	38.9168	35.8530	25.2138	20.7149
	EHAF	-	-	-	-
	POLP	5.7663	4.5229	3.2795	3.2462
	Prior. stablo	2.1370	1.9722	1.8079	1.6518
	TCAM	1.4604	1.1683	1.1683	1.1683
	TCAM prom. kofe	1.4671	1.1742	1.1728	1.1728
	M-12Wb	1.8603	0.9522	0.5724	0.5724
	HASH	2.0224	1.7724	1.6552	1.6005
	PFHT	4.0043	3.0043	2.4064	1.9376
	BPFL	1.0753	0.4662	0.2414	0.2122
	BPFLSM	1.1285	0.5194	0.1864	0.1504
	BPFLSS	1.3447	0.6017	0.2719	0.1242
175399	FIPL	42.3647	39.1040	28.5238	23.0901
	EHAF	-	-	-	-
	POLP	6.2901	4.9334	3.5768	3.5405
	Prior. stablo	2.4464	2.2581	2.0723	1.8924
	TCAM	1.6727	1.3382	1.3382	1.3382
	TCAM prom. kofe	1.6798	1.3444	1.3429	1.3429
	M-12Wb	2.1679	1.1036	0.6670	0.6670
	HASH	2.5957	2.0644	1.9473	1.8926
	PFHT	5.6678	3.5428	2.8490	2.3802
	BPFL	1.2099	0.5271	0.2759	0.2424
	BPFLSM	1.2699	0.5871	0.2134	0.1716
	BPFLSS	1.5384	0.6638	0.2937	0.1397
210039	FIPL	53.4192	49.2368	35.3221	28.5659
	EHAF	-	-	-	-
	POLP	8.2151	6.4889	4.7627	4.7143
	Prior. stablo	2.9295	2.7084	2.4926	2.2793
	TCAM	2.0031	1.6025	1.6025	1.6025

	TCAM prom. kofe	2.0106	1.6091	1.6075	1.6075
	M-12Wb	2.5071	1.2833	0.7714	0.7714
	HASH	2.7939	2.4426	2.3255	2.2708
	PFHT	6.2423	4.1173	3.3095	2.8408
	BPFL	1.4621	0.6461	0.3545	0.3143
	BPFLSM	1.5379	0.7219	0.2626	0.2123
	BPFLSS	1.8526	0.8400	0.3895	0.1726
240958	FIPL	55.2636	50.9650	36.9578	30.2509
	EHAF	-	-	-	-
	POLP	8.4930	6.7083	4.9236	4.8735
	Prior. stablo	3.3608	3.1117	2.8683	2.6283
	TCAM	2.2980	1.8384	1.8384	1.8384
	TCAM prom. kofe	2.3061	1.8455	1.8438	1.8438
	M-12Wb	3.0228	1.5390	0.9301	0.9301
	HASH	3.2784	2.7471	2.6300	2.5753
	PFHT	6.6272	4.5022	3.5843	3.1155
	BPFL	1.6280	0.7120	0.3856	0.3384
	BPFLSM	1.7088	0.7928	0.2940	0.2351
	BPFLSS	2.1230	0.8959	0.4019	0.1863
273791	FIPL	64.6045	59.3479	41.0710	33.6750
	EHAF	-	-	-	-
	POLP	10.1165	7.9900	5.8635	5.8039
	Prior. stablo	3.8840	3.6100	3.3377	3.0762
	TCAM	2.6111	2.0889	2.0889	2.0889
	TCAM prom. kofe	2.6200	2.0967	2.0948	2.0948
	M-12Wb	3.2544	1.6575	1.0013	1.0113
	HASH	3.8769	3.3456	3.0956	2.9785
	PFHT	8.0971	5.9721	4.9367	3.9367
	BPFL	1.8235	0.7923	0.4419	0.3877
	BPFLSM	1.9138	0.8826	0.3356	0.2686
	BPFLSS	2.3946	1.0515	0.5046	0.2072
307915	FIPL	73.0405	67.0188	46.5808	37.6687
	EHAF	-	-	-	-
	POLP	11.5191	9.0952	6.6713	6.6034
	Prior. stablo	4.3681	4.0660	3.7646	3.4721
	TCAM	2.9365	2.3492	2.3492	2.3492
	TCAM prom. kofe	2.9459	2.3575	2.3555	2.3555
	M-12Wb	3.5446	1.7978	1.0907	1.0907
	HASH	4.1973	3.6659	3.4160	3.2989
	PFHT	8.4499	6.3249	5.1630	4.1630
	BPFL	2.0419	0.8854	0.5002	0.4398
	BPFLSM	2.1426	0.9861	0.3791	0.3046
	BPFLSS	2.6806	1.2079	0.6008	0.2306
348866	FIPL	76.3446	70.0894	48.8444	39.6781
	EHAF	-	-	-	-
	POLP	12.0383	9.5072	6.9761	6.9051
	Prior. stablo	4.9490	4.6134	4.2827	3.9555
	TCAM	3.3270	2.6616	2.6616	2.6616
	TCAM prom. kofe	3.3380	2.6710	2.6689	2.6689
	M-12Wb	4.1923	2.1253	1.2899	1.2899
	HASH	5.3815	4.2565	4.0065	3.8893
	PFHT	12.0110	7.5110	6.1821	5.1821
	BPFL	2.2567	0.9736	0.5483	0.4784
	BPFLSM	2.3657	1.0826	0.4214	0.3359
	BPFLSS	3.0218	1.2529	0.5917	0.2489

365309	FIPL	74.6681	68.7223	50.8226	40.2319
	EHAF	-	-	-	-
	POLP	11.7177	9.2541	6.7905	6.7214
	Prior. stablo	5.1822	4.8323	4.4883	4.1454
	TCAM	3.4839	2.7871	2.7871	2.7871
	TCAM prom. kofe	3.4943	2.7963	2.7941	2.7941
	M-12Wb	4.7441	2.4194	1.4597	1.4597
	HASH	5.5640	4.4390	4.1890	4.0718
	PFHT	12.2884	7.7884	6.3980	5.3980
	BPFL	2.3757	1.0123	0.5713	0.4983
	BPFLSM	2.4896	1.1262	0.4396	0.3500
	BPFLSS	3.2431	1.2733	0.5867	0.2559

Predloženi lukap algoritmi su najbolji po pitanju ukupnih memorijskih zahteva u slučaju IPv6 tabela. Osnovni TCAM algoritam i TCAM algoritam sa kofama promenljive veličine, koji su bili najbolji u slučaju IPv4 tabela, sada su lošiji od predloženih lukap algoritama u slučaju većih IPv6 tabela, dok su za manje IPv6 tabele podjednaki. POLP algoritam sada ima znatno veće memorijske zahteve isto kao i FIPL. Razlog je što se prefiksi sada nalaze na većim dubinama pa strukture stabla ova dva algoritma sadrže mnogo praznih čvorova koji znatno povećavaju memorijske zahteve. Otuda POLP i FIPL nisu adekvatni za IPv6 lukap. S druge strane stablo sa prioritetima i algoritmi zasnovani na heširanju imaju slične memorijske zahteve kao i u IPv4 slučaju, ali i dalje su značajno lošiji po pitanju memorijskih zahteva u odnosu na predložene lukap algoritme, pogotovo sa stanovišta internih memorijskih zahteva. M-12Wb algoritam ima najmanje interne memorijske zahteve od postojećih lukap algoritama, ali su oni i dalje značajno veći od internih memorijski zahteva predloženih algoritama – oko 2-3 puta veći za slučaj kada su dve eksterne memorije na raspolaganju, odnosno 3-5 puta za slučaj kada su tri eksterne memorije na raspolaganju.

Kompleksnost ažuriranja lukap tabele je takođe veoma važan aspekt lukap algoritama. Ažuriranje se vrši samo u slučaju promena u topologiji mreže, pa je ažuriranje manje intezivno, a samim tim i manje kritično od samog lukapa. Takođe, ažuriranje lukap tabele je tipično centralizovano i tako da se najveći deo proračuna izvršava u kontrolnoj ravni na procesorskom modulu.

FIPL je sa stanovišta kompleksnosti ažuriranja veoma jednostavan s obzirom da FIPL u stvari predstavlja m-arno stablo pa se novi prefiksi lako dodaju, a takođe i lako brišu.

EHAF ne koristi veoma kompresovanu strukturu što se i vidi iz njegovih memorijskih zahteva, tako da ažuriranje nije veoma komplikovano i svodi se na ažuriranje odgovarajućih bitmapa.

U POLP-u većina slučajeva ažuriranja je identična ažuriranjima koja odgovaraju lukap algoritmima zasnovanim na binarnim stablima. Međutim, postoje slučajevi ažuriranja koji su veoma kompleksni. Na primer, u slučaju kada se dodaje novi prefiks može doći do dodavanja novih čvorova u memorije faza. Najgori slučaj je kada memorije faza odgovarajućeg pajplajna nemaju dovoljno slobodnog prostora za sve nove čvorove, pa može doći do pomeranja većeg brojeg čvorova da bi se oslobodio dovoljan prostor za smeštanje novih čvorova.

U slučaju prioritetnih stabala ažuriranje nije komplikovano jer se u slučaju dodavanja i brisanja prefiksa vrše veoma slične operacije kao u običnom binarnom stablu.

Kod TCAM algoritama je bitno voditi računa o sortiranosti zapisa u TCAM memoriji što može komplikovati ažuriranje. U slučaju TCAM algoritma sa kofama promenljive veličine i M-12Wb algoritma ažuriranje nije jednostavno jer se mora voditi računa o optimalnom izboru pokrivajućih prefiksa da bi se postigli optimalni memorijski zahtevi. Otuda u slučaju dodavanja/brisanja prefiksa može doći do pomeranja većeg broja prefiksa.

Kod osnovnog heš algoritma ažuriranje nije preterano kompleksno jer se ažuriranje svodi na brisanje/dodavanje novog para (prefiks, ID izlaznog porta) na lokaciji na koju ukazuje izračunati heš ažuriranog prefiksa. PFHT je komplikovan u slučaju brisanja prefiksa jer on utiče na brojače u Blum filtru tako što se odgovarajući brojači dekrementiraju. Kao posledica može doći do premeštanja određenih prefiksa jer su neke lokacije sada povoljnije usled manje vrednosti brojača. Pri tome sam proces određivanja koji prefiksi se trebaju premestiti je komplikovan.

Predloženi lukap algoritmi (BPFL, BPFLSM i BPFLSS) su u većini slučajeva veoma jednostavni za ažuriranje kao što je i opisano u prethodnim potpoglavljima. Međutim, u slučaju dodavanja ili brisanja podstabla može doći do pomeranja većeg broja čvorova u balansiranim stablima u cilju očuvanja balansiranosti.

Izvršene analiza pokazuje prednost predloženih lukap algoritama koji ostvaruju optimalan protok lukap modula, imajući pri tome male memorijske zahteve i

podržavajući efikasno i IPv4 i IPv6 tabele. Zbog malih memorijskih resursa koje zahtevaju, predloženi algoritmi su skalabilniji od postojećih lukap algoritama, odnosno mogu da podrže veći broj prefiksa u IP tabelama.

5.3. FPGA implementacija predloženih lukap algoritama

U okviru ovog potpoglavlja je izložena FPGA implementacija predloženih lukap algoritama: BPFL, BPFLSM i BPFLSS. Korišćen je Xilinx-ov čip Virtex5-SX240T [35], koji je izabran da se omogući smeštanje i najveće lukap tabele. Korišćeno je razvojno okruženje ISE Design Suite 12.2 [35]. Implementacija je izvršena za tri veličine lukap tabele: 104779, 210039 i 365309 prefiksa. Sve tri tabele su korišćene i u analizi u prethodnom potpoglavlju. Implementacija je izvršena i za IPv4 i za IPv6 tabele. Iskorišćeni resursi FPGA čipa u okviru implementacija predloženih lukap algoritama za IPv4 lukap tabele su prikazani u tabeli 5.3.1.

Tabela 5.3.1. – Iskorišćenost resursa FPGA čipa u okviru implementacija predloženih lukap algoritama za IPv4 lukap tabele

Veličina tabele	Broj ekster. mem.	Algoritam	$Reg.$	LE	$Mem.[Mb]$	$f_{max}[MHz]$
104779	1	BPFL	4.9K (3.3%)	8.2K (5.5%)	3.2 (17.2%)	150.9
		BPFLSM	4.4K (2.9%)	6.9K (4.6%)	2.5 (13.4%)	151.4
		BPFLSS	4.5K (3.0%)	3.9K (2.6%)	2.2 (11.8%)	149.3
	2	BPFL	4.9K (3.3%)	8.0K (5.3%)	2.5 (13.4%)	156.2
		BPFLSM	4.4K (2.9%)	6.9K (4.6%)	1.7 (9.1%)	162.6
		BPFLSS	4.5K (3.0%)	3.8K (2.5%)	1.4 (7.5%)	179.1
	3	BPFL	4.9K (3.3%)	8.0K (5.3%)	2.3 (12.4%)	158.9
		BPFLSM	4.4K (2.9%)	6.9K (4.6%)	1.6 (8.6%)	160.5
		BPFLSS	4.5K (3.0%)	3.7K (2.5%)	0.4 (2.2%)	179.1
210039	1	BPFL	12.7K (8.5%)	13.1K (8.7%)	4.9 (26.3%)	146.1
		BPFLSM	12.2K (8.1%)	11.4K (7.6%)	4.1 (22.0%)	143.0
		BPFLSS	12.3K (8.2%)	8.4K (5.6%)	4.1 (22.0%)	142.0
	2	BPFL	12.7K (8.5%)	12.9K (8.6%)	3.5 (18.8%)	158.8
		BPFLSM	12.2K (8.1%)	11.3K (7.5%)	2.5 (13.4%)	158.5
		BPFLSS	12.3K (8.2%)	8.3K (5.5%)	2.5 (13.4%)	162.2
	3	BPFL	12.7K (8.5%)	12.9K (8.6%)	3.2 (17.2%)	158.8
		BPFLSM	12.2K (8.1%)	11.3K (7.5%)	2.3 (12.4%)	158.5
		BPFLSS	12.3K (8.2%)	8.1K (5.4%)	0.5 (2.7%)	181.7
365309	1	BPFL	17.9K (11.9%)	16.6K (11.1%)	7.6 (40.9%)	141.9
		BPFLSM	17.4K (11.6%)	15.0K (10.0%)	6.9 (37.1%)	139.2
		BPFLSS	17.5K (11.7%)	11.4K (7.6%)	6.1 (32.8%)	138.9
	2	BPFL	17.9K (11.9%)	16.4K (10.9%)	4.8 (25.8%)	152.1
		BPFLSM	17.4K (11.6%)	14.7K (9.8%)	3.6 (19.4%)	146.3
		BPFLSS	17.5K (11.7%)	11.2K (7.5%)	2.7 (14.5%)	162.2
	3	BPFL	17.9K (11.9%)	16.4K (10.9%)	4.5 (24.2%)	152.1
		BPFLSM	17.4K (11.6%)	14.7K (9.8%)	3.3 (17.7%)	146.3
		BPFLSS	17.5K (11.7%)	11.1K (7.4%)	0.8 (4.3%)	170.7

Na osnovu zazetih resursa čipa prikazanih u tabeli 5.3.1 se vidi da je najkritičniji resurs interna memorija čipa za slučaj kada je samo izlazna memorija smeštena u eksternu memoriju. Međutim, čak i u slučaju najveće tabele i dalje je zauzeto manje od 50% memorijskih resursa čipa što govori u prilog skalabilnosti predloženih lukap algoritama. Sa povećanjem broja eksternih memorija na raspolaganju značajno se smanjuju zauzeti memorijski resursi FPGA čipa, pri čemu je smanjenje najizraženije kod BPFLSS algoritma. Potrošnja logičkih elemenata je nešto manja kod BPFLSS-a, a potrošnja registarskih bita je podjednaka kod sva tri algoritma što je posledica činjenice da blok za nalaženje podstabla troši najveći deo registarskih bita, a on je identičan kod sva tri predložena algoritma. Maksimalna frekvencija takta se kreće u granicama od 141.9MHz do 181.7MHz što odgovara protoku lukapa u granicama od 141.9 miliona lukapa po sekundi do 181.7 lukapa u sekundi. Ako se pretpostavi da je najkraća dužina okvira 64B onda se podržane brzine linkova kreću u granicama od 72.7Gb/s do 93Gb/s.

Tabela 5.3.2. – Iskorišćenost resursa FPGA čipa u okviru implementacija predloženih lukap algoritama za IPv6 lukap tabele

Veličina tabele	Broj ekster. mem.	Algoritam	$Reg.$	LE	$Mem.[Mb]$	$f_{max}[MHz]$
104779	1	BPFL	11.4K (7.6%)	13.0K (8.7%)	6.5 (34.9%)	147.0
		BPFLSM	10.5K (7.0%)	9.0K (6.0%)	5.0 (26.9%)	147.3
		BPFLSS	10.7K (7.1%)	8.7K (5.8%)	4.9 (26.3%)	140.4
	2	BPFL	11.4K (7.6%)	13.0K (8.7%)	5.1 (27.4%)	162.4
		BPFLSM	10.5K (7.0%)	8.7K (5.8%)	2.8 (15.1%)	162.6
		BPFLSS	10.7K (7.1%)	8.5K (5.7%)	2.7 (14.5%)	180.3
	3	BPFL	11.4K (7.6%)	13.0K (8.7%)	4.8 (25.8%)	162.4
		BPFLSM	10.5K (7.0%)	8.7K (5.8%)	2.5 (13.4%)	162.6
		BPFLSS	10.7K (7.1%)	8.4K (5.6%)	2.0 (10.8%)	181.7
210039	1	BPFL	22.9K (15.3%)	19.4K (13.0%)	9.4 (50.5%)	140.1
		BPFLSM	21.9K (14.6%)	16.1K (10.7%)	8.5 (45.7%)	130.0
		BPFLSS	22.2K (14.8%)	15.2K (10.1%)	9.1 (48.9%)	126.8
	2	BPFL	22.9K (15.3%)	19.2K (12.8%)	6.6 (35.5%)	152.7
		BPFLSM	21.9K (14.6%)	15.6K (10.4%)	3.9 (21.0%)	149.1
		BPFLSS	22.2K (14.8%)	14.7K (9.8%)	4.5 (24.2%)	162.2
	3	BPFL	22.9K (15.3%)	19.1K (12.8%)	5.9 (31.7%)	155.7
		BPFLSM	21.9K (14.6%)	15.5K (10.3%)	3.4 (18.3%)	154.9
		BPFLSS	22.2K (14.8%)	14.6K (9.7%)	2.7 (14.5%)	173.3
365309	1	BPFL	31.9K (21.3%)	24.7K (16.5%)	15.3 (82.3%)	128.6
		BPFLSM	31.0K (20.7%)	22.1K (14.8%)	12.1 (65.1%)	127.3
		BPFLSS	31.2K (20.8%)	20.4K (13.6%)	12.9 (69.4%)	126.3
	2	BPFL	31.9K (21.3%)	24.3K (16.2%)	9.6 (51.6%)	145.0
		BPFLSM	31.0K (20.7%)	21.5K (14.4%)	6.3 (33.9%)	149.1
		BPFLSS	31.2K (20.8%)	19.8K (13.2%)	7.0 (37.6%)	153.9
	3	BPFL	31.9K (21.3%)	24.2K (16.2%)	8.2 (44.1%)	148.5
		BPFLSM	31.0K (20.7%)	21.4K (14.3%)	5.8 (31.2%)	162.4
		BPFLSS	31.2K (20.8%)	19.5K (13.0%)	4.2 (22.6%)	170.1

U tabeli 5.3.2 su prikazani zauzeti resursi FPGA čipa za slučaj IPv6 tabela. Potrošnja resursa je veća od slučaja IPv4 tabela pre svega zbog dužih prefiksa, ali i zbog većeg broja nepraznih podstabala koji je posledica manje prosečne gustine popunjenosti podstabala. I dalje su najkritičniji memorijski resursi FPGA čipa koji su u najgorem slučaju zauzeti 82.3%. Odnosi između predloženih lukap algoritama su veoma slični kao u IPv4 slučaju. Najveća razlika je što je prednost u zauzetim memorijskim resursima BPFLSS-a u odnosu na BPFL i BPFLSM sada manja za slučaj kada su tri eksterne memorije na raspolaganju. Maksimalna frekvencija takta se kreće u granicama od 126.3MHz do 181.7MHz što odgovara protoku lukapa u granicama od 126.3 miliona lukapa po sekundi do 181.7 lukapa u sekundi, odnosno podržane brzine linkova su u granicama od 64.6Gb/s do 93Gb/s.

Na osnovu prikazanih performansi FPGA implementacije predloženih lukap algoritama može se videti da su oni veoma skalabilni. Mogu se implementirati upotrebom jednog FPGA čipa i jedne eksterne memorije. Pri tome treba imati u vidu da moderni FPGA čipovi mogu da sadrže do 80Mb memorije i do 2 miliona logičkih elemenata i registara [35], tako da se predloženi lukap algoritmi upotrebom ovakvih čipova mogu smestiti na jedan FPGA čip zajedno sa ostalim funkcijama paketskog procesiranja. U analiziranim slučajevima podržana brzina linka se kreće u granicama od 64.6Gb/s do 93Gb/s što znači da su podržani 40Gb/s linkovi, dok se za podršku 100Gb/s linkova može koristiti jači FPGA čip (npr. iz Xilinx-ove serije Virtex7) ili ASIC čip.

6. ZAKLJUČAK

U okviru ove doktorske teze implementirane su funkcije paketskog procesiranja koje čine osnovni deo ravni podataka rutera koja je neophodna za obavljanje osnovne funkcije rutera – usmeravanja paketa kroz Internet mrežu. Implementirani paketski procesor je integrisan u prototip Internet rutera koji je rezultat projekta „Sistemska integracija Internet rutera“ koje je podržalo Ministarstvo nauke Republike Srbije. Razvijeni prototip rutera omogućava istraživanje i razvoj naprednih funkcionalnosti rutera koje nisu moguće u okviru komercijalnih rutera drugih proizvođača. Takođe, prototip se koristi i u edukativne svrhe kako bi studenti mogli steći precizniju sliku o načinu rada rutera.

Pored implementacije funkcija paketskog procesiranja izvršena je i njihova analiza u cilju određivanja najkritičnijih funkcija paketskog procesiranja koje predstavljaju potencijalna uska grla u unapređivanju kapaciteta resursa, odnosno skalabilnosti rutera. Lukap koji na osnovu odredišne IP adrese paketa određuje izlazni port rutera na koji paket treba da se usmeri predstavlja jednu od kritičnih funkcija. Druga kritična funkcija je baferisanje paketa dok čekaju na svoj red za prosleđivanje ka odgovarajućem izlaznom portu.

U tezi su predloženi novi i napredni lukap algoritmi (BPFL, BPFLSM i BPFLSS) koji podržavaju IPv4 i IPv6 tabele i omogućavaju veliki protok paketa čime su podržani linkovi velikih brzina. Predloženi lukap algoritmi imaju minimalne memorijske zahteve čime je omogućena njihova skalabilnost tj. podrška velikih lukap tabela. Predloženi lukap algoritmi su poređeni sa postojećim rešenjima i utvrđeno je da predloženi lukap algoritmi imaju najbolje performanse po pitanju zahtevanih memorijskih resursa, a postignuta prednost je izrazita u slučaju velikih IPv6 tabela. Takođe, predloženi lukap algoritmi su implementirani na FPGA čipu pri čemu su analizirane performanse implementacije.

LITERATURA

[1] H.J. Chao, B. Liu, *High performance switches and routers*, Wiley, 2007.

[2] I. Elhanany, M. Hamdi, *High-performance packet switching architectures*, Springer, 2007.

[3] N. McKeown, „Fast switched backplane for a gigabit switched router," *Business Communications Review*, December 1997.

[4] J. Aweya, „IP router architectures: An overview," *Journal of Systems Architecture*, vol. 46(6), pp. 483-511, April 2000.

[5] A. Smiljanić, Z. Čiča, R. Đenić „Prototip Internet rutera," *POSTEL 2010*, Beograd, Decembar 2010.

[6] M. Antić, Z. Čiča, N. Maksić, A. Smiljanić, „Testiranje prototipa skalabilnog Internet rutera," *ETRAN 2011*, Banja Vrućica – Teslić, Jun 2011.

[7] A. Smiljanić, *Predavanja za predmet Arhitektura svičeva i rutera*

[8] A. Bianco, R. Birke, J. Finochietto, L. Giraudo, F. Marenco, M. Mellia, A. Khan, D. Manjuath, „Control and management plane in a multi-stage software router architecture," *Proc. of IEEE Workshop on High Performance Switching and Routing 2008*, Beijing, China, May 2008.

[9] I. Houidi, W. Louati, D. Zeghlache, „An extensible software router data-path for dynamic low-level service deployment," *Proc. of IEEE Workshop on High Performance Switching and Routing 2006*, Poznan, Poland, June 2006.

[10] R. Giladi, N. Yemini, „A programmable, generic forwarding element approach for dynamic network functionality," *Proc. of PRESTO 2009*, Barcelona, Spain, August 2009.

[11] A. Bianco, J. Finochietto, G. Gavilanes, F. Neri, „A control and management plane for large packet switches," *Proc. of IT-NEWS 2008*, Venice, Italy, February 2008.

[12] S. Floyd, V. Jacobson, „Random early detection gateways for congestion avoidance,“ *IEEE/ACM Transactions on Networking*, vol. 1(4), pp. 397-413, August 1993.

[13] A. Parekh, R. Gallager, „A generalized processor sharing approach to flow control in integrated services networks: The single-node case,“ *IEEE/ACM Transactions on Networking*, vol. 1(3), pp. 344-357, June 1993.

[14] M. Shreedhar, G. Varghese, „Efficient fair queueing using deficit round robin,“ *IEEE/ACM Transactions on Networking*, vol. 4(3), pp. 375-385, June 1996.

[15] J. Bennett, H. Zhang, „Hierarchical packet fair queueing algorithms,“ *IEEE/ACM Transactions on Networking*, vol. 5(5), pp. 675-689, October 1997.

[16] J.P. Yang, Y.S. Chu, M.C. Liang, „Threshold-based selective drop for shared buffer packet switches,“ *IEEE Communication Letters*, vol. 7(4), pp. 183-85, April 2003.

[17] B. Matthews, I. Elhanany, „A scalable memory-efficient architecture for parallel shared memory switches,“ *Proc. of IEEE Workshop on High Performance Switching and Routing 2007*, New York, USA, May 2007.

[18] Z. Dong, R. Rojas-Cessa, „Output-based shared-memory crosspoint-buffered packet switch for multicast services,“ *Proc. of IEEE ICC 2008*, Beijing, China, May 2008.

[19] T. Anderson, S. Owicki, J. Saxe, C. Thacker, „High speed switch scheduling for local area networks,“ *ACM Trans. Comput. Syst.*, vol. 11(4), pp. 319–352, November 1993.

[20] N. McKeown, „The iSLIP scheduling algorithm for input-queued switches,“ *IEEE/ACM Transanctions on Networking*, vol. 7(2), pp. 188–201, April 1999.

[21] V. Nitnaware, S. Limaye, „Time efficient arbiter in the design of scheduler embodying iSLIP algorithm for on-chip interconnection,“ *International Journal of Advanced Science and Technology*, vol. 21, pp. 69–82, August 2010.

[22] M. Petrović, A. Smiljanić, M. Blagojević, „Design of the switching controller for the high-capacity non-blocking Internet router,“ *IEEE Transanctions on VLSI Systems*, vol. 17(8), pp. 1157–1161, August 2009.

[23] M. Petrović, A. Smiljanić, „Optimization of the scheduler for the non-blocking high-capacity router,“ *IEEE Communications Letters*, vol. 11(6), pp. 534–536, June 2007.

[24] M. Petrović, A. Smiljanić, „Design of the scheduler for the high-capacity non-blocking packet switch,“ *Proc of IEEE Workshop on High Performance Switching and Routing 2006*, Poznan, Poland, June 2006.

[25] C.S. Chang, D.S. Lee, Y.S. Jou, „Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering, “ *Computer Communications*, vol. 25(6), pp. 611-622, April 2002.

[26] C.S. Chang, D.S. Lee, C.M. Lien, „Load balanced Birkhoff-von Neumann switches, part II: multi-stage buffering, “ *Computer Communications*, vol. 25(6), pp. 623-634, April 2002.

[27] W.J. Dally, B.P. Towles, *Principles and practices of interconnection networks*, Morgan Kaufmann, 2003.

[28] A. Smiljanić, „Rate and delay guarantees provided by Clos packet switches with load balancing,“ *IEEE/ACM Transactions on Networking*, vol. 16(1), pp. 170-181, February 2008.

[29] A. Smiljanić, M. Petrović, „Speedup of Clos packet switches that provide delay guarantees,“ *Proc of IEEE Workshop on High Performance Switching and Routing 2005*, Hong Kong, China, May 2005.

[30] A. Smiljanić, „Performance of load balancing algorithms in Clos packet switches,“ *Proc of IEEE Workshop on High Performance Switching and Routing 2004*, Phoenix, USA, April 2004.

[31] M. Blagojević, A. Smiljanić, „Design of multicast controller for high-capacity Internet router,“ *IET Electronics Letters*, vol. 44(3), pp. 255-256, January 2008.

[32] M. Blagojević, A. Smiljanić, M. Petrović „Design of the multicast controller for the high-capacity Internet router,“ *Proc. of IEEE Workshop on High Performance Switching and Routing 2007*, New York, USA, May 2007.

[33] J.W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, J. Luo, „NetFPGA - An open platform for gigabit-rate network switching and routing,“ *Proc. of IEEE MSE 2007*, San Diego, USA, June 2007.

[34] G. Gibb, J.W. Lockwood, J. Naous, P. Hartke, N. McKeown, „NetFPGA - An open platform for teaching how to build gigabit-rate network switches and routers,“ *IEEE Transactions on Education*, vol. 51(3), pp. 364-369, August 2008.

[35] *http://www.xilinx.com*

[36] *http://www.cypress.com*

[37] *http://www.micron.com*

[38] *http://www.marvell.com*

[39] *LVDS application and data handbook*, Texas Instruments, 2002.

[40] M. Carević, Z. Čiča, „FPGA implementacija segmentacije i rekonstrukcije IP paketa u Internet ruteru,“ *ETRAN 2009*, Vrnjačka Banja, Jun 2009.

[41] M. Carević, Z. Čiča, „FPGA Implementation of IP Packet Segmentation and Reassembly in Internet Router,“ *Serbian Journal of Electrical Engineering*, vol. 6(3), pp. 399-407, December 2009.

[42] D.E. Comer, *Networks System Design Using Network Processors Intel IXP Version*, Prentice Hall, 2004.

[43] A. Tokalić, Z. Čiča, A. Smiljanić, „Performance analysis of the IP lookup table updating,“ *Proc. of TELSIKS 2011*, Niš, October 2011.

[44] *http://bgp.potaroo.net*

[45] S. Iyer, R. R. Compella, N. McKeown, „Designing buffers for router line cards,“ *Stanford University HPNG Technical Report - TR02-HPNG-031001*, 2002.

[46] S. Iyer, R. R. Compella, N. McKeown, „Designing buffers for router line cards,“ *IEEE/ACM Transactions on Networking*, vol. 16(3), pp. 705-717, June 2008.

[47] S. Iyer, R. R. Compella, N. McKeown, „Analysis of a memory architecture for fast packet buffers,“ *Proc. of IEEE Workshop on High Performance Switching and Routing 2001*, Dallas, USA, May 2001.

[48] J. Garcia, J. Corbal, L. Cerda, M. Valero, „Design and implementation of high-performance memory systems for future packet buffers,“ *Proc. of the 36th International Symposium on Microarchitecture (MICRO-36 2003)*, San Diego, USA, December 2003.

[49] M. Kabra, S. Saha, B. Lin, „Fast Buffer Memory with Deterministic Packet Departures,“ *Proc. of HOTI 2006*, Stanford, USA, August 2006.

[50] A.Wilen, J. Schade, R. Thornburg, *Introduction to PCI Express*, Intel Press, 2003.

[51] *http://www.rapidio.org/specs/current*

[52] *http://www.sata-io.org*

[53] *http://www.cisco.com*

[54] *http://www.ietf.org/rfc/rfc2328.txt*

[55] *http://www6.ietf.org/rfc/rfc4271*

[56] *http://tools.ietf.org/rfc/rfc2453.txt*

[57] *http://tools.ietf.org/rfc/rfc3031.txt*

[58] *http://tools.ietf.org/rfc/rfc3945.txt*

[59] Z. Čiča, A. Smiljanić, „Balanced parallelised frugal IPv6 lookup algorithm,“ *IET Electronics Letters*, vol. 47(17), pp. 963-965, August 2011.

[60] Z. Čiča, L.Milinković, A. Smiljanić, „FPGA implementation of lookup algorithms,“ *Proc. of IEEE Workshop on High Performance Switching and Routing 2011*, Cartagena, Spain, July 2011.

[61] Z. Čiča, A. Smiljanić, „Frugal IP lookup based on a parallel search,“ *Proc. of IEEE Workshop on High Performance Switching and Routing 2009*, Paris, France, June 2009.

[62] W. Jiang, Q. Wang, V. K. Prasanna, „Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup,“ *Proceedings of IEEE INFOCOM 2008*, Phoenix, USA, April 2008.

[63] D. Taylor, J. Lockwood, T. Sproull, J. Turner, D. Parlour, „Scalable IP lookup for programmable routers,“ *Proceedings of IEEE INFOCOM 2002*, New York, USA, June 2002.

[64] D. Pao, C. Liu, A. Wu, L. Yeung, K.S. Chan, „Efficient hardware architecture for fast IP address lookup,“ *IEE Proceedings on Computers and Digital Techniques*, vol. 150(1), pp. 43-52, January 2003.

[65] R. Rojas-Cessa, L. Ramesh, Z. Dong, L.Cai, N. Ansari, „Parallel-search trie-based scheme for fast IP lookup," *Proceedings of GLOBECOM 2007*, Washington D.C., USA, November 2007.

[66] W. Eatherton, Z. Dittia, G. Varghese, „Tree bitmap: hardware/software IP lookups with incremental updates," *ACM SIGCOMM Computer Communication Review*, vol. 34(2), pp. 97-122, April 2004.

[67] N. Yazdani, P. Min, „Fast and scalable schemes for the IP address lookup problem," *Proceedings of IEEE Conference on High Performance Switching and Routing 2000*, Heidelberg, Germany, June 2000.

[68] S. Kumar, M. Becchi, P. Crowley, J. Turner, „CAMP: Fast and efficient IP lookup architecture," *Proceedings of ANCS '06*, San Jose, USA, December 2006.

[69] H. Song, J. Turner, J. Lockwood, „Shape shifting tries for faster IP route lookup," *Proceedings of ICNP 2005*, Boston, USA, September 2005.

[70] S. Sahni, K.S. Kim, „Efficient construction of multibit tries for IP lookup," *IEEE/ACM Transactions on Networking*, vol. 11(4), pp. 650-662, August 2003.

[71] R. Sangireddy, N. Futamura, S. Aluru, A. Somani, „Scalable, memory efficient, high-speed IP lookup algorithms," *IEEE/ACM Transactions on Networking*, vol. 13(4), pp. 802-812, August 2005.

[72] M. Degermark, A. Brodnik, S. Carlsson, S. Pink, „Small forwarding tables for fast routing lookups," *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[73] M. Ruiz-Sanchez, E. Biersack, W. Dabbous, „Survey and taxonomy of IP address lookup algorithms," *IEEE Network*, vol. 15(2), pp. 8-23, March/April 2001.

[74] V. Srinivasan, G. Varghese, „Fast address lookups using controlled prefix expansion," *Proceedings of ACM Sigmetrics 98*, Madison, USA, June 1998.

[75] S. Nilsson, G. Karlsson, „IP-address lookup using LC-tries," *IEEE Journal on Selected Areas in Communications*, vol. 17(6), pp. 1083–1092, June 1999.

[76] H. Lim, C. Yim, E. Swartzlander, „Priority tries for IP address lookup," *IEEE Transactions on Computers*, vol. 59(6), pp. 784-794, June 2010.

[77] S.Y. Hsieh, Y.C. Yang, „A classified multi-suffix trie for IP lookup and update," *IEEE Transactions on Computers*, vol. 61(5), pp. 726-731, April 2011.

[78] M. Berger, „IP lookup with low memory requirement and fast update," *Proceedings of HPSR 2003*, Torino, Italy, June 2003.

[79] W. Lu, S. Sahni, „Low-power TCAMs for very large forwarding tables," *IEEE/ACM Transactions on Networking*, vol. 18(3), pp. 948-959, June 2010.

[80] B. Agrawal, T. Sherwood, „Modeling TCAM power for next generation network devices," *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software - ISPASS 2006*, Austin, USA, March 2006.

[81] D. Lin et al., „Route table partitioning and load balancing for parallel searching with TCAMs," *Proceedings of 21st IEEE International Parallel and Distributed Processing Symposium 2007*, Long Beach, USA, March 2007.

[82] V.C. Ravikumar, R.N. Mahapatra, L.N. Bhuyan, „EaseCAM: An energy and storage efficient TCAM-based router architecture for IP lookup," *IEEE Transactions on Computers*, vol. 54(5), pp. 521-533, May 2005.

[83] X. Zhang, B. Liu, W. Li, Y. Xi, D. Berminghem, X. Wang, „IPv6-oriented 4*OC768 packet classification with deriving-merging partition and field-variable encoding algorithm," *Proceedings of INFOCOM 2006*, Barcelona, Spain, April 2006.

[84] F. Zane, G. Narlikar, A. Basu, „CoolCAMs: Power-efficient TCAMs for forwarding engines," *Proceeding of INFOCOM 2003*, San Francisco, USA, March/April 2003.

[85] K. Zheng, C. Hu, H. Liu, B. Liu, „An ultra-high throughput and power efficient TCAM-based IP lookup engine," *Proceedings of INFOCOM 2004*, Hong Kong, China, March 2004.

[86] H. Song, S. Dharmapurikar, J. Turner, J. Lockwood, „Fast hash table lookup using extended Bloom filter: An aid to network processing," *Proceedings of SIGCOMM '05*, Philadelphia, USA, August 2005.

[87] A. Broder, M. Mitzenmacher, „Using multiple hash functions to improve IP lookups," *Proceedings of IEEE INFOCOM 2001*, Anchorage, USA, April 2001.

[88] M. Bando, N.S. Artan, J. Chao, „FlashLook: 100-Gbps hash-tuned route lookup architecture,“ *Proceedings of International Conference on High Performance Switching and Routing 2009*, Paris, France, June 2009.

[89] K. Huang, G. Xie, Y. Li, A.X. Liu, „Offset addressing approach to memory-efficient IP address lookup,“ *Proceedings of INFOCOM 2011*, Shanghai, China, April 2011.

[90] G. Antichi, A. Di Pietro, D. Ficara, S. Giordano, G. Procissi, F. Vitucci, „A heuristic and hybrid hash-based approach to fast lookup,“ *Proc. of IEEE Workshop on High Performance Switching and Routing 2009*, Paris, France, June 2009.

[91] B.H. Bloom, „Space/time trade-offs in hash coding with allowable errors,“ *Communication of the ACM*, vol. 13(7), pp. 422-426, July 1970.

[92] S. Agarwal, A. Trachtenberg, „Approximating the number of differences between remote sets,“ *Proceedings of ITW 2006*, Punta del Este, Uruguay, April 2006.

[93] *http://data.ris.ripe.net*

[94] *http://tools.ietf.org/id/draft-ietf-grow-mrt-17.txt*

[95] M. Wang, S. Deering, T. Hain, L. Dunn, „Non-random generator for IPv6 tables,“ *Proceedings of IEEE Symposium on High-Performance Interconnects 2004*, Stanford, USA, August 2004.

SPISAK SKRAĆENICA

ARP	*Address Resolution Protocol*
ASIC	*Application-Specific Integrated Circuit*
BGP	*Border Gateway Protocol*
BPFL	*Balanced Parallelized Frugal Lookup*
BPFLSM	*Balanced Parallelized Frugal Lookup with Shared Memory*
BPFLSS	*Balanced Parallelized Frugal Lookup with Subtree Splitting*
DDR2 SDRAM	*Double Data Rate Synchronous Dynamic Random Access Memory*
DRAM	*Dynamic Random Access Memory*
DSRAM	*Data SRAM*
DTCAM	*Data TCAM*
EHAF	*Efficient Hardware Architecture for Fast IP Lookup*
FCS	*Frame Check Sequence*
FIFO	*First In First Out*
FIPL	*Fast Internet Protocol Lookup*
FPGA	*Field Programmable Gate Array*
GMPLS	*Generalized Multi-Protocol Label Switching*
IFG	*InterFrame Gap*
IP	*Internet Protocol*
ISRAM	*Index SRAM*
ITCAM	*Index TCAM*
LPM	*Longest Prefix Matching*
M-12Wb	*Many-1 2-Level TCAM with Wide Memory Lookup*
MAC	*Media Access Control*
MPLS	*Multi-Protocol Label Switching*
MRT	*Multi-threaded Routing Toolkit*
OSI	*Open Systems Interconnect*
OSPF	*Open Shortest Path First*
PFHT	*Prunned Fast Hash Table Lookup*
PIM	*Parallel Iterative Matching*
PLL	*Phase-Locked Loop*
POLP	*Parallelized Optimal Linear Pipeline*
POS	*Packet Over SDH*
RIP	*Routing Information Protocol*
SDH	*Synchronous Digital Hierarchy*
SERDES	*SERializer/DESerializer*
SFD	*Start Frame Delimiter*
SFP	*Small Form-factor Pluggable Transceiver*
SGS	*Sequential Greedy Scheduler*
SNMP	*Simple Network Management Protocol*
SRAM	*Static Random Access Memory*
TB	*Tree Bitmap*
TCAM	*Ternary Content Addressable Memory*
TDM	*Time Division Multiplex*
ToS	*Type of Service*
TTL	*Time To Live*

A. MEMORIJSKI ZAHTEVI LUKAP ALGORITAMA ZA ANALIZIRANE IPv4 I IPv6 LUKAP TABELE

U ovom prilogu će prvo biti dati rezultati za postojeće lukap algoritme, a potom i za lukap algoritme predložene u okviru ove teze. Rezultati su dati u skladu sa matematičkom analizom lukap algoritama datih u okviru ove teze, a pored memorijskih zahteva su date i vrednosti karakterističnih parametara svakog od lukap algoritama.

A.1. Postojeći lukap algoritmi

U okviru ovog dela će biti predstavljeni memorijski zahtevi postojećih lukap algoritama koristeći njihovu analizu iz četvrtog poglavlja. U okviru ove analize za konkretne lukap tabele, su napisane odgovarajuće aplikacije koje na osnovu strukture lukap tabele vrše proračun parametara analiziranih lukap algoritama. Za svaki algoritam su date i vrednosti karakterističnih parametara algoritma za sve analizirane lukap tabele.

A.1.1. FIPL algoritam

FIPL algoritam je zasnovan na TB tehnici pri čemu koristi strajdove dužine $m = 4$ bita. Na osnovu (4.3.7.1.1)-(4.3.7.1.2) su izvršeni proračuni memorijskih zahteva za analizirane lukap tabele. U tabeli A.1.1.1 su prikazani zahtevani memorijski resursi za smeštanje čvorova m-arnog stabla M_{ms} i izlaznu memoriju M_{iz} za analizirane IPv4 tabele. U tabeli A.1.1.1 su takođe date vrednosti parametara N_n (broj postojećih nizova od 2^m čvorova za IPv4 tabele) i N_c (broj čvorova čiji interni bitmap ukazuje na postojanje bar jednog čvora sa validnim ID-em izlaznog porta). U tabeli A.1.1.2 je prikazana distribucija postojećih nizova od 2^m članova po nivoima, tj. vrednost parametra N_n^j za IPv4 tabele. Analogno, u tabeli A.1.1.3 su dati zahtevani memorijski resursi M_{ms} i M_{iz}, kao i vrednosti parametara N_n i N_c za IPv6 tabele, a u tabeli A.1.1.4 je prikazana distribucija postojećih nizova od 2^m članova po nivoima, tj. vrednost parametra N_n^j za IPv6 tabele. Može se uočiti da su memorijski zahtevi u IPv6 znatno veći od memorijskih zahteva u IPv4 slučaju, što je posledica činjenice da se prefiksi

nalaze na većim dubinama u slučaju IPv6 tabela, pa je samim tim i broj postojećih nizova od 2^m članova znatno veći u IPv6 slučaju.

Tabela A.1.1.1. – Zahtevani memorijski resursi FIPL algoritma za IPv4 lukap tabele

Veličina tabele	N_n	N_c	M_{ms}[MB]	M_{nh}[MB]
104779	9887	40358	1.2069	1.1547
130287	12175	48110	1.4862	1.3764
153126	13335	54530	1.6278	1.5601
175399	14877	60257	1.8160	1.7240
210039	18268	70319	2.2300	2.0118
240958	18800	77886	2.2949	2.2283
273791	20770	86633	2.5354	2.4786
307915	22233	94648	2.7140	2.7079
348866	25833	106785	3.1534	3.0551
365309	26362	110625	3.2180	3.1650

Tabela A.1.1.2. – Distribucija postojećih nizova od 2^m članova po nivoima stabla za IPv4 tabele

Veličina tabele / Parametri	105K	130K	153K	175K	210K	241K	274K	308K	349K	365K
N_n^1	7964	9814	10881	12030	13638	14875	16259	17503	19605	20182
N_n^2	1599	1911	2111	2350	2648	2952	3155	3354	3665	3757
N_n^3	189	207	228	249	1337	380	517	523	1783	1766
N_n^4	75	161	58	183	345	301	492	490	385	374
N_n^5	37	59	32	39	273	264	318	333	365	253
N_n^6	21	21	23	24	25	26	27	28	28	28
N_n^7	2	2	2	2	2	2	2	2	2	2

Tabela A.1.1.3. – Zahtevani memorijski resursi FIPL algoritma za IPv6 lukap tabele

Veličina tabele	N_n	N_c	M_{ms}[MB]	M_{nh}[MB]
104779	201484	65406	24.5952	1.8713
130287	264351	93564	32.2694	2.6769
153126	293708	107089	35.8530	3.0638
175399	320340	113968	39.1040	3.2607
210039	403348	146185	49.2368	4.1824
240958	417505	150247	50.9650	4.2986
273791	486178	183731	59.3479	5.2566
307915	549018	210474	67.0188	6.0217
348866	574172	218634	70.0894	6.2552
365309	562973	207820	68.7223	5.9458

Tabela A.1.1.4. – Distribucija postojećih nizova od 2^m članova po nivoima stabla za IPv6 tabele

Veličina tabele / Parametri	105K	130K	153K	175K	210K	241K	274K	308K	349K	365K
N_n^1	47938	73504	87156	86673	113989	114747	149724	167428	174039	146634
N_n^2	28421	34601	36855	44513	55347	54943	60588	73008	75090	86759
N_n^3	22951	32313	35053	37942	48808	50022	59373	67053	70737	65557
N_n^4	21748	26700	29083	33265	40986	43851	46811	53301	56930	60949
N_n^5	21653	26375	28576	32755	39619	42980	46253	52302	55484	57415
N_n^6	19110	23409	26035	29010	35201	38258	41772	46132	50079	51657

N_n^7	17771	21662	24063	26725	32196	34819	37849	41669	44910	46329
N_n^8	14680	17524	19045	21047	24756	26397	28245	30689	32717	33471
N_n^9	6340	6835	7030	7284	7643	7682	7795	7889	7969	7992
N_n^{10}	512	512	512	512	2436	1373	2830	3550	3575	3811
N_n^{11}	142	400	116	320	1263	1014	2315	2841	1535	1395
N_n^{12}	108	289	89	179	558	873	2077	2610	561	512
N_n^{13}	76	193	61	81	512	512	512	512	512	458
N_n^{14}	32	32	32	32	32	32	32	32	32	32
N_n^{15}	2	2	2	2	2	2	2	2	2	2

A.1.2. EHAF algoritam

Zahtevani memorijski resursi po nivoima M_n^i, kao i memorijski zahtevi izlazne memorije M_{iz} za EHAF algoritam su računati na osnovu (4.3.7.2.1)-(4.3.7.2.5) i prikazani su u tabeli A.1.2.1 za IPv4 analizirane tabele. Takođe, u tabeli A.1.2.1 su date vrednosti karakterističnih parametara EHAF algoritma: N_{pod}^3(broj nepraznih podstabala na nivoua 3), N_{pod}^3 (broj nepraznih podstabala na nivoua 4) i N_{gpod}^4 (broj grupa podstabala četvrtog nivoa koja sadrže bar jedno neprazno podstablo). Ovaj algoritam nije predviđen za rad sa IPv6 adresama jer bi u tom slučaju bili potrebni veoma dugi bitmap vektori za podstabla na dubljim nivoima. Dužina tih vektora bi zahtevala prevelik broj bita (za najdublja podstabla i preko milijardu bita) tako da praktična implementacija EHAF algoritma za IPv6 nije moguća.

Tabela A.1.2.1. – Zahtevani memorijski resursi EHAF algoritma za IPv4 lukap tabele

Veličina tabele	N_{pod}^3	N_{pod}^4	N_{gpod}^4	M_n^1[MB]	M_n^2[MB]	M_n^3[MB]	M_n^4[MB]	M_{iz}[MB]
104779	11248	99	56	0.0001	0.0156	0.5607	0.1087	5.7689
130287	13833	217	131	0.0001	0.0156	0.6393	0.1636	7.0835
153126	15560	86	49	0.0001	0.0156	0.6917	0.1068	7.8598
175399	17524	262	148	0.0001	0.0156	0.7671	0.1736	8.9006
210039	20228	2029	763	0.0001	0.0156	0.8493	0.4741	11.0752
240958	22655	692	245	0.0001	0.0156	0.9230	0.2208	11.6054
273791	24474	1091	333	0.0001	0.0156	0.9783	0.2856	12.6841
307915	26577	1235	310	0.0001	0.0156	1.0423	0.2822	13.7770
348866	29345	2976	1031	0.0001	0.0156	1.1264	0.6399	15.9701
365309	30256	3083	1060	0.0001	0.0156	1.1541	0.6538	16.4652

A.1.3. POLP algoritam

Vršena je analiza POLP algoritma za različite brojeve pajplajnova. Broj faza u pajplajnu zavisi od dubine D originalnog stabla koja je uzeta za početak podstabala koja se razmeštaju po pajplajnovima. Što je veća dubina, minimalno neophodan broj faza u okviru pajplajna je manji pa je ukupan broj memorija po jednom pajplajnu manji, što je od ogromnog značaja za slučajeve kad interni memorijski resursi integrisanog čipa

(FPGA, ASIC) nisu dovoljni, pošto bi onda i broj eksternih memorija bio manji. U ovoj analizi je usvojeno $D = 16$, jer bi veće vrednosti bile problematične sa stanovišta ažuriranja, pošto se u korene postojećih podstabala moraju gurati kraći prefiksi. Minimalan broj faza je stoga $F = 17$ za IPv4 lukap tabele, odnosno $F = 49$ za IPv6 lukap tabele, pošto njihov broj mora da bude jednak najdužoj mogućoj putanji kroz podstablo (tada svaka faza sadrži po jedan čvor putanje). U tabeli A.1.3.1 su prikazani ukupni memorijski zahtevi usmerivača M_u, pajplajnova M_p i izlaznih memorija M_{iz} za POLP algoritam za analizirane IPv4 lukap tabele koji su proračunati na osnovu (4.3.7.3.1)-(4.3.7.3.3). Takođe, u tabeli A.1.3.1 su date i vrednosti parametara: P (broj pajplajnova), F (broj faza) i N_c (broj lokacija memorije u okviru jedne faze).

Iz tabele A.1.3.1 se može videti da promena broja pajplajnova ne utiče u značajnoj meri na memorijske zahteve, ali utiče na broj lokacija u okviru jedne faze (N_c). Što je veći broj pajplajnova manje su memorije po fazama, što je posledica manjeg ukupnog broja čvorova po pajplajnovima. U tabeli A.1.3.2 su prikazani memorijski zahtevi, kao i vrednosti parametara P, F i N_c za slučaj IPv6 lukap tabela. Može se uočiti da i ovde broj pajplajnova ne utiče značajno na memorijske zahteve. Memorijski zahtevi su značajno veći nego u IPv4 slučaju što je posledica značajno većeg broja čvorova podstabala. Broj čvorova podstabala je veći jer se prefiksi sada nalaze na većim dubinama nego u IPv4 slučaju, pa se sada čuva značajno veći broj praznih čvorova.

Tabela A.1.3.1. – Zahtevani memorijski resursi POLP algoritma za IPv4 lukap tabele

Veličina tabele	P	F	N_c	M_u[MB]	M_p[MB]	M_{nh}[MB]
104779	2	17	8910	0.0156	0.7222	0.5778
	4	17	4458	0.0625	0.6866	0.5782
	6	17	2975	0.1406	0.6511	0.5788
	8	17	2231	0.1875	0.6511	0.5787
130287	2	17	10809	0.0156	0.8762	0.7010
	4	17	5405	0.0625	0.8325	0.7010
	6	17	3604	0.1406	0.7888	0.7012
	8	17	2705	0.1875	0.7894	0.7017
153126	2	17	12442	0.0156	1.0086	0.8069
	4	17	6221	0.0625	0.9581	0.8069
	6	17	4152	0.1406	0.9592	0.8078
	8	17	3114	0.1875	0.9087	0.8078
175399	2	17	14025	0.0156	1.1369	0.9095
	4	17	7015	0.0625	1.0804	0.9098
	6	17	4680	0.1406	1.0812	0.9105
	8	17	3509	0.1875	1.0240	0.9102
210039	2	17	16651	0.0156	1.4173	1.0798
	4	17	8326	0.0625	1.3498	1.0799
	6	17	5551	0.1406	1.2824	1.0799

	8	17	4165	0.1875	1.2830	1.0804
240958	2	17	18614	0.0156	1.5843	1.2071
	4	17	9308	0.0625	1.5091	1.2072
	6	17	6206	0.1406	1.4338	1.2074
	8	17	4657	0.1875	1.4345	1.2080
273791	2	17	20985	0.0156	1.7861	1.3609
	4	17	10497	0.0625	1.7018	1.3615
	6	17	6998	0.1406	1.6167	1.3615
	8	17	5249	0.1875	1.6169	1.3616
307915	2	17	23191	0.0156	1.9739	1.5039
	4	17	11597	0.0625	1.8802	1.5041
	6	17	7735	0.1406	1.7870	1.5048
	8	17	5800	0.1875	1.7866	1.5045
348866	2	17	26265	0.0156	2.2356	1.7033
	4	17	13138	0.0625	2.1300	1.7040
	6	17	8758	0.1406	2.1298	1.7039
	8	17	6571	0.1875	2.0241	1.7045
365309	2	17	27322	0.0156	2.3255	1.7718
	4	17	13665	0.0625	2.2154	1.7723
	6	17	9112	0.1406	2.2159	1.7727
	8	17	6834	0.1875	2.1051	1.7727

Tabela A.1.3.2. – Zahtevani memorijski resursi POLP algoritma za IPv6 lukap tabele

Veličina tabele	P	F	N_c	M_u[MB]	M_p[MB]	M_{nh}[MB]
104779	2	49	8883	0.0156	2.1793	1.6604
	4	49	4444	0.0625	2.0767	1.6613
	6	49	2963	0.1406	1.9731	1.6615
	8	49	2223	0.1875	1.9737	1.6621
130287	2	49	11843	0.0156	2.9055	2.2140
	4	49	5928	0.0625	2.7701	2.2161
	6	49	3951	0.1406	2.6310	2.2156
	8	49	2972	0.1875	2.6388	2.2221
153126	2	49	13304	0.0156	3.2639	2.4868
	4	49	6651	0.0625	3.1080	2.4864
	6	49	4442	0.1406	3.1136	2.4909
	8	49	3332	0.1875	2.9584	2.4913
175399	2	49	14516	0.0156	3.5612	2.7133
	4	49	7262	0.0625	3.3935	2.7148
	6	49	4843	0.1406	3.3947	2.7158
	8	49	3634	0.1875	3.2265	2.7171
210039	2	49	18470	0.0156	4.7471	3.4524
	4	49	9250	0.0625	4.5387	3.4580
	6	49	6164	0.1406	4.3207	3.4565
	8	49	4631	0.1875	4.3281	3.4625
240958	2	49	19096	0.0156	4.9080	3.5694
	4	49	9560	0.0625	4.6908	3.5739
	6	49	6373	0.1406	4.4672	3.5737
	8	49	4780	0.1875	4.4674	3.5739
273791	2	49	22753	0.0156	5.8479	4.2530
	4	49	11398	0.0625	5.5926	4.2610
	6	49	7601	0.1406	5.3279	4.2623
	8	49	5705	0.1875	5.3319	4.2655
307915	2	49	25935	0.0156	6.6557	4.8478
	4	49	12979	0.0625	6.3683	4.8521
	6	49	8659	0.1406	6.3730	4.8556

	8	49	6501	0.1875	6.0758	4.8607
348866	2	49	27082	0.0156	6.9605	5.0622
	4	49	13543	0.0625	6.6451	5.0629
	6	49	9045	0.1406	6.6571	5.0721
	8	49	6772	0.1875	6.3291	5.0633
365309	2	49	26360	0.0156	6.7749	4.9272
	4	49	13192	0.0625	6.4729	4.9317
	6	49	8798	0.1406	6.4753	4.9336
	8	49	6599	0.1875	6.1674	4.9339

A.1.4. Stablo sa prioritetima

Zahtevani memorijski resursi za stablo sa prioritetima M_{ps} su računati na osnovu (4.3.7.4.1) i prikazani su u tabelama A.1.4.1 i A.1.4.3 za IPv4 i IPv6 analizirane tabele, respektivno. U tabelama A.1.4.1 i A.1.4.3 su date i vrednosti maksimalne dubine stabla sa prioritetima D_{max}, kao i ukupnog broja čvorova stabla sa prioritetima N_c. U tabelama A.1.4.2 i A.1.4.4 je data distribucija čvorova po nivoima stabla sa prioritetima, tj. vrednost parametra N_c^j za IPv4 i IPv6 analizirane tabele, respektivno.

Tabela A.1.4.1. – Zahtevani memorijski resursi stabla sa prioritetima za IPv4 lukap tabele

Veličina tabele	D_{max}	N_c	M_{ps}[MB]
104779	26	104779	1.0367
130287	30	130287	1.2891
153126	29	153126	1.5516
175399	27	175399	1.7773
210039	30	210039	2.1283
240958	30	240958	2.4416
273791	32	273791	2.8395
307915	32	307915	3.1934
348866	28	348866	3.6182
365309	29	365309	3.7887

Tabela A.1.4.2. – Distribucija čvorova stabla sa prioritetima po nivoima za IPv4 tabele

Veličina tabele / Parametri	105K	130K	153K	175K	210K	241K	274K	308K	349K	365K
N_c^1	14342	18297	21350	24311	29495	34249	39687	44961	52302	55278
N_c^2	13959	17437	20833	24011	28976	34139	39101	43769	50977	53650
N_c^3	13460	16482	19970	23475	28545	32218	36035	40297	46820	49443
N_c^4	12817	16035	18108	20394	25321	29759	34942	39801	45480	47436
N_c^5	11153	13858	17236	20028	23277	25963	28567	31221	35104	36306
N_c^6	9081	11299	12651	14065	17649	20795	24473	29286	33536	35560
N_c^7	6940	9145	11696	14064	16377	18318	19950	21818	24261	25022
N_c^8	6922	8537	9583	10751	12402	13771	14937	16228	17795	18345
N_c^9	5561	6577	7289	8047	9140	10070	10830	11561	12898	13805
N_c^{10}	3699	4219	4634	5045	6429	7676	9232	11281	12587	12899
N_c^{11}	2255	3163	4126	5035	5611	6134	6578	7052	7608	7768
N_c^{12}	2087	2355	2585	2831	3123	3433	3680	3942	4253	4359
N_c^{13}	1130	1275	1399	1538	1694	1850	2009	2159	2346	2410

N_c^{14}	609	687	747	817	898	991	1076	1155	1252	1293
N_c^{15}	326	370	404	440	487	540	580	618	674	693
N_c^{16}	183	199	220	238	266	288	330	458	363	373
N_c^{17}	105	112	124	135	147	161	310	444	195	201
N_c^{18}	62	64	70	76	82	101	310	401	128	150
N_c^{19}	36	37	41	43	45	93	294	350	108	110
N_c^{20}	20	22	22	24	25	90	261	327	56	56
N_c^{21}	12	21	12	13	14	78	170	274	55	55
N_c^{22}	6	20	6	7	13	72	161	179	28	28
N_c^{23}	4	17	6	4	7	57	93	109	14	22
N_c^{24}	4	16	4	3	4	50	60	99	11	15
N_c^{25}	4	15	4	2	3	26	50	52	7	14
N_c^{26}	2	12	2	1	2	14	27	28	4	7
N_c^{27}	0	6	2	1	2	9	16	17	2	5
N_c^{28}	0	4	1	0	2	7	14	14	2	4
N_c^{29}	0	4	1	0	2	4	7	7	0	2
N_c^{30}	0	2	0	0	1	2	5	4	0	0
N_c^{31}	0	0	0	0	0	0	4	2	0	0
N_c^{32}	0	0	0	0	0	0	2	1	0	0

Broj čvorova u stablu sa prioritetima je jednak broju prefiksa. Time se ostvaruje značajna ušteda sa stanovišta ukupnog broja čvorova u odnosu na binarna i m-arna stabla jer nema praznih čvorova što je od posebnog značaja za IPv6 tabele gde su prefiksi na dubljim nivoima. Povećanje memorijskih resursa za IPv6 tabele je posledica dužih prefiksa pošto čvorovi u stablu sa prioritetima moraju da čuvaju kompletan prefiks ako su konfigurisani kao prioritetni. Ali, relativno povećanje je znatno manje nego kod prethodna dva algoritma (POLP i FIPL) jer u ovom algoritmu nema praznih čvorova koji čine najveći deo strukture stabla u prethodna dva algoritma u IPv6 slučaju. Takođe, maksimalna dubina stabla sa prioritetima D_{max} je manja nego originalnog binarnog stabla (maksimalne dužine prefiksa su 32 i 64 bita u IPv4 i IPv6 tabelama, respektivno), ali to smanjenje nije od velikog praktičnog značaja pošto je dubina i dalje prevelika, a to je posledica nesimetrično popunjenog originalnog binarnog stabla. To ujedno predstavlja i najveći problem ovog algoritma.

Tabela A.1.4.3. – Zahtevani memorijski resursi stabla sa prioritetima za IPv6 lukap tabele

Veličina tabele	D_{max}	N_c	M_{ps}[MB]
104779	46	104779	1.4364
130287	47	130287	1.7861
153126	47	153126	2.1370
175399	48	175399	2.4464
210039	47	210039	2.9295
240958	48	240958	3.3608
273791	47	273791	3.8840
307915	48	307915	4.3681
348866	48	348866	4.9490
365309	48	365309	5.1822

Tabela A.1.4.4. – Distribucija čvorova stabla sa prioritetima po nivoima za IPv6 tabele

Veličina tabele / Parametri	105K	130K	153K	175K	210K	241K	274K	308K	349K	365K
N_c^1	8490	10553	11814	13503	15855	17858	19312	21298	23656	24662
N_c^2	8446	10188	11782	13322	15471	17455	19196	21243	23314	24253
N_c^3	8191	10129	11191	12898	15294	17205	18439	20622	23062	24175
N_c^4	7599	9533	11160	12378	14330	16456	18324	19968	21905	23058
N_c^5	7541	8707	10275	12026	14043	15194	17282	19383	21790	22389
N_c^6	6779	8687	9400	10847	13144	15193	15969	18035	20320	21582
N_c^7	5955	7730	9285	10175	11946	13881	15898	17188	18792	19779
N_c^8	5882	6788	8348	9612	11191	12505	14468	16462	18269	18634
N_c^9	5200	6440	7341	8475	10651	11751	12952	14879	17121	17914
N_c^{10}	4468	5886	6719	7451	9470	11107	12080	13388	15380	16187
N_c^{11}	3813	5034	6402	7032	8283	9866	11488	12665	14048	14631
N_c^{12}	3772	4425	5549	6551	7419	8879	10394	11996	13167	13380
N_c^{13}	3227	3960	4885	5851	7407	8184	9501	10931	12966	13307
N_c^{14}	2889	3915	4308	5362	6780	7787	8860	10087	12083	12313
N_c^{15}	2702	3604	3966	5130	6394	7532	8343	9568	11285	11451
N_c^{16}	2688	3503	3937	5034	6164	7488	7875	9032	10662	10610
N_c^{17}	2666	3477	3729	4772	5861	6875	7630	8238	9664	9905
N_c^{18}	2634	3405	3586	4190	5015	5598	7187	7756	7875	8950
N_c^{19}	2333	2924	3465	4024	4061	4111	5742	6712	7731	7900
N_c^{20}	2040	2045	3254	2919	3572	4067	4078	4796	5555	7321
N_c^{21}	1637	1894	2485	2047	2375	3035	3955	4086	4755	5363
N_c^{22}	1024	1090	2043	1819	2061	2541	3279	3954	4324	5207
N_c^{23}	838	1024	1306	1515	2048	2216	2897	3707	4092	4837
N_c^{24}	621	934	1144	1328	1936	2048	2564	3352	3872	4092
N_c^{25}	556	852	1024	1147	1709	1926	2300	2927	3509	3988
N_c^{26}	512	695	912	1024	1298	1659	2048	2280	3013	2788
N_c^{27}	496	512	647	972	1024	1351	1994	2048	2320	2048
N_c^{28}	373	479	512	721	880	1067	1581	1687	2048	1956
N_c^{29}	256	327	439	519	628	1024	1141	1250	1731	1538
N_c^{30}	235	256	308	512	512	767	1024	1153	1326	1312
N_c^{31}	163	235	260	360	491	560	995	1102	1282	1267
N_c^{32}	128	181	256	271	484	512	957	1055	1278	1192
N_c^{33}	126	139	238	256	477	494	893	1024	1256	1176
N_c^{34}	101	128	228	238	426	473	821	986	1201	1111

N_c^{35}	92	127	213	216	392	449	797	971	1045	1077
N_c^{36}	89	119	210	191	325	448	512	682	1024	1024
N_c^{37}	64	112	156	160	256	376	396	512	734	962
N_c^{38}	58	82	128	134	128	299	256	305	512	681
N_c^{39}	32	64	73	128	99	256	128	256	316	512
N_c^{40}	26	32	64	99	64	157	100	128	256	267
N_c^{41}	16	32	32	64	32	128	64	71	128	256
N_c^{42}	8	16	19	47	16	64	32	64	73	128
N_c^{43}	6	8	16	32	12	54	16	32	64	64
N_c^{44}	4	8	8	17	8	32	9	16	32	32
N_c^{45}	2	4	4	16	4	16	8	8	16	16
N_c^{46}	1	2	3	8	2	8	4	6	8	8
N_c^{47}	0	2	2	4	1	4	2	4	4	4
N_c^{48}	0	0	0	2	0	2	0	2	2	2

A.1.5. Osnovni TCAM algoritam

Tabela A.1.5.1. – Zahtevani memorijski resursi osnovnog TCAM algoritma za IPv4 lukap tabele

Veličina tabele	N_p	L	M_t[MB]	M_s[MB]
104779	104779	32	0.3997	0.1999
130287	130287	32	0.4970	0.2485
153126	153126	32	0.5841	0.2921
175399	175399	32	0.6691	0.3345
210039	210039	32	0.8012	0.4006
240958	240958	32	0.9192	0.4596
273791	273791	32	1.0444	0.5222
307915	307915	32	1.1746	0.5873
348866	348866	32	1.3308	0.6654
365309	365309	32	1.3935	0.6968

Na osnovu (4.4.3.1.1)-(4.4.3.1.2) su za osnovni TCAM algoritam proračunati zahtevani memorijski resursi za TCAM memoriju M_t i za SRAM memoriju M_s koji su prikazani u tabelama A.1.5.1 i A.1.5.2 za analizirane IPv4 i IPv6 tabele, respektivno. U tabelama A.1.5.1 i A.1.5.2 su takođe date i vrednosti ukupnog broja prefiksa N_p i maksimalne dužine prefiksa L. Može se primetiti da se u slučaju IPv6 tabela troše veći resursi za TCAM memoriju usled dužih prefiksa, ali SRAM resursi ostaju nepromenjeni usled toga što dužina ID-a izlaznog porta nije promenjena. Može se videti da su memorijski zahtevi znatno manji u odnosu na prethodno analizirane algoritme, ali je ovde uključena i TCAM memorija koja je znatno složenija i skuplja od ‘običnih’

memorija (tipično interne i eksterne SRAM memorije). Pri tome je potrošnja velika pogotovo u slučaju velikih tabela jer TCAM tada sadrži velik broj prefiksa.

Tabela A.1.5.2. – Zahtevani memorijski resursi osnovnog TCAM algoritma za IPv6 lukap tabele

Veličina tabele	N_p	L	M_t[MB]	M_s[MB]
104779	104779	64	0.7994	0.1999
130287	130287	64	0.9940	0.2485
153126	153126	64	1.1683	0.2921
175399	175399	64	1.3382	0.3345
210039	210039	64	1.6025	0.4006
240958	240958	64	1.8384	0.4596
273791	273791	64	2.0889	0.5222
307915	307915	64	2.3492	0.5873
348866	348866	64	2.6616	0.6654
365309	365309	64	2.7871	0.6968

A.1.6. TCAM algoritam sa kofama promenljive veličine

U ovom algoritmu DTCAM memorija sadrži svih N_p postojećih prefiksa i potencijalno čak i dodatne prefikse (tzv. pokrivajuće prefikse) tako da njeni memorijski resursi sigurno nisu manji od osnovnog TCAM algoritma. Međutim, ti resursi su podeljeni na tzv. kofe pri čemu se u pretrazi vrši pretraga samo jedne kofe čime se značajno smanjuje potrošnja. Idealno bi bilo kada bi sve kofe bile iste veličine K (što bi istovremeno bila i maksimalna veličina kofe kako je originalno i definisan parametar K), pa bi tada logično bilo da je optimalna veličina kofe $K = \sqrt{N_p}$, jer bi tada veličina ITCAM memorije i svih kofa u DTCAM memoriji iznosila $\sqrt{N_p}$. Međutim, s obzirom da su kofe promenljive veličine, veličina formiranih kofa se kreće u granicama od $K/2$ do K i u [79] je pokazano da je optimalna maksimalna veličina kofe $K = \sqrt{2 \cdot N_p}$. U tabelama A.1.6.1 i A.1.6.2 su prikazani proračunati zahtevani memorijski resursi za ITCAM i DTCAM memorije (M_{it} i M_{dt}), kao i za ISRAM i DSRAM memorije (M_{is} i M_{ds}) za analizirane IPv4 i IPv6 tabele, respektivno koristeći (4.4.3.2.1)-(4.4.3.2.3). Pri tome je proračun rađen i za varijantu $K = \sqrt{N_p}$ i za $K = \sqrt{2 \cdot N_p}$ da bi se pokazalo i numerički da je bolja druga jednačina za K sa stanovišta najgoreg slučaja potrošnje. U tabelama A.1.6.1 i A.1.6.2 su date i vrednosti maksimalne veličine kofe K, broja kofa N_k, i ukupnog broja lokacija u DTCAM memoriji, odnosno ukupnog broja prefiksa u svim kofama DTCAM memorije $\sum_{i=1}^{N_k} N_p^i$. Parametar N_p^i predstavlja broj prefiksa u i-toj kofi.

Tabela A.1.6.1. – Zahtevani memorijski resursi TCAM algoritma sa kofama promenljive veličine za IPv4 lukap tabele

Veličina tabele	Varijanta	K	N_k	$\sum_{i=1}^{N_k} N_p^i$	$M_{it}[MB]$	$M_{dt}[MB]$	$M_{is}[MB]$	$M_{ds}[MB]$
104779	$K = \sqrt{N_p}$	324	501	105269	0.0016	0.4016	0.0016	0.2008
	$K = \sqrt{2 \cdot N_p}$	458	351	105124	0.0011	0.4010	0.0011	0.2005
130287	$K = \sqrt{N_p}$	359	553	130824	0.0017	0.4991	0.0017	0.2495
	$K = \sqrt{2 \cdot N_p}$	510	391	130669	0.0012	0.4985	0.0012	0.2492
153126	$K = \sqrt{N_p}$	391	596	153711	0.0018	0.5864	0.0019	0.2932
	$K = \sqrt{2 \cdot N_p}$	552	417	153536	0.0012	0.5857	0.0014	0.2928
175399	$K = \sqrt{N_p}$	419	655	176041	0.0020	0.6715	0.0021	0.3358
	$K = \sqrt{2 \cdot N_p}$	592	446	175838	0.0013	0.6708	0.0015	0.3354
210039	$K = \sqrt{N_p}$	458	722	210741	0.0022	0.8039	0.0023	0.4020
	$K = \sqrt{2 \cdot N_p}$	648	490	210521	0.0015	0.8031	0.0016	0.4015
240958	$K = \sqrt{N_p}$	491	775	241716	0.0024	0.9221	0.0025	0.4610
	$K = \sqrt{2 \cdot N_p}$	694	532	241483	0.0016	0.9212	0.0018	0.4606
273791	$K = \sqrt{N_p}$	523	840	274612	0.0025	1.0476	0.0029	0.5238
	$K = \sqrt{2 \cdot N_p}$	739	575	274357	0.0017	1.0466	0.0020	0.5233
307915	$K = \sqrt{N_p}$	555	876	308767	0.0026	1.1779	0.0030	0.5889
	$K = \sqrt{2 \cdot N_p}$	783	615	308519	0.0018	1.1769	0.0021	0.5885
348866	$K = \sqrt{N_p}$	591	927	349769	0.0028	1.3343	0.0032	0.6671
	$K = \sqrt{2 \cdot N_p}$	835	654	349509	0.0019	1.3333	0.0023	0.6670
365309	$K = \sqrt{N_p}$	604	953	366220	0.0028	1.3970	0.0033	0.6985
	$K = \sqrt{2 \cdot N_p}$	855	663	365948	0.0020	1.3960	0.0023	0.6980

Zbir kolona K i N_k predstavlja zbir lokacija koje se aktiviraju u ITCAM i DTCAM memoriji u jednom lukapu u najgorem slučaju. Zbir tih kolona se je uvek manji u slučaju kada je $K = \sqrt{2 \cdot N_p}$ što znači da je taj slučaj energetski efikasniji jer je manji broj lokacija aktiviran. Treba primetiti da je broj lokacija u DTCAM u oba slučaja

veći od ukupnog broja prefiksa što je posledica upisa pokrivajućih prefiksa u DTCAM memoriju. Iako je rešen problem potrošnje ostaje problem memorijskih zahteva TCAM memorija koji su sada čak i veći nego u osnovnom TCAM lukap algoritmu što čini ovo rešenje i dalje skupim.

Tabela A.1.6.2. – Zahtevani memorijski resursi TCAM algoritma sa kofama promenljive veličine za IPv6 lukap tabele

Veličina tabele	Varijanta	K	N_k	$\sum_{i=1}^{N_k} N_p^i$	$M_{it}[MB]$	$M_{dt}[MB]$	$M_{is}[MB]$	$M_{ds}[MB]$
104779	$K = \sqrt{N_p}$	324	480	105258	0.0015	0.8031	0.0015	0.2008
	$K = \sqrt{2 \cdot N_p}$	458	351	105129	0.0011	0.8021	0.0011	0.2005
130287	$K = \sqrt{N_p}$	361	543	130829	0.0017	0.9981	0.0017	0.2495
	$K = \sqrt{2 \cdot N_p}$	510	395	130681	0.0012	0.9970	0.0012	0.2493
153126	$K = \sqrt{N_p}$	391	584	153709	0.0018	1.1727	0.0019	0.2932
	$K = \sqrt{2 \cdot N_p}$	553	420	153545	0.0013	1.1715	0.0014	0.2929
175399	$K = \sqrt{N_p}$	419	634	176032	0.0020	1.3430	0.0020	0.3358
	$K = \sqrt{2 \cdot N_p}$	589	449	175847	0.0013	1.3416	0.0015	0.3354
210039	$K = \sqrt{N_p}$	458	696	210734	0.0022	1.6078	0.0022	0.4019
	$K = \sqrt{2 \cdot N_p}$	648	478	210516	0.0014	1.6061	0.0016	0.4015
240958	$K = \sqrt{N_p}$	491	756	241713	0.0023	1.8441	0.0024	0.4610
	$K = \sqrt{2 \cdot N_p}$	693	512	241469	0.0015	1.8423	0.0017	0.4606
273791	$K = \sqrt{N_p}$	522	795	274585	0.0024	2.0949	0.0027	0.5237
	$K = \sqrt{2 \cdot N_p}$	740	558	274348	0.0017	2.0931	0.0019	0.5233
307915	$K = \sqrt{N_p}$	555	840	308754	0.0025	2.3556	0.0028	0.5889
	$K = \sqrt{2 \cdot N_p}$	785	589	308503	0.0018	2.3537	0.0020	0.5884
348866	$K = \sqrt{N_p}$	591	899	349764	0.0027	2.6685	0.0030	0.6671
	$K = \sqrt{2 \cdot N_p}$	834	636	349501	0.0019	2.6670	0.0021	0.6670
365309	$K = \sqrt{N_p}$	604	911	366219	0.0027	2.7940	0.0030	0.6985
	$K = \sqrt{2 \cdot N_p}$	855	659	365967	0.0020	2.7921	0.0022	0.6980

A.1.7. M-12Wb

U slučaju analize M-12Wb algoritma , dužina lokacije u obe SRAM memorije je postavljena na $L_s = 144$ bita kao u [79]. Takođe je korišćen rezultat iz [79] koji precizira da je najoptimalnije rešenje kada su veličina jedne kofe i ukupan broj kofa jednaki (napomena: kofe su konstantne tj. fiksne veličine). U tabelama A.1.7.1 i A.1.7.2 su prikazani zahtevani memorijski resursi za ITCAM i DTCAM memorije (M_{it} i M_{dt}), kao i za ISRAM i DSRAM memorije (M_{is} i M_{ds}) za analizirane IPv4 i IPv6 tabele, respektivno. Zahtevani memorijski resursi su proračunati koristeći (4.4.3.3.1)-(4.4.3.3.3). U tabelama A.1.7.1 i A.1.7.2 su takođe date i vrednosti veličine kofe K, broja kofa N_k, dužine lokacije u SRAM memorijama L_s i broja pokrivajućih prefiksa N_{pp}. Može se uočiti da su ukupni zahtevani memorijski resursi za TCAM memorije značajno manji nego u prethodna dva analizirana TCAM algoritma. Ali, s druge strane ITCAM memorija sadrži znatno veći broj lokacija nego ITCAM memorija TCAM algoritma sa kofama promenljive veličine analiziranom u odeljku A.1.6 što znači da je potrošnja znatno veća od optimalne. Naravno, ona je i dalje za red veličine niža od osnovnog TCAM algoritma. Odatle se može zaključiti da ovaj algoritam smanjuje i potrošnju i TCAM memorijske zahteve u odnosu na osnovni TCAM algoritam, ali se optimizacija vrši po memorijskim resursima, tako da ostvarena ušteda u potrošnji snage nije optimalna.

Tabela A.1.7.1. – Zahtevani memorijski resursi M-12Wb algoritma za IPv4 lukap tabele

Veličina tabele	K	N_k	L_s	N_{pp}	M_{it}[MB]	M_{dt}[MB]	M_{is}[MB]	M_{ds}[MB]
104779	182	182	144	10582	0.0404	0.1264	0.1817	0.5686
130287	203	203	144	13392	0.0511	0.1572	0.2299	0.7074
153126	220	220	144	15692	0.0599	0.1846	0.2694	0.8308
175399	235	235	144	18071	0.0689	0.2107	0.3102	0.9480
210039	257	257	144	22418	0.0855	0.2520	0.3848	1.1338
240958	274	274	144	25532	0.0974	0.2864	0.4383	1.2888
273791	292	292	144	28862	0.1101	0.3253	0.4954	1.4637
307915	308	308	144	32153	0.1227	0.3619	0.5519	1.6284
348866	328	328	144	36409	0.1389	0.4104	0.6250	1.8468
365309	336	336	144	38074	0.1452	0.4307	0.6536	1.9380

Tabela A.1.7.2. – Zahtevani memorijski resursi M-12Wb algoritma za IPv6 lukap tabele

Veličina tabele	K	N_k	L_s	N_{pp}	M_{it}[MB]	M_{dt}[MB]	M_{is}[MB]	M_{ds}[MB]
104779	198	198	144	16341	0.1247	0.2991	0.2805	0.6730
130287	212	212	144	18905	0.1442	0.3429	0.3245	0.7715
153126	230	230	144	22124	0.1688	0.4036	0.3798	0.9081
175399	249	249	144	25431	0.1940	0.4730	0.4366	1.0643

210039	267	267	144	29818	0.2275	0.5439	0.5119	1.2238
240958	294	294	144	35471	0.2706	0.6595	0.6089	1.4838
273791	305	305	144	38226	0.2916	0.7097	0.6562	1.5969
307915	319	319	144	41191	0.3143	0.7764	0.7071	1.7468
348866	347	347	144	48665	0.3713	0.9186	0.8354	2.0670
365309	368	368	144	55908	0.4265	1.0332	0.9597	2.3247

A.1.8. Osnovni lukap heš algoritam

Osnovni lukap heš algoritam je ispitivan za slučajeve kad je broj lokacija koje se mogu dobiti heš funkcijom veći od broja prefiksa određene dužine dva, četiri i osam puta da bi se videla zavisnost maksimalne dužine liste prefiksa od veličine prostora u koji se mapiraju heširani prefiksi. Pošto heš funkcija generiše adresne bite onda je za dati prefiks određen prvo broj adresnih bita A takav da je $2^A \geq 2 \cdot N_p^i$, gde je N_p^i broj prefiksa dužine i i ova vrednost je odgovarala povećanju od dva puta. Za povećanje prostora od četiri i osam puta koristili su se jedan, odnosno dva dodatna adresna bita. Usled ove tehnike realna povećanja prostora su nešto veća od navedenih vrednosti (2, 4 i 8). Na osnovu (4.5.2.1.1) su proračunati zahtevani memorijski resursi za analizirane IPv4 i IPv6 tabele. Proračunati zahtevani memorijski resursi za memorije pokazivača M_p i memorije listi M_l su prikazani u tabelama A.1.8.1 i A.1.8.3 za IPv4 i IPv6 tabele, respektivno. Sa N_{lmax} je označena maksimalna dužina liste gledano po svim dužinama prefiksa, pri čemu brojevi u zagradi označavaju na koliko dužina prefiksa su se pojavile liste sa tolikim brojem članova. Vrednosti parametara N_p^i (broj prefiksa dužine i) i N_m^i (ukupan broj različitih memorijskih lokacija koje mogu da se dobiju heš funkcijom koja se vrši nad prefiksima dužine i) su prikazani u tabelama A.1.8.2 i A.1.8.4 za IPv4 i IPv6 tabele, respektivno. Pri tome, u tabelama A.1.8.2 i A.1.8.4, parametar N_m^i je dat samo za povećanje od dva puta. Vrednost parametra N_m^i za slučaj uvećanja od četiri puta je dva puta veća od vrednosti parametra N_m^i datih u tabelama A.1.8.2 i A.1.8.4. Vrednost parametra N_m^i za slučaj uvećanja od osam puta je četiri puta veća od vrednosti parametra N_m^i datih u tabelama A.1.8.2 i A.1.8.4.

Sa povećanjem memorijskog prostora, koji je na raspolaganju za heširanje, se dobijaju bolji rezultati po pitanju maksimalne dužine liste, ali su memorijski resursi neophodni za memoriju pokazivača tada znatno veći i to jedan od velikih problema heš funkcija koje zahtevaju veće memorijske resurse za bolje performanse tj. manji broj kolizija. U slučaju IPv6 tabela pozitivna osobina je da zahtevani memorijski resursi ne

rastu u značajnoj meri u odnosu na zahtevane memorijske resurse dobijene u slučaju IPv4 tabela.

Tabela A.1.8.1. – Zahtevani memorijski resursi osnovnog lukap heš algoritma za IPv4 lukap tabele

Veličina tabele	Povećanje	N_{lmax}	$M_p[MB]$	$M_l[MB]$
104779	2	6(1)	0.4509	0.6609
	4	4(3)	0.9019	0.6609
	8	4(1)	1.8038	0.6609
130287	2	5(2)	0.8213	0.8322
	4	4(5)	1.6426	0.8322
	8	3(8)	3.2853	0.8322
153126	2	5(4)	0.8579	0.9800
	4	4(6)	1.7159	0.9800
	8	4(2)	3.4318	0.9800
175399	2	5(4)	0.9015	1.1242
	4	4(7)	1.8030	1.1242
	8	4(3)	3.6059	1.1242
210039	2	6(1)	1.0347	1.3494
	4	5(2)	2.0694	1.3494
	8	4(2)	4.1389	1.3494
240958	2	6(1)	1.1068	1.5504
	4	5(2)	2.2135	1.5504
	8	4(2)	4.4271	1.5504
273791	2	6(1)	1.7734	1.7827
	4	5(1)	3.5468	1.7827
	8	4(1)	7.0935	1.7827
307915	2	6(2)	1.8656	2.0085
	4	4(5)	3.7313	2.0085
	8	4(1)	7.4625	2.0085
348866	2	7(1)	2.0043	2.2794
	4	5(2)	4.0086	2.2794
	8	4(3)	8.0172	2.2794
365309	2	6(1)	2.1362	2.3910
	4	5(1)	4.2724	2.3910
	8	4(2)	8.5448	2.3910

Tabela A.1.8.2. – Vrednosti parametara osnovnog heš algoritma u slučaju dvostrukog povećanja za IPv4 tabele

Veličina tabele / Parametri	105K	130K	153K	175K	210K	241K	274K	308K	349K	365K
N_p^8	18	19	19	18	19	18	20	22	23	27
N_m^8	64	64	64	64	64	64	64	64	64	64
N_p^9	5	4	3	5	10	9	9	10	10	12
N_m^9	16	16	8	16	32	32	32	32	32	32
N_p^{10}	7	6	7	8	13	15	20	25	25	28
N_m^{10}	16	16	16	32	32	32	64	64	64	64
N_p^{11}	11	15	15	20	30	40	55	65	71	78
N_m^{11}	32	32	32	64	64	128	128	256	256	256
N_p^{12}	37	57	59	86	114	135	159	181	213	221
N_m^{12}	128	128	128	256	256	512	512	512	512	512
N_p^{13}	91	103	140	189	238	276	313	379	430	459

N_m^{13}	256	256	512	512	512	1024	1024	1024	1024	1024
N_p^{14}	229	278	314	344	412	491	564	652	759	786
N_m^{14}	512	1024	1024	1024	1024	1024	2048	2048	2048	2048
N_p^{15}	389	493	561	662	800	979	1100	1232	1355	1388
N_m^{15}	1024	1024	2048	2048	2048	2048	4096	4096	4096	4096
N_p^{16}	7036	7497	8212	8590	9166	9760	10283	10825	11566	11791
N_m^{16}	16384	16384	32768	32768	32768	32768	32768	32768	32768	32768
N_p^{17}	1359	1830	2343	2931	3669	4269	4508	5103	5668	5897
N_m^{17}	4096	4096	8192	8192	8192	16384	16384	16384	16384	16384
N_p^{18}	2446	3425	4045	4939	6010	6947	7694	8632	9375	9807
N_m^{18}	8192	8192	8192	16384	16384	16384	16384	32768	32768	32768
N_p^{19}	7141	8849	10039	11060	13061	14938	16374	17850	19240	19810
N_m^{19}	16384	32768	32768	32768	32768	32768	32768	65536	65536	65536
N_p^{20}	6252	9268	10615	12182	14624	17092	19239	21706	25371	26944
N_m^{20}	16384	32768	32768	32768	32768	65536	65536	65536	65536	65536
N_p^{21}	4684	6701	8143	10292	13149	16160	18785	21507	25127	26534
N_m^{21}	16384	16384	16384	32768	32768	32768	65536	65536	65536	65536
N_p^{22}	7042	9340	11750	13439	16585	20239	24046	27797	32809	34538
N_m^{22}	16384	32768	32768	32768	65536	65536	65536	65536	131072	131072
N_p^{23}	8692	10926	13233	14653	18032	21471	24594	27971	31900	33578
N_m^{23}	32768	32768	32768	32768	65536	65536	65536	65536	65536	131072
N_p^{24}	59188	71045	83504	95634	111544	126635	142937	160106	181140	189401
N_m^{24}	131072	262144	262144	262144	262144	262144	524288	524288	524288	524288
N_p^{25}	22	65	22	99	708	165	237	327	1049	1199
N_m^{25}	64	256	64	256	2048	512	512	1024	4096	4096
N_p^{26}	14	54	7	54	538	214	313	430	1139	1349
N_m^{26}	32	128	16	128	2048	512	1024	1024	4096	4096
N_p^{27}	14	46	11	60	451	79	128	127	703	817
N_m^{27}	32	128	32	128	1024	256	256	256	2048	2048
N_p^{28}	22	59	19	47	283	119	220	242	309	142
N_m^{28}	64	128	64	128	1024	256	512	512	1024	512
N_p^{29}	11	55	10	43	193	362	556	743	273	140
N_m^{29}	32	128	32	128	512	1024	2048	2048	1024	512
N_p^{30}	29	101	32	22	293	418	836	978	217	180
N_m^{30}	64	256	128	64	1024	1024	2048	2048	512	512
N_p^{31}	0	1	0	0	0	4	6	3	1	11
N_m^{31}	0	2	0	0	0	8	16	8	2	32
N_p^{32}	40	50	23	22	97	123	795	1002	93	155
N_m^{32}	128	128	64	64	256	256	2048	2048	256	512

Tabela A.1.8.3. – Zahtevani memorijski resursi osnovnog lukap heš algoritma za IPv6 lukap tabele

Veličina tabele	Povećanje	N_{lmax}	M_p[MB]	M_l[MB]
104779	2	6(1)	0.4988	0.9279
	4	5(2)	0.9975	0.9279
	8	4(2)	1.9950	0.9279
130287	2	5(7)	0.6011	1.1559
	4	4(8)	1.2022	1.1559
	8	4(1)	2.4044	1.1559
153126	2	5(6)	0.6607	1.3617
	4	5(1)	1.3213	1.3617
	8	4(2)	2.6426	1.3617
175399	2	7(1)	1.0241	1.5716

	4	5(1)	2.0481	1.5716
	8	4(4)	4.0962	1.5716
210039	2	6(2)	1.0894	1.8845
	4	6(1)	2.1789	1.8845
	8	4(5)	4.3578	1.8845
240958	2	6(2)	1.1162	2.1622
	4	5(2)	2.2325	2.1622
	8	4(4)	4.4650	2.1622
273791	2	6(1)	1.4067	2.4702
	4	4(10)	2.8134	2.4702
	8	4(4)	5.6269	2.4702
307915	2	6(2)	1.4175	2.7798
	4	5(1)	2.8350	2.7798
	8	4(3)	5.6701	2.7798
348866	2	6(1)	2.2098	3.1717
	4	5(1)	4.4196	3.1717
	8	4(4)	8.8393	3.1717
365309	2	5(11)	2.2414	3.3226
	4	5(1)	4.4829	3.3226
	8	4(8)	8.9658	3.3226

Tabela A.1.8.4. – Vrednosti parametara osnovnog heš algoritma u slučaju dvostrukog povećanja za IPv6 tabele

Veličina tabele / **Parametri**	**105K**	**130K**	**153K**	**175K**	**210K**	**241K**	**274K**	**308K**	**349K**	**365K**
N_p^{21}	31	31	31	31	31	31	31	31	31	31
N_m^{21}	64	64	64	64	64	64	64	64	64	64
N_p^{22}	9	11	9	14	21	30	49	51	50	64
N_m^{22}	32	32	32	32	64	64	128	128	128	128
N_p^{23}	10	19	15	24	54	51	66	92	95	119
N_m^{23}	32	64	32	64	128	128	256	256	256	256
N_p^{24}	28	40	48	68	80	105	117	129	166	169
N_m^{24}	64	128	128	256	256	256	256	512	512	512
N_p^{25}	17	24	33	49	56	68	70	96	113	119
N_m^{25}	64	64	128	128	128	256	256	256	256	256
N_p^{26}	74	79	107	140	182	208	243	283	317	340
N_m^{26}	256	256	256	512	512	512	512	1024	1024	1024
N_p^{27}	61	68	87	78	117	117	148	165	166	181
N_m^{27}	128	256	256	256	256	256	512	512	512	512
N_p^{28}	168	210	227	266	295	374	416	487	593	605
N_m^{28}	512	512	512	1024	1024	1024	1024	1024	2048	2048
N_p^{29}	100	127	139	171	214	231	253	307	351	333
N_m^{29}	256	256	512	512	512	512	512	1024	1024	1024
N_p^{30}	289	366	422	491	586	748	847	925	1004	1055
N_m^{30}	1024	1024	1024	1024	2048	2048	2048	2048	2048	4096
N_p^{31}	1773	1828	1966	2108	2296	2496	2602	2739	2796	2969
N_m^{31}	4096	4096	4096	8192	8192	8192	8192	8192	8192	8192
N_p^{32}	5263	5669	6246	6482	6870	7264	7681	8086	8770	8822
N_m^{32}	16384	16384	16384	16384	16384	16384	16384	16384	32768	32768
N_p^{33}	328	496	576	734	957	1035	1133	1280	1407	1460
N_m^{33}	1024	1024	2048	2048	2048	4096	4096	4096	4096	4096
N_p^{34}	1031	1334	1767	2197	2712	3234	3375	3823	4261	4437

N_m^{34}	4096	4096	4096	8192	8192	8192	8192	8192	16384	16384
N_p^{35}	591	851	997	1256	1412	1679	1936	2145	2341	2416
N_m^{35}	2048	2048	2048	4096	4096	4096	4096	8192	8192	8192
N_p^{36}	1855	2574	3048	3683	4598	5268	5758	6487	7034	7391
N_m^{36}	4096	8192	8192	8192	16384	16384	16384	16384	16384	16384
N_p^{37}	1779	2222	2477	2706	3152	3690	4130	4547	4770	4885
N_m^{37}	4096	8192	8192	8192	8192	8192	16384	16384	16384	16384
N_p^{38}	5362	6627	7562	8354	9909	11248	12244	13303	14470	14925
N_m^{38}	16384	16384	16384	32768	32768	32768	32768	32768	32768	32768
N_p^{39}	1576	2267	2629	2979	3589	4250	4762	5354	6291	6711
N_m^{39}	4096	8192	8192	8192	8192	16384	16384	16384	16384	16384
N_p^{40}	4676	7001	7986	9203	11035	12842	14477	16352	19080	20233
N_m^{40}	16384	16384	16384	32768	32768	32768	32768	32768	65536	65536
N_p^{41}	1111	1754	2010	2537	3373	3998	4683	5418	6240	6693
N_m^{41}	4096	4096	4096	8192	8192	8192	16384	16384	16384	16384
N_p^{42}	3573	4947	6133	7755	9776	12162	14102	16089	18887	19841
N_m^{42}	8192	16384	16384	16384	32768	32768	32768	32768	65536	65536
N_p^{43}	1774	2337	2947	3374	4011	4962	5862	6907	8237	8608
N_m^{43}	4096	8192	8192	8192	8192	16384	16384	16384	32768	32768
N_p^{44}	5268	7003	8803	10065	12574	15277	18184	20890	24572	25930
N_m^{44}	16384	16384	32768	32768	32768	32768	65536	65536	65536	65536
N_p^{45}	2159	2737	3337	3748	4487	5414	6025	6953	8082	8492
N_m^{45}	8192	8192	8192	8192	16384	16384	16384	16384	16384	32768
N_p^{46}	6533	8189	9896	10905	13545	16057	18569	21018	23818	25086
N_m^{46}	16384	16384	32768	32768	32768	32768	65536	65536	65536	65536
N_p^{47}	14716	17556	20808	23781	27890	31572	35711	39776	45196	47167
N_m^{47}	32768	65536	65536	65536	65536	65536	131072	131072	131072	131072
N_p^{48}	44472	53489	62696	71853	83654	95063	107226	120330	135944	142234
N_m^{48}	131072	131072	131072	262144	262144	262144	262144	262144	524288	524288
N_p^{49}	5	12	2	33	181	37	60	87	276	299
N_m^{49}	16	32	4	128	512	128	128	256	1024	1024
N_p^{50}	17	53	20	66	527	128	177	240	773	900
N_m^{50}	64	128	64	256	2048	512	512	512	2048	2048
N_p^{51}	2	12	1	11	141	60	74	114	274	344
N_m^{51}	4	32	2	32	512	128	256	256	1024	1024
N_p^{52}	12	42	6	43	397	154	239	316	865	1005
N_m^{52}	32	128	16	128	1024	512	512	1024	2048	2048
N_p^{53}	3	15	2	13	108	25	33	39	181	200
N_m^{53}	8	32	4	32	256	64	128	128	512	512
N_p^{54}	11	31	9	47	343	54	95	88	522	617
N_m^{54}	32	64	32	128	1024	128	256	256	2048	2048
N_p^{55}	5	18	4	9	77	26	51	51	84	40
N_m^{55}	16	64	8	32	256	64	128	128	256	128
N_p^{56}	17	41	15	38	206	93	169	191	225	102
N_m^{56}	64	128	32	128	512	256	512	512	512	256
N_p^{57}	2	12	2	13	51	81	125	179	71	44
N_m^{57}	4	32	4	32	128	256	256	512	256	128
N_p^{58}	9	43	8	30	142	281	431	564	202	96
N_m^{58}	32	128	16	64	512	1024	1024	2048	512	256
N_p^{59}	8	26	6	2	76	123	208	242	52	41
N_m^{59}	16	64	16	4	256	256	512	512	128	128

N_p^{60}	21	75	26	20	217	295	628	736	165	139
N_m^{60}	64	256	64	64	512	1024	2048	2048	512	512
N_p^{61}	0	0	0	0	0	1	2	2	0	3
N_m^{61}	0	0	0	0	0	2	4	4	0	8
N_p^{62}	0	1	0	0	0	3	4	1	1	8
N_m^{62}	0	2	0	0	0	8	8	2	2	16
N_p^{63}	6	10	4	7	24	33	199	257	17	45
N_m^{63}	16	32	8	16	64	128	512	1024	64	128
N_p^{64}	34	40	19	15	73	90	596	745	76	110
N_m^{64}	128	128	64	32	256	256	2048	2048	256	256

A.1.9. PFHT algoritam

Za PFHT algoritam je za svaku dužinu prefiksa korišćen memorijski prostor veći osam puta od broja prefiksa dotične dužine, pri čemu je korišćeno $N_{bf} = 8$ heš funkcija. Zahtevani memorijski resursi za memorije pokazivača M_p i memorije listi M_l, kao i za registre R za analizirane IPv4 i IPv6 tabele, respektivno su prikazani u tabelama A.1.9.1 i A.1.9.3, respektivno. Vrednosti parametara N_p^i (broj prefiksa dužine i) i N_{br}^i (ukupan broj brojača koji se koristi na dužini prefiksa i) su prikazani u tabelama A.1.9.2 i A.1.9.4 za IPv4 i IPv6 tabele, respektivno.

Može se primetiti da nema značajnih razlika u memorijskim zahtevima između IPv4 i IPv6 tabela što predstavlja veoma dobru osobinu PFHT algoritma sa aspekta tranzicije na duže IPv6 adrese. Međutim, velik problem i za IPv4 tabele i za IPv6 tabele predstavlja realizacija brojača koji zahtevaju velik broj registarskih bita čime postaje problematična praktična implementacija PFHT algoritma.

Tabela A.1.9.1. – Zahtevani memorijski resursi PFHT algoritma za IPv4 lukap tabele

Veličina tabele	R[Mb]	M_p[MB]	M_l[MB]
104779	3.1261	1.8038	0.6609
130287	5.3039	3.2853	0.8322
153126	5.5558	3.4318	0.9800
175399	5.8589	3.6059	1.1242
210039	6.7300	4.1389	1.3494
240958	7.1825	4.4271	1.5504
273791	10.8171	7.0935	1.7827
307915	11.4114	7.4625	2.0085
348866	12.2470	8.0172	2.2794
365309	13.0245	8.5448	2.3910

Tabela A.1.9.2. – Vrednosti parametara PFHT algoritma za IPv4 tabele

Veličina tabele / Param.	105K	130K	153K	175K	210K	241K	274K	308K	349K	365K
N_p^8	18	19	19	18	19	18	20	22	23	27
N_{br}^8	256	256	256	256	256	256	256	256	256	256
N_p^9	5	4	3	5	10	9	9	10	10	12
N_{br}^9	64	64	32	64	128	128	128	128	128	128
N_p^{10}	7	6	7	8	13	15	20	25	25	28
N_{br}^{10}	64	64	64	128	128	128	256	256	256	256
N_p^{11}	11	15	15	20	30	40	55	65	71	78
N_{br}^{11}	128	128	128	256	256	512	512	1024	1024	1024
N_p^{12}	37	57	59	86	114	135	159	181	213	221
N_{br}^{12}	512	512	512	1024	1024	2048	2048	2048	2048	2048
N_p^{13}	91	103	140	189	238	276	313	379	430	459
N_{br}^{13}	1024	1024	2048	2048	2048	4096	4096	4096	4096	4096
N_p^{14}	229	278	314	344	412	491	564	652	759	786
N_{br}^{14}	2048	4096	4096	4096	4096	4096	8192	8192	8192	8192
N_p^{15}	389	493	561	662	800	979	1100	1232	1355	1388
N_{br}^{15}	4096	4096	8192	8192	8192	8192	16384	16384	16384	16384
N_p^{16}	7036	7497	8212	8590	9166	9760	10283	10825	11566	11791
N_{br}^{16}	65536	65536	131072	131072	131072	131072	131072	131072	131072	131072
N_p^{17}	1359	1830	2343	2931	3669	4269	4508	5103	5668	5897
N_{br}^{17}	16384	16384	32768	32768	32768	65536	65536	65536	65536	65536
N_p^{18}	2446	3425	4045	4939	6010	6947	7694	8632	9375	9807
N_{br}^{18}	32768	32768	32768	65536	65536	65536	65536	131072	131072	131072
N_p^{19}	7141	8849	10039	11060	13061	14938	16374	17850	19240	19810
N_{br}^{19}	65536	131072	131072	131072	131072	131072	131072	262144	262144	262144
N_p^{20}	6252	9268	10615	12182	14624	17092	19239	21706	25371	26944
N_{br}^{20}	65536	131072	131072	131072	131072	262144	262144	262144	262144	262144
N_p^{21}	4684	6701	8143	10292	13149	16160	18785	21507	25127	26534
N_{br}^{21}	65536	65536	65536	131072	131072	131072	262144	262144	262144	262144
N_p^{22}	7042	9340	11750	13439	16585	20239	24046	27797	32809	34538
N_{br}^{22}	65536	131072	131072	131072	262144	262144	262144	262144	524288	524288
N_p^{23}	8692	10926	13233	14653	18032	21471	24594	27971	31900	33578
N_{br}^{23}	131072	131072	131072	131072	262144	262144	262144	262144	262144	524288
N_p^{24}	59188	71045	83504	95634	111544	126635	142937	160106	181140	189401
N_{br}^{24}	524288	1048576	1048576	1048576	1048576	1048576	2097152	2097152	2097152	2097152
N_p^{25}	22	65	22	99	708	165	237	327	1049	1199
N_{br}^{25}	256	1024	256	1024	8192	2048	2048	4096	16384	16384
N_p^{26}	14	54	7	54	538	214	313	430	1139	1349
N_{br}^{26}	128	512	64	512	8192	2048	4096	4096	16384	16384
N_p^{27}	14	46	11	60	451	79	128	127	703	817
N_{br}^{27}	128	512	128	512	4096	1024	1024	1024	8192	8192
N_p^{28}	22	59	19	47	283	119	220	242	309	142
N_{br}^{28}	256	512	256	512	4096	1024	2048	2048	4096	2048
N_p^{29}	11	55	10	43	193	362	556	743	273	140
N_{br}^{29}	128	512	128	512	2048	4096	8192	8192	4096	2048
N_p^{30}	29	101	32	22	293	418	836	978	217	180
N_{br}^{30}	256	1024	512	256	4096	4096	8192	8192	2048	2048

N_p^{31}	0	1	0	0	0	4	6	3	1	11
N_{br}^{31}	0	8	0	0	0	32	64	32	8	128
N_p^{32}	40	50	23	22	97	123	795	1002	93	155
N_{br}^{32}	512	512	256	256	1024	1024	8192	8192	1024	2048

Tabela A.1.9.3. – Zahtevani memorijski resursi PFHT algoritma za IPv6 lukap tabele

Veličina tabele	R[Mb]	M_p[MB]	M_l[MB]
104779	3.5621	1.9950	0.9279
130287	4.2617	2.4044	1.1559
153126	4.6602	2.6426	1.3617
175399	6.8178	4.0962	1.5716
210039	7.3060	4.3578	1.8845
240958	7.4912	4.4650	2.1622
273791	9.3154	5.6269	2.4702
307915	9.4089	5.6701	2.7798
348866	13.8440	8.8393	3.1717
365309	14.0587	8.9658	3.3226

Tabela A.1.9.4. – Vrednosti parametara PFHT algoritma za IPv6 tabele

Veličina tabele / Parametri	105K	130K	153K	175K	210K	241K	274K	308K	349K	365K
N_p^{21}	31	31	31	31	31	31	31	31	31	31
N_{br}^{21}	256	256	256	256	256	256	256	256	256	256
N_p^{22}	9	11	9	14	21	30	49	51	50	64
N_{br}^{22}	128	128	128	128	256	256	512	512	512	512
N_p^{23}	10	19	15	24	54	51	66	92	95	119
N_{br}^{23}	128	256	128	256	512	512	1024	1024	1024	1024
N_p^{24}	28	40	48	68	80	105	117	129	166	169
N_{br}^{24}	256	512	512	1024	1024	1024	1024	2048	2048	2048
N_p^{25}	17	24	33	49	56	68	70	96	113	119
N_{br}^{25}	256	256	512	512	512	1024	1024	1024	1024	1024
N_p^{26}	74	79	107	140	182	208	243	283	317	340
N_{br}^{26}	1024	1024	1024	2048	2048	2048	2048	4096	4096	4096
N_p^{27}	61	68	87	78	117	117	148	165	166	181
N_{br}^{27}	512	1024	1024	1024	1024	1024	2048	2048	2048	2048
N_p^{28}	168	210	227	266	295	374	416	487	593	605
N_{br}^{28}	2048	2048	2048	4096	4096	4096	4096	4096	8192	8192
N_p^{29}	100	127	139	171	214	231	253	307	351	333
N_{br}^{29}	1024	1024	2048	2048	2048	2048	2048	4096	4096	4096
N_p^{30}	289	366	422	491	586	748	847	925	1004	1055
N_{br}^{30}	4096	4096	4096	4096	8192	8192	8192	8192	8192	16384
N_p^{31}	1773	1828	1966	2108	2296	2496	2602	2739	2796	2969
N_{br}^{31}	16384	16384	16384	32768	32768	32768	32768	32768	32768	32768
N_p^{32}	5263	5669	6246	6482	6870	7264	7681	8086	8770	8822
N_{br}^{32}	65536	65536	65536	65536	65536	65536	65536	65536	131072	131072
N_p^{33}	328	496	576	734	957	1035	1133	1280	1407	1460
N_{br}^{33}	4096	4096	8192	8192	8192	16384	16384	16384	16384	16384
N_p^{34}	1031	1334	1767	2197	2712	3234	3375	3823	4261	4437
N_{br}^{34}	16384	16384	16384	32768	32768	32768	32768	32768	65536	65536
N_p^{35}	591	851	997	1256	1412	1679	1936	2145	2341	2416

N_{br}^{35}	8192	8192	8192	16384	16384	16384	16384	32768	32768	32768
N_{p}^{36}	1855	2574	3048	3683	4598	5268	5758	6487	7034	7391
N_{br}^{36}	16384	32768	32768	32768	65536	65536	65536	65536	65536	65536
N_{p}^{37}	1779	2222	2477	2706	3152	3690	4130	4547	4770	4885
N_{br}^{37}	16384	32768	32768	32768	32768	32768	65536	65536	65536	65536
N_{p}^{38}	5362	6627	7562	8354	9909	11248	12244	13303	14470	14925
N_{br}^{38}	65536	65536	65536	131072	131072	131072	131072	131072	131072	131072
N_{p}^{39}	1576	2267	2629	2979	3589	4250	4762	5354	6291	6711
N_{br}^{39}	16384	32768	32768	32768	32768	65536	65536	65536	65536	65536
N_{p}^{40}	4676	7001	7986	9203	11035	12842	14477	16352	19080	20233
N_{br}^{40}	65536	65536	65536	131072	131072	131072	131072	131072	262144	262144
N_{p}^{41}	1111	1754	2010	2537	3373	3998	4683	5418	6240	6693
N_{br}^{41}	16384	16384	16384	32768	32768	32768	65536	65536	65536	65536
N_{p}^{42}	3573	4947	6133	7755	9776	12162	14102	16089	18887	19841
N_{br}^{42}	32768	65536	65536	65536	131072	131072	131072	131072	262144	262144
N_{p}^{43}	1774	2337	2947	3374	4011	4962	5862	6907	8237	8608
N_{br}^{43}	16384	32768	32768	32768	32768	65536	65536	65536	131072	131072
N_{p}^{44}	5268	7003	8803	10065	12574	15277	18184	20890	24572	25930
N_{br}^{44}	65536	65536	131072	131072	131072	131072	262144	262144	262144	262144
N_{p}^{45}	2159	2737	3337	3748	4487	5414	6025	6953	8082	8492
N_{br}^{45}	32768	32768	32768	32768	65536	65536	65536	65536	65536	131072
N_{p}^{46}	6533	8189	9896	10905	13545	16057	18569	21018	23818	25086
N_{br}^{46}	65536	65536	131072	131072	131072	131072	262144	262144	262144	262144
N_{p}^{47}	14716	17556	20808	23781	27890	31572	35711	39776	45196	47167
N_{br}^{47}	131072	262144	262144	262144	262144	262144	524288	524288	524288	524288
N_{p}^{48}	44472	53489	62696	71853	83654	95063	107226	120330	135944	142234
N_{br}^{48}	524288	524288	524288	1048576	1048576	1048576	1048576	1048576	2097152	2097152
N_{p}^{49}	5	12	2	33	181	37	60	87	276	299
N_{br}^{49}	64	128	16	512	2048	512	512	1024	4096	4096
N_{p}^{50}	17	53	20	66	527	128	177	240	773	900
N_{br}^{50}	256	512	256	1024	8192	2048	2048	2048	8192	8192
N_{p}^{51}	2	12	1	11	141	60	74	114	274	344
N_{br}^{51}	16	128	8	128	2048	512	1024	1024	4096	4096
N_{p}^{52}	12	42	6	43	397	154	239	316	865	1005
N_{br}^{52}	128	512	64	512	4096	2048	2048	4096	8192	8192
N_{p}^{53}	3	15	2	13	108	25	33	39	181	200
N_{br}^{53}	32	128	16	128	1024	256	512	512	2048	2048
N_{p}^{54}	11	31	9	47	343	54	95	88	522	617
N_{br}^{54}	128	256	128	512	4096	512	1024	1024	8192	8192
N_{p}^{55}	5	18	4	9	77	26	51	51	84	40
N_{br}^{55}	64	256	32	128	1024	256	512	512	1024	512
N_{p}^{56}	17	41	15	38	206	93	169	191	225	102
N_{br}^{56}	256	512	128	512	2048	1024	2048	2048	2048	1024
N_{p}^{57}	2	12	2	13	51	81	125	179	71	44
N_{br}^{57}	16	128	16	128	512	1024	1024	2048	1024	512
N_{p}^{58}	9	43	8	30	142	281	431	564	202	96
N_{br}^{58}	128	512	64	256	2048	4096	4096	8192	2048	1024
N_{p}^{59}	8	26	6	2	76	123	208	242	52	41
N_{br}^{59}	64	256	64	16	1024	1024	2048	2048	512	512
N_{p}^{60}	21	75	26	20	217	295	628	736	165	139

N_{br}^{60}	256	1024	256	256	2048	4096	8192	8192	2048	2048
N_{p}^{61}	0	0	0	0	0	1	2	2	0	3
N_{br}^{61}	0	0	0	0	0	8	16	16	0	32
N_{p}^{62}	0	1	0	0	0	3	4	1	1	8
N_{br}^{62}	0	8	0	0	0	32	32	8	8	64
N_{p}^{63}	6	10	4	7	24	33	199	257	17	45
N_{br}^{63}	64	128	32	64	256	512	2048	4096	256	512
N_{p}^{64}	34	40	19	15	73	90	596	745	76	110
N_{br}^{64}	512	512	256	128	1024	1024	8192	8192	1024	1024

A.2. Predloženi novi lukap algoritmi

U okviru ovog dela će biti predstavljeni memorijski zahtevi predloženih novih lukap algoritama koristeći njihovu analizu iz petog poglavlja. U okviru ove analize za konkretne lukap tabele, su napisane odgovarajuće aplikacije koje na osnovu strukture lukap tabele vrše proračun parametara analiziranih lukap algoritama. U svim predloženim novim lukap algoritmima je korišćena dubina podstabala $D = 8$. Takođe je u svim algoritmima parametar Δ postavljen na $\Delta = 8$.

A.2.1. BPFL i njegove modifikacije (BPFLSM i BPFLSS)

U tabeli A.2.1.1 su za analizirane IPv4 tabele predstavljeni zahtevani memorijski resursi BPFL-a i njegove modifikacije BPFLSM: R_{bs} (registarski resursi neophodni za balansirana stabla blokova za nalaženje podstabla svih nivoa), M_{bs} (memorijski resursi neophodni za balansirana stabla blokova za nalaženje podstabla svih nivoa), M_{pm} (memorijski resursi za početne memorije), $M_{dm}^{1..k}$ (memorijski resursi dodatnih memorija koje čuvaju indekse popunjenih čvorova), M_{dm}^{k+1} (memorijski resursi dodatnih memorija koje čuvaju kompletne bitmap vektore), M_{iz} (memorijski resursi za izlaznu memoriju). BPFL i BPFLSM su stavljeni u isti tabelu jer su gotovo identične strukture, pa su i memorijski proračuni gotovo identični. Sami parametri algoritama: N_{bs}^{i} (broj balansiranih stabala u nivou i), N_{c}^{i} (broj čvorova u jednom balansiranom stablu u nivou i) i $N_{m}^{i,j}$ (broj lokacija u dodatnoj memoriji j nivoa i) za analizirane IPv4 tabele su prikazani u tabeli A.2.1.2.

Može se uočiti da BPFLSM unosi uvećanje memorijskih zahteva samo u početnoj memoriji pošto je u svakoj njenoj lokaciji pridodat dodatni ID izlaznog porta. Uvećanje memorijskih zahteva BPFLSM-a nije veliko, ali zato doprinosi većoj fleksibilnosti lukap algoritma tako što omogućava izmeštanje početne memorije u eksternu memoriju čime se dodatno rasterećuju interni memorijski resursi.

Tabela A.2.1.1. – Zahtevani memorijski resursi BPFL i BPFLSM algoritama za IPv4 lukap tabele

Veličina tabele	Varijanta	$R_{bs}[b]$	$M_{bs}[MB]$	$M_{pm}[MB]$	$M_{dm}^{1..k}[MB]$	$M_{dm}^{k+1}[MB]$	$M_{iz}[MB]$
104779	BPFL	3696	0.0127	0.0685	0.0403	0.0381	0.7833
	BPFLSM	3696	0.0127	0.0815	0.0403	0.0381	0.7833
130287	BPFL	4768	0.0158	0.0842	0.0500	0.0498	1.0117
	BPFLSM	4768	0.0158	0.1001	0.0500	0.0498	1.0117
153126	BPFL	4880	0.0173	0.0929	0.0574	0.0607	1.2128
	BPFLSM	4880	0.0173	0.1103	0.0574	0.0607	1.2128
175399	BPFL	5952	0.0199	0.1061	0.0628	0.0726	1.4319
	BPFLSM	5952	0.0199	0.1262	0.0628	0.0726	1.4319
210039	BPFL	11456	0.0269	0.1349	0.0711	0.0887	1.7494
	BPFLSM	11456	0.0269	0.1621	0.0711	0.0887	1.7494
240958	BPFL	8448	0.0256	0.1379	0.0769	0.1085	2.0787
	BPFLSM	8448	0.0256	0.1642	0.0769	0.1085	2.0787
273791	BPFL	9824	0.0288	0.1527	0.0838	0.1279	2.4207
	BPFLSM	9824	0.0288	0.1816	0.0838	0.1279	2.4207
307915	BPFL	10560	0.0311	0.1650	0.0907	0.1463	2.7460
	BPFLSM	10560	0.0311	0.1962	0.0907	0.1463	2.7460
348866	BPFL	16336	0.0385	0.1950	0.1014	0.1669	3.1419
	BPFLSM	16336	0.0385	0.2341	0.1014	0.1669	3.1419
365309	BPFL	16640	0.0395	0.2008	0.1044	0.1770	3.3179
	BPFLSM	16640	0.0395	0.2407	0.1044	0.1770	3.3179

Tabela A.2.1.2. – Vrednosti parametara BPFL i BPFLSM algoritama za IPv4 lukap tabele

Veličina tabele / Parametri	105K	130K	153K	175K	210K	241K	274K	308K	349K	365K
N_{bs}^{2}	7	7	8	9	10	11	11	12	13	13
N_{c}^{2}	15	15	15	15	15	15	15	15	15	15
$N_{m}^{2,1}$	7	5	3	7	7	6	5	5	5	2
$N_{m}^{2,2}$	5	1	4	4	5	3	4	5	4	7
$N_{m}^{2,3}$	1	2	2	2	5	6	5	4	5	6
$N_{m}^{2,4}$	3	5	6	7	7	6	5	3	4	4
$N_{m}^{2,5}$	67	75	82	90	98	112	125	135	146	150
N_{bs}^{3}	103	126	141	159	182	203	219	237	261	269
N_{c}^{3}	63	63	63	63	63	63	63	63	63	63
$N_{m}^{3,1}$	1275	1482	1696	1695	1883	2018	2206	2305	2514	2603
$N_{m}^{3,2}$	630	812	882	1009	1109	1211	1258	1348	1520	1527
$N_{m}^{3,3}$	403	491	556	645	679	754	817	916	993	1055
$N_{m}^{3,4}$	230	300	380	405	520	539	603	651	809	819
$N_{m}^{3,5}$	559	744	917	1104	1361	1672	1973	2263	2599	2762
N_{bs}^{4}	6	13	5	15	114	37	55	58	162	163
N_{c}^{4}	15	15	15	15	15	15	15	15	15	15
$N_{m}^{4,1}$	3	4	0	2	5	10	39	45	4	8
$N_{m}^{4,2}$	0	0	0	1	0	8	28	33	1	3
$N_{m}^{4,3}$	0	2	0	0	1	2	9	11	0	1
$N_{m}^{4,4}$	0	2	0	0	0	5	2	9	0	0
$N_{m}^{4,5}$	0	0	0	0	0	0	5	8	0	0

Zahtevani memorijski resursi za BPFL i BPFLSM za analizirane IPv6 tabele su dati u tabeli A.2.1.3, a vrednosti parametara u tabeli A.2.1.4. Isti zaključci važe kao i u slučaju IPv4 tabela.

Tabela A.2.1.3. – Zahtevani memorijski resursi BPFL i BPFLSM algoritama za IPv6 lukap tabele

Veličina tabele	Varijanta	R_{bs}[b]	M_{bs}[MB]	M_{pm}[MB]	$M_{dm}^{1..k}$[MB]	M_{dm}^{k+1}[MB]	M_{iz}[MB]
104779	BPFL	9456	0.0918	0.2054	0.0354	0.0044	0.4377
	BPFLSM	9456	0.0918	0.2451	0.0354	0.0044	0.4377
130287	BPFL	12128	0.1121	0.2511	0.0445	0.0052	0.5348
	BPFLSM	12128	0.1121	0.2994	0.0445	0.0052	0.5348
153126	BPFL	12672	0.1242	0.2798	0.0562	0.0060	0.6091
	BPFLSM	12672	0.1242	0.3330	0.0562	0.0060	0.6091
175399	BPFL	14528	0.1397	0.3137	0.0676	0.0061	0.6828
	BPFLSM	14528	0.1397	0.3737	0.0676	0.0061	0.6828
210039	BPFL	20720	0.1726	0.3835	0.0836	0.0064	0.8160
	BPFLSM	20720	0.1726	0.4593	0.0836	0.0064	0.8160
240958	BPFL	21728	0.1863	0.4180	0.0999	0.0078	0.9160
	BPFLSM	21728	0.1863	0.4988	0.0999	0.0078	0.9160
273791	BPFL	27232	0.2072	0.4567	0.1193	0.0091	1.0312
	BPFLSM	27232	0.2072	0.5470	0.1193	0.0091	1.0312
307915	BPFL	31200	0.2306	0.5063	0.1383	0.0102	1.1565
	BPFLSM	31200	0.2306	0.6070	0.1383	0.0102	1.1565
348866	BPFL	28976	0.2489	0.5522	0.1611	0.0114	1.2831
	BPFLSM	28976	0.2489	0.6612	0.1611	0.0114	1.2831
365309	BPFL	29712	0.2559	0.5727	0.1697	0.0140	1.3634
	BPFLSM	29712	0.2559	0.6866	0.1697	0.0140	1.3634

Tabele A.2.1.5 i A.2.1.6 prikazuju zahtevane memorijske resurse za BPFLSS u slučaju analiziranih IPv4 i IPv6 tabela, respektivno. Takođe, u tabelama A.2.1.5 i A.2.1.6 su date i vrednosti parametara specifičnih za BPFLSS: $N_{D/2}$ (broj blokova veličine $2^{D/2}$ u dodatnoj memoriji za smeštanje bitmapa nepraznih podstabala donjeg nivoa) i N_{D+1} (broj blokova veličine 2^{D+1} u dinamički alociranom delu izlazne memorije). Memorijski resursi se ne razlikuju u značajnoj meri u odnosu na BPFLSM, ali je dodatna prednost što postoji samo jedna dodatna memorija pa ona kao i početna memorija u slučaju potrebe može da se izmesti u eksternu memoriju, što pruža dodatni stepen fleksibilnosti BPFLSS-u. Takođe, interesantno je da BPFLSS u slučaju IPv4 tabela ima manje memorijske zahteve za izlaznu memoriju, dok je u slučaju IPv6 tabela obrnuta situacija. Razlog je u razlici struktura podstabala u IPv4 i IPv6 slučaju. U IPv6 tabelama je manji udeo gušće popunjenih podstabala, pa kod BPFL i BPFLSM u IPv6 slučaju dolazi do rezervisanja manjeg broja neiskorišćenih lokacija u izlaznoj memoriji. Sa stanovišta najgoreg slučaja u BPFL i BPFLSM po pitanju neiskorišćenih lokacija je formiranje kompletnog bitmapa za veoma gusta stabla jer on rezerviše lokacije u

izlaznoj memoriji za sve čvorove koji su indeksirani u bitmapu. Bitmap adresira 510 čvorova u konkretnom slučaju za $D = 8$, a veoma retko broj popunjenih čvorova u podstablu bude veći od 200, tako da se u IPv4 tabelama na taj način rezerviše velik broj neiskorišćenih lokacija u izlaznoj memoriji. U IPv6 slučaju se takva situacija ređe dešava pa je samim tim i manje neiskorišćenih lokacija što se može videti po zahtevanim resursima za izlaznu memoriju u tabeli A.2.1.3. S druge strane ponašanje BPFLSS je konzistentnije tj. isto je ponašanje i za IPv4 i za IPv6 tabele pa samim tim dobijamo manje zahtevane resurse za izlaznu memoriju u IPv4 slučaju, a veće u IPv6 slučaju u odnosu na BPFL i BPFLSM.

Tabela A.2.1.4. – Vrednosti parametara BPFL i BPFLSM algoritama za IPv6 lukap tabele

Veličina tabele / **Parametri**	**105K**	**130K**	**153K**	**175K**	**210K**	**241K**	**274K**	**308K**	**349K**	**365K**
N_{bs}^{3}	1	2	1	4	2	1	2	2	5	3
N_{c}^{3}	3	3	3	3	3	3	3	3	3	3
$N_{m}^{3,1}$	0	0	0	1	2	0	0	3	3	2
$N_{m}^{3,2}$	0	1	0	1	1	0	2	0	4	2
$N_{m}^{3,3}$	0	1	0	1	0	0	0	0	4	0
$N_{m}^{3,4}$	0	0	0	1	0	0	0	0	1	0
$N_{m}^{3,5}$	1	1	1	0	1	3	3	2	1	5
N_{bs}^{4}	7	7	7	8	10	11	11	12	12	13
N_{c}^{4}	31	31	31	31	31	31	31	31	31	31
$N_{m}^{4,1}$	25	31	21	32	52	57	49	45	46	68
$N_{m}^{4,2}$	24	20	27	27	28	38	38	45	53	47
$N_{m}^{4,3}$	19	18	10	21	30	33	32	35	39	38
$N_{m}^{4,4}$	19	11	18	19	23	28	25	32	34	31
$N_{m}^{4,5}$	69	82	91	91	91	108	105	113	122	132
N_{bs}^{5}	55	73	81	89	104	117	119	132	142	146
N_{c}^{5}	63	63	63	63	63	63	63	63	63	63
$N_{m}^{5,1}$	509	667	778	935	1126	1305	1440	1614	1771	1859
$N_{m}^{5,2}$	73	109	157	188	249	311	344	428	485	488
$N_{m}^{5,3}$	19	19	25	40	55	63	100	106	133	155
$N_{m}^{5,4}$	3	5	7	5	9	8	25	22	35	42
$N_{m}^{5,5}$	1	1	1	2	1	4	11	11	16	14
N_{bs}^{6}	63	75	85	95	109	122	131	144	159	163
N_{c}^{6}	255	255	255	255	255	255	255	255	255	255
$N_{m}^{6,1}$	2230	2737	3397	3899	4681	5506	6316	7035	8142	8506
$N_{m}^{6,2}$	366	509	652	846	1066	1265	1546	1879	2143	2295
$N_{m}^{6,3}$	68	75	123	156	203	273	399	510	616	678
$N_{m}^{6,4}$	8	16	26	32	49	73	110	137	196	192
$N_{m}^{6,5}$	1	2	5	8	12	13	31	42	49	80
N_{bs}^{7}	1	4	1	4	29	9	14	17	46	51
N_{c}^{7}	63	63	63	63	63	63	63	63	63	63
$N_{m}^{7,1}$	0	0	0	0	0	0	0	0	0	0

$N_m^{7,2}$	0	0	0	0	0	0	0	0	0	0
$N_m^{7,3}$	0	0	0	0	0	0	0	0	0	0
$N_m^{7,4}$	0	0	0	0	0	0	0	0	0	0
$N_m^{7,5}$	0	0	0	0	0	0	0	0	0	0
N_{bs}^{8}	3	6	2	3	18	27	64	80	18	15
N_c^{8}	31	31	31	31	31	31	31	31	31	31
$N_m^{8,1}$	0	0	0	0	0	0	0	0	0	0
$N_m^{8,2}$	0	0	0	0	0	0	0	0	0	0
$N_m^{8,3}$	0	0	0	0	0	0	0	0	0	0
$N_m^{8,4}$	0	0	0	0	0	0	0	0	0	0
$N_m^{8,5}$	0	0	0	0	0	0	0	0	0	0

Tabela A.2.1.5. – Zahtevani memorijski resursi BPFLSS algoritma za IPv4 lukap tabele

Veličina tabele	$N_{D/2}$	N_{D+1}	$R_{bs}[b]$	$M_{bs}[MB]$	$M_{pm}[MB]$	$M_{dm}[MB]$	$M_{iz}[MB]$
104779	1387	387	3696	0.0127	0.0805	0.0794	0.4799
130287	1645	481	4768	0.0158	0.0992	0.0941	0.5954
153126	1900	568	4880	0.0173	0.1104	0.1087	0.6932
175399	2109	654	5952	0.0199	0.1274	0.1207	0.7970
210039	2447	779	11456	0.0269	0.1636	0.1400	0.9641
240958	2771	898	8448	0.0256	0.1659	0.1586	1.0831
273791	3100	1024	9824	0.0288	0.1816	0.1774	1.2256
307915	3402	1152	10560	0.0311	0.1981	0.1947	1.3689
348866	3803	1310	16336	0.0385	0.2364	0.2176	1.5703
365309	3960	1376	16640	0.0395	0.2428	0.2266	1.6426

Tabela A.2.1.6. – Zahtevani memorijski resursi BPFLSS algoritma za IPv6 lukap tabele

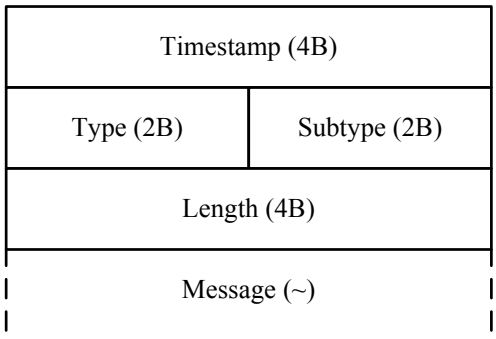
Veličina tabele	$N_{D/2}$	N_{D+1}	$R_{bs}[b]$	$M_{bs}[MB]$	$M_{pm}[MB]$	$M_{dm}[MB]$	$M_{iz}[MB]$
104779	1404	253	9456	0.0918	0.2373	0.0803	0.5508
130287	2116	257	12128	0.1121	0.2965	0.1211	0.6231
153126	2581	337	12672	0.1242	0.3298	0.1477	0.7430
175399	2691	420	14528	0.1397	0.3701	0.1540	0.8746
210039	3791	458	20720	0.1726	0.4505	0.2169	1.0126
240958	3768	628	21728	0.1863	0.4940	0.2156	1.2271
273791	5197	686	27232	0.2072	0.5469	0.2974	1.3431
307915	6469	743	31200	0.2306	0.6071	0.3702	1.4727
348866	5990	978	28976	0.2489	0.6612	0.3428	1.7689
365309	5782	1160	29712	0.2559	0.6866	0.3308	1.9698

B. MRT FORMAT LUKAP TABELA RUTERA

Da bi se obezbedila efikasna analiza rada Internet mreže, istraživačima je trebalo omogućiti da imaju uvid u razmene poruka protokola rutiranja, kao i uvid u sadržaj lukap tabela Internet rutera, pri čemu je trebalo omogućiti da istim podacima može da pristupa i da ih koristi velik broj istraživača. Usled postojanja navedenih potreba istraživača, razvijen je MRT format koji standardizuje prikaz rada protokola rutiranja u vidu razmene poruka, kao i prikaz sadržaja lukap tabele [94].

U okviru ove teze su korišćeni MRT zapisi lukap tabela dostupni u vidu fajlova za preuzimanje na [93]. Cilj ovog priloga je da izloži strukturu MRT zapisa lukap tabela.

Jedan fajl sadrži više MRT zapisa. Struktura zaglavlja jednog MRT zapisa je data na slici B.1. Polje *Timestamp* označava trenutak formiranja zapisa i dužine je 4 bajta. Polje *Type* predstavlja tip zapisa i dužine je dva bajta. Polje *Subtype*, dužine dva bajta, predstavlja podtip zapisa koji se koristi u sprezi sa poljem *Type* radi preciznijeg određivanja prirode zapisa, odnosno njegove strukture. Polje *Length* određuje dužinu zapisa u bajtovima, pri čemu se bajtovi zaglavlja ne uzimaju u obzir. Polje *Length* je dužine četiri bajta. Polje *Message* sadrži sam zapis.



Slika B.1. – Zaglavlje MRT zapisa

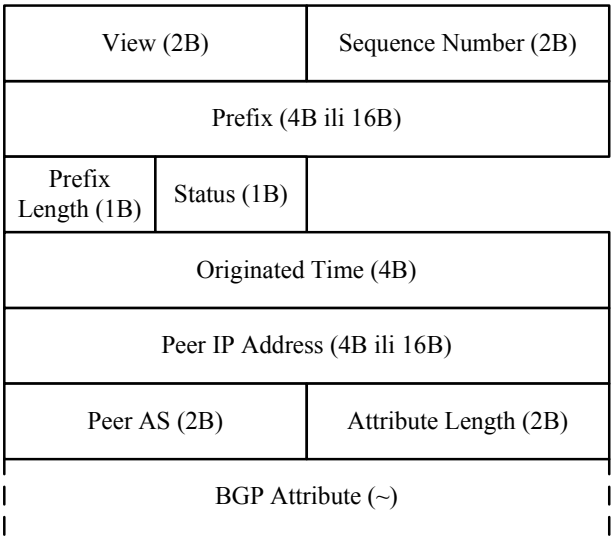
Kao što je rečeno polje *Type* definiše tip zapisa. U preporuci [94] je definisano devet tipova, pri čemu je svakom tipu dodeljena odgovarajuća vrednost polja *Type*. U tabeli B.1 su navedeni svi definisani tipovi zapisa sa odgovarajućim vrednostima polja *Type*.

Tabela B.1. – Definisani tipovi MRT zapisa

Vrednost polja *Type*	Tip MRT zapisa
11	OSPFv2
12	TABLE_DUMP
13	TABLE_DUMP_V2
16	BG4MP
17	BG4MP_ET
32	ISIS
33	ISIS_ET
48	OSPFv3
49	OSPFv3_ET

Tipovi TABLE_DUMP i TABLE_DUMP_V2 predstavljaju zapise u kojima se čuva sadržaj lukap tabele. Preporučuje se upotreba novijeg tipa TABLE_DUMP_V2 za prikaz strukture lukap tabele rutera. Međutim, na sajtu [93] gde su dostupni sadržaji lukap tabela rutera u MRT formatu, fajlovi koji čuvaju veoma stare sadržaje lukap tabela (osam godina i više) koriste stariji TABLE_DUMP tip, pa će stoga oba tipa biti izložena u nastavku.

Stariji TABLE_DUMP tip ima dve moguće vrednosti za polje *Subtype*. Ukoliko polje *Subtype* ima vrednost 1, tada MRT zapis sadrži IPv4 prefiks, a ukoliko polje *Subtype* ima vrednost 2, tada MRT zapis sadrži IPv6 prefiks. Struktura jednog MRT zapisa za TABLE_DUMP tip je data na slici B.2.



Slika B.2. – Struktura jednog MRT zapisa za TABLE_DUMP tip

Polje *View*, dužine dva bajta, ima tipično vrednost 0 u većini slučajeva. Ovo polje je namenjeno za slučaj višestrukih lukap tabela, poput servera ruta, da bi se različite lukap tabele iz istog uređaja mogle međusobno razlikovati. Polje *Sequence*

Number, dužine dva bajta, predstavlja redni broj zapisa u lukap tabeli. U slučaju da je broj zapisa veći od 2^{16}, polje *Sequence Number* koristi kružni princip brojanja, po kome se svaki put nakon dostizanja maksimalne vrednosti ponovo broji od nule. Polje *Prefix* predstavlja prefiks iz lukap tabele. Ovo polje ima dužinu četiri bajta za slučaj IPv4 prefiksa, odnosno, šesnaest bajtova za slučaj IPv6 prefiksa. Polje *Prefix Length*, dužine jedan bajt, određuje dužinu prefiksa, odnosno, broj bita od interesa iz polja *Prefix* koji zaista i čine prefiks. Očigledno je da predstava prefiksa nije optimalna u slučaju TABLE_DUMP tipa, pošto se uvek čuva onoliko bita kolika je dužina IP adrese (32b za IPv4, odnosno 128b za IPv6 prefikse). Videćemo u nastavku da TABLE_DUMP_V2 tip nema takvu manu. Polje *Status*, dužine jedan bajt, se ne koristi i ovo polje je uvek postavljeno na vrednost 1. Polje *Originated Time*, dužine četiri bajta, predstavlja vreme kada je dotični zapis prvi put kreiran u lukap tabeli. Polje *Peer IP Address*, predstavlja IP adresu '*peer*' rutera od kog je dobijen podatak o dotičnom prefiksu. Dužina ovog polja je četiri ili šesnaest bajtova, u zavisnosti da li je u pitanju IPv4 ili IPv6 adresa. Polje *Peer AS*, dužine dva bajta, predstavlja adresu autonomnog sistema kome pripada '*peer*' ruter od kog je dobijeno obaveštenje o prefiksu. Polje *Attribute Length*, dužine dva bajta, određuje dužinu BGP atributa koji je smešten u polje *BGP Attribute*.

Noviji TABLE_DUMP_V2 tip ima šest mogućih vrednosti za polje *Subtype* i one su date u tabeli B.2.

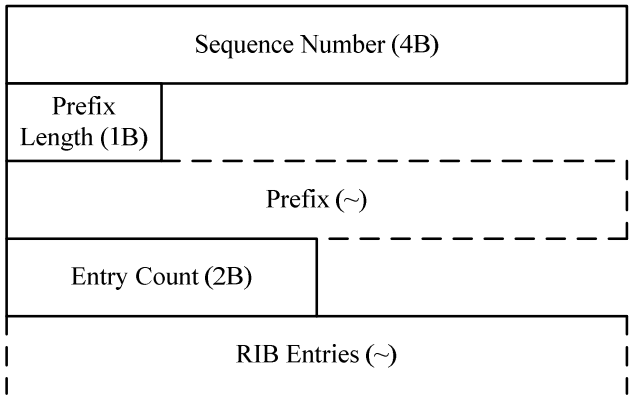
Tabela B.2. – Vrednosti *Subtype* polja za TABLE_DUMP_V2 tip

Vrednost polja *Subype*	Naziv podtipa
1	PEER_INDEX_TABLE
2	RIB_IPV4_UNICAST
3	RIB_IPV4_MULTICAST
4	RIB_IPV6_UNICAST
5	RIB_IPV6_MULTICAST
6	RIB_GENERIC

Podtip PEER_INDEX_TABLE predstavlja listu indeksiranih '*peer*' rutera. MRT zapis sa ovim podtipom se obično stavlja na početak liste MRT zapisa u jednom fajlu. Indeksi '*peer*' rutera se koriste potom u zapisima ostalih podtipova navedenih u tabeli B.2. Na ovaj način je izbegnuto dupliranje informacija o '*peer*' ruterima koje se javlja u slučaju starijeg TABLE_DUMP tipa. Sada se sve informacije o '*peer*' ruterima nalaze na samo jednom mestu, a potom se u ostalim zapisima koriste indeksi za identifikovanje '*peer*' rutera koji su mnogo kraći. Podtip RIB_IPV4_UNICAST predstavlja zapis koji

odgovara IPv4 unikast prefiksu. Podtip RIB_IPV4_MULTICAST predstavlja zapis koji odgovara IPv4 multikast prefiksu. Podtip RIB_IPV6_UNICAST predstavlja zapis koji odgovara IPv6 unikast prefiksu. Podtip RIB_IPV6_MULTICAST predstavlja zapis koji odgovara IPv6 multikast prefiksu. Podtip RIB_GENERIC se koristi za sva slučajeve koji nisu pokriveni nekim od prethodno navedena četiri podtipa i njega nećemo razmatrati u nastavku.

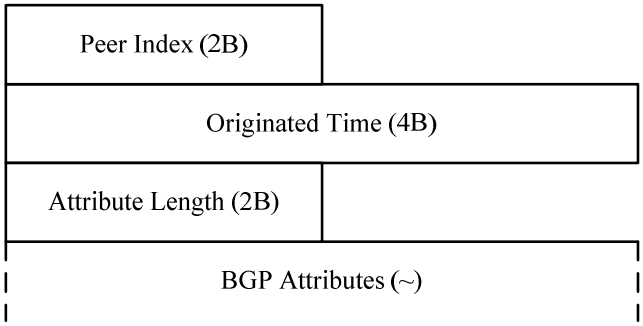
Zapisi podtipova RIB_IPV4_UNICAST, RIB_IPV4_MULTICAST, RIB_IPV6_UNICAST, RIB_IPV6_MULTICAST imaju dodatno zaglavlje tzv. zaglavlje unosa prikazano na slici B.3.



Slika B.3. – Zaglavlje unosa

Polje *Sequence Number*, dužine četiri bajta, predstavlja redni broj zapisa u lukap tabeli. Polje *Prefix Length*, dužine jedan bajt, predstavlja dužinu odgovarajućeg prefiksa iz lukap tabele. Polje *Prefiks* predstavlja vrednost prefiksa, pri čemu ovo polje ima onoliko bita koliko je dugačak sam prefiks. Kao što vidimo ovde nema čuvanja bita koji ne ulaze u prefiks, kao što je to bilo kod starijeg TABLE_DUMP tipa. Polje *Entry Count*, dužine dva bajta, predstavlja broj unosa koji odgovaraju dotičnom prefiksu. Naime, u starijem TABLE_DUMP tipu je za svaki prefiks moglo da postoji više zapisa, npr. ako je ruter dobio informaciju o istom prefiksu od više ’*peer*’ rutera ili ima više navedenih BGP atributa u lukap tabeli. Ovakav način čuvanja nije bio naročito efikasan, pa je uveden noviji TABLE_DUMP_V2 tip koji znatno efikasnije čuva informacije o lukap tabeli. U okviru TABLE_DUMP_V2 tipa za isti prefiks se može formirati više tzv. unosa koji se čuvaju u okviru samo jednog zapisa, onog koji odgovara dotičnom prefiksu. Polje *RIB Entries* je promenljive dužine i sadrži same unose.

Struktura jednog unosa je data na slici B.4. Polje *Peer Index*, dužine dva bajta, sadrži indeks ’peer’ rutera od koga je dobijen unos. Polje *Originated Time*, dužine četiri bajta, predstavlja vreme kada je unos prvi put kreiran u lukap tabeli. Polje *Attribute Length*, dužine dva bajta, predstavlja dužinu svih BGP atributa, koji se čuvaju u polju *BGP Attributes*.



Slika B.4. – Struktura jednog unosa

BIOGRAFIJA

Čiča Zoran je rođen 18.3.1979. u Zagrebu. Osnovno obrazovanje je započeo u Zagrebu. Zbog rata, 1991. je izbegao u Srbiju, u Sremsku Mitrovicu gde je završio osnovnu školu, a potom i gimnaziju. 1997. godine je upisao Elektrotehnički fakultet u Beogradu i diplomirao 27.5.2002. sa ukupnom srednjom ocenom 9.50, na diplomskom ispitu ocena 10. Tokom studija honorarno radio kao demonstrator u Laboratoriji za elektroniku Elektrotehničkog fakulteta u Beogradu. Postdiplomske studije - smer Telekomunikacione i računarske mreže na Elektrotehničkom fakultetu je upisao 2002. Ispite na postdiplomskim studijama položio je sa prosečnom ocenom 10. Magistarsku tezu „Primena determinističke teorije servisnih sistema u planiranju transportnih mreža“, čiji je mentor bio redovni profesor dr Petrović Grozdan, odbranio je decembra 2007. godine.

U 2002. stekao je zvanje asistenta-pripravnika na Elektrotehničkom fakultetu u Beogradu. U 2008. je stekao zvanje asistenta. Držao je računske i lab vežbe iz predmeta Telekomunikacione mreže, Projektovanje digitalnih telefonskih centrala, Računarske telekomunikacije. Trenutno je angažovan na predmetima Računarske osnove i primene Interneta, Komutacioni sistemi, Projektovanje komunikacionog hardvera, Internet programiranje, Arhitektura svičeva i rutera. Takođe je bio angažovan kao asistent na predmetu Računarske mreže i komunikacije na Vojno-tehničkoj akademiji. Učestvovao je na nekoliko projekata finansiranih od strane Ministarstva za nauku.

Zoran Čiča je autor više radova na konferencijama i časopisima, od kojih su dva bila nagrađena kao Najbolji radovi mladih autora na konferencijama ETRAN 2007 i 2009. Jedan rad je objavljen u časopisu sa SCI liste - IET Electronics Letters sa impakt faktorom 1.004.

Zoran Čiča je bio i recenzent za međunarodne časopise IEEE Communication Letters i IEEE Transactions on Communications, za međunarodnu konferenciju IEEE Workshop on High Performance Switching and Routing, kao i za domaće konferencije ETRAN i TELFOR. U 2010. dobio je priznanje Exemplary reviewer za časopis IEEE Communication Letters.

Прилог 1.

Изјава о ауторству

Потписани-а ЖОРАН Г. ЧИЧА

број уписа /

Изјављујем

да је докторска дисертација под насловом

ИМПЛЕМЕНТАЦИЈА ФУНКЦИЈА ПАКЕТСКОГ ПРОЦЕСИРАЊА У ИНТЕРНЕТ РУТЕРИМА ВЕЛИКОГ КАПАЦИТЕТА

- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио интелектуалну својину других лица.

Потпис докторанда

У Београду, 26. 4. 2012.

Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора ŽORAN ČIČA

Број уписа /

Студијски програм /

Наслов рада IMPLEMENTACIJA FUNKCIJA PAKETSKOG PROCESIRANJA U INTERNET RUTERIMA VELIKOG KAPACITETA

Ментор DR ALEKSANDRA SMIĆANIĆ

Потписани ŽORAN ČIČA

изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

У Београду, 26.4.2012.

Потпис докторанда

Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић" да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

IMPLEMENTACIJA FUNKCIJA PAKETSKOG PROCESIRANJA U INTERNET RUTERIMA VELIKOG KAPACITETA

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство

(2.) Ауторство - некомерцијално

3. Ауторство – некомерцијално – без прераде

4. Ауторство – некомерцијално – делити под истим условима

5. Ауторство – без прераде

6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Београду, 26. 4. 2012.