ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

SIMULACIJA TORUSNE ARHITEKTURE

– Diplomski rad –

Kandidat:

Milan Puzović 2009/394

Mentor:

doc. dr Zoran Čiča

Beograd, Septembar 2014.

SADRŽAJ

1. UVOD

Na probijanje paketske tehnologije na telekomunikaciono tržište dominantno je uticao tehnološki razvoj jer je tek sa razvojem integrisanih čipova bilo moguće efikasno procesirati pakete i omogućiti razvoj paketske komutacije do današnjeg nivoa. Kako se celokupno procesiranje paketa vrši na čvorovima (i zavisi od razvoja same tehnologije) javila se potreba da se razvijaju različite arhitekture paketskih komutatora kako bi se dobili što bolji rezultati u pogledu sledećih karakteristika:

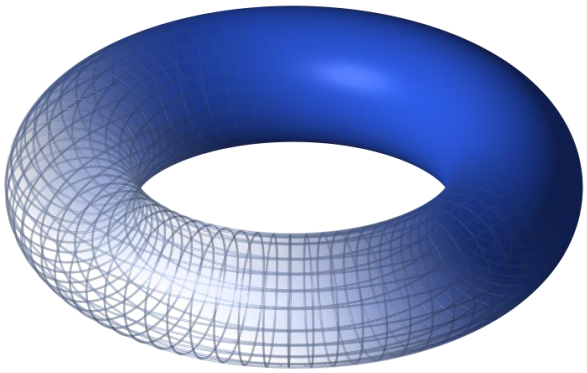
- podršla multikast saobraćaja,
- blokada,
- skalabilnost,
- ubrzanje komutatora,
- propusnost komutatora,
- cena,
- kompleksnost,
- tip paketa koji se komutira,
- baferi.

Ovaj rad se bavi jednom od tih arhitektura, torusnom arhitekturom, koja se primenjuje u pojedinim mrežnim uređajima. Glavna ideja ove arhitekture je da se teret komutacije prebaci na same portove. Tačnije sada svaki port pretstavlja mali mrežni čvor koji je povezan sa svojih šest suseda po x, y i z osi. Takođe, paket unutar same strukture može putovati po više različitih putanja od izvorišnog do odredišnog čvora. Sama struktura je zanimljiva jer je jednostavna, skalabilna i jeftina stoga ćemo se u ovom radu pozabaviti funkcionisanjem torusne arhitekture. U tu svrhu biće kreirana simulacija torusne arhitekture (koja će biti stavljena na test i biti analizirana) pomoću koje ćemo pratiti njeno ponašanje prilikom opterećenja saobraćajem, protoke po linkovima kao i uticaj na sam rad povećavanjem njenih dimenzija. Za potrebe rada, kod simulacije je pisan u programskom jeziku C, koji se nametnuo kao prirodan izbor jer ispunjava sve zahteve za programiranjem simulacije kao i zbog svoje jednostavnosti i razumljive sintakse. Kod je pisan u Microsoft Visual Studio Ultimate 2013 okruženju jer pruža sve neophodne alate za ugodno pisanje i testiranje koda.

Rad je organizovan tako da ćemo se prvo opširnije pozabaviti samom torusnom arhitekturom, njenom strukturom, (načinom) principom funkcionisanja, prednostima i manama i to će biti obuhvaćeno u drugom poglavlju. U trećem poglavljućemo prikazati kod simulacije, bavićemo se njegovom analizom i komentarisati primenjena rešenja i metode u kodu. Četvrto poglavlje je rezervisano za analizu dobijenih rezultata prilikom testiranja simulacije i njihovo komentarisanje da bi u petom poglavlju izveli zaključke i zapažanja vezana za torusnu arhitekturu.

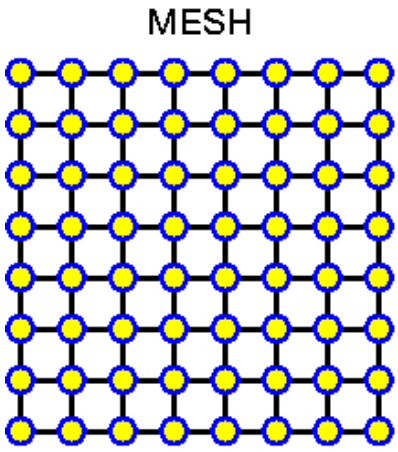
2. TORUSNA ARHITEKTURA

Torusna arhitektura je, kako joj samo ime kaže, ideju dobila od geometrijskog tela, torusa - obrtna površ koja se dobija kada se rotira kružnica u trodimenzionalnom prostoru oko ose koplanarne sa kružnicom, a koja ne dodiruje krug, poput unutrašnje gume automobila. Slika 2.1. prikazuje torusnu strukturu.



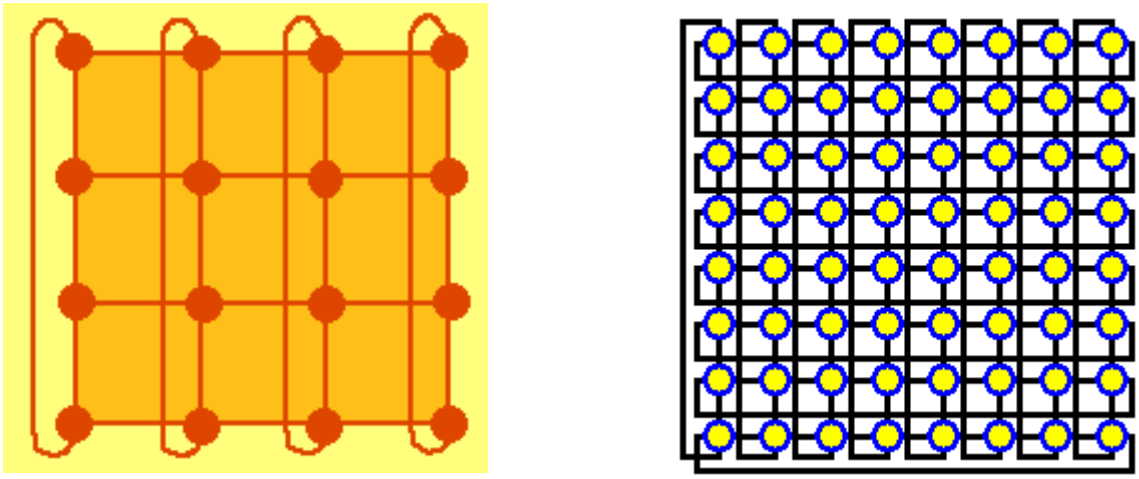
Slika 2.1. Torusna površ[4].

Kako bismo bolje razumeli povezanost arhitekture i geometrijskog tela krenimo od ravne ploče. Izdelićemo je na podjednake delove oblika kvadrata tako što ćemo povući jednak broj vertikalnih i horizontalnih linija i tako dobiti mrežu. Sada zamislimo da se na obodima ploče koji su ispresecani tim linijama kao i na presecima samih linija nalaze mali mrežni čvorovi. Na ovaj način smo dobili jednu meš mrežu mrežnih čvorova (Slika 2.2).



Slika 2.2. Čvorovi povezani u meš mrežu[5].

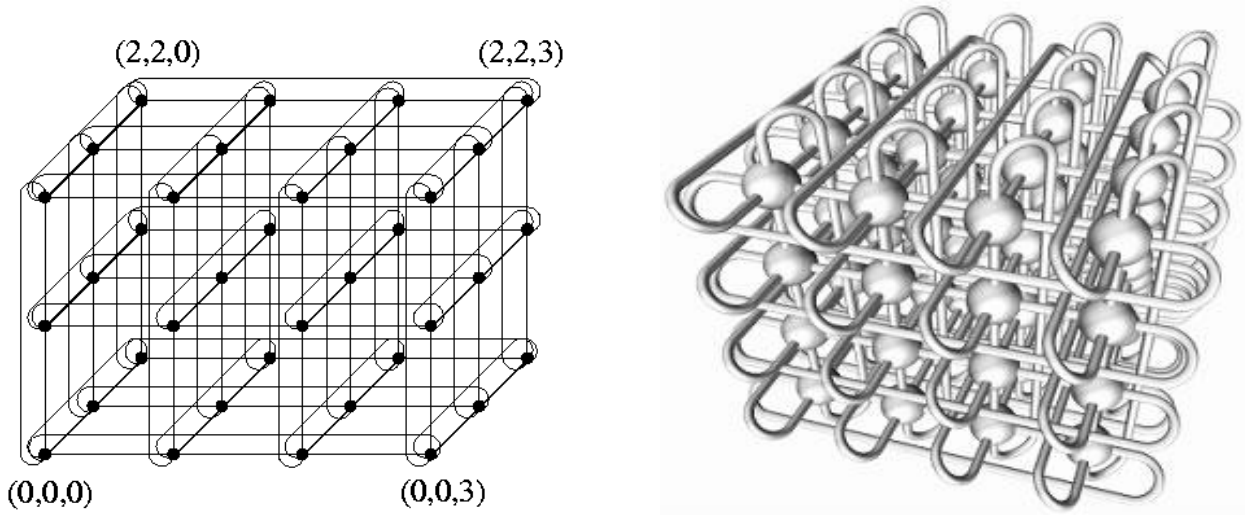
Primećujemo da, ako bi čvor na jednom kraju mreže poslao paket čvoru na drugom kraju, taj paket bi morao, u zavisnosti od dimenzija mreže, da pređe dug put kroz celu mrežu. Ali ako dodamo jedan link između ta dva čvora, ceo put paketa se svodi na prelazak tog linka čime se značajno ubrzava proces prenosa (Slika 2.3.a). Kada isti princip primenimo na svaka dva krajnja čvora ove mreže, koja su postavljeni jedan naspram drugog, dobićemo 2D torusnu strukturu (Slika 2.3b).



a) b)

Slika 2.3. Primer dvodimenzionalne torusne strukture: pod a) vidimo način povezivanja krajnjih čvorova[6]; b) dvodimenzionalna torusna struktura[5].

Slaganjem ovih torusnih struktura, jedna na drugu, tj. dodavanjem treće dimenzije i povezivanjem čvorova po istom principu, dobijamo 3D torusnu strukturu (Slika 2.4) koja povezuje NxNxN čvorova od kojih svaki čvor ima po 6 suseda po svim osama dekartovog koordinatnog sistema (-x, +x, -y, +y, -z, +z) izraženih u paketskim skokovima (hop).



a) b)

Slika 2.4. Primer trodimenzionalne torusne strukture: a) spoj više 2D struktura u jednu 3D strukturu[7]; b) model 3D torusne strukture[3].

Kako bi smo primetili prednost uvođenja treće dimenzije u torusnoj arhitekturi, poredićemo meš mrežu sa 32x32 čvora i 3D torus sa 10x10x10 čvorova. Zadatak je da se pošalje paket od čvora koji se nalazi na obodu mreže do čvora koji se nalazi u središtu. Kako bi paket došao do čvora u središtu meš mreže on mora, ako je poslat sa čvora koji se nalazi u gornjem levom kraju mreže, da napravi 16 skokova po dužini, a zatim još toliko po širini, što je ukupno 32 skoka. Kod 3D torusne arhitekture, ako paket krene iz iste pozicije, napraviće pet skokova po dužini, pet skokova po širini i još pet po dubini, što je ukupno 15 skokova naspram 32 skoka kod meš mreže. Više skokova uzrokuje veća kašnjenja što loše utiče na rad cele mreže. Bitno je napomenuti da idealan torusni oblik ima jednak broj čvorova po svim dimenzijama, inače cela struktura postaje nebalansirana.

Torusna arhitektura funkcioniše tako što se celokupna komutacija paketa prebacuje na portove predstavljene mrežnim čvorovima. Ti mrežni čvorovi se ponašaju kao mali ruteri od kojih svaki ima po sedam portova, jedan eksterni i šest internih ka svojim susedima. Ruteri su spojeni kratkim linkovima na kojima se nalaze baferi. Razlog postavljanja bafera je priroda paketskog saobraćaja jer može doći do slučaja kada veći broj paketa bude namenjen istom izlazu što, usled zagušenja, dovodi do odbacivanja paketa. Baferi koji su primenjeni u simulaciji su baferi na izlaznim portovima FIFO (*First In First Out*) tipa. Kod ovog tipa bafera paketi u istom redosledu ulaze i izlaze iz bafera.

Postoji tri tipa rutiranja unutar torusne arhitekture:

- fiksno,
- adaptivno,
- balansirano.

Fiksno rutiranje podrazumeva da postoji obrazac, redosled, kojim se rutiranje obavlja i ono se unapred definiše. Npr. prvo se paket usmerava po osi x, zatim y osi i na kraju z osi dok ne stigne do odredišnog čvora.

Adaptivno rutiranje funkcioniše tako što se posmatra opterećenost linkova saobraćajem. Oni koji su manje opterećeni biće korišćeni za prenos paketa dok će linkovi sa većim opterećenjem biti zaobilaženi.

Balansirano rutiranje vrši predikciju putanja paketa na osnovu težinskih koeficijenata, cena, koji se dodeljuju linkovima.

Glavna prednost torusne arhitekture je njena mogućnost da se lako proširuje bez povećavanja kompleksnosti komutacije jer ona ne zavisi od broja ulaznih i izlaznih portova. Takođe, jednostavnost arhitekture utiče na cenu koja raste linearno sa brojem portova što je još jedna od prednosti.

Mana ove arhitekture je da je blokirajuća. Može se desiti da dođe do zagušenja linkova i nepotrebnog odbacivanja paketa iako izlazni portovi nisu preopterećeni. Još jedna bitna mana je i kašnjenje paketa koji prolaze kroz ovu strukturu. Ono zavisi od udaljenosti samih portova, a sa povećanjem dimenzija mreže postaje značajno veliko.

3. KODNA REALIZACIJA

U ovom poglavlju ćemo se detaljno baviti samom simulacijom torusne arhitekture i analizom programskog koda. Pratićemo sve funkcije i komentarisati korake izvršavanja simulacije.

Kao što je pomenuto, simulacija torusne arhitekture je rađena u programskom jeziku C dok je kod razvijan u Microsoft Visual Studio okruženju. Simulacija bi trebalo da oponaša rad torusnog komutatora koji se nalazi pod saobraćajnim opterećenjem. Koraci simulacije se izvršavaju u diskretnim vremenskim trenucima jednake dužine (vremenski slot). Paketi, koji se generišu u čvorovima, generišu se po Bernulijevoj distribuciji. Destinacija paketa, mrežni čvor, predstavljena je koordinatama tog odredišnog čvora i upisana u zaglavlju paketa. Prilikom generisanja paketa, koordinate odredišta su formirane na osnovu uniformne raspodele. Generisani paketi su fiksne dužine (ćelija) i za potrebe ove simulacije korisnički sadržaj paketa nije bitan. Baferi su postavljeni na izlaznim portovima i FIFO su tipa. Rutiranje paketa je fiksno, paketi se prvo usmeravaju po x osi, zatim po y osi i na kraju z osi dok ne stignu do odredišnog čvora. U simulaciji je omogućeno:

- podešavanje dimenzija torusne arhitekture po svim koordinatama,
- zadavanje verovatnoće kojom će se paketi generisati u čvorovima (time se podešava opterećenje na ulaznim eksternim linkovima),
- određivanje trajanja simulacije izraženog u vremenskim slotovima.

Za potrebe analize simulacije testiraćemo ponašanje torusne arhitekture tako što ćemo pokretati simulaciju sa različitim podacima koje smo prethodno naveli i porediti dobijene rezultate. U tu svrhu posmatraćemo sledeće parametre:

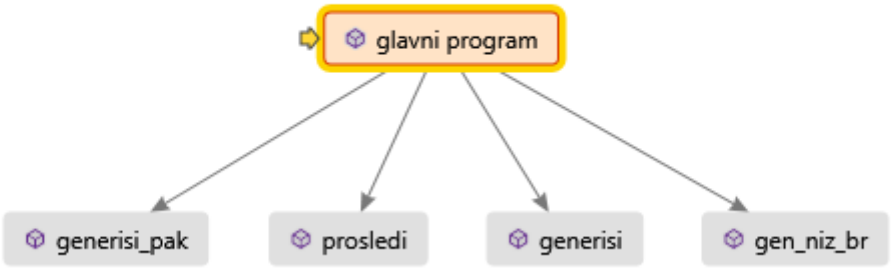
- maksimalni protok paketa,
- minimalni protok paketa,
- srednji protok paketa,
- maksimalna popunjenost bafera,
- minimalna popunjenost bafera
- srednja popunjenost bafera.

Paketski protok se posmatra na svim internim linkovima torusne arhitekture za vreme izvršavanja simulacije. Popunjenost bafera se odnosi na broj paketa koji se nalazi u baferima tokom i nakon izvršavanja simulacije.

Prilikom izrade koda simulacije bilo je potrebno rešiti sledeće zahteve:

- generisanje torusa, mrežnih čvorova i linkova,
- formirati bafere FIFO tipa na portovima čvorova,
- generisanje paketa,
- prosleđivanje paketa između čvorova,
- prikupljanje podataka.

Kako bi se izašlo u susret ovim zahtevima, programski kod simulacije organizovan je tako da se sastoji od pet celina, glavnog programa i četiri funkcije. (Slika 3.1)



Slika 3.1. Stablo programskog koda simulacije izrađeno uMicrosoft Visual Studio Ultimate 2013.

Strelica prikazuje glavni program kao koren stabla jer se u njemu vrši definisanje i pozivanje preostalih funkcija. Nakon izvršenja funkcije, korak se vraća nazad u glavni program odakle se može pristupiti ostalim funkcijama. Stablo prikazuje da su četri funkcije grane nezavisne jedna od druge i pristupa im se isključivo preko glavnog programa. Kao što se može zaključiti na osnovu stabla koda, svaka funkcija izvršava po jednu od navedenih stavki osim formiranja bafera i baferisanja kod kojih je primenjeno drugo rešenje koje će biti objašnjeno nešto kasnije. U daljem tekstu bavićemo se svakom celinom posebno i komentarisati primenjena rešenja.

3.1. Glavni program

Sa opisom koda simulacije krenućemo od njegovog korena, glavnog programa. U glavnom programu su definisane sve bitnije promenljive, pokazivači i funkcije. Takođe, vrši učitavanje podataka kao i ispis dobijenih rezultata.

Radi lakšeg rada na simulaciji i testiranja unos podataka se obavlja putem tekstualnog dokumenta koji je prethodno popunjen potrebnim podacima.

```
FILE *ptr_file;
ptr_file=fopen("input.txt","r");
if(!ptr_file){
      printf("Greska prilikom otvaranja fajla!\n");
      return1;
}
fscanf(ptr_file,"%d%d%d",&x,&y,&z);
fscanf(ptr_file,"%d",&p);
fscanf(ptr_file,"%d",&max_slot);
fclose(ptr_file);
```

Iz tekstualnog dokumenta `input.txt` učitavaju se x, y, z koordinata, verovatnoća kojom se generišu paketi u rasponu od 0 do 10 i broj vremenskih slotova, respektivno. Nakon što su učitani podaci, definišu se sve funkcije koje će se koristiti. Prve funkcije koje se pozivaju su funkcija za kreiranje torusa i funkcija koja rezerviše memorijski prostor za pohranjivanje podataka simulacije. Detaljan opis funkcija biće predstavljen u narednim odeljcima teksta.

Sledeća bitna stavka koju je potrebno uraditi je kreiranje petlje unutar koje bi se obavljale funkcije u određenom redosledu. Te funkcije simuliraju procese u torusnoj strukturi kao što su generisanje i prosleđivanje paketa u trajanju od jednog vremenskog slota. Kao rešenje, korišćena je `while` petlja u kojoj bi se funkcije izvršavale sve dok brojač vremenskih slotova q ne dostigne vrednost koja je definisana kao maksimalan broj vremenskih slotova (trajanje simulacije). Unutar `while` petlje izvršava se pozivanje dve funkcije `generisi_pak` i `prosledi` kao i inkrementiranje rednog broja vremenskog slota `q`. Funkcija `generisi_pak` generiše pakete u čvorovima i smešta ih u bafere na odgovarajućem portu. Funkcijom `prosledi` se paketi smešteni u baferima, prosleđuju do odgovarajućih čvorova.

```
while(q<max_slot){
      prosledi(x, y, z, q, pokaz, protok, bafer,&max_ext_baf, ptr);
      generisi_pak(x, y, z, q, pokaz, p, bafer,&max_baf, count_gen);
      q++;
}
```

Nakon izvršenja `while` petlje simulacija se završava i sledi deo koda koji je zadužen za proračun i ispis rezultata. Kao što je naglašeno na početku poglavlja po završetku simulacije potrebno je izračunati maksimalni, minimalni i srednji protok paketa po linkovima kao i maksimalnu, minimalnu i srednju popunjenost bafera paketima. Podaci dobijeni simulacijom se čuvaju u dvodimenzionom nizu matrica. Razlog tome je što prve tri dimenzije određuju posmatrani čvor, a četvrta dimenzija definiše izlazni port. Zbog navedenog načina čuvanja podataka, određivanje maksimalnih i minimalnih vrednosti se vrši po sledećem principu. Za trenutnu vrednost minimuma/maksimuma se postavi prvi element dvodimenzionog niza matrica. Koristeći četvorostruku `for` petlju prolazi se kroz sve elemente niza i svaki element se poredi sa trenutnom vrednošću minimuma/maksimuma. Ako je element veći od trenutne vrednosti, u slučaju maksimuma, taj element postaje novi maksimum, u slučaju minimuma se ignoriše. Ako je element manji od trenutne vrednosti, a traži se minimum, taj elemet će postati nova trenutna vrednost. U slučaju maksimuma se ignoriše. Srednje vrednosti su dobijene kao aritmetička sredina svih elemenata.

```
for(i =0; i<x; i++)
for(j =0; j<y; j++)
for(k =0; k<z; k++){
fprintf(ptr_file_i2,"\n\n%d%d%d\n\n", i, j, k);
for(l =0; l <6; l++){
      if(protok[i][j][k][l]> max_p) max_p = protok[i][j][k][l];
      if(protok[i][j][k][l]< min_p) min_p = protok[i][j][k][l];
      if(protok[i][j][k][l]!=0){
            sum_link = sum_link + protok[i][j][k][l];
            cnt1++;
      }
      if(bafer[i][j][k][l]< min_baf) min_baf = bafer[i][j][k][l];
      sum_baf = sum_baf + bafer[i][j][k][l];
      cnt2++;
      fprintf(ptr_file_i2,"%d%d%d\n", protok[i][j][k][l],
      bafer[i][j][k][l], count_gen[i][j][k][l]);
      fprintf(ptr,"%d ", count_gen[i][j][k][l]);
}
sred_p = sum_link / cnt1;
sred_b = sum_baf / cnt2;
```

Specijalan slučaj je maksimalna popunjenost bafera `max_baf` koja se računa za vreme izvršavanja simulacije. Inicijalna vrednost `max_baf` se postavi na 0. Tokom izvršavanja funkcije `generisi_pak` ako se paket generiše i smesti u bafer, broj paketa u tom baferu se poredi sa `max_baf`. Ako je broj paketa u baferu veći od `max_baf`, tada `max_baf` uzima tu vrednost, u suprotnom ostaje nepromenjeno. Nakon proračuna, rezultati se ispisuju u komandnom prozoru standardnom funkcijom printf i oslobađaju se resursi rezervisani za porebe simulacije.

3.2. Funkcija za generisanje torusa

Prvu funkciju koju ćemo opisati je funkcija za generisanje torusa. Ova funkcija treba da definiše čvorove i linkove između čvorova. Prvobitna ideja je bila da se čvorovi predstave kao strukturni podaci (strukture u jeziku C su složeni tipovi podataka koji se sastoje od određenog broja elemenata[2]) u kome bi elementi te strukture bili koordinate čvora, odgovarajući baferi za svaki link i informacije o susedima. Za potebe simulacije ovakva složena struktura nije bila neophodna, zato je problemu definisanja čvora pristupljeno na jednostavniji način. Pokazalo se da je sasvim dovoljno i jednostavnije, čvorove definisati putem njihovih koordinata. To je urađeno korišćenjem trostruke for petlje čime je formiran niz matrica čiji su elementi pokazivači na niz pokazivača na elemente tipa Paket. Niz pokazivača na elemente tipa `Paket` predstavljaju čvorove.

```
pokaz= malloc(x *sizeof(Paket***));
for(i =0; i < x; i++){
      *(pokaz + i)= malloc(y *sizeof(Paket**));
      for(j =0; j < y; j++){
            *(*(pokaz + i)+ j)= malloc(z *sizeof(Paket*));
            for(k =0; k < z; k++){
                  *(*(*(pokaz + i)+ j)+ k)= malloc(7*sizeof(Paket));
```

Kao što možemo primetiti za definisanjem linkova nije bilo potrebe jer poznajemo koordinate svakog čvora i paketi se kreću po torusu isključivo prateći koordinate. Kašnjenje po linkovima je fiksno i transport paketa između dva susedna čvora traje tačno jedan vremenski slot.

3.2.1. Paketi

Paket, kao osnovna jedinica komunikacije unutar digitalne mreže, mora da nosi određene podatke. Za potrebe simulacije podaci koji su sadržani u svakom paketu su:

- koordinate odredišnog čvora,
- redni broj vremenskog slota u kome je generisan paket,
- pokazivač na podatke tipa *Paket*.

Iz navedenih zahteva, zaključuje se da je opravdano rešenje pakete predstaviti kao strukture čiji će elementi biti traženi podaci. Koordinate odredišnog čvora se definišu kao tri podatka tipa *integer*, svaki za po jednu koordinatu. Prilikom prosleđivanja i generisanja paketa ovo su polja koja će čvorovi proveravati kako bi znali na koji izlazni link treba proslediti paket. Sledeći element je redni broj vremenskog slota u kome je generisan paket. Ovo polje je informacionog karaktera i obavlja funkciju pomoćnog parametra koji zbog sekvencijalnog izvršavanja koda mora da spreči generisanje paketa i njegovo prosleđivanje susednom čvoru u istom vremenskom slotu. Poslednji element unutar paketa je pokazivač na strukturni podatak tipa *Paket*, njegova uloga je veoma bitna za funkciju bafera čija će realizacija biti opisana u sledećem odeljku.

```
typedef struct paket {int x, y, z, vrem_slot;struct paket *sled;} Paket;
```

3.3. Baferi

Prilikom pisanja koda simulacije, baferima je posvećena posebna pažnja jer su jedini dinamički element unutar torusne arhitekture. Kao što je već naglašeno, baferi koji su primenjeni u simulaciji su FIFO tipa. FIFO tip je odabran zbog svoje jednostavnosti i lake programske realizacije. Baferi su postavljeni na svih sedam izlaznih portova svakog čvorova, od toga šest internih i jedan eksterni. Za potrebe simulacije, kapacitet bafera nije ograničen čime se uticaj odabranog tipa bafera izopštava iz krajnjih rezultata.

Sa pespektive koda, baferi su realizovani kao linearne liste (to su dinamičke strukture podataka odnosno niz podataka strukturnog tipa čiji su elementi logički povezani i mogu se dinamički stvarati i uništavati [2]). Bafer čine pokazivač na poslednji paket u baferu i paketi. Sada dolazimo do bitne funkcije pokazivača u samom paketu. Kako imamo pokazivač na poslednji paket, znamo koji je paket poslednji ušao u bafer i znamo o kom se baferu radi, ali da bi znali koji paket je prethodio trenutnom potreban nam je pokazivač. Na ovaj način bafer predstavljamo kao lanac paketa od kojih svaki paket pokazuje na njegovog prethodnika, osim prvog, što omogućava praćenje redosleda dolaska paketa u bafer i njihovog prosleđivanja.

Potrebno je naglasiti da je prilikom pisanja koda simulacije odlučeno da, nakon izvršavanja funkcije za generisanje torusa, pokazivač na poslednji paket bafera pokazuje na strukturni podatak tipa paket umesto vrednosti `NULL`. Kako bafer može biti prazan,poslednji paket se popunjava nevalidnim vrednostima, tj. vrednost koordinate x unutar paketa, postavlja se na negativnu konstantu (broj -9). Na ovaj način, ispitivanjem koordinate x poslednjeg paketa u baferu, lako se utvrđuje da li je bafer prazan. Negativna vrednost znači da je bafer prazan, a pozitivna da u baferu ima paketa. Zbog sličnih razloga se polja koja označavaju vremenski slot i sledeći paket postavljaju na vrednosti -1 i `NULL` respektivno. U kodu je korišćena `for` petlja sa sedam ciklusa, unutar koje se definišu prazni FIFO baferi na svakom od sedam izlaznih linkova, od kojih su šest linkova interni i jedan eksterni.

```
for(l =0; l <7; l++){
       (*(*(*(pokaz + i)+ j)+ k)+ l)->x =-9;
       (*(*(*(pokaz + i)+ j)+ k)+ l)->vrem_slot =-1;
       (*(*(*(pokaz + i)+ j)+ k)+ l)-> sled =NULL;
}
```

3.4. Funkcija generisanje paketa

Generisanje paketa je jedna od dve glavne funkcije koju simulirani torus treba da obavlja. U suštini, paket za koji kažemo da je generisan u čvoru, je u stvari paket koji je, krećući se kroz mrežu, došao na ulazni port torusa. Kako posmatramo samo torusnu strukturu, a ne celokupnu mrežu, čija ona može biti deo, smatraćemo da se paketi generišu na ulaznim portovima torusa. Uslovi koje funkcija generisanja paketa treba da ispuni su:

- generisanje paketa po Bernulijevoj distribuciji,
- izvorišni čvor paketa ne sme istovremeno biti i odredišni,
- usmeravanje paketa na odgovarajući izlazni port.

3.4.1. Generisanje paketa po Bernulijevoj distribuciji

Da bi se paket u čvoru torusa generisao po Bernulijevoj distribuciji potrebno je definisati promenljivu *p*. Ovaj parametar predstavlja verovatnoću da se paket generiše. Pošto programski jezik C u svojim bibliotekama nema funkciju koja bi ispunila ovaj zahtev, koristićemo se jednostavnom alternativom uz upotrebu matematičke funkcije `rand()` (vrednost `rand()` funkcije je pseudoslučajan broj s ravnomernom raspodelom u opsegu (0,RAND_MAX). RAND_MAX je simbolička konstanta čija se vrednost menja od računara do računara, ali koja ne sme da bude manja od 32767[2]).

```
if(rand()%10>=10-p){
```

Kao što možemo videti iz priložene linije koda, korišćena je osnovna selekcija, naredba **`if`**. Ako je uslov naredbe ispunjen, paket se generiše, u suprotnom se ne generiše i posmatrani čvor se preskače. Uslov je definisan tako što funkcija `rand()` generiše pseudoslučajan broj u rasponu od 0 do RAND_MAX. Zatim se od tog broja nađe modul od 10 i na taj način se opseg ograniči na vrednosti od 0 do 9. Dobijeni broj se poredi sa razlikom broja 10 i promenljivom *p* (da bi uslov bio pravilan, promenljiva *p* mora da je celobroja vrednost između 0 i 10. Ovo je analogno verovatnoći od 0 do 1 samo što se umesto koraka 0.1 koriste koraci od po 1). U slučaju da je dobijeni broj veći ili jednak razlici uslov je ispunjen. Na sledećem primeru posmatraćemo ponašanje uslova.

Neka je funkcija `rand()` generisala broj 5428 i neka je vrednost promenljive *p*=5 (što znači da je verovatnoća generisanja paketa jednaka 0.5) . Potražićemo vrednost tog broja po modulu 10 i kao rezultat dobijamo broj 8. Kako je 8 veće od 10 - 5 = 5 paket će biti generisan. Zaključujemo da, sa povećanjem verovatnoće p smanjuje se razlika sa desne strane nejednakosti, čime se povećava verovatnoća da uslov bude ispunjen.

3.4.2. Kontrolisanje generisanih paketa

Sledeći zahtev koji je potrebno ispuniti je da odredišni čvor generisanog paketa ne bude čvor u kome je generisan. Razlog za uvođenje ovog ograničenja je što bi, u slučaju da ograničenja nema, generisani paket odmah bio usmeren na eksterni link izvornog čvora. Na taj način paket ne bi uticao na saobraćaj u torusu, što ovaj slučaj, iz perspektive simulacije i traženih rezultata, čini suvišnim. U priloženom delu koda možemo videti primenjeno rešenje.

```
mark=1;
Paket *tekuci =malloc(sizeof(Paket));
while(mark){
	tekuci->x = rand()% x;
	tekuci->y = rand()% y;
	tekuci->z = rand()% z;
	if(i == tekuci->x && j == tekuci->y && k == tekuci->z) mark =1;
	else mark =0;
}
```

Promenljiva `mark` je kontrolni parametar i u zavisnosti da li ima vrednost 1 ili 0 **`while`** petlja će biti izvršena ili ne. Početna vrednost parametra `mark` je 1. Unutar **`while`** petlje se vrši generisanje koordinata odredišnog čvora koristeći matematičku funkciju `rand()`. Potom, u uslovu naredbe **`if`** ispitujemo da li su koordinate odredišnog čvora jednake koordinatama čvora u kom se paket generiše. Ako je uslov ispunjen vrednost parametra `mark` ostaje 1 što znači da su koordinate jednake, ponovo se ulazi u petlju i generišu se nove koordinate odredišnog čvora. Proces se ponavlja sve dok se uslov ne ispuni i vrednost promenljive `mark` postane 0 što uslovljava izlazak iz petlje.

3.4.3. Usmeravanje paketa

Nakon što je paket uspešno generisan sa poznatim koordinatama odredišnog čvora potrebno je, shodno dodeljenim koordinatama, usmeriti paket na jedan od 6 izlaznih portova. Kada je port određen, paket se smešta u bafer koji odgovara tom portu. U baferu paket dolazi na vrh steka po pravilu funkcionisanja bafera FIFO tipa, gde čeka svoj red na prosleđivanje. Navedeni koraci predstavljaju proces usmeravanja paketa pa je iz tog razloga i prilikom pisanja koda, problem usmeravanja paketa podeljen na dva dela:

- određivanje izlaznog porta
- smeštanje generisanih paketa u bafer

i) Određivanje porta

Prva stavka koju treba ispuniti je određivanje izlaznog porta. Svaki čvor ima 7 portova od kojih su 6 za interne linkove i jedan za eksterni. Kako smo u prethodnom delu teksta pomenuli da čvorovi ne mogu slati pakete sami sebi, posmatraćemo samo 6 portova za interne linkove. Sledeće što treba uraditi je indeksirati portove kako bi znali kojim portom u kom pravcu i smeru šaljemo paket. Izbor indeksa za portove je bio proizvoljan, ali jednom definisani indeksi važe za svaki čvor u torusu. Vodeći se prethodnim uslovom portovi svih čvorova su indeksirani na sledeći način:

- port sa indeksom 0 – paket se šalje po x koordinati u negativnom smeru,
- port sa indeksom 1 – paket se šalje po x koordinati u pozitivnom smeru,
- port sa indeksom 2 – paket se šalje po y koordinati u negativnom smeru,
- port sa indeksom 3 – paket se šalje po y koordinati u pozitivnom smeru,
- port sa indeksom 4 – paket se šalje po z koordinati u negativnom smeru,
- port sa indeksom 5 – paket se šalje po z koordinati u pozitivnom smeru,
- port sa indeksom 6 – paket se šalje na eksterni link.

Izabrani način rutiranja je fiksni pa je prva koordinata kojom se usmerava paket, x koordinata. Ako x koordinata generišućeg čvora odgovara x koordinati odredišnog čvora paket se usmerava po y koordinati. Ako i y koordinata odgovara, paket se šalje po z koordinati. Kako je redosled usmeravanja paketa po koordinatama definisan,potrebno je utvrditi smer izabrane koordinate.

Ovo je nešto komplikovaniji problem jer zbog prirode torusne arhitekture, ne postoje ivični čvorovi. To znači da ako se krene od proizvoljnog čvora samo u jednom smeru npr. pozitivnom smeru x koordinate, moguće je doći u čvor od koga se krenulo. Zbog potrebe za efikasnim i brzim prosleđivanjem u interesu je da paket prelazi što kraći put.

Da bi zahtevani uslov bio ispunjen prvo moramo odrediti okolinu čvora koji generiše paket, odnosno odrediti koji su čvorovi bliži po negativnom, a koji po pozitivnom smeru kretanja. Za primer ćemo uzeti koordinatu x i neka je dimenzija torusa po toj koordinati 6 (dimenzije se izražavaju u paketskim skokovima).

Prvo što treba uraditi je naći polovinu dimenzije torusa po željenoj koordinati. Ako se dobije broj koji nije ceo, uzima se prvi veći, celi broj. U našem slučaju je to 3. Zatim se proverava da li je x koordinata odredišnog čvora paketa manja od x koordinate izvorišnog čvora. U slučaju da jeste, dolazimo do poduslova. Poduslov proverava da li je x koordinata odredišnog čvora veća od razlike x koordinate izvorišnog čvora i polovine dimenzije torusa (broja 3).

Ako je uslov ispunjen, paket se šalje u negativnom smeru x ose, odnosno na port 0, u suprotnom paket se šalje na port 1, odnosno u pozitivnom smeru. Za slučaj da glavni uslov nije ispunjen javlja se novi poduslov kojim se ispituje da li je x koordinata odredišnog čvora paketa manja od zbira koordinate izvorišnog čvora i polovine dimenzije torusa. Ispunjavanjem poduslova paket se usmerava u pozitivnom smeru (port 1). Ukoliko je neispunjen poduslov, paket se usmerava na negativni smer port 0. Na sličan način funkcioniše usmeravanje i po ostalim koordinatama.

ii) Smeštanje paketa u bafer

Kao što smo pomenuli u odeljku 3.4, baferi su linearne liste, odnosno čine ih pokazivač na poslednji paket u baferu i lanac paketa. Nakon što smo kreirali paket i usmerili ga na odgovarajući link, paket se smešta u bafer. Prilikom smeštanja paketa javljaju se dva slučaja:

- paket se smešta u prazan bafer,
- paket se smešta u bafer u kome već ima paketa.

Da li je bafer prazan ili ima paketa proveravamo tako što ispitujemo vrednost x koordinate poslednjeg paketa u baferu. Negativna vrednost koordinate označava da je bafer prazan, a pozitivna da u baferu ima paketa. U slučaju da je bafer prazan, vrednosti generisanog paketa se prepisuju u poslednji paket na koji pokazuje pokazivač, a koji je do tada bio nevažeći. Polje pokazivača unutar paketa je **NULL**.

Za slučaj da u baferu ima paketa primenjuje se drugo rešenje. Generiše se novi pomoćni paket koji će pamtiti vrednosti poslednjeg paketa, a koji je do dolaska novog bio na vrhu steka. U poslednji paket se sada upisuju nove vrednosti sa tim da polje pokazivača na paket `sled` sada pokazuje na pomoćni paket. Na ovaj način je novi paket postavljen na vrh steka, a da lanac paketa nije prekinut.

```
if(mark==0&&((i>tekuci->x && tekuci->x >= i-x/2)|| tekuci->x > i+x/2)) {
      if((*(*(*(pokaz + i)+ j)+ k)+0)->x >=0){
      sledeci = malloc(sizeof(Paket));
      *sledeci =*(*(*(*(pokaz + i)+ j)+ k)+0);
      tekuci->sled = sledeci;
      }
      *(*(*(*(pokaz + i)+ j)+ k)+0)=*tekuci;
      bafer[i][j][k][0]++;
      mark=1;
}
elseif(mark==0&&((i<tekuci->x && tekuci->x<=i+x/2)|| tekuci->x<i-x/2)){
      if((*(*(*(pokaz + i)+ j)+ k)+1)->x >=0){
      sledeci = malloc(sizeof(Paket));
      *sledeci =*(*(*(*(pokaz + i)+ j)+ k)+1);
      tekuci->sled = sledeci;
      }
      *(*(*(*(pokaz + i)+ j)+ k)+1)=*tekuci;
      bafer[i][j][k][1]++;
      mark =1;
}
```

U priloženom kodu prikazan je opisani postupak koji se odnosi na x koordinatu. Za y i z koordinate kod je realizovan po istom principu.

3.5. Funkcija prosleđivanja paketa

Pored funkcije generisanja paketa, funkcija za prosleđivanje paketa je druga glavna funkcija koju simulirani torus treba da obavlja. Sam proces prosleđivanja paketa realizovan je tako da nema prioriteta između čvorova kao i bafera na linkovima između čvorova. Da bi paket bio uspešno prosleđen odgovarajućem čvoru, funkcija koja obavlja proces prosleđivanja treba da ispuni sledeće zahteve:

- ispituje da li je bafer prazan,
- preuzima paket sa dna steka unutar bafera,
- prosleđuje paket do sledećeg čvora,
- utvrđuje da li je novi čvor odredišni čvor,
- vrši usmeravanje paketa na odgovarajući port.

Shodno navedenim zahtevima, kod funkcije je definisan tako da se sastoji iz dve celine:

- preuzimanje paketa iz bafera,
- prosleđivanje paketa.

Prva celina ispunjava prva dva zahteva dok se preostala tri zahteva realizuju u drugoj celini.

3.5.1. Preuzimanje paketa iz bafera

Prva celina koda funkcije za prosleđivanje paketa se bavi analizom stanja bafera i pripremom paketa za njegovo prosleđivanje. Kao prvi zadatak potrebno je ispitati da li je posmatrani bafer prazan. Tom prilikom korišćena je `if` naredba u čijem uslovu se ispituje x koordinata odredišnog čvora poslednjeg paketa u baferu. U slučaju da tražena koordinata ima negativnu vrednost, smatra se da je bafer prazan. Takav bafer se preskače i prelazi se na sledeći.

Bitno je napomenuti, da je prilikom testiranja koda uočeno da se neki paketi nakon generisanja u čvoru, odmah prosledjuju u istom vremenskom slotu, čak i više puta. Kao uzrok ove pojave javlja se nemogućnost za istovremeno obavljanje procesa u svim čvorovima usled sekvencijalnog izvršavanja koda. To znači da se prvo prosleđuju paketi iz čvorova sa nižim koordinatama. Zatim se inkrementira promenljiva, koja sad nosi vrednost koordinate sledećeg čvora (promenljive *i, j, k, l*). Potom se paketi prosleđuju iz novog čvora i proces se ponavlja dok se ne obiđu svi čvorovi. Na ovaj način paket može preći iz čvora sa nižim koordinatama u čvor sa višim, a potom taj čvor može u istom vremenskom slotu tek primljeni paket odmah proslediti sledećem čvoru koji tek treba da dođe na red.

Rešenje navedenog problema predstavlja polje *vremenski slot*. Pomenuli smo da se ovo polje nalazi u paketskoj strukturi i nosi informaciju o rednom broju vremenskog slota u kom je paket generisan ili prosleđen. Ako se vrednost tog polja u paketu poklopi sa trenutnom vrednošću vremenskog slota u kom se prosleđivanje obavlja, paket će biti ignorisan. Uslov za ispitivanje polja *vremenski slot* u paketu dodat je `if` naredbi za ispitivanje popunjenosti bafera.

```
if((*(*(*(pokaz + i)+ j)+ k)+ l)->x >=0
        &&(*(*(*(pokaz + i)+ j)+ k)+ l)->vrem_slot < q){
```

U slučaju da posmatrani bafer nije prazan, potrebno je pronaći paket koji se nalazi na kraju steka bafera. Za potrebe pretrage bafera korišćena je **while** petlja. Uslov petlje ispituje da li je polje pokazivača na sledeći paket u posmatranom paketu različito od **NULL**. Ispunjen uslov znači da posmatrani paket nije i prvi paket u baferu pa se **while** petlja ponovo izvršava. Unutar petlje dva pokazivača na pakete, `prosli` i `temp` se pomeraju duž lanca paketa dok ne dođu do kraja lanca. Kada dođu do kraja lanca `temp` će pokazivati na prvi paket koji je stigao u bafer, a pokazivač `prosli` na drugi paket po redu. Polje `sled` paketa na koji pokazuje pokazivač `prosli` dobija vrednost **NULL**. Na taj način poslednji paket je oslobođen i njegove vrednosti se upisuju u novi paket `tekuci`. Nakon unosa vrednosti trenutnog vremenskog slota, novi paket je spreman za prosleđivanje.

```
temp=(*(*(*(pokaz + i)+ j)+ k)+ l);
prosli =NULL;
while(temp -> sled !=NULL){
      prosli= temp;
      temp= prosli -> sled;
}
tekuci= malloc(sizeof(Paket));
*tekuci =*temp;
tekuci->vrem_slot = q;
```

Ako u baferu postoji samo samo jedan paket, njegove vrednosti se prepisuju u paket `tekuci` koji se prosleđuje. Zatim se u polje x koordinate poslednjeg paketa u baferu upisuje negativna vrednost čime se označava da je bafer prazan.

```
if(prosli !=NULL) prosli->sled =NULL;
else{
      (*(*(*(pokaz + i)+ j)+ k)+ l)->x =-9;
      (*(*(*(pokaz + i)+ j)+ k)+ l)->sled =NULL;
}
```

3.5.2. Prosleđivanje paketa

Nakon što je paket uspešno izdvojen iz bafera i pripremljen za prosleđivanje, počinjesa izvršavanjem druga celina koda funkcije za prosleđivanje paketa. Zadatak ove celine je da odredi čvor kojem treba proslediti izdvojeni paket, zatim da ispita da li je novi čvor odredišni i u zavisnosti od ishoda usmeri paket na odgovarajući port i smesti ga u bafer.

Za potrebe ove celine korišćena je naredba **switch**. Naredba **switch** je zgodna jer omogućava izvršavanje različitih koraka koda u zavisnosti od uslova. Ova mogućnost nam odgovara jer usmeravanje paketa u novom čvoru zavisi od izlaznog porta čvora sa kog je paket prosleđen.

Kao primer, uzećemo paket koji se preuzima sa porta 0. Zadržana je ista numeracija portova koja je primenjena u funkciji za generisanje paketa. U baferu na portu 0 se smeštaju paketi koji će biti prosleđeni po x koordinati u negativnom smeru. Kada paket bude prosleđen u novi čvor, koji nije odredišni, paket se može usmeriti na sve portove osim porta 1. Razlog tome je što na port 1 izlaze paketi koji se šalju u pozitivnom smeru. Ako se paket koji je prosleđen sa porta 0 opet prosledi, ali preko porta 1, vratiće se u početni čvor.

Dakle, `switch` naredba nam omogućava da razlikujemo slučajeve u zavisnosti sa kog porta je paket došao. Kako je navedeno u prethodnom primeru, za svaki izlazni port izvorišnog čvora postoji 5 potencijalnih portova susednog čvora na koje paket može doći.

Proces usmeravanja paketa ka izlaznom portu je isti kao i kod funkcije generisanja paketa. Međutim, postoji bitna razlika jer se određivanje izlaznog porta na čvoru, u koji paket treba da stigne, vrši pre prosleđivanja samog paketa. Tačnije, određuje se destinacija sledećeg skoka paketa, iako prvi skok još nije napravljen. To može dovesti do slučaja da paket bude prosleđen na nepostojeći čvor.

Problem se dešava ako se, npr. u torusu dimenzija 5x5x5, preuzima paket sa porta 0 čvora, čija x koordinataima vrednost 0 i šalje na čvor sa x koordinatom 5, gde je 5 najveća vrednost koordinate x koju čvor može da ima. Prema uslovu prosleđivanja, najkraći put do čvora sa koordinatom 5 je preko porta 0. Međutim, na ovaj način, paket se prosleđuje na čvor sa koordinatom -1, koji ne postoji. Kako bi se sprečio ovaj slučaj, uvedena je naredba `if` čiji uslov će ispitivati da li je koordinata sledećeg čvora manja od 0. Ako je uslov ispunjen, za sledeći čvor se uzima čvor na suprotnom kraju torusa, odnosno čvor čija koordinata ima najveću vrednost.

```
if(i -1<0) crd = x -1;
else crd = i -1;
```

Isti princip važi i za čvor čija x koordinata ima najvišu vrednost, gde bi port sa brojem 1 vodio do nepostojećeg čvora.

```
if(i +1> x-1) crd =0;
else crd = i +1;
```

U zavisnosti od porta, uslov je primenljiv i na y i z koordinatama.

Nakon što je ispunjen uslov za usmeravanje i određen port čvora na koji se prosleđuje paket, smeštanje u bafer se obavlja po poznatoj proceduri. Prvo se ispita da li je bafer prazan. Ako jeste, poslednji paket u baferu (nevažeći paket) uzima vrednosti paketa koji se prosleđuje. Ako nije, generiše se pomoćni paket koji uzima vrednost poslednjeg paketa u lancu. Poslednji paket u lancu, zatim, preuzima vrednost paketa koji se prosleđuje i usmerava pokazivač na pomoćni paket. Na ovaj način lanac paketa je ponovo povezan sa novim paketom na vrhu.

```
if((*(*(*(pokaz + crd)+ j)+ k)+0)->x >=0){
      sledeci  = malloc(sizeof(Paket));
      *sledeci =*(*(*(*(pokaz + crd)+ j)+ k)+0);
      tekuci->sled = sledeci;
}
*(*(*(*(pokaz + crd)+ j)+ k)+0)=*tekuci;
```

Ako nijedan od uslova za usmeravanje nije ispunjen, smatra se da je posmatrani čvor ujedno i odredišni čvor paketa. Paket se šalje na port 6 i ubacuje u bafer na eksternom linku, gde završava svoj put kroz torus.

4. ANALIZA REZULTATA SIMULACIJE

U ovom poglavlju ćemo se baviti analizom rezultata dobijenih prilikom testiranja simulacije. Poredićemo torusne arhitekture različitih dimenzija i njihovo ponašanje u simulaciji prilikom različitih saobraćajnih opterećenja, takođe, biće opisana i kodna realizacija metode prikupljanja podataka na osnovu koje je izvršena analiza rada torusne arhitekture rutera.

4.1. Kodna realizacija metode prikupljanja podataka

Kao što smo pomenuli u prethodnom poglavlju, za potrebe analize ponašanja torusne arhitekture i poređenja torusa različitih dimenzija posmatrani su sledeći parametri:

- maksimalni protok paketa,
- minimalni protok paketa,
- srednji protok paketa,
- maksimalna popunjenost bafera,
- minimalna popunjenost bafera
- srednja popunjenost bafera.

Da bi navedeni parametri mogli biti izmereni potrebno je uvesti brojače, tj. promenljive koje će pamtiti vrednosti i ažurirati ih shodno njihovim promenama tokom simulacije. Brojači se formiraju uz pomoć funkcije `gen_niz_br` koja generiše četvorodimenzionalne nizove kod kojih prve tri dimenzije određuju čvor, a četvrta dimenzija određuje port na kome se vrši merenje. Proces kojim funkcija `gen_niz_br` formira brojače je isti kao i prilikom kreiranja torusa sa razlikom u polju niza koje je sad tipa *integer* umesto *Paket*.

```
int****gen_niz_br(int x,int y,int z){
	int i, j, k, l;
	int****protok;
	protok = malloc(x *sizeof(int***));
	for(i =0; i < x; i++){
		*(protok + i)= malloc(y *sizeof(int**));
		for(j =0; j < y; j++){
			*(*(protok + i)+ j)= malloc(z *sizeof(int*));
			for(k =0; k < z; k++){
				*(*(*(protok + i)+ j)+ k)= malloc(7*sizeof(int));
				for(l =0; l <7; l++){
					protok[i][j][k][l]=0;
				}
			}
		}
	}
	return protok;
	}
```

Za potrebe proračuna u simulaciji definisana su tri tipa takvih brojača:

```
protok = gen_niz_br(x, y, z);
bafer = gen_niz_br(x, y, z);
count_gen = gen_niz_br(x, y, z);
```

Zadatak brojača `protok` je da pamti broj paketa koji su prošli preko određenog linka. Kako bi se brojač `protok` inkrementirao samo kad se paket pošalje na sledeći čvor, on mora biti smešten unutar funkcije `prosledi` ispod uslova koji proverava da li ima paketa za prosleđivanje unutar bafera. Nakon svakog uspešnog prosleđivanja brojač se inkrementira i pamti broj paketa koji je prosleđen.

Brojač `bafer` ima ulogu da meri broj paketa koji se nalazi u baferu na svakom portu čvora. Kako se popunjenost bafera menja prilikom generisanja paketa u čvorovima kao i prilikom prosleđivanja paketa, brojač će biti smešten u funkcijama `generisi_pak` i `prosledi`. Unutar funkcije `generisi_pak` brojač se smešta u svaki od mogućih ishoda uslova koji ispituje kom izlaznom portu je generisani paket namenjen. U zavisnosti od porta inkrementira se njemu odgovarajući brojač. Kod funkcije `prosledi` brojač `bafer` se smešta na isto mesto kao i brojač `protok` sa tim što se dekrementira, jer uspešno prosleđivanje znači da je paket uzet iz bafera. Takođe, brojač se postavlja i na moguće ishode uslova koji ispituje kom čvoru se paket prosleđuje, gde se inkrementira jer paket ulazi u bafer novog čvora.

Brojač `count_gen` je zadužen da meri koliko se paketa generisalo u svakom čvoru i koliko je od generisanih paketa prosleđeno na svaki port čvora. Brojač je smešten u funkciju `generisi_pak` ispod uslova koji ispituje da li je paket generisan i u slučaju da jeste inkrementira se. Informacija o ukupnom broju generisanih paketa čvora se čuva u polju niza koji odgovara eksternom linku čvora. Kako čvor ne generiše pakete za svoj eksterni link, prethodno rešenje je moguće izvesti. Ostala polja brojača se inkrementiraju u zavisnosti od porta na koji se paket usmerava. Iz tog razloga brojač se smešta i u ishode uslova koji ispituje na koji port treba usmeriti paket.

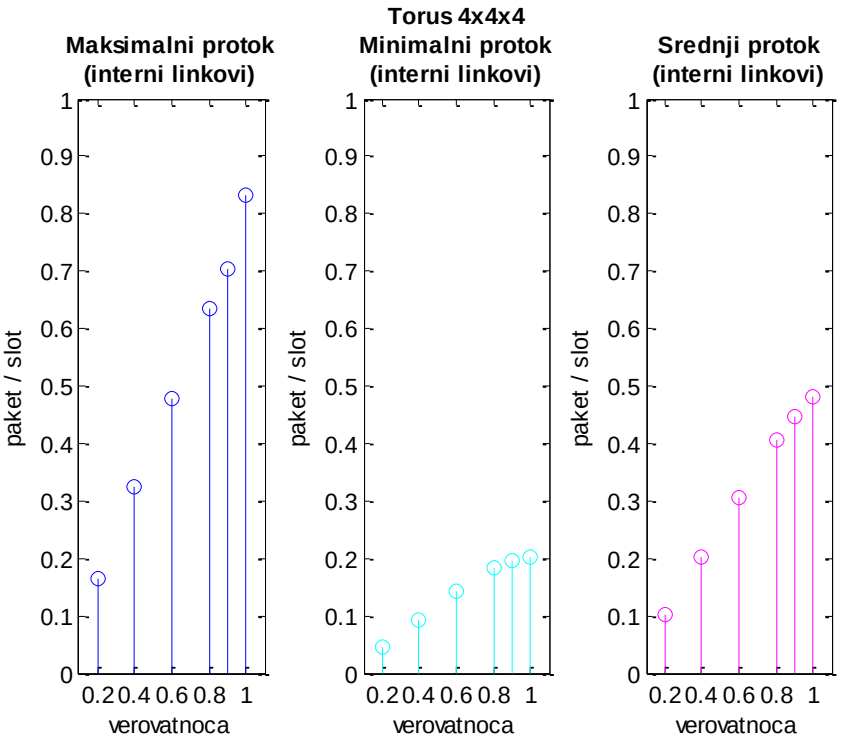
4.2. Analiza rezulata dobijenih u simulaciji

Nakon što smo opisali način prikupljanja podataka, bavićemo se analizom dobijenih rezultata pri različitim ulaznim parametrima. Za potrebe rada analiziraćemo toruse dimenzija 4x4x4, 5x5x5 i 8x8x8 jer torusi sa ovim dimenzijama predstavljaju strukture sa 64, 125 i 512 portova, tj. poredimo strukture od kojih svaka sledeća ima približno dvostruko, odnosno četvorostruko više portova. Takođe, posmatraćemo kako se svaka od navedenih struktura ponaša pod različitim saobraćajnim opterećenjem.

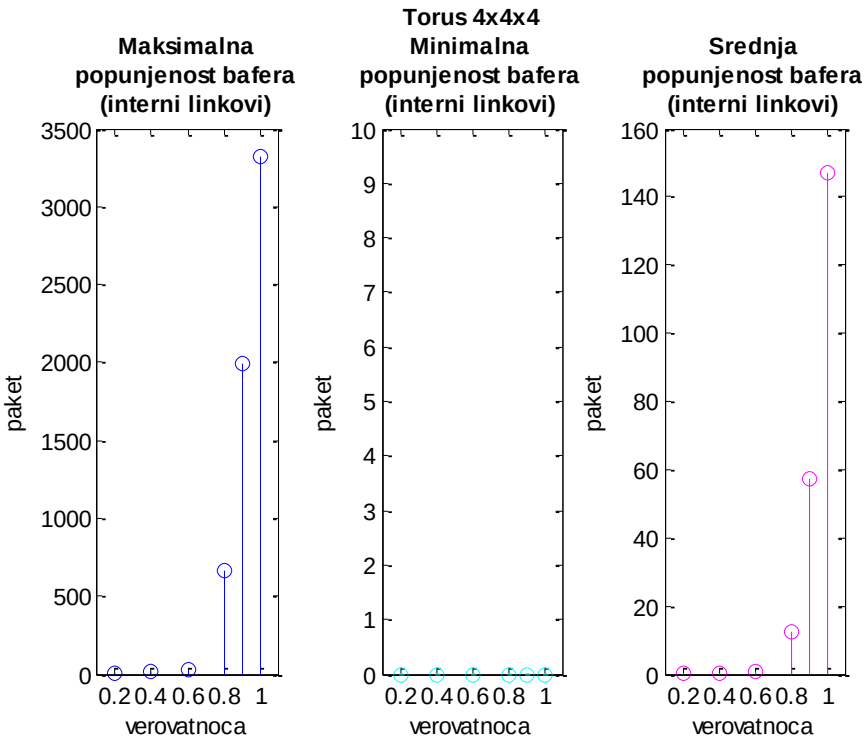
4.2.1. Ponašanje torusa pod različitim opterećenjem

Prilikom testiranja simulacije analizirane su sve tri navedene dimenzije torusa. Trajanje simulacije je ograničeno na 10,000 vremenskih slotova. Opterećenja na čvorovima, tj. verovatnoće generisanja paketa pri kojima je vršena simulacija su 0.2, 0.4, 0.6, 0.8, 0.9 i 1. Pored navedenih uslova, napravljena je razlika između podataka dobijenih za interne i eksterne linkove, kao i za sve linkove zbirno. Sledeći grafikoni prikazuju maksimalni, minimalni i srednji protok kao i maksimalnu, minimalnu i srednju popunjenost bafera.

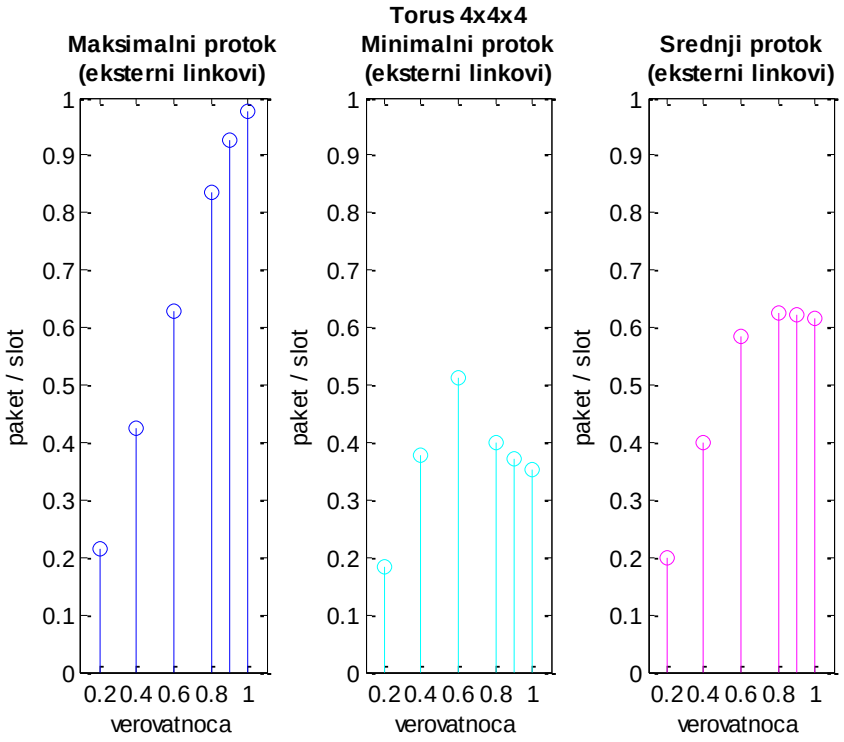
i) Torus dimenzija 4x4x4



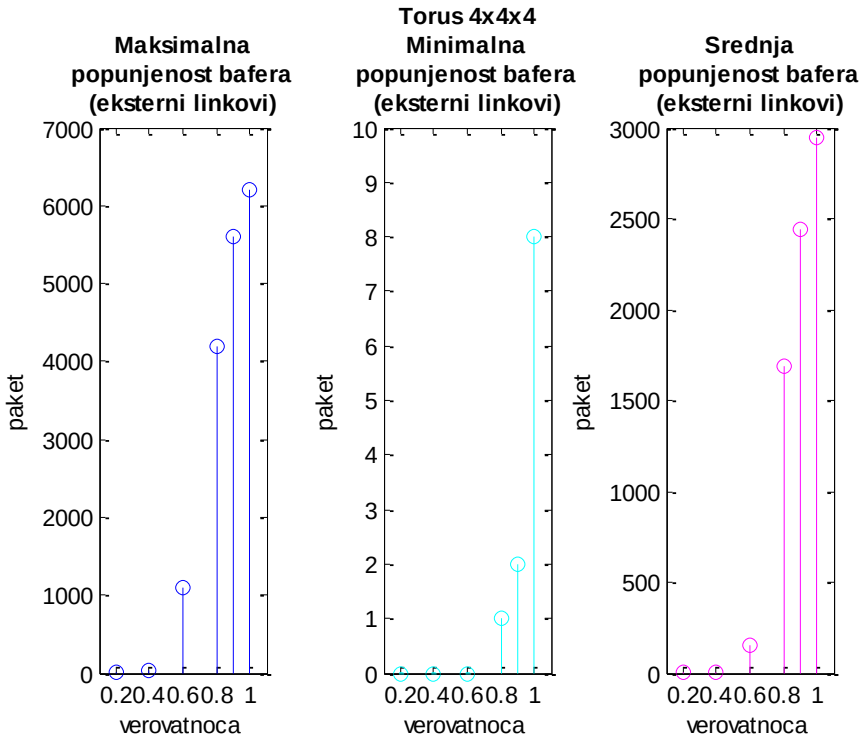
Grafik 4.2.1.1 Maksimalni, minimalni i srednji protok na internim linkovima za torus dimenzija 4x4x4.



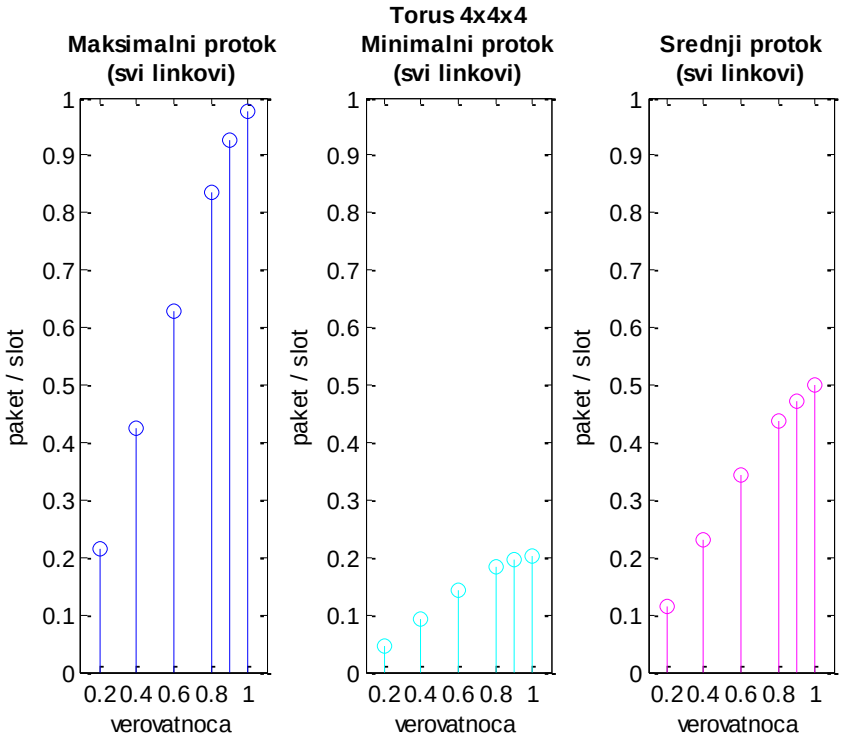
Grafik 4.2.1.2 Maksimalna, minimalna i srednja popunjenost bafera na internim linkovima za torus dimenzija 4x4x4.



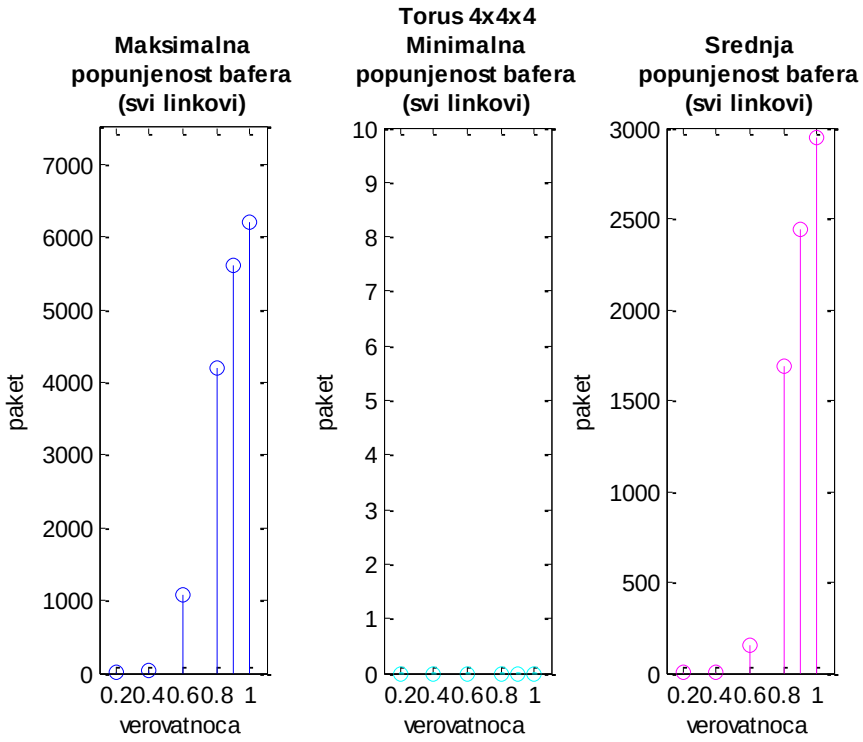
Grafik 4.2.1.3 Maksimalni, minimalni i srednji protok na eksternim linkovima za torus dimenzija 4x4x4.



Grafik 4.2.1.4 Maksimalna, minimalna i srednja popunjenost bafera na eksternim linkovima za torus dimenzija 4x4x4.

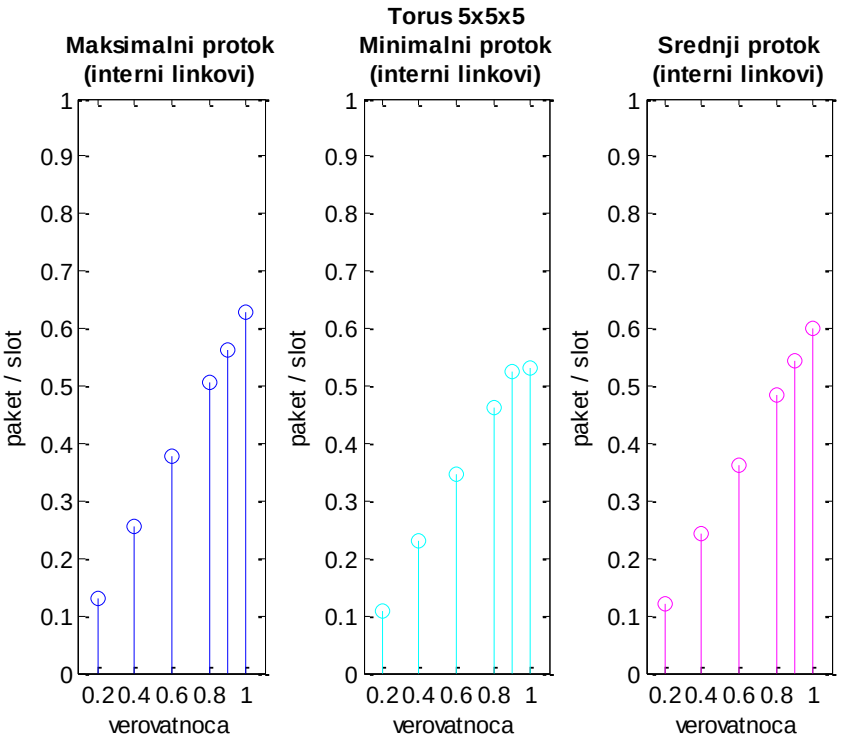


Grafik 4.2.1.5 Maksimalni, minimalni i srednji protok na svim linkovima za torus dimenzija 4x4x4.

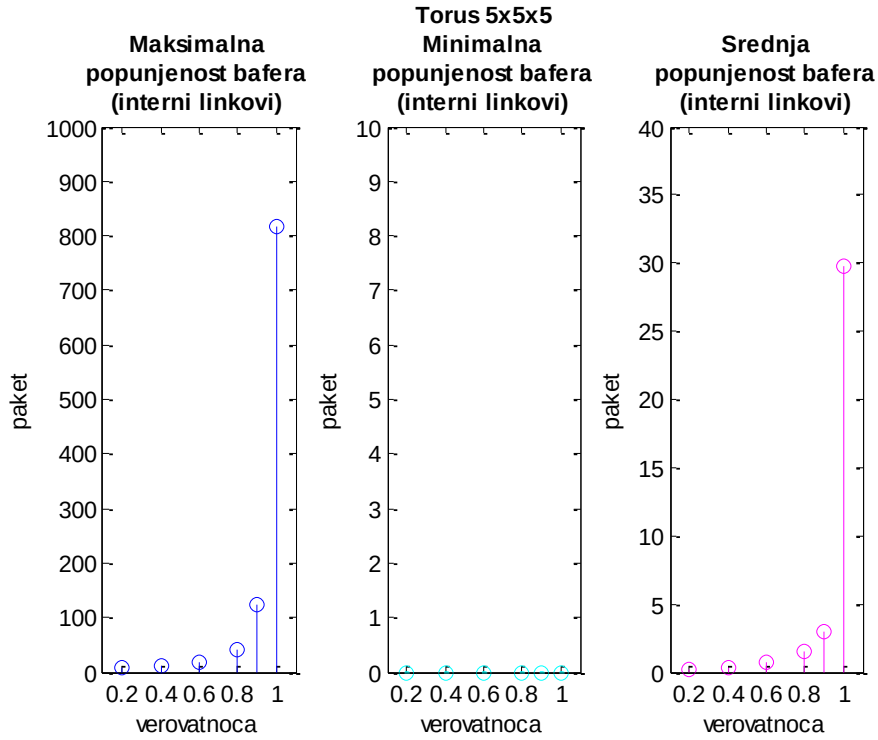


Grafik 4.2.1.6 Maksimalna, minimalna i srednja popunjenost bafera na svim linkovima za torus dimenzija 4x4x4.

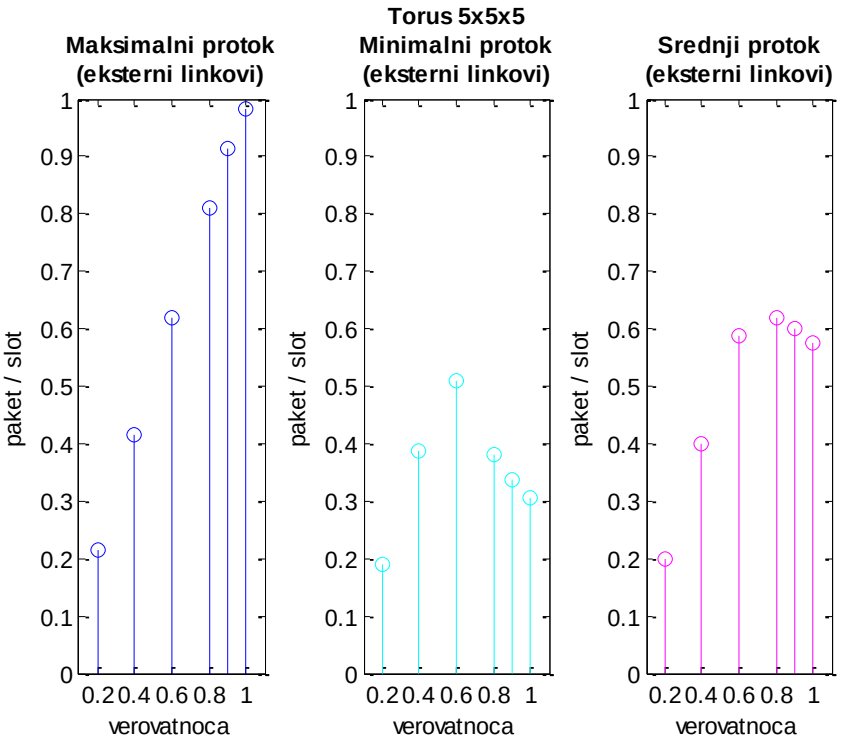
ii) Torus dimenzija 5x5x5



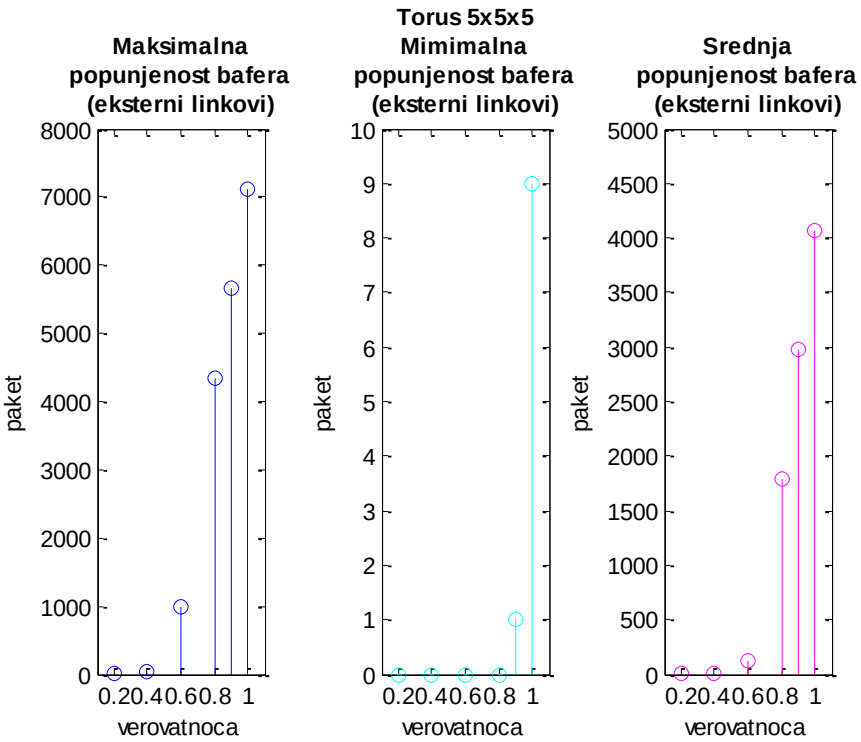
Grafik 4.2.1.7 Maksimalni, minimalni i srednji protok na internim linkovima za torus dimenzija 5x5x5.



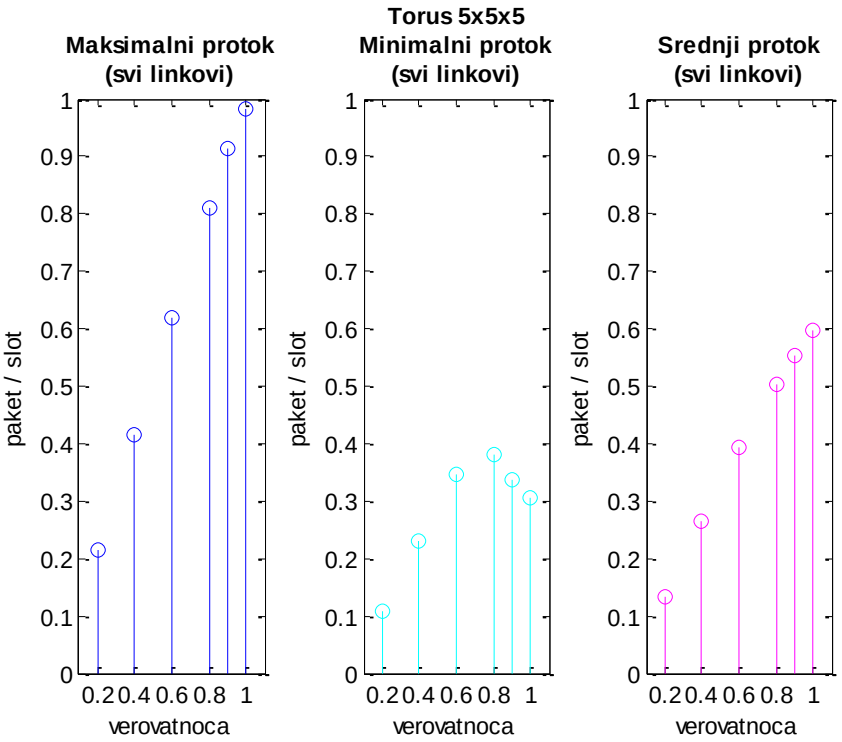
Grafik 4.2.1.8 Maksimalna, minimalna i srednja popunjenost bafera na internim linkovima za torus dimenzija 5x5x5.



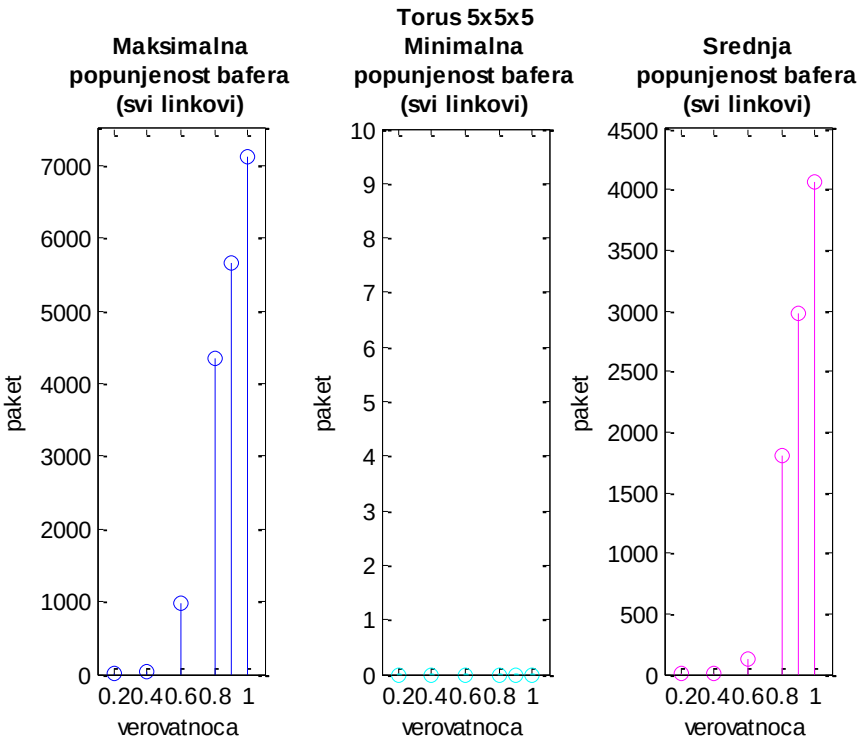
Grafik 4.2.1.9 Maksimalni, minimalni i srednji protok na eksternim linkovima za torus dimenzija 5x5x5.



Grafik 4.2.1.10 Maksimalna, minimalna i srednja popunjenost bafera na eksternim linkovima za torus dimenzija 5x5x5.

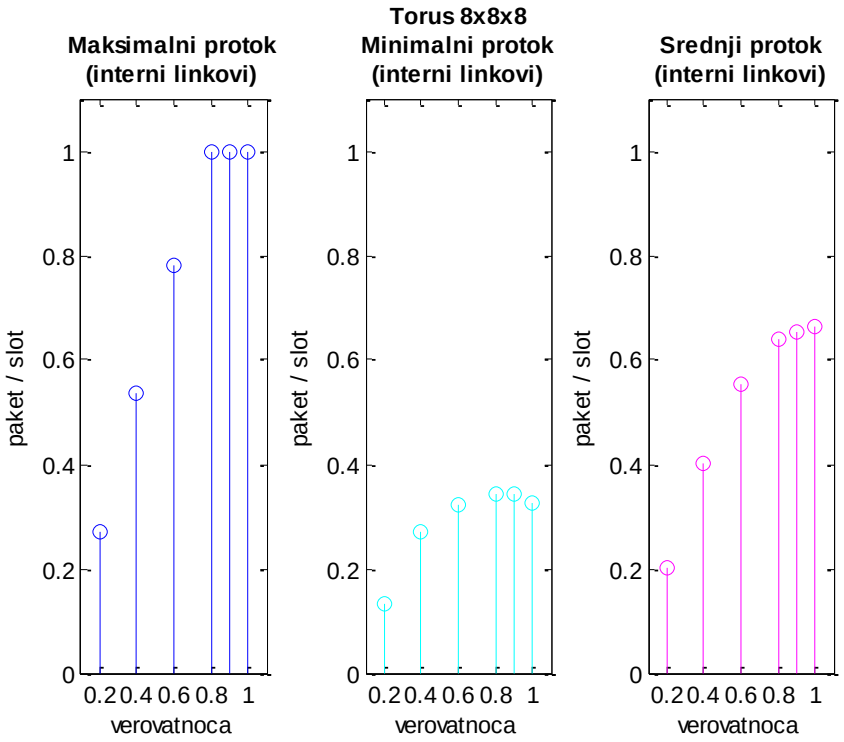


Grafik 4.2.1.11 Maksimalni, minimalni i srednji protok na svim linkovima za torus dimenzija 5x5x5.

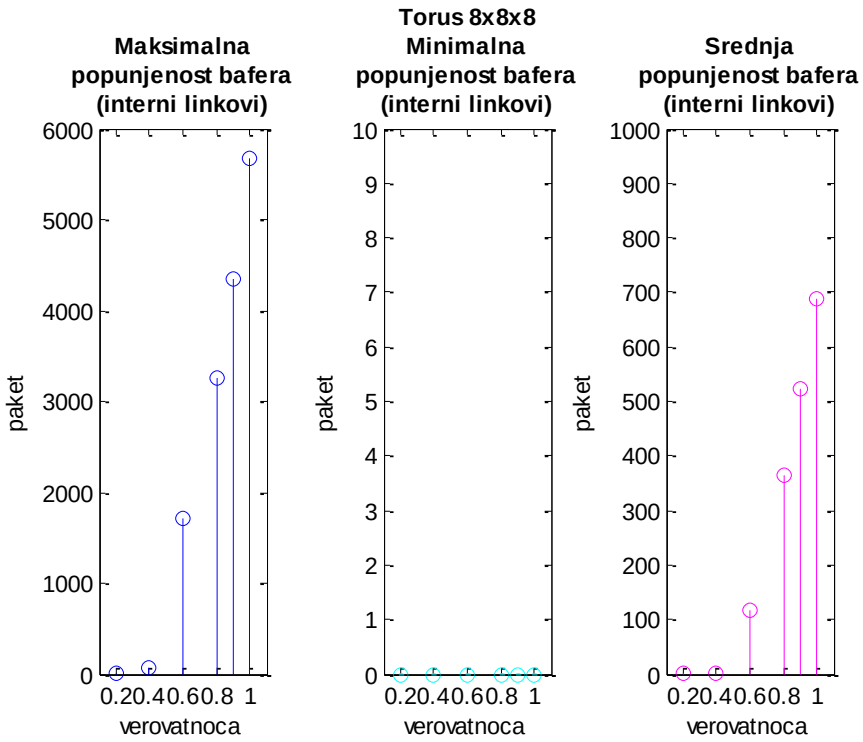


Grafik 4.2.1.12 Maksimalna, minimalna i srednja popunjenost bafera na svim linkovima za torus dimenzija 5x5x5.

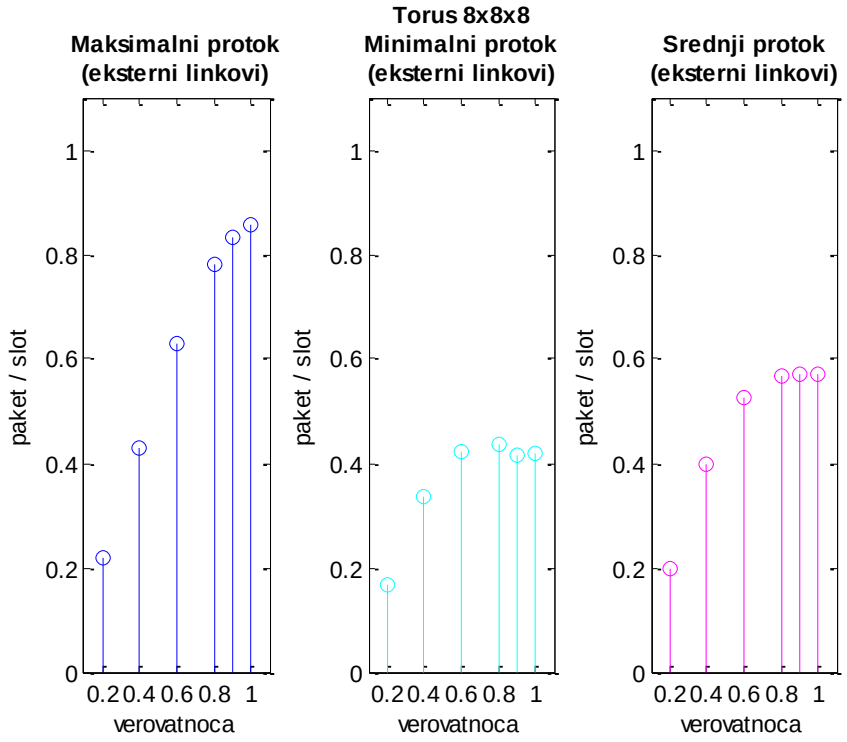
iii) Torus dimenzija 8x8x8



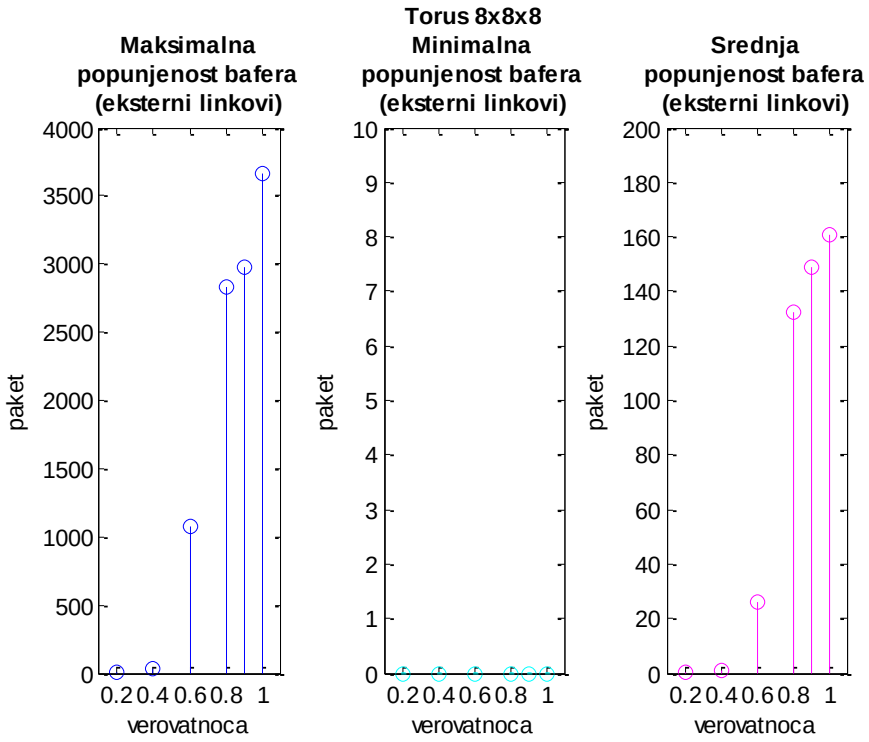
Grafik 4.2.1.13 Maksimalni, minimalni i srednji protok na internim linkovima za torus dimenzija 8x8x8.



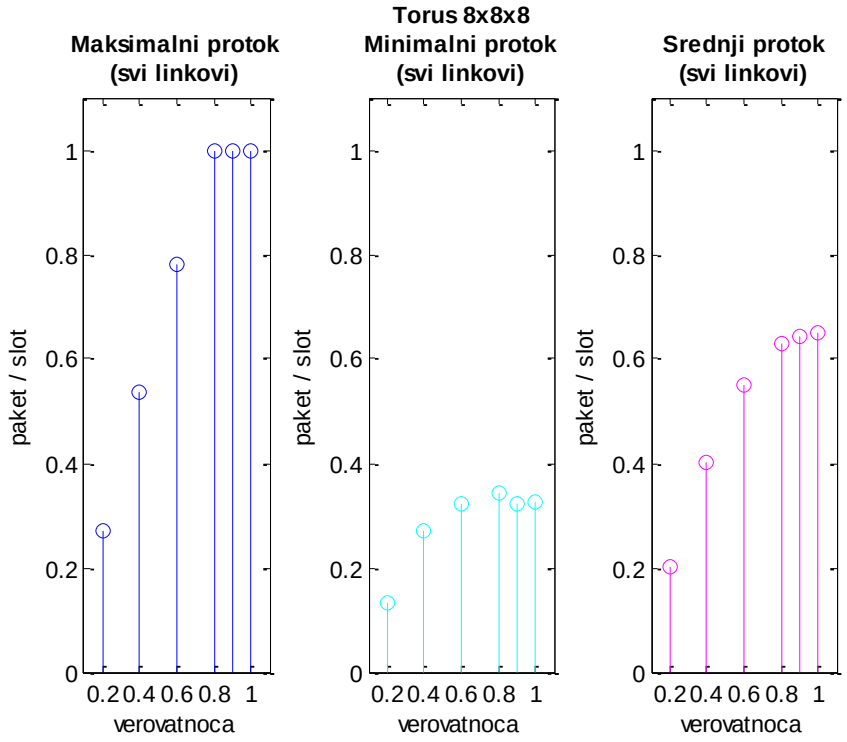
Grafik 4.2.1.14 Maksimalna, minimalna i srednja popunjenost bafera na internim linkovima za torus dimenzija 8x8x8.



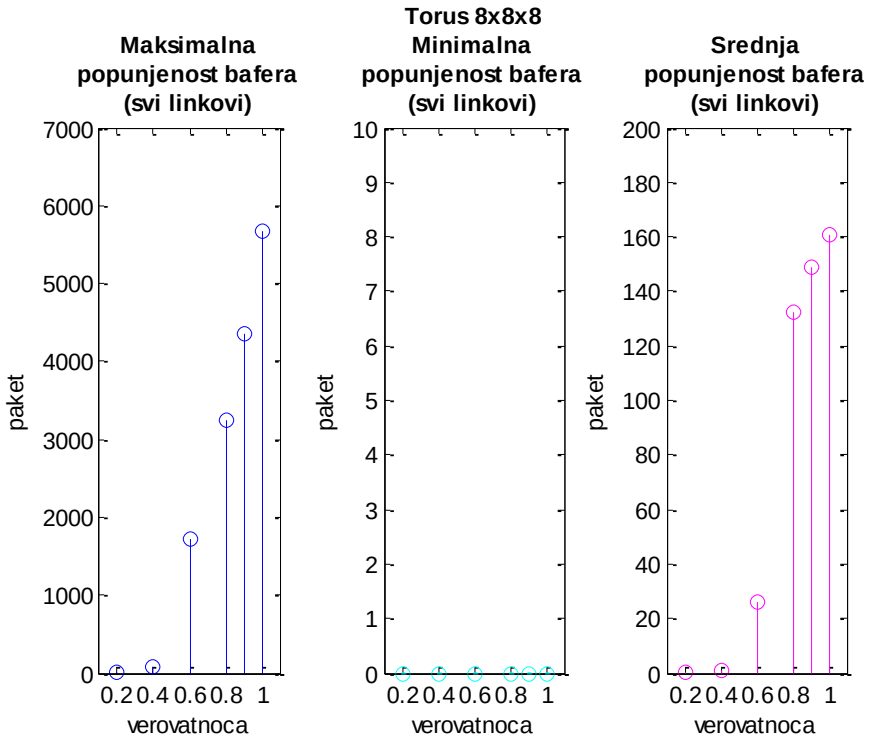
Grafik 4.2.1.15 Maksimalni, minimalni i srednji protok na eksternim linkovima za torus dimenzija 8x8x8.



Grafik 4.2.1.16 Maksimalna, minimalna i srednja popunjenost bafera na eksternim linkovima za torus dimenzija 8x8x8.



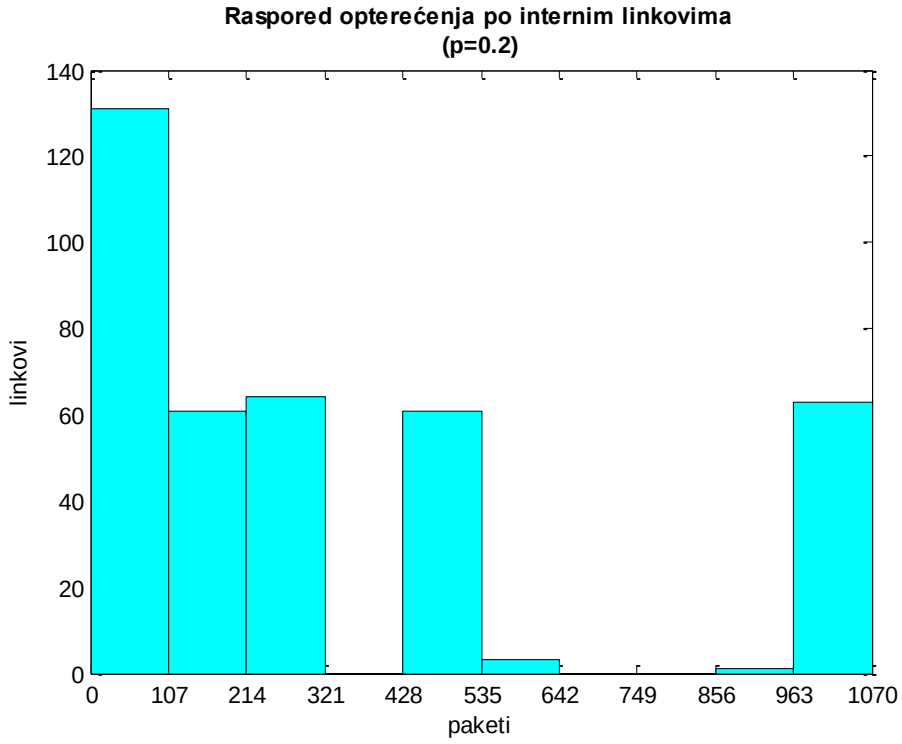
Grafik 4.2.1.17 Maksimalni, minimalni i srednji protok na svim linkovima za torus dimenzija 8x8x8.



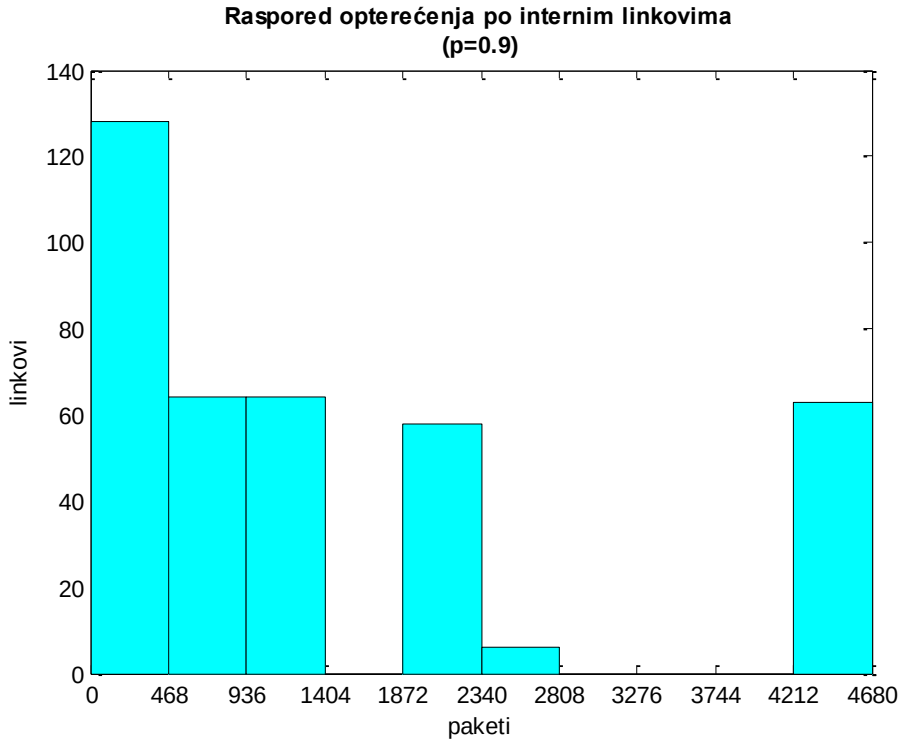
Grafik 4.2.1.18 Maksimalna, minimalna i srednja popunjenost bafera na svim linkovima za torus dimenzija 8x8x8.

Prvo ćemo posmatrati grafike dobijene za torus dimenzija 4x4x4. Kao što možemo primetiti, povećanjem opterećenja protok po internim linkovima približno linearno raste što se i moglo očekivati jer se povećava broj paketa koji se prosleđuju (Grafik 4.2.1.1). Takođe, na velikim opterećenjima dolazi i do značajnog opterećenja bafera. Što se eksternih linkova tiče, primećuje se da na velikim opterećenjima srednji protok stagnira (Grafik 4.2.1.3). Razlog tome je što eksterni linkovi postaju preopterećeni usled velikog broja paketa što pokazuje i srednja popunjenost bafera koja za opterećenje p=1 beleži preko 2,900 paketa po baferu dok se na nekim baferima meri maksimalna popunjenost preko 6,000 paketa (Grafik 4.2.1.4).

Posmatrajući grafike možemo primetiti da postoji veliko odstupanje minimalnih i maksimalnih vrednosti od srednje vrednosti. Dolazimo do pretpostavke da prilikom generisanja paketa određeni portovi su više opterećeni nego drugi što za posledicu ima pomenuta odstupanja. U tu svrhu posmatraćemo dva histograma koji pokazuju opterećenost linkova pri malim, odnosno velikim opterećenjima za torus dimenzija 4x4x4. Na *x* koordinati grafika očitavamo broj paketa koji su došli na link, u jednakim opsezima dok *y* koordinata pokazuje koliko linkova spada u isti opseg opterećenja.



Grafik 4.2.1.19 Raspored opterećenja po linkovima torusa dimenzija 4x4x4 pri malim opterećenjima.



Grafik 4.2.1.19 Raspored opterećenja po linkovima torusa dimeznija 4x4x4 pri velikim opterećenjima.

Prikazani grafici pokazuju da linkovi zaista nisu podjednako opterećeni. Primećujemo da najveći broj linkova nije prenosio pakete ili je bio slabo opterećen. Zanimljivo je da skoro šestina ukupnog broja linkova upada u poslednji opseg koji predstavlja linkove sa najvećim opterećenjem. Upoređujući grafike za velika i mala opterećenja ne uočavaju se velike razlike u samom obliku grafika dok se broj paketa po opsegu povećava shodno povećanju opterećenja. Slični rezultati se dobijaju i za toruse dimenzija 5x5x5 i 8x8x8.

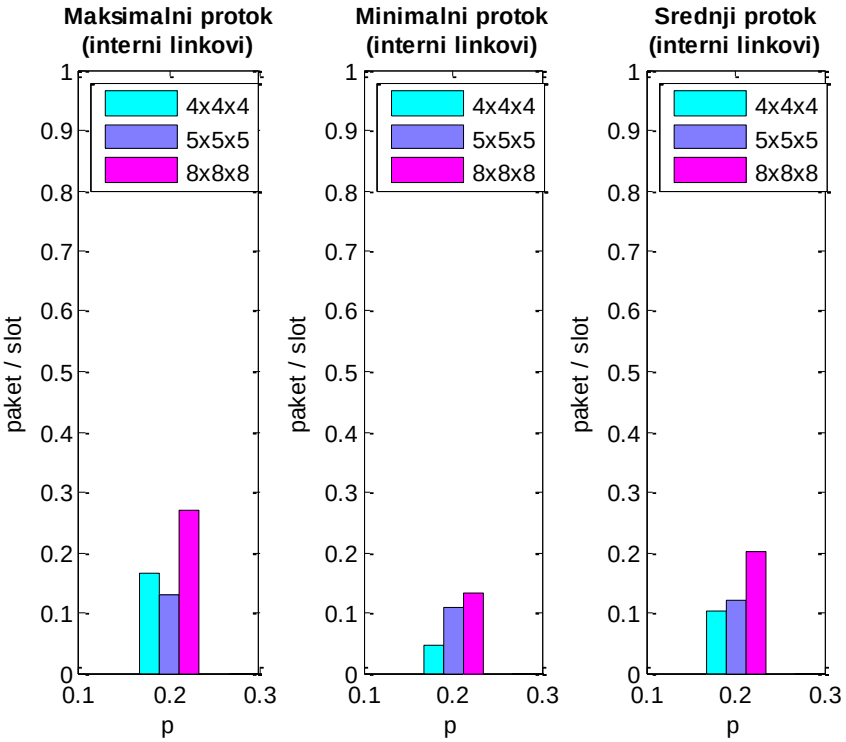
Na grafiku protoka po internim linkovima (Grafik 4.2.1.7), koji odgovara torusu dimenzija 5x5x5, dobijamo približno linearan porast protoka sa tim što su razlike između minimalne, maksimalne i srednje vrednosti znatno manje što znači da imamo pravilno raspoređen protok po svim internim linkovima. To se može primetiti i po popunjenosti bafera (Grafik 4.2.1.8) gde se veliki broj paketa beleži tek na najvećem opterećenju (p=1). Na eksternim linkovima se zapaža sličan problem kao kod 4x4x4, tj. javlja se blagi pad srednjeg protoka na velikim opterećenjima dok popunjenost bafera značajno raste.

Kod torusa dimenzija 8x8x8 beleže se maksimalni mogući protoci od 1 paket/slot na internim linkovima pri najvećim opterećenjima (Grafik 4.2.1.13). Takođe, primećuju se veće razlike između maksimalne, minimalne i srednje vrednosti. Razlog tome su veća opterećenja na pojedinim internim linkovima što se može primetiti na Grafiku 4.2.1.14 gde se značajne popunjenosti bafera beleže već pri srednjem opterećenju. Sa druge strane, na eksternim linkovima se beleže niži protoci dok je i srednja popunjenost bafera mala, oko 160 paketa (Grafik 4.2.1.16).

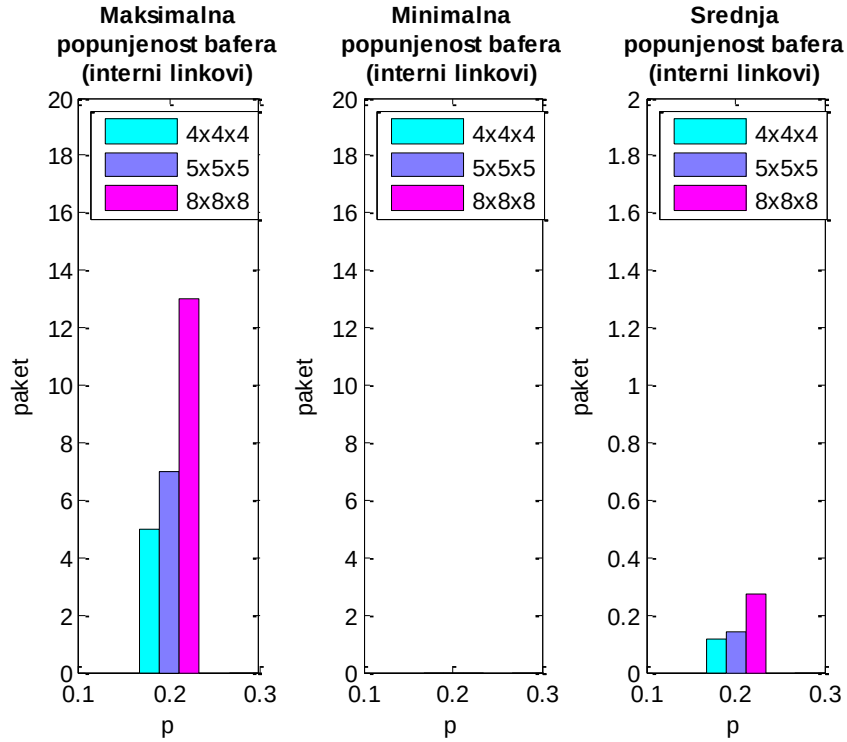
4.2.2. Poređenje torusa različitih dimenzija

Sledeći grafikoni prikazuju ponašanja torusa različitih dimenzija pri opterećenjima 0.2 (malo opterećenje), 0.6 (srednje opterećenje), 0.9 i 1 (veliko opterećenje).

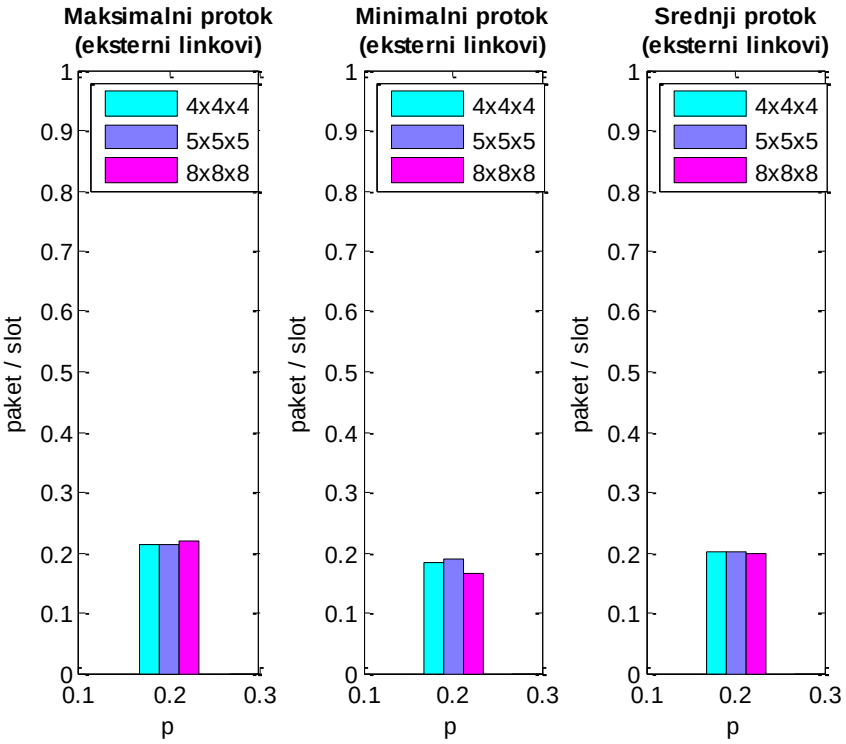
i) Malo opterećenje (p = 0.2)



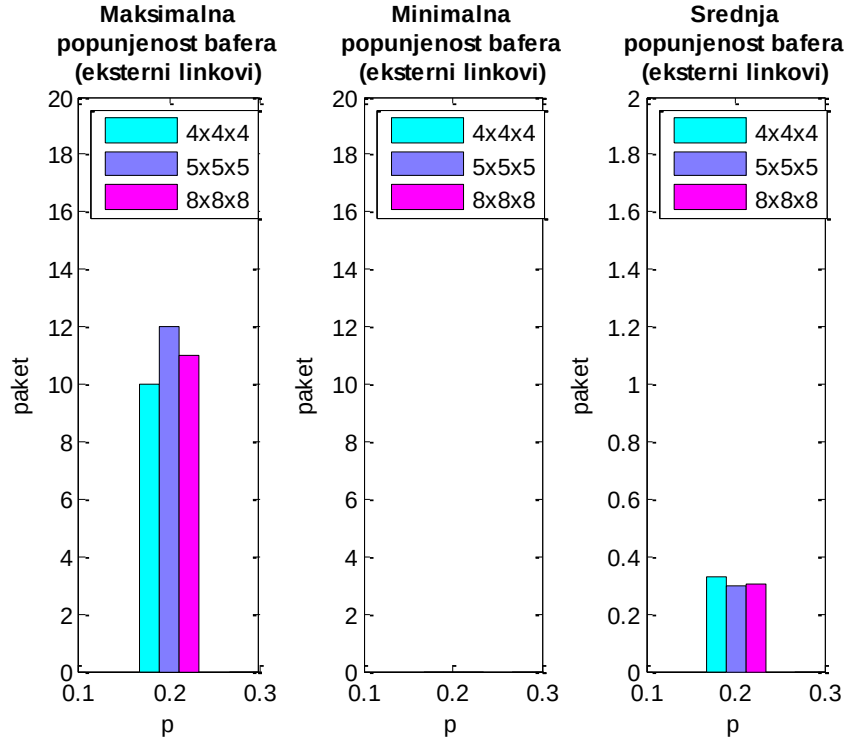
Grafik 4.2.2.1 Maksimalni, minimalni i srednji protok na internim linkovima torusa pri malom opterećenju.



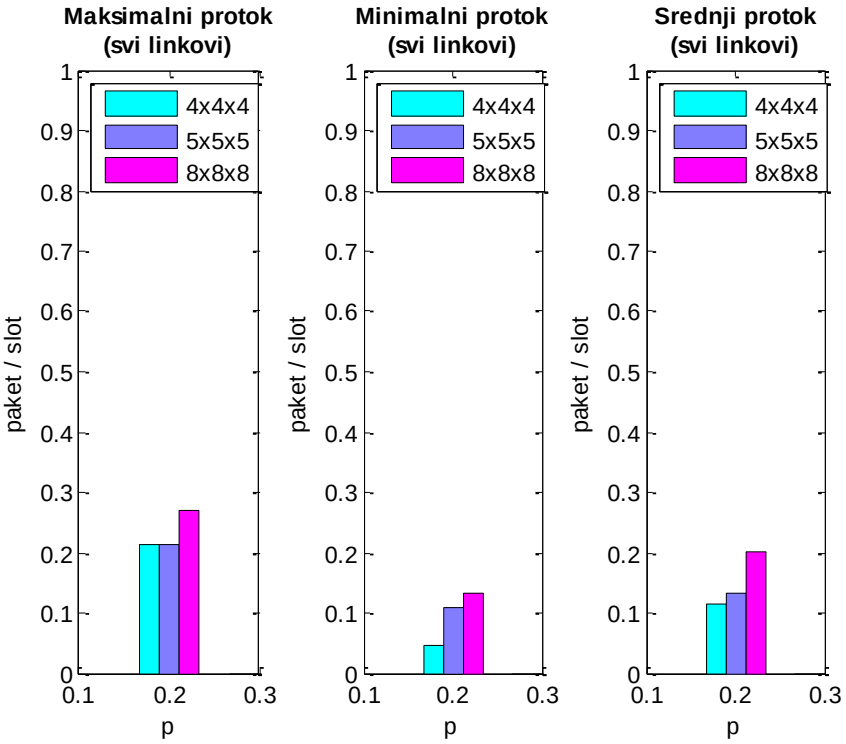
Grafik 4.2.2.2 Maksimalna, minimalna i srednja popunjenost bafera na internim linkovima torusa pri malom opterećenju.



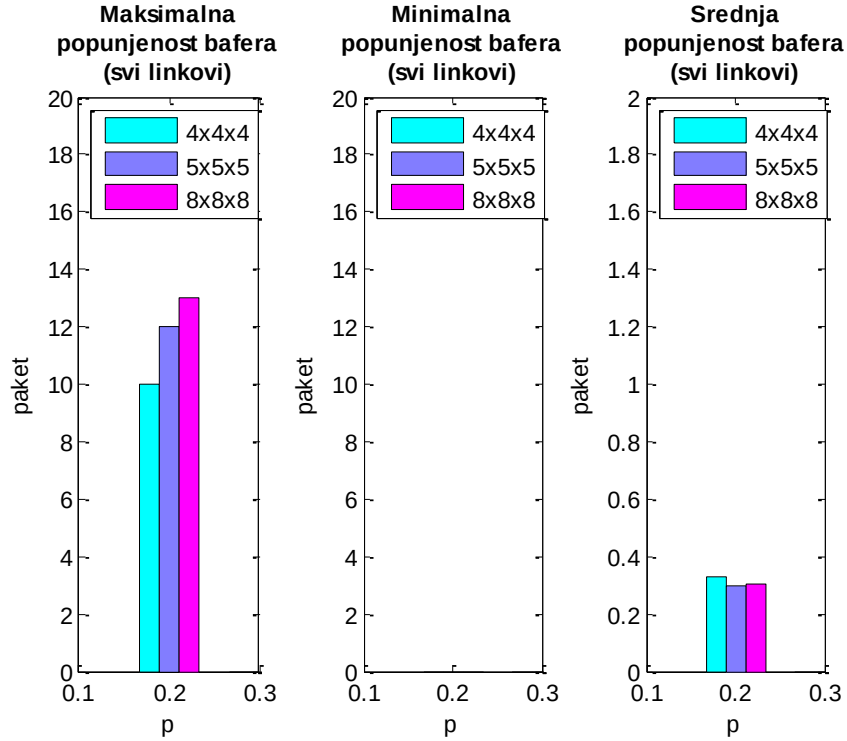
Grafik 4.2.2.3 Maksimalni, minimalni i srednji protok na eksternim linkovima torusa pri malom opterećenju.



Grafik 4.2.2.4 Maksimalna, minimalna i srednja popunjenost bafera na eksternim linkovima torusa pri malom opterećenju.

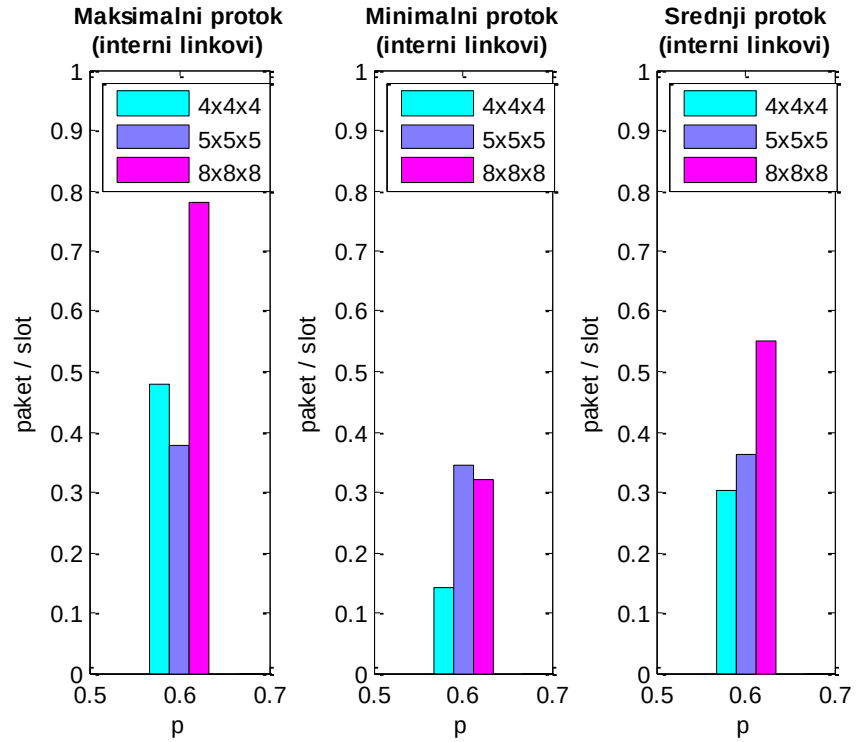


Grafik 4.2.2.5 Maksimalni, minimalni i srednji protok na svim linkovima torusa pri malom opterećenju.

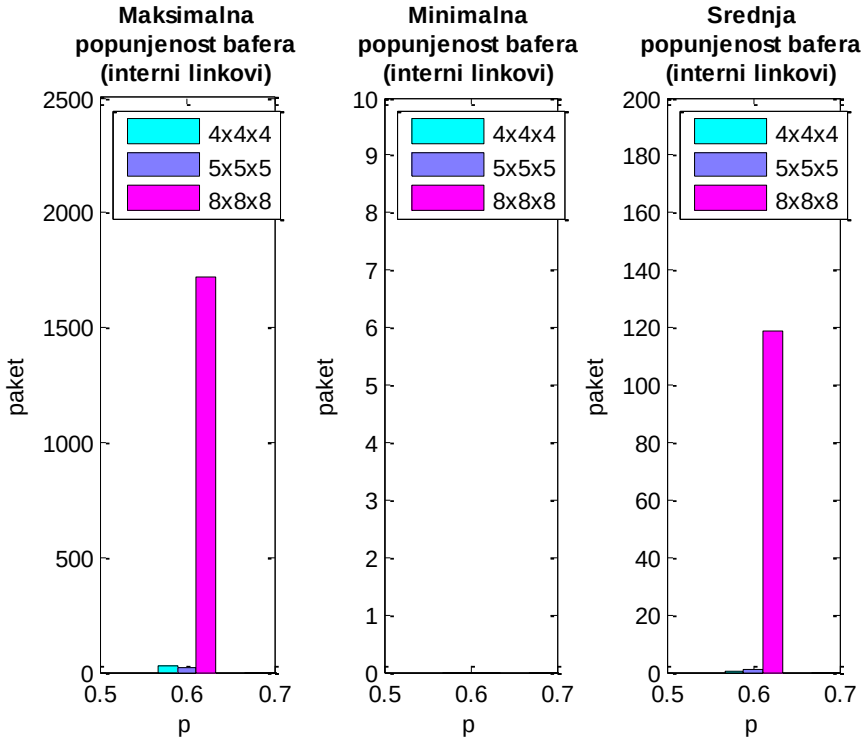


Grafik 4.2.2.6 Maksimalna, minimalna i srednja popunjenost bafera na svim linkovima torusa pri malom opterećenju.

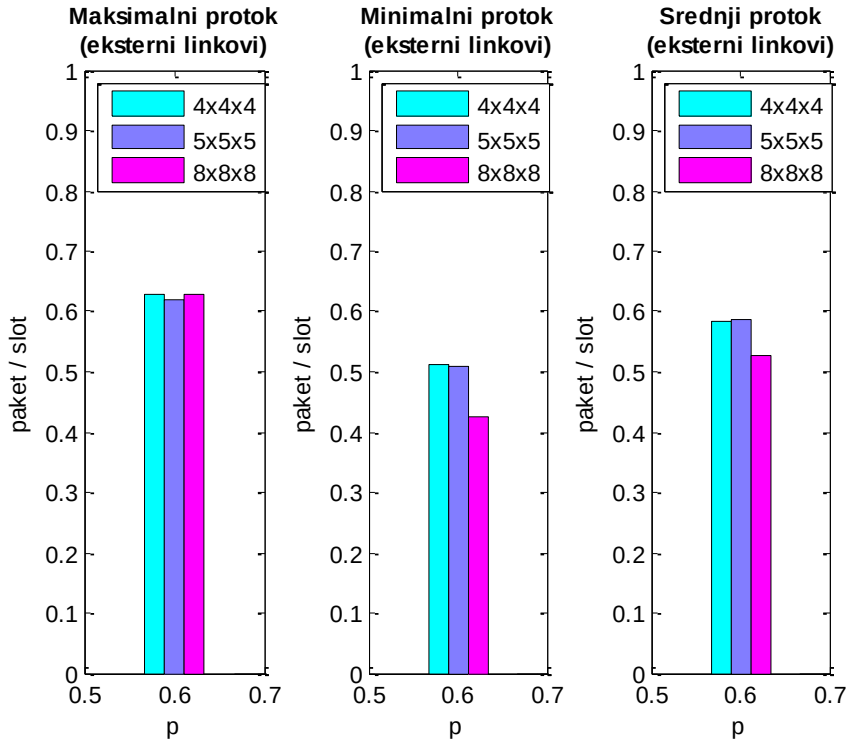
ii) Srednje opterećenje (p = 0.6)



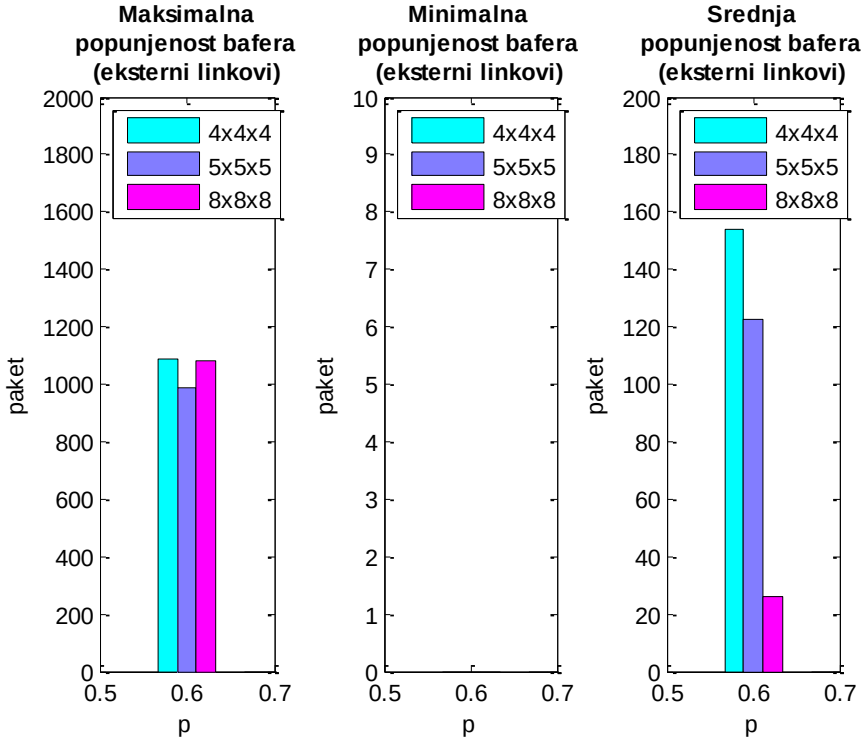
Grafik 4.2.2.7 Maksimalni, minimalni i srednji protok na internim linkovima torusa pri srednjem opterećenju.



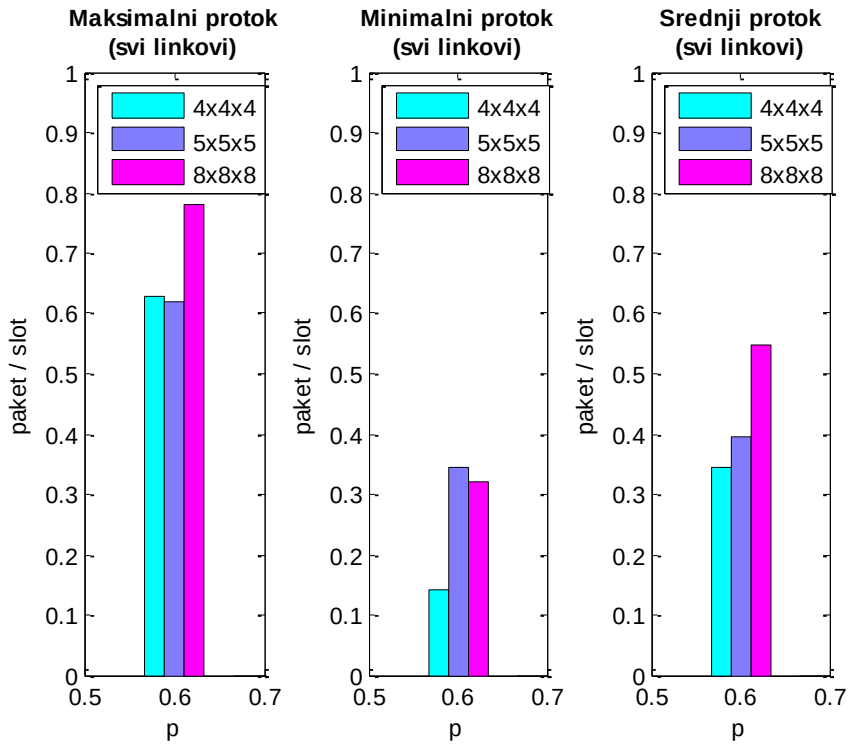
Grafik 4.2.2.8 Maksimalna, minimalna i srednja popunjenost bafera na internim linkovima torusa pri srednjem opterećenju.



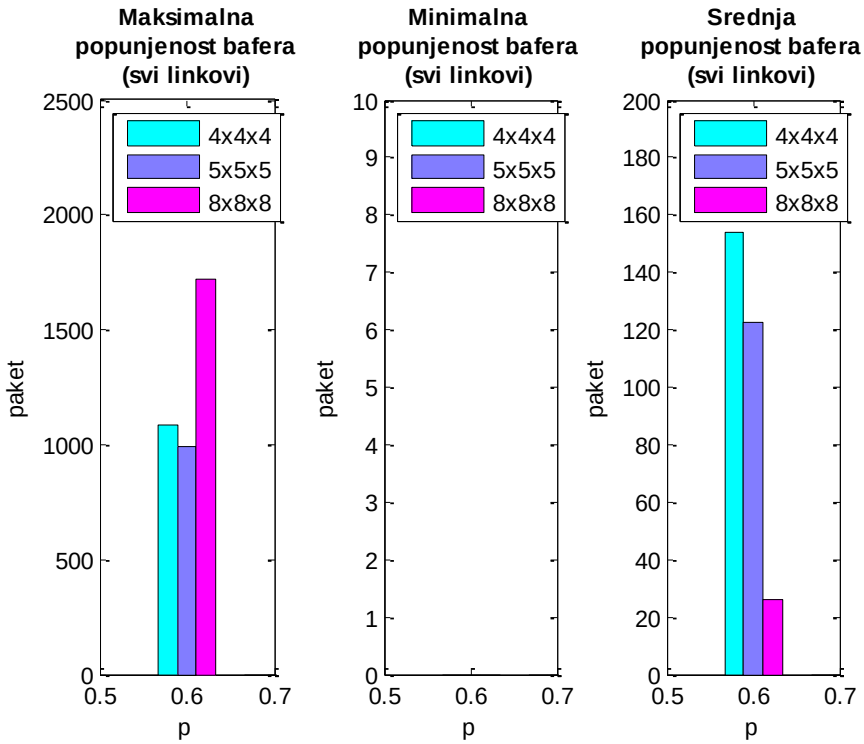
Grafik 4.2.2.9 Maksimalni, minimalni i srednji protok na eksternim linkovima torusa pri srednjem opterećenju.



Grafik 4.2.2.10 Maksimalna, minimalna i srednja popunjenost bafera na eksternim linkovima torusa pri srednjem opterećenju.

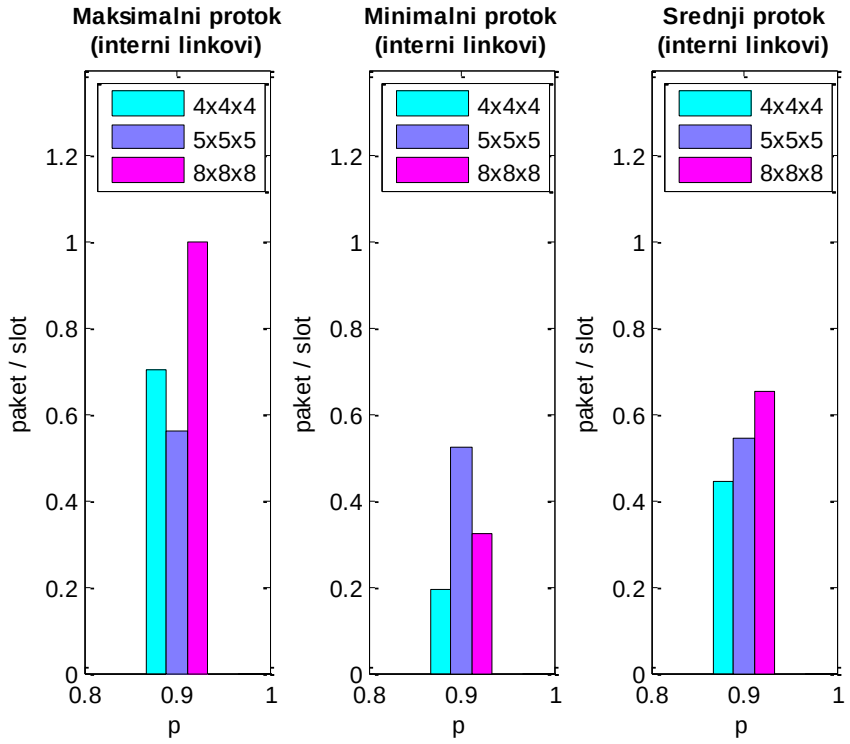


Grafik 4.2.2.11 Maksimalni, minimalni i srednji protok na svim linkovima torusa pri srednjem opterećenju.

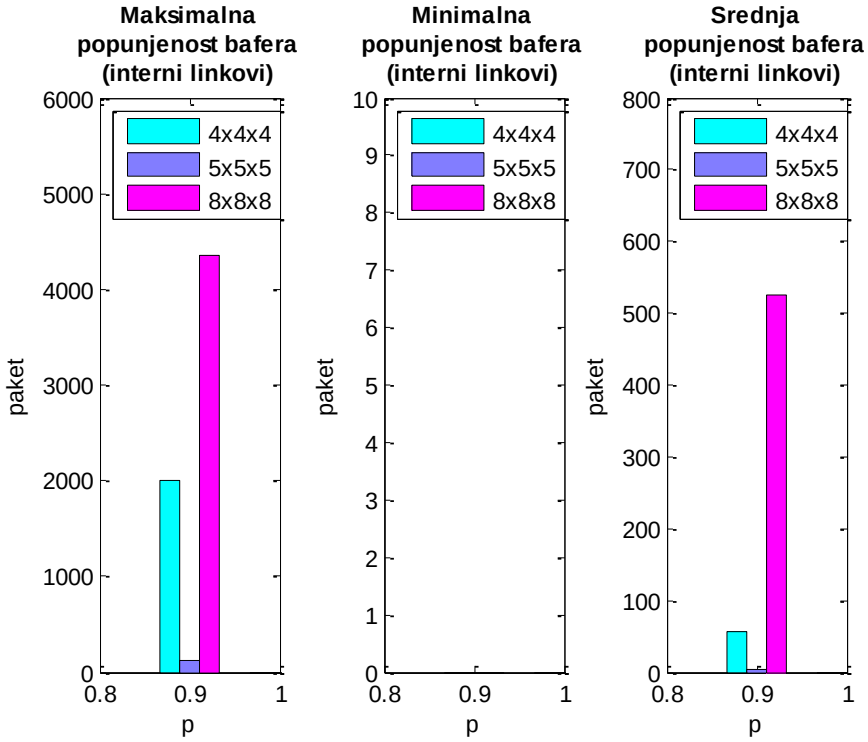


Grafik 4.2.2.12 Maksimalna, minimalna i srednja popunjenost bafera na svim linkovima torusa pri srednjem opterećenju.

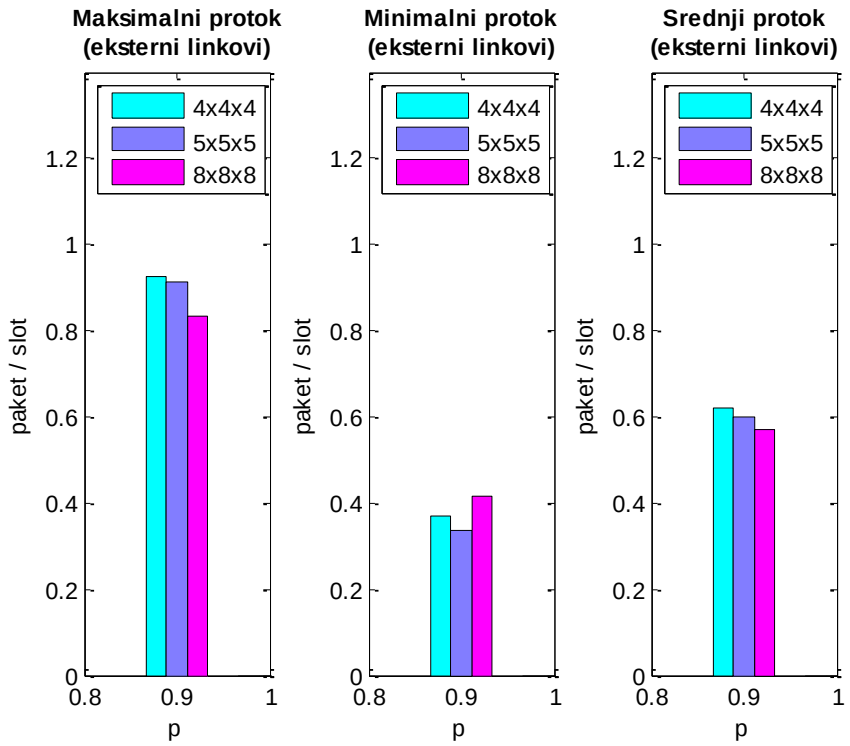
iii) Veliko opterećenje (p = 0.9)



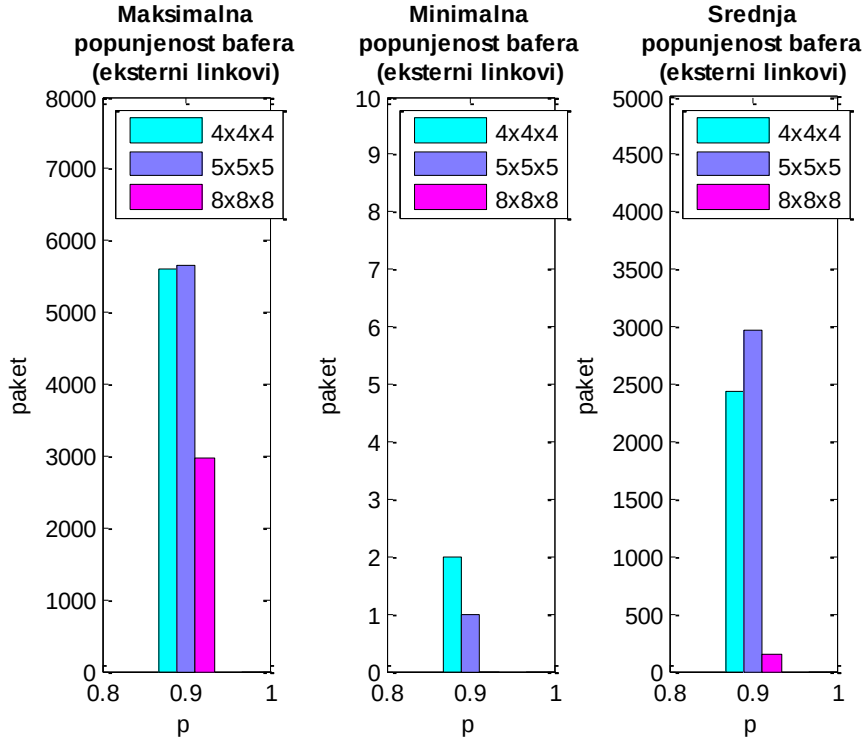
Grafik 4.2.2.13 Maksimalni, minimalni i srednji protok na internim linkovima torusa pri velikom opterećenju.



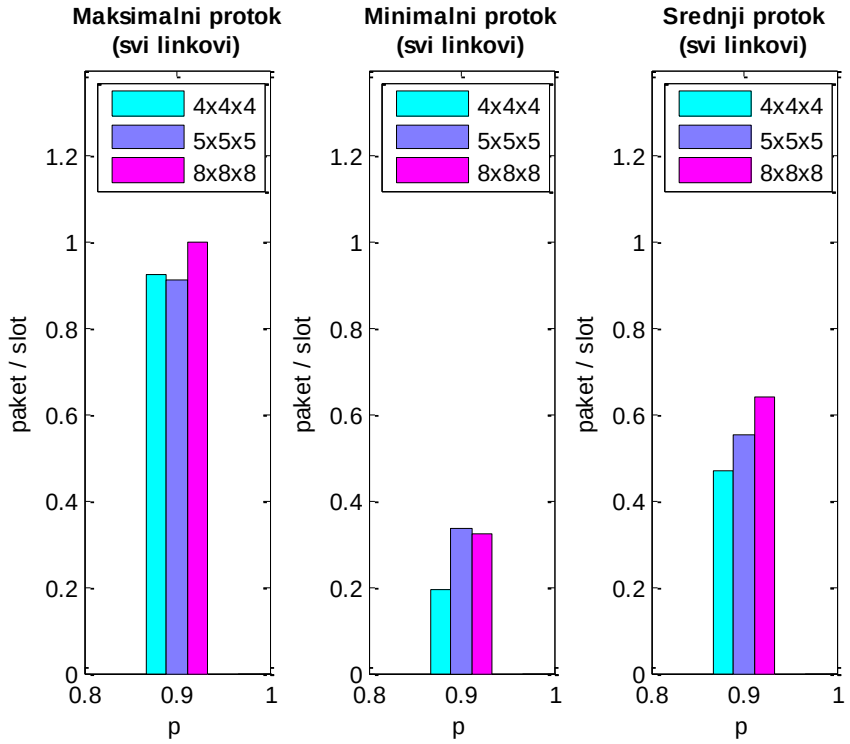
Grafik 4.2.2.14 Maksimalna, minimalna i srednja popunjenost bafera na internim linkovima torusa pri velikom opterećenju.



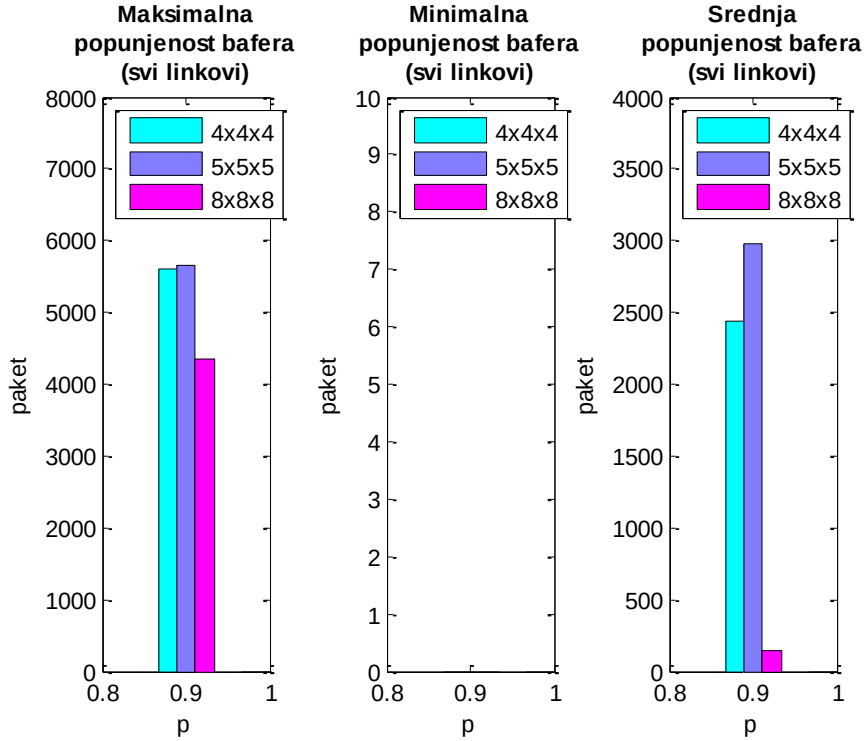
Grafik 4.2.2.15 Maksimalni, minimalni i srednji protok na eksternim linkovima torusa pri velikom opterećenju.



Grafik 4.2.2.16 Maksimalna, minimalna i srednja popunjenost bafera na eksternim linkovima torusa pri velikom opterećenju.

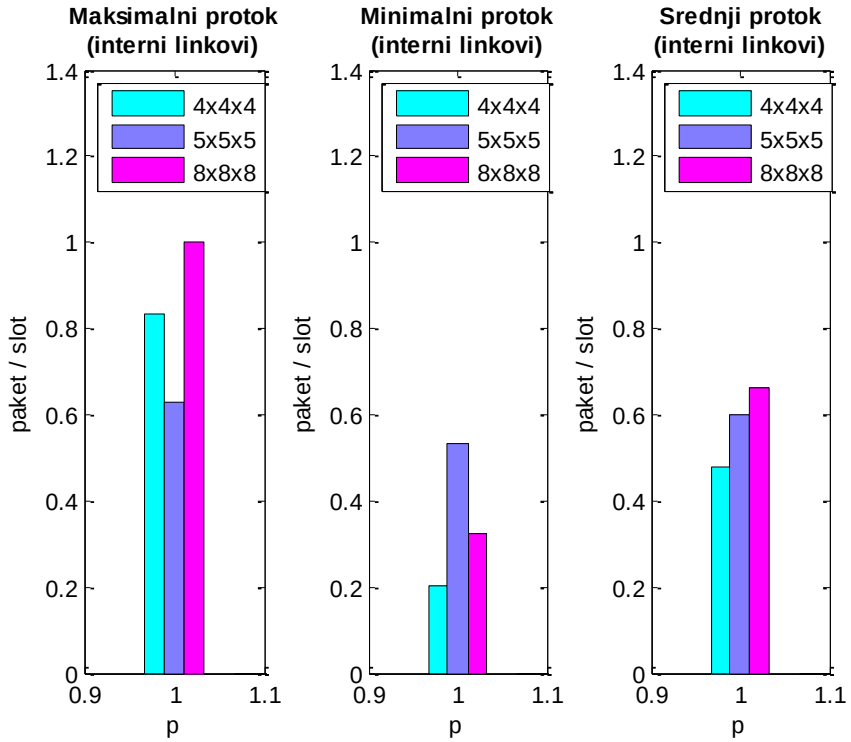


Grafik 4.2.2.17 Maksimalni, minimalni i srednji protok na svim linkovima torusa pri velikom opterećenju.

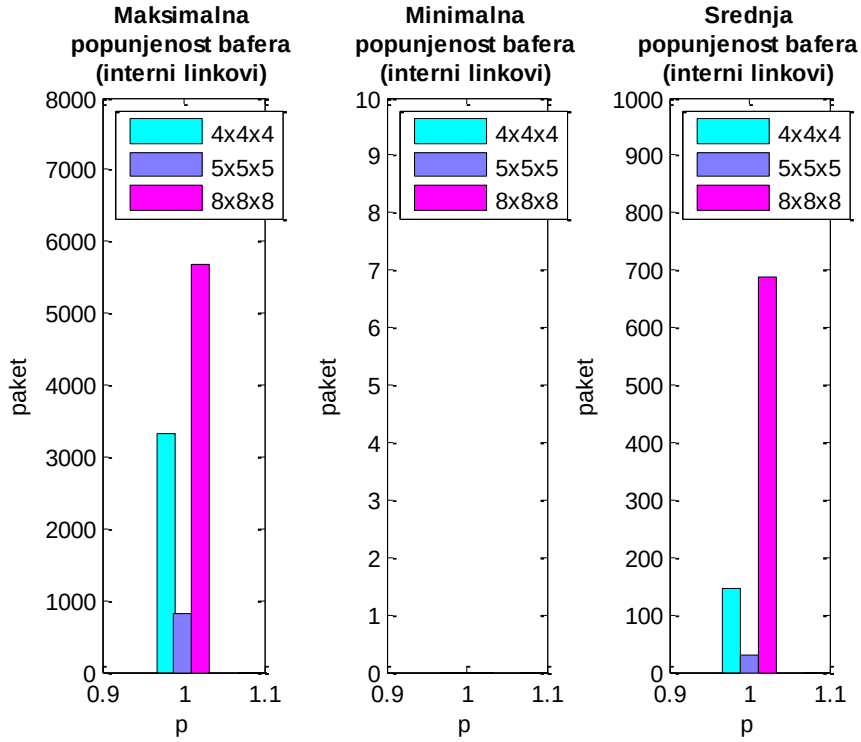


Grafik 4.2.2.18 Maksimalna, minimalna i srednja popunjenost bafera na svim linkovima torusa pri velikom opterećenju.

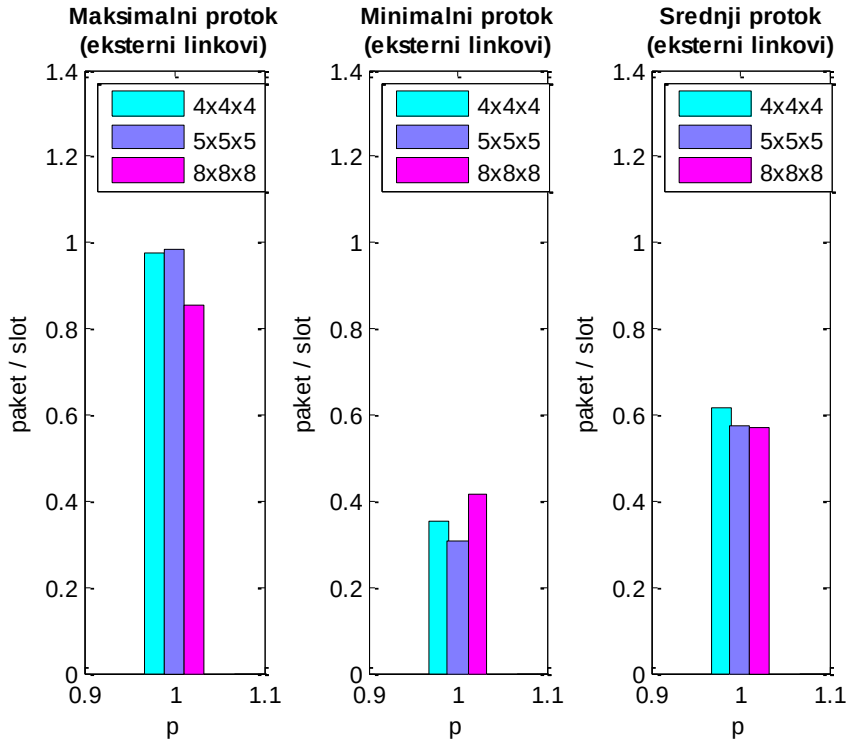
iv) Veliko opterećenje (p = 1)



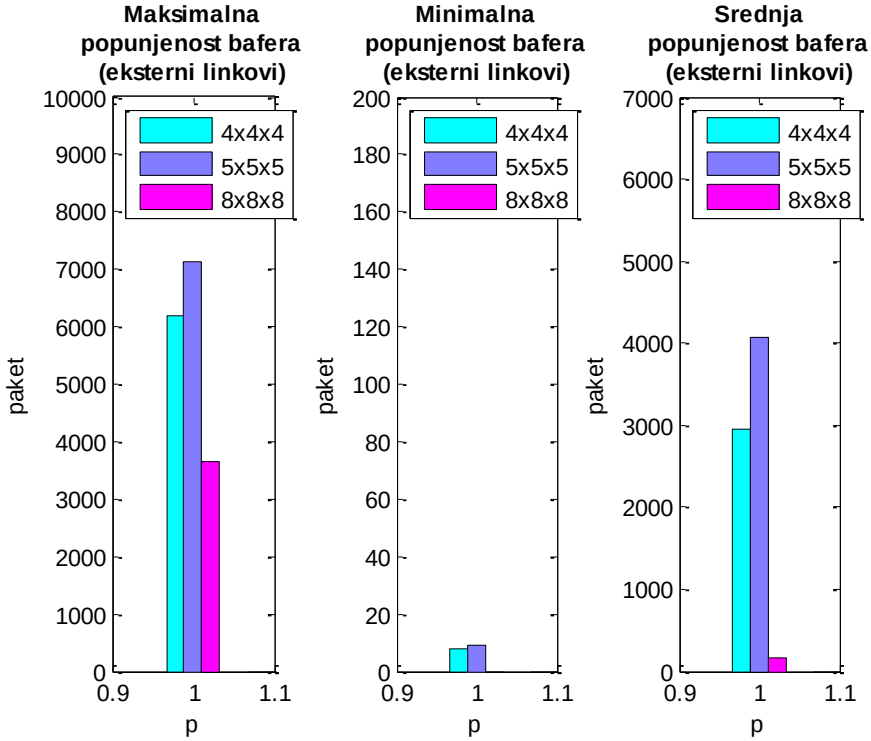
Grafik 4.2.2.19 Maksimalni, minimalni i srednji protok na internim linkovima torusa pri velikom opterećenju.



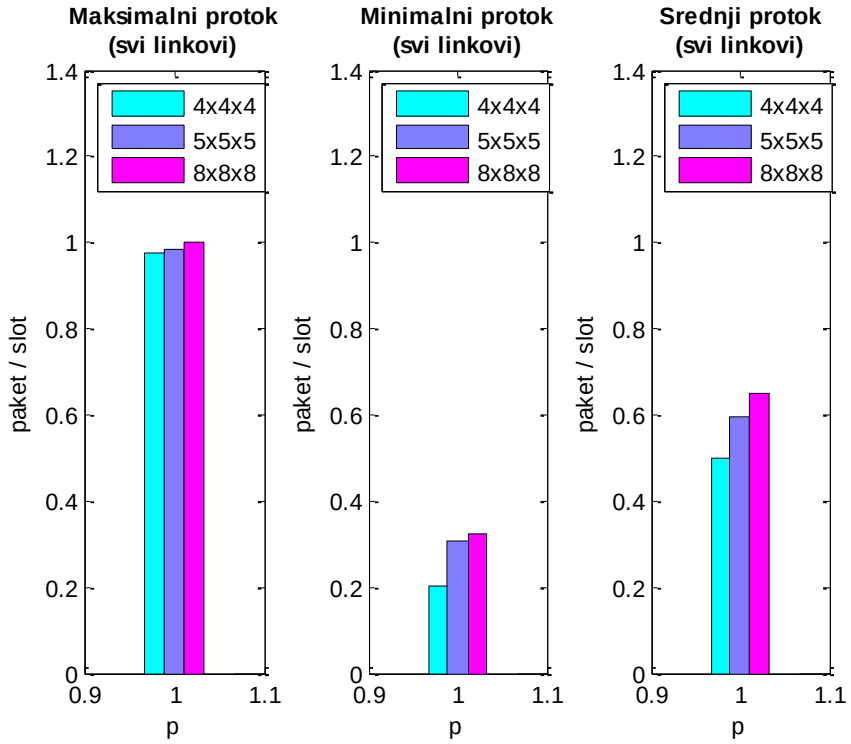
Grafik 4.2.2.20 Maksimalna, minimalna i srednja popunjenost bafera na internim linkovima torusa pri velikom opterećenju.



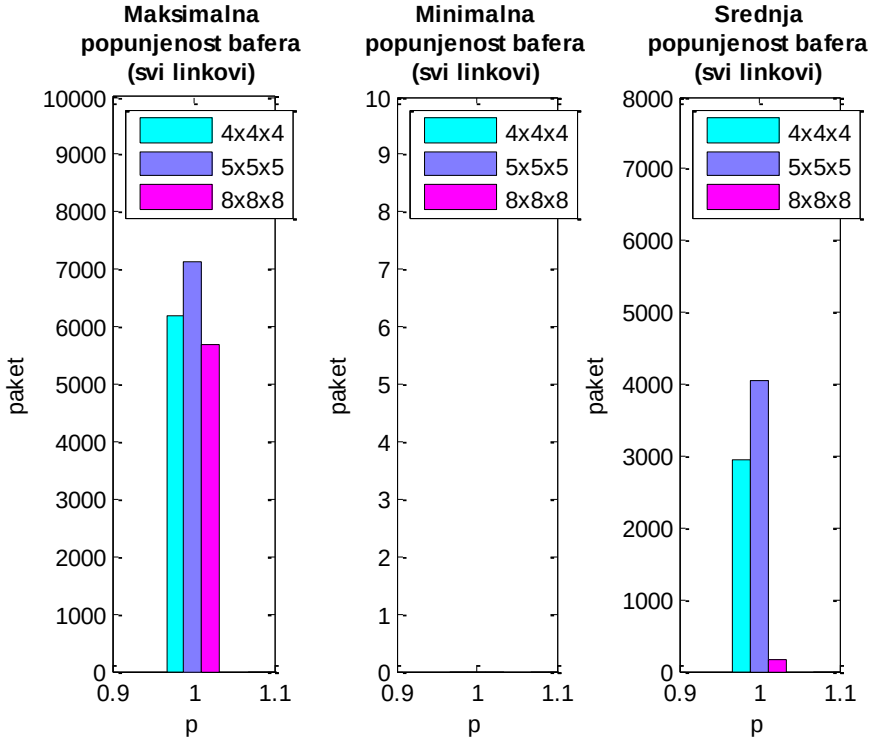
Grafik 4.2.2.21 Maksimalni, minimalni i srednji protok na eksternim linkovima torusa pri velikom opterećenju.



Grafik 4.2.2.22 Maksimalna, minimalna i srednja popunjenost bafera na eksternim linkovima torusa pri velikom opterećenju.



Grafik 4.2.2.23 Maksimalni, minimalni i srednji protok na svim linkovima torusa pri velikom opterećenju.



Grafik 4.2.2.24 Maksimalna, minimalna i srednja popunjenost bafera na svim linkovima torusa pri velikom opterećenju.

Prethodni grafici prikazuju direktno poređenje torusa različitih dimenzija pri promeni opterećenja. Prvih šest grafika pokazuju rezultate za mala opterećenja (p=0.2). Na njima se primećuje nešto veći protok po internim linkovima koji odgovara torusu dimenzija 8x8x8 u odnosu na preostale dve dimenzije (Grafik 4.2.2.1). Što se tiče eksternih linkova ne primećuju se značajne razlike u protoku i popunjenosti bafera među pomenutim torusima.

U slučaju srednjeg opterećenja (p=0.6), razlike između torusa 8x8x8 i preostala dva torusa postaju izraženije. Na grafikonima 4.2.2.7 i 4.2.2.8 vidi se da torus 8x8x8 ima veći srednji protok kao i popunjenost bafera na internim linkovima u odnosu na 4x4x4 i 5x5x5. Kod eksternih linkova razlika je nešto manja sa tim što sada torusi 5x5x5 i 4x4x4 dominiraju.

Pri velikim opterećenjima (p=0.9 i p=1), torus 8x8x8 ostaje dominantan u pogledu protoka i popunjenosti bafera na internim linkovima sa tim što je razlika u srednjem protoku manja između torusa 8x8x8 i 5x5x5 (Grafik 4.2.2.13). Na eksternim linkovima (Grafik 4.2.2.14) se primećuje da torus 4x4x4 postiže nešto veći srednji protok. Što se tiče popunjenosti bafera na eksternim linkovima torus 5x5x5 beleži najveću srednju popunjenost oko 3,000 paketa, torus 4x4x4 ima nešto nižu, dok torus 8x8x8 ima značajno manju srednju popunjenost.

Prilikom testiranja simulacije uočena je interesantna odlika torusa sa dimenzijama 5x5x5. Za razliku od torusa 4x4x4 i 8x8x8, torus 5x5x5 ima manji maksimalni protok kao i popunjenost bafera na internim linkovima. Kako je razlika između maksimalnog i minimalnog protoka na internim linkovima mala, što ukazuje da torus 5x5x5 ima ravnomernije raspoređeno saobraćajno opterećenje na internim linkovima.

Sva prethodna testiranja su rađena pri trajanju simulacije od 10,000 vremenskih slotova. Da bi se potvrdila verodostojnost rezultata izvršena su i testiranja pri trajanju simulacije od 80,000 slotova. Utvrđeno je da nema značajne razlike između grafika dobijenih u simulacijama, tj. trajanje simulacije nije uticalo na prikazane rezultate odnosa dimenzija torusa.

5. ZAKLJUČAK

Torusna arhitektura je jedno od jednostavnijih rešenja koje se može koristiti prilikom projektovanja komutacionog polja. U prilog torusnoj arhitekturi ide i mogućnost lakog proširivanja bez povećavanja kompleksnosti komutacije. Data karakteristika se mogla primetiti prilikom programiranja simulacije jer se nije javila potreba za dodavanjem novih funkcija niti je bilo potrebno izmeniti postojeće funkcije prilikom povećavanja dimenzija torusa.

Sa druge strane u prikazanim rezultatima uočena je i glavna mana ove arhitekture, a to je problem zagušenja na internim linkovima pri srednjim i velikim opterećenjima iako na eksternim linkovima nije bilo zagušenja. Ovaj problem je bio značajan kod torusa sa većim dimenzijama i bivao je izraženiji sa daljim povećanjem dimenzija.

Zanimljivo je da je torus sa dimenzijama 5x5x5 pokazao mala odstupanja od srednjeg protoka po internim linkovima i male popunjenosti bafera, što znači da je, za razliku od torusa sa dimenzijama 4x4x4 i 8x8x8, imao najbolje raspoređen saobraćaj po internim linkovima pri malom i srednjem opterećenju. Bilo bi poželjno daljim testiranjem ispitati ovu karakteristiku, ali bi se izašlo iz okvira teme rada.

Važno je napomenuti, da se postojeća tema može dodatno proširiti uključivanjem dodatnih opcija i ograničenja u simulaciju, kao što su adaptivno i balansirano rutiranje, zatim baferi ograničenog kapaciteta i paketi koji nisu fiksne dužine, kao i generisanje paketa u burstovima i merenje kašnjenja kroz torus, koji bi se mogli ispitivati u nekim od budućih radova.

LITERATURA

[1] Z. Čiča, *Komutacioni sistemi* – predavanja, [Online]. Available: http://telekomunikacije.etf.bg.ac.rs/predmeti/te4ks/ks.php

[2] L. Kraus, *Programski jezik C sa rešenim zadacima*, Akademska misao, Beograd, 2008.

[3] Cluster Design, *Torus Networks Design*, [Online]. Available: http://clusterdesign.org/torus/

[4] Wikipedia, *Torus*, [Online]. Available: http://en.wikipedia.org/wiki/Torus

[5] http://users.informatik.uni-halle.de/~jopsi/dpar03/chap2.shtml

[6] http://sankofa.loc.edu/savur/web/Parallel.html

[7] http://pages.cs.wisc.edu/~tvrdik/5/html/Section5.html

PRILOG

i) Glavni program.c

```
#include<stdio.h>
#include<stdlib.h>

typedefstruct paket { int x, y, z, vrem_slot; struct paket *sled;} Paket;  // FIFO bafer

int main () {
	int x, y, z, max_slot, w, q = 0, i, j, k, l;
	int p, ****protok, max_p, min_p, ****bafer, max_baf = 0, min_baf, ****count_gen;
	int max_ext_p, min_ext_p, max_ext_baf = 0, min_ext_baf;
	int max_uk_p, min_uk_p, max_uk_baf, min_uk_baf;
	double sred_uk_p, sum_uk_link = 0, cnt_uk_p = 0, sred_uk_baf, sum_uk_baf = 0,
cnt_uk_b = 0;
	double cnt_e_p = 0, sum_ext_link = 0, sred_ext_p, cnt_e_b = 0, sum_ext_baf = 0,
sred_ext_baf;
	double sred_p, sred_b, cnt1 = 0, cnt2 = 0, sum_baf = 0, sum_link = 0;
	Paket ****pokaz;
	Paket ****generisi(int x, int y, int z);
	int ****gen_niz_br(int x, int y, int z);

	FILE *ptr_file, *ptr_file_i1, *ptr_file_i2;
	ptr_file = fopen("input.txt", "r");
	if (!ptr_file) {
		printf("Greska prilikom otvaranja fajla!\n");
		return 1;
	}

	fscanf(ptr_file, "%d%d%d", &x, &y, &z);
	fscanf(ptr_file, "%d", &p);
	fscanf(ptr_file, "%d", &max_slot);
	fclose(ptr_file);

	void prosledi(int x, int y, int z, int q, Paket ****pokaz, int ****protok, int
****bafer, int *max_ext_baf, FILE *ptr);
	void generisi_pak(int x, int y, int z, int q, Paket ****pokaz, int p, int ****bafer,
int *max_baf, int ****count_gen);

	FILE *ptr;
	ptr = fopen("Rezultati_3.txt", "w");
	if (!ptr) {
		printf("Greska prilikom otvaranja fajla!\n");
		return 1;
	}

	pokaz = generisi(x,y,z);
	protok = gen_niz_br(x, y, z);
	bafer = gen_niz_br(x, y, z);
	count_gen = gen_niz_br(x, y, z);

	while (q<max_slot) {
		prosledi(x, y, z, q, pokaz, protok, bafer, &max_ext_baf, ptr);
```

```
			generisi_pak(x, y, z, q, pokaz, p, bafer, &max_baf, count_gen);
			q++;
		}

		ptr_file_i2 = fopen("Rezultati_2.txt", "w");
		if (!ptr_file_i2) {
			printf("Greska prilikom otvaranja fajla!\n");
			return 1;
		}
		max_p = protok[0][0][0][0];
		min_p = protok[0][0][0][0];
		min_baf = bafer[0][0][0][0];
		max_ext_p = protok[0][0][0][6];
		min_ext_p = protok[0][0][0][6];
		min_ext_baf = bafer[0][0][0][6];

		for (i = 0; i<x; i++)
		for (j = 0; j<y; j++)
		for (k = 0; k<z; k++) {
		fprintf(ptr_file_i2, "\n\n%d%d%d\n\n", i, j, k);
		for (l = 0; l < 6; l++) {
			if (protok[i][j][k][l] > max_p) max_p = protok[i][j][k][l];
			if (protok[i][j][k][l] < min_p) min_p = protok[i][j][k][l];
			if (protok[i][j][k][l] != 0) {
				sum_link = sum_link + protok[i][j][k][l];
				cnt1++;
			}
			if (bafer[i][j][k][l] < min_baf) min_baf = bafer[i][j][k][l];
				sum_baf = sum_baf + bafer[i][j][k][l];
				cnt2++;
			fprintf(ptr_file_i2, "%d%d%d\n", protok[i][j][k][l], bafer[i][j][k][l],
count_gen[i][j][k][l]);

			fprintf(ptr, "%d ", count_gen[i][j][k][l]);
			}

			if (protok[i][j][k][6] > max_ext_p) max_ext_p = protok[i][j][k][6];
			if (protok[i][j][k][6] < min_ext_p) min_ext_p = protok[i][j][k][6];
			sum_ext_link = sum_ext_link + protok[i][j][k][6];
			cnt_e_p++;

			if (bafer[i][j][k][6] < min_ext_baf) min_ext_baf = bafer[i][j][k][6];
			sum_ext_baf = sum_ext_baf + bafer[i][j][k][6];
			cnt_e_b++;

			for (l = 0; l < 7; l++) {
				sum_uk_link = sum_uk_link + protok[i][j][k][l];
				cnt_uk_p++;
				sum_uk_baf = sum_uk_baf + bafer[i][j][k][6];
				cnt_uk_b++;
			}

			fprintf(ptr, "%d\n", count_gen[i][j][k][6]);
			fprintf(ptr_file_i2, "%d%d%d\n", protok[i][j][k][6], bafer[i][j][k][6],
count_gen[i][j][k][6]);
				}

		sred_p = sum_link / cnt1;
		sred_b = sum_baf / cnt2;
		sred_ext_p = sum_ext_link / cnt_e_p;
```

```
	sred_ext_baf = sum_ext_baf / cnt_e_b;
	sred_uk_p = sum_uk_link / cnt_uk_p;
	sred_uk_baf = sum_uk_baf / cnt_uk_b;

	max_uk_p = max_ext_p;
	if (max_uk_p < max_p) max_uk_p = max_p;
	min_uk_p = min_ext_p;
	if (min_uk_p > min_p) min_uk_p = min_p;
	max_uk_baf = max_ext_baf;
	if (max_uk_baf < max_baf) max_uk_baf = max_baf;
	min_uk_baf = min_ext_baf;
	if (min_uk_baf > min_baf) min_uk_baf = min_baf;

	ptr_file_i1 = fopen("Rezultati_1.txt", "w");
	if (!ptr_file_i1) {
		printf("Greska prilikom otvaranja fajla!\n");
		return 1;
	}

	fprintf(ptr_file_i1, "\nMinimalni protok na internim linkovima je: %d\n", min_p);
	fprintf(ptr_file_i1, "\nMaksimalni protok na internim linkovima je: %d\n", max_p);
	fprintf(ptr_file_i1, "\nSrednji protok na internim linkovima je: %f\n", sred_p);

	fprintf(ptr_file_i1, "\nMinimalni protok na eksternim linkovima je: %d\n",
min_ext_p);
	fprintf(ptr_file_i1, "\nMaksimalni protok na eksternim linkovima je: %d\n",
max_ext_p);
	fprintf(ptr_file_i1, "\nSrednji protok na eksternim linkovima je: %f\n", sred_ext_p);
	fprintf(ptr_file_i1, "\nMinimalni protok na svim linkovima je: %d\n", min_uk_p);
	fprintf(ptr_file_i1, "\nMaksimalni protok na svim linkovima je: %d\n", max_uk_p);
	fprintf(ptr_file_i1, "\nSrednji protok na svim linkovima je: %f\n", sred_uk_p);

	fprintf(ptr_file_i1, "\nNajmanja popunjenost bafera na internim linkovima: %d
paketa\n", min_baf);
	fprintf(ptr_file_i1, "\nNajveca popunjenost bafera na internim linkovima: %d
paketa\n", max_baf);
	fprintf(ptr_file_i1, "\nSrednja popunjenost bafera na internim linkovima: %f
paketa\n", sred_b);

	fprintf(ptr_file_i1, "\nNajmanja popunjenost bafera na externim linkovima: %d
paketa\n", min_ext_baf);
	fprintf(ptr_file_i1, "\nNajveca popunjenost bafera na externim linkovima: %d
paketa\n", max_ext_baf);
	fprintf(ptr_file_i1, "\nSrednja popunjenost bafera na externim linkovima: %f
paketa\n", sred_ext_baf);

	fprintf(ptr_file_i1, "\nNajmanja popunjenost bafera na svim linkovima: %d paketa\n",
min_uk_baf);
	fprintf(ptr_file_i1, "\nNajveca popunjenost bafera na svim linkovima: %d paketa\n",
max_uk_baf);
	fprintf(ptr_file_i1, "\nSrednja popunjenost bafera na svim linkovima: %f paketa\n",
sred_uk_baf);

	fclose(ptr_file_i1);
	fclose(ptr_file_i2);
	fclose(ptr);
	printf("Simulacija je uspesno zavrsena!\n");
	scanf("%d", &w);
	free (pokaz);
	}
```

ii) F-ja generisi.c

```
#include<stdio.h>
#include<stdlib.h>

typedefstruct paket { int x, y, z, vrem_slot; struct paket *sled; } Paket;

Paket ****generisi(int x, int y, int z) {
	int i, j, k, l;
	Paket ****pokaz;
	pokaz = malloc(x * sizeof(Paket***));
	for (i = 0; i < x; i++) {
		*(pokaz + i) = malloc(y * sizeof(Paket**));
		for (j = 0; j < y; j++){
			*(*(pokaz + i) + j) = malloc(z * sizeof(Paket*));
			for (k = 0; k < z; k++){
				*(*(*(pokaz + i) + j) + k) = malloc(7 * sizeof(Paket));
				for (l = 0; l < 7; l++) {
					(*(*(*(pokaz + i) + j) + k) + l)->x = -9;
//postavlja vrednost koordinate x na negativnu vrednost kao oznaku da se radi o praznom
baferu
					(*(*(*(pokaz + i) + j) + k) + l)->vrem_slot = -1;
//nepostojeci vremenski slot
					(*(*(*(pokaz + i) + j) + k) + l) -> sled = NULL;
				}
			}
		}
	}
	return pokaz;
}
```

iii) F-ja gener_rac_protok.c

```
#include<stdio.h>
#include<stdlib.h>


int ****gen_niz_br(int x, int y, int z) {
	int i, j, k, l;
	int ****protok;
	protok = malloc(x * sizeof(int***));
	for (i = 0; i < x; i++) {
		*(protok + i) = malloc(y * sizeof(int**));
		for (j = 0; j < y; j++){
			*(*(protok + i) + j) = malloc(z * sizeof(int*));
			for (k = 0; k < z; k++){
				*(*(*(protok + i) + j) + k) = malloc(7 * sizeof(int));
				for (l = 0; l < 7; l++) {
					protok[i][j][k][l]=0;
				}
			}
		}
	}
	return protok;
}
```

iv) Generisi_pak.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

typedefstruct paket { int x, y, z, vrem_slot; struct paket *sled; } Paket;

void generisi_pak(int x, int y, int z, int q, Paket ****pokaz, int p, int ****bafer, int
*max_baf, int ****count_gen) {
	Paket *sledeci = NULL;
	int i, j, k, mark;
	for (i = 0; i < x; i++)
	for (j = 0; j < y; j++)
	for (k = 0; k < z; k++) {
		if (rand() % 10 >= 10-p) {
//u zavisnosti od p generise paket ili ne generise (Bernuli)
		mark = 1;
//mark ne dozvoljava da se jedan isti paket koji je generisan prekopira u vise bafera
		Paket *tekuci = malloc(sizeof(Paket));
//alocira se memorijski prostor velicine praznog paketa
		while (mark) {
//while petlja dodeljuje odredisnu adresu paket po uniformnoj raspodeli
			tekuci->x = rand() % x;
			tekuci->y = rand() % y;
			tekuci->z = rand() % z;
			if (i == tekuci->x && j == tekuci->y && k == tekuci->z) mark = 1;
//ovaj uslov ne dozvoljava da cvor generise paket za samog sebe
			else mark = 0;
		}
		tekuci ->vrem_slot = q;
//generisanom paketu dodeljujem red.br. vremenskog slota u kom je generisan (kako se ne bi
doslo da paket bude prosledjen u istom vremenskom slotu u kome je i generisan
		tekuci->sled = NULL;
		count_gen[i][j][k][6]++;
//broji koliko je jedan cvor generisao paketa
//-------------------------------------------------------------------------------------
		if (mark == 0 && ((i > tekuci->x && tekuci->x >= i - x/2) || tekuci->x > i +
x/2)) {
			if ((*(*(*(pokaz + i) + j) + k) + 0)->x >= 0) {
//ispituje da li je odgovarajuci bafer pun
				sledeci = malloc(sizeof(Paket));
//alocira memorijski prostor velicine paketa
				*sledeci = *(*(*(*(pokaz + i) + j) + k) + 0);
//u alocirani mem. prostor se upisuju vrednosti paketa koji je poslednji usao u bafer
				tekuci->sled = sledeci;
//sada pokazivac novogenerisanog paketa pokazuje na prethodno alocirani mem. prostor
			}
//ceo ovaj proces se izvrsava zato sto je pokazivac na bafer fiksiran na prethodno
rezervisani memorijski prostor i ne mozemo ga preusmeravati vec jedino mozemo menjati
vrednos mem.prostoru na koji on pokazuje
			*(*(*(*(pokaz + i) + j) + k) + 0) = *tekuci;
//pokazivac na bafer sada pokazuje na novogenerisani paket
			count_gen[i][j][k][0]++;
//brojac meri koliko je od generisanih paketa poslato u posmatrani bafer
			bafer[i][j][k][0]++;
//brojac pamti da je u bafer ubacen paket
			if (bafer[i][j][k][0] > *max_baf) *max_baf = bafer[i][j][k][0];
			mark = 1;
```

```
		}
		elseif (mark == 0 && ((i < tekuci->x && tekuci->x <= i + x/2) || tekuci->x < i
- x/2)) {
			if ((*(*(*(pokaz + i) + j) + k) + 1)->x >= 0) {
				sledeci = malloc(sizeof(Paket));
				*sledeci = *(*(*(*(pokaz + i) + j) + k) + 1);
				tekuci->sled = sledeci;
			}
			*(*(*(*(pokaz + i) + j) + k) + 1) = *tekuci;
			count_gen[i][j][k][1]++;
			bafer[i][j][k][1]++;
			if (bafer[i][j][k][1] > *max_baf) *max_baf = bafer[i][j][k][1];
			mark = 1;
		}
		elseif (mark == 0 && ((j > tekuci->y && tekuci->y >= j - y/2) || tekuci->y > j
+ y/2)) {
			if ((*(*(*(pokaz + i) + j) + k) + 2)->x >= 0) {
				sledeci = malloc(sizeof(Paket));
				*sledeci = *(*(*(*(pokaz + i) + j) + k) + 2);
				tekuci->sled = sledeci;
			}
			*(*(*(*(pokaz + i) + j) + k) + 2) = *tekuci;
			count_gen[i][j][k][2]++;
			bafer[i][j][k][2]++;
			if (bafer[i][j][k][2] > *max_baf) *max_baf = bafer[i][j][k][2];
			mark = 1;
		}
		elseif (mark == 0 && ((j < tekuci->y && tekuci->y <= j + y/2) || tekuci->y < j
- y/2)) {
			if ((*(*(*(pokaz + i) + j) + k) + 3)->x >= 0) {
				sledeci = malloc(sizeof(Paket));
				*sledeci = *(*(*(*(pokaz + i) + j) + k) + 3);
				tekuci->sled = sledeci;
			}
			*(*(*(*(pokaz + i) + j) + k) + 3) = *tekuci;
			count_gen[i][j][k][3]++;
			bafer[i][j][k][3]++;
			if (bafer[i][j][k][3] > *max_baf) *max_baf = bafer[i][j][k][3];
			mark = 1;
		}
		elseif (mark == 0 && ((k > tekuci->z && tekuci->z >= k - z/2) || tekuci->z > k
+ z/2)) {
			if ((*(*(*(pokaz + i) + j) + k) + 4)->x >= 0) {
				sledeci = malloc(sizeof(Paket));
				*sledeci = *(*(*(*(pokaz + i) + j) + k) + 4);
				tekuci->sled = sledeci;
			}
			*(*(*(*(pokaz + i) + j) + k) + 4) = *tekuci;
			count_gen[i][j][k][4]++;
			bafer[i][j][k][4]++;
			if (bafer[i][j][k][4] > *max_baf) *max_baf = bafer[i][j][k][4];
			mark = 1;
		}
		elseif (mark == 0 && ((k < tekuci->z && tekuci->z <= k + z/2) || tekuci->z < k
- z/2)) {
			if ((*(*(*(pokaz + i) + j) + k) + 5)->x >= 0) {
				sledeci = malloc(sizeof(Paket));
				*sledeci = *(*(*(*(pokaz + i) + j) + k) + 5);
				tekuci->sled = sledeci;
			}
```

```
                        *(*(*(*(pokaz + i) + j) + k) + 5) = *tekuci;
                        count_gen[i][j][k][5]++;
                        bafer[i][j][k][5]++;
                        if (bafer[i][j][k][5] > *max_baf) *max_baf = bafer[i][j][k][5];
                        mark = 1;
                }
        }
        }
}
```

v) Prosledi.c

```
#include<stdio.h>
#include<stdlib.h>

typedefstruct paket { int x, y, z, vrem_slot; struct paket *sled; } Paket;

//#############################################################################################
// l=0 - '-x'; l=1 - '+x'; l=2 - '-y'; l=3 - '+y'; l=4 - '-z'; l=5 - '+z'; l=6 - 'izlaz'-
'osa'
//#############################################################################################
//bafer je prazan ako je pokaz[i][j][k][l].x=-9 i pokaz[i][j][k][l].sled=NULL

void prosledi(int x, int y, int z, int q, Paket ****pokaz, int ****protok, int ****bafer,
int *max_ext_baf, FILE *ptr) {
        Paket *tekuci, *sledeci = NULL, *prosli = NULL, *temp = NULL;
        int i, j, k, l, mark, crd;
        for (i = 0; i<x; i++)
        for (j = 0; j<y; j++)
        for (k = 0; k<z; k++)
        for (l = 0; l<7; l++) {
                if ((*(*(*(pokaz + i) + j) + k) + l)->x >= 0 && (*(*(*(pokaz + i) + j) + k) +
l)->vrem_slot < q) {
//ispituje da li ima paketa u baferu i da li je taj paket vec prosledjen u tekucem
vremenskom slotu
                        temp = (*(*(*(pokaz + i) + j) + k) + l);
//temp se postavlja na pocetak bafera(liste)
                        prosli = NULL;
                        while (temp -> sled != NULL) {
//while petlja postavlja temp na poslednji element liste, sto je ustvari paket koji treba
prosleditи
                                prosli = temp;
//prosli je sad "novi" poslednji paket u baferu koji ce u sledecem krugu biti prosledjen
                                temp = prosli -> sled;
                        }
                        tekuci = malloc(sizeof(Paket));
//tekuci sad postaje slobodan paket koji mozemo preneti
                        *tekuci = *temp;
                        tekuci->vrem_slot = q;
//ovde se u paket upisuje vrednost tekuceg vremenskog slota u kome ce biti prosledjen
                        if (prosli != NULL) prosli->sled = NULL;
//ukoliko je ispunjen uslov "novi" poslednji paket raskida vezu sa starim
                        else {
                                (*(*(*(pokaz + i) + j) + k) + l)->x = -9;
//ukoliko nije ispunjeno, a to znaci da smo u baferu imali samo jedan paket, pokazivac na
ulaz bafera sa vraca na default vrednost(prazan)
                                (*(*(*(pokaz + i) + j) + k) + l)->sled = NULL;
                        }
```

```
                    mark = 0;
                    crd = 0;
                    protok[i][j][k][l]++;
                    bafer[i][j][k][l]--;
                    switch (l) {
//-------------------------------------------------------------------------------------
                    case 0:if (i - 1 < 0) crd = x - 1;
                           else crd = i - 1;
                           if (mark == 0 && ((crd > tekuci->x && tekuci->x >= crd - x/2) ||
tekuci->x > crd + x/2)) {
//jer bi u suprotnom paket isao levo a onda skrenuo desno
                                  if ((*(*(*(pokaz + crd) + j) + k) + 0)->x >= 0) {
                                         sledeci  = malloc(sizeof(Paket));
                                         *sledeci = *(*(*(*(pokaz + crd) + j) + k) + 0);
//paket ide u bafer l=0 (levo)
                                         tekuci->sled = sledeci;
                                  }
                                  *(*(*(*(pokaz + crd) + j) + k) + 0) = *tekuci;
                                  bafer[crd][j][k][0]++;
                                  mark = 1;
//mark sprecava da se isti paket pomeri vise put u jednoj iteraciji
                           }
                           elseif (mark == 0 && ((j > tekuci->y && tekuci->y >= j - y/2) ||
tekuci->y > j + y/2)) {
//ovde se vec smatra da je x cvora = x paketa
                                  if ((*(*(*(pokaz + crd) + j) + k) + 2)->x >= 0) {
                                         sledeci = malloc(sizeof(Paket));
                                         *sledeci = *(*(*(*(pokaz + crd) + j) + k) + 2);
//pa sad ga saljemo po y osi; paket ide u bafer l=2 (nazad)
                                         tekuci->sled = sledeci;
                                  }
                                  *(*(*(*(pokaz + crd) + j) + k) + 2) = *tekuci;
                                  bafer[crd][j][k][2]++;
                                  mark = 1;
                           }
                           elseif (mark == 0 && ((j < tekuci->y && tekuci->y <= j + y/2) ||
tekuci->y < j - y/2)) {
//paket ide u bafer l=3 (napred)
                                  if ((*(*(*(pokaz + crd) + j) + k) + 3)->x >= 0) {
                                         sledeci  = malloc(sizeof(Paket));
                                         *sledeci = *(*(*(*(pokaz + crd) + j) + k) + 3);
                                         tekuci->sled = sledeci;
                                  }
                                  *(*(*(*(pokaz + crd) + j) + k) + 3) = *tekuci;
                                  bafer[crd][j][k][3]++;
                                  mark = 1;
                           }
                           elseif (mark == 0 && ((k > tekuci->z && tekuci->z >= k - z/2) ||
tekuci->z > k + z/2)) {
                                  if ((*(*(*(pokaz + crd) + j) + k) + 4)->x >= 0) {
                                         sledeci  = malloc(sizeof(Paket));
                                         *sledeci = *(*(*(*(pokaz + crd) + j) + k) + 4);
//y cvora = y paketa pa sad saljemo po z osi
                                         tekuci->sled = sledeci;
//paket ide u bafer l=4 (nadole)
                                  }
                                  *(*(*(*(pokaz + crd) + j) + k) + 4) = *tekuci;
                                  bafer[crd][j][k][4]++;
                                  mark = 1;
                           }
```

```
                        elseif (mark == 0 && ((k < tekuci->z && tekuci->z <= k + z/2) ||
tekuci->z < k - z/2)) {
//paket ide u bafer l=5 (nagore)
                                if ((*(*(*(pokaz + crd) + j) + k) + 5)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + crd) + j) + k) + 5);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + crd) + j) + k) + 5) = *tekuci;
                                bafer[crd][j][k][5]++;
                                mark = 1;
                        }
                        elseif (mark == 0) {
//ovde se smatra da su koorinate x,y,z paketa = x,y,z cvora
                                if ((*(*(*(pokaz + crd) + j) + k) + 6)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + crd) + j) + k) + 6);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + crd) + j) + k) + 6) = *tekuci;
                                bafer[crd][j][k][6]++;
                                if (bafer[crd][j][k][6] > *max_ext_baf) *max_ext_baf =
bafer[crd][j][k][6];
                                mark = 1;
                        }
                        break;
//-------------------------------------------------------------------------------------
                case 1:if (i + 1 > x-1) crd = 0;
                        else crd = i + 1;
                        if (mark == 0 && ((crd < tekuci->x && tekuci->x <= crd + x / 2)
|| tekuci->x < crd - x / 2)) {
                                if ((*(*(*(pokaz + crd) + j) + k) + 1)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + crd) + j) + k) + 1);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + crd) + j) + k) + 1) = *tekuci;
                                bafer[crd][j][k][1]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((j > tekuci->y && tekuci->y >= j - y/2) ||
tekuci->y > j + y/2)) {
                                if ((*(*(*(pokaz + crd) + j) + k) + 2)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + crd) + j) + k) + 2);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + crd) + j) + k) + 2) = *tekuci;
                                bafer[crd][j][k][2]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((j < tekuci->y && tekuci->y <= j + y/2) ||
tekuci->y < j - y/2)) {
                                if ((*(*(*(pokaz + crd) + j) + k) + 3)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + crd) + j) + k) + 3);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + crd) + j) + k) + 3) = *tekuci;
                                bafer[crd][j][k][3]++;
```

```
                                mark = 1;
                        }
                        elseif (mark == 0 && ((k > tekuci->z && tekuci->z >= k - z/2) ||
tekuci->z > k + z/2)) {
                                if ((*(*(*(pokaz + crd) + j) + k) + 4)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + crd) + j) + k) + 4);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + crd) + j) + k) + 4) = *tekuci;
                                bafer[crd][j][k][4]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((k < tekuci->z && tekuci->z <= k + z/2) ||
tekuci->z < k - z/2)) {
                                if ((*(*(*(pokaz + crd) + j) + k) + 5)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + crd) + j) + k) + 5);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + crd) + j) + k) + 5) = *tekuci;
                                bafer[crd][j][k][5]++;
                                mark = 1;
                        }
                        elseif (mark == 0) {
                                if ((*(*(*(pokaz + crd) + j) + k) + 6)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + crd) + j) + k) + 6);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + crd) + j) + k) + 6) = *tekuci;
                                bafer[crd][j][k][6]++;
                                if (bafer[crd][j][k][6] > *max_ext_baf) *max_ext_baf =
bafer[crd][j][k][6];
                                mark = 1;
                        }
                        break;
//-------------------------------------------------------------------------------------
                case 2:if (j - 1 < 0) crd = y - 1;
                        else crd = j - 1;
                        if (mark == 0 && ((i > tekuci->x && tekuci->x >= i - x/2) ||
tekuci->x > i + x/2)) {
                                if ((*(*(*(pokaz + i) + crd) + k) + 0)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + crd) + k) + 0);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + crd) + k) + 0) = *tekuci;
                                bafer[i][crd][k][0]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((i < tekuci->x && tekuci->x <= i + x/2) ||
tekuci->x < i - x/2)) {
                                if ((*(*(*(pokaz + i) + crd) + k) + 1)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + crd) + k) + 1);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + crd) + k) + 1) = *tekuci;
                                bafer[i][crd][k][1]++;
```

```
                                mark = 1;
                        }
                        elseif (mark == 0 && ((crd > tekuci->y && tekuci->y >= crd -
y/2) || tekuci->y > crd + y/2)) {
                                if ((*(*(*(pokaz + i) + crd) + k) + 2)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + crd) + k) + 2);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + crd) + k) + 2) = *tekuci;
                                bafer[i][crd][k][2]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((k > tekuci->z && tekuci->z >= k - z/2) ||
tekuci->z > k + z/2)) {
                                if ((*(*(*(pokaz + i) + crd) + k) + 4)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + crd) + k) + 4);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + crd) + k) + 4) = *tekuci;
                                bafer[i][crd][k][4]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((k < tekuci->z && tekuci->z <= k + z/2) ||
tekuci->z < k - z/2)) {
                                if ((*(*(*(pokaz + i) + crd) + k) + 5)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + crd) + k) + 5);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + crd) + k) + 5) = *tekuci;
                                bafer[i][crd][k][5]++;
                                mark = 1;
                        }
                        elseif (mark == 0) {
                                if ((*(*(*(pokaz + i) + crd) + k) + 6)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + crd) + k) + 6);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + crd) + k) + 6) = *tekuci;
                                bafer[i][crd][k][6]++;
                                if (bafer[i][crd][k][6] > *max_ext_baf) *max_ext_baf =
bafer[i][crd][k][6];
                                mark = 1;
                        }
                        break;
//-------------------------------------------------------------------------------------
                case 3:if (j + 1 > y - 1) crd = 0;
                        else crd = j + 1;
                        if (mark == 0 && ((i > tekuci->x && tekuci->x >= i - x/2) ||
tekuci->x > i + x/2)) {
                                if ((*(*(*(pokaz + i) + crd) + k) + 0)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + crd) + k) + 0);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + crd) + k) + 0) = *tekuci;
                                bafer[i][crd][k][0]++;
```

```
                                mark = 1;
                        }
                        elseif (mark == 0 && ((i < tekuci->x && tekuci->x <= i + x/2) ||
tekuci->x < i - x/2)) {
                                if ((*(*(*(pokaz + i) + crd) + k) + 1)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + crd) + k) + 1);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + crd) + k) + 1) = *tekuci;
                                bafer[i][crd][k][1]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((crd < tekuci->y && tekuci->y <= crd +
y/2) || tekuci->y < crd - y/2)) {
                                if ((*(*(*(pokaz + i) + crd) + k) + 3)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + crd) + k) + 3);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + crd) + k) + 3) = *tekuci;
                                bafer[i][crd][k][3]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((k > tekuci->z && tekuci->z >= k - z/2) ||
tekuci->z > k + z/2)) {
                                if ((*(*(*(pokaz + i) + crd) + k) + 4)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + crd) + k) + 4);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + crd) + k) + 4) = *tekuci;
                                bafer[i][crd][k][4]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((k < tekuci->z && tekuci->z <= k + z/2) ||
tekuci->z < k - z/2)) {
                                if ((*(*(*(pokaz + i) + crd) + k) + 5)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + crd) + k) + 5);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + crd) + k) + 5) = *tekuci;
                                bafer[i][crd][k][5]++;
                                mark = 1;
                        }
                        elseif (mark == 0) {
                                if ((*(*(*(pokaz + i) + crd) + k) + 6)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + crd) + k) + 6);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + crd) + k) + 6) = *tekuci;
                                bafer[i][crd][k][6]++;
                                if (bafer[i][crd][k][6] > *max_ext_baf) *max_ext_baf =
bafer[i][crd][k][6];
                                mark = 1;
                        }
                        break;
//-------------------------------------------------------------------------------
```

```
                    case 4:if (k - 1 < 0) crd = z - 1;
                           else crd = k - 1;
                           if (mark == 0 && ((i > tekuci->x && tekuci->x >= i - x/2) ||
tekuci->x > i + x/2)) {
                                  if ((*(*(*(pokaz + i) + j) + crd) + 0)->x >= 0) {
                                         sledeci  = malloc(sizeof(Paket));
                                         *sledeci = *(*(*(*(pokaz + i) + j) + crd) + 0);
                                         tekuci->sled = sledeci;
                                  }
                                  *(*(*(*(pokaz + i) + j) + crd) + 0) = *tekuci;
                                  bafer[i][j][crd][0]++;
                                  mark = 1;
                           }
                           elseif (mark == 0 && ((i < tekuci->x && tekuci->x <= i + x/2) ||
tekuci->x < i - x/2)) {
                                  if ((*(*(*(pokaz + i) + j) + crd) + 1)->x >= 0) {
                                         sledeci  = malloc(sizeof(Paket));
                                         *sledeci = *(*(*(*(pokaz + i) + j) + crd) + 1);
                                         tekuci->sled = sledeci;
                                  }
                                  *(*(*(*(pokaz + i) + j) + crd) + 1) = *tekuci;
                                  bafer[i][j][crd][1]++;
                                  mark = 1;
                           }
                           elseif (mark == 0 && ((j > tekuci->y && tekuci->y >= j - y/2) ||
tekuci->y > j + y/2)) {
                                  if ((*(*(*(pokaz + i) + j) + crd) + 2)->x >= 0) {
                                         sledeci  = malloc(sizeof(Paket));
                                         *sledeci = *(*(*(*(pokaz + i) + j) + crd) + 2);
                                         tekuci->sled = sledeci;
                                  }
                                  *(*(*(*(pokaz + i) + j) + crd) + 2) = *tekuci;
                                  bafer[i][j][crd][2]++;
                                  mark = 1;
                           }
                           elseif (mark == 0 && ((j < tekuci->y && tekuci->y <= j + y/2) ||
tekuci->y < j - y/2)) {
                                  if ((*(*(*(pokaz + i) + j) + crd) + 3)->x >= 0) {
                                         sledeci  = malloc(sizeof(Paket));
                                         *sledeci = *(*(*(*(pokaz + i) + j) + crd) + 3);
                                         tekuci->sled = sledeci;
                                  }
                                  *(*(*(*(pokaz + i) + j) + crd) + 3) = *tekuci;
                                  bafer[i][j][crd][3]++;
                                  mark = 1;
                           }
                           elseif (mark == 0 && ((crd > tekuci->z && tekuci->z >= crd -
z/2) || tekuci->z > crd + z/2)) {
                                  if ((*(*(*(pokaz + i) + j) + crd) + 4)->x >= 0) {
                                         sledeci  = malloc(sizeof(Paket));
                                         *sledeci = *(*(*(*(pokaz + i) + j) + crd) + 4);
                                         tekuci->sled = sledeci;
                                  }
                                  *(*(*(*(pokaz + i) + j) + crd) + 4) = *tekuci;
                                  bafer[i][j][crd][4]++;
                                  mark = 1;
                           }
                           elseif (mark == 0) {
                                  if ((*(*(*(pokaz + i) + j) + crd) + 6)->x >= 0) {
                                         sledeci  = malloc(sizeof(Paket));
```

```
                                        *sledeci = *(*(*(*(pokaz + i) + j) + crd) + 6);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + j) + crd) + 6) = *tekuci;
                                bafer[i][j][crd][6]++;
                                if (bafer[i][j][crd][6] > *max_ext_baf) *max_ext_baf =
bafer[i][j][crd][6];
                                mark = 1;
                        }
                        break;
//-------------------------------------------------------------------------------------
                case 5:if (k + 1 > z - 1) crd = 0;
                        else crd = k + 1;
                        if (mark == 0 && ((i > tekuci->x && tekuci->x >= i - x/2) ||
tekuci->x > i + x/2)) {
                                if ((*(*(*(pokaz + i) + j) + crd) + 0)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + j) + crd) + 0);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + j) + crd) + 0) = *tekuci;
                                bafer[i][j][crd][0]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((i < tekuci->x && tekuci->x <= i + x/2) ||
tekuci->x < i - x/2)) {
                                if ((*(*(*(pokaz + i) + j) + crd) + 1)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + j) + crd) + 1);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + j) + crd) + 1) = *tekuci;
                                bafer[i][j][crd][1]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((j > tekuci->y && tekuci->y >= j - y/2) ||
tekuci->y > j + y/2)) {
                                if ((*(*(*(pokaz + i) + j) + crd) + 2)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + j) + crd) + 2);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + j) + crd) + 2) = *tekuci;
                                bafer[i][j][crd][2]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((j < tekuci->y && tekuci->y <= j + y/2) ||
tekuci->y < j - y/2)) {
                                if ((*(*(*(pokaz + i) + j) + crd) + 3)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + j) + crd) + 3);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + j) + crd) + 3) = *tekuci;
                                bafer[i][j][crd][3]++;
                                mark = 1;
                        }
                        elseif (mark == 0 && ((crd < tekuci->z && tekuci->z <= crd +
z/2) || tekuci->z < crd - z/2)) {
                                if ((*(*(*(pokaz + i) + j) + crd) + 5)->x >= 0) {
```

```
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + j) + crd) + 5);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + j) + crd) + 5) = *tekuci;
                                bafer[i][j][crd][5]++;
                                mark = 1;
                        }
                        elseif (mark == 0) {
                                if ((*(*(*(pokaz + i) + j) + crd) + 6)->x >= 0) {
                                        sledeci  = malloc(sizeof(Paket));
                                        *sledeci = *(*(*(*(pokaz + i) + j) + crd) + 6);
                                        tekuci->sled = sledeci;
                                }
                                *(*(*(*(pokaz + i) + j) + crd) + 6) = *tekuci;
                                bafer[i][j][crd][6]++;
                                if (bafer[i][j][crd][6] > *max_ext_baf) *max_ext_baf =
bafer[i][j][crd][6];
                                mark = 1;
                        }
                        break;
//-------------------------------------------------------------------------------------
                default: break;
                }
            free(tekuci);
            }
      }
}
```