Универзитет у Београду
Електротехнички факултет

Симулација распоређивања пакета на основу приоритета

Дипломски рад

Ментор:

Др Зоран Чича, ванр. проф.

Кандидат:

Тамара Јовановић
2017/0158

Београд, Септембар 2021.

САДРЖАЈ

1. Увод

Како је у почетку размена података у мрежи била базирана на техници комутације кола, где је за сваку комуникацију између корисника било неопходно успостављење везе, ресурски који су на тај начин били заузети нису били доступни за друге потенцијалне размене података у мрежи. Ресурси би били доступни тек након завршетка комуникације између тих корисника, односно након раскидања успостављене везе. То је узроковало слабу искоришћеност ресурса. Као решење тог проблема започето је са коришћењем размене података на бази технике комутације пакета, што је омогућило много већу искоришћеност доступних ресурса.

У овом раду биће представљена симулација пакетског комутатора у ком се распоређивање пакета врши на основу приоритета. Симулација је извршена у програмском језику *C++*.

У другом поглављу биће речи о пакетској комутацији, зашто је углавом погоднија од комутације кола. Такође биће издвојене неке од битнијих карактеристика пакетског комутатора и биће извршена његова класификација са становишта позиције бафера са акцентом на пакетским комутаторима са баферима на улазним портовима.

Треће поглавље биће посвећено самој симулацији и објашњењу битнијих делова кода и биће приказан један пример резултата симулације.

Поглавље 4 ће обухватити бројне вредности кашњења и пропусности на нивоу целе симулације за различите улазне параметре.

Последње поглавље овог рада биће закључак у коме ће бити дат завршни коментар овог рада.

2. ПАКЕТСКИ КОМУТАТОР

У овом поглављу биће речи о пакетским комутаторима, класификацији тих комутатора са становишта позиције бафера, а посебна пажња биће обраћена на комутаторе са баферима на улазу.

2.1.Пакетска комутација

Пренос података се првенствено обављао коришћењем технике комутације кола. Та техника се заснива на принципу заузимања ресурса у мрежи за везу између корисника који желе међусобно да комуницирају, при чему се заузети ресурси могу користити само за размену података у тој успостављеној вези. Иако комутација кола гарантује квалитет сервиса кориснику, велика мана ове технике је слаба искоришћеност заузетих ресурса. Из тог разлога потребна су знатно већа улагања оператера у проширење своје инфраструктуре да би се покрио још већи број корисника. Додавање нових корисника на тај начин би било отежано и успорено, а највероватније би и развој нових сервиса био успоренији.

Као решење претходно наведених проблема технике комутације кола, започето је коришћење технике комутације пакета. Подаци који се размењују се деле на пакете, па се у том облику преносе између корисника кроз мрежу, при чему пролазе кроз мрежне чворове. По уласку у мрежни чвор, одређује се на који излаз мрежног чвора треба послати дотични пакет. Пакет затим чека на свој редослед за слање заједно са пакетима других веза који се шаљу преко истог излаза мрежног чвора. Уколико постоје пакети за слање у мрежном чвору, они се увек прослеђују ка излазима тако да уколико и постоји пауза у једној вези, она не утиче на искоришћеност линка који ће тада да преноси пакете других веза које су у току. Комутација пакета постиже високу искоришћеност ресурса у мрежи, али сада недостаје гаранција квалитета сервиса. На пример, може доћи до преоптерећења појединих делова мреже услед комуникације превеликог броја корисника, а самим тим и до потенцијалних губитака пакета у мрежи. Закључује се да парамери попут кашњења, оствареног протока и др. у великој мери зависе од тренутне ситуације у мрежи, односно тренутних комуникација у току.

У овом делу су издвојене неке од карактеристика које су битне за пакетске комутаторе. Особина скалабилности одређује колико је лако проширивати капацитет комутатора, па је комутатор скалабилнији што је лакше проширивати његов капацитет. Цена представља економичност израде комутатора, док комплексност дефинише како комплексност имплементације комутатора, тако и сложеност управљања комутатором. Следећа битна карактеристика пакетских комутатора јесте подршка *multicast* саобраћаја, па постоје *unicast* и *multicast* пакети.

Са становишта блокаде разликујемо неблокирајуће и блокирајуће пакетске комутаторе. Неблокирајући комутатор је онај код ког се може успоставити веза између произвољног слободног улаза и произвољног слободног излаза без обзира на постојеће

успостављене конекције, у супротном комутатор је блокирајући. Ефекти блокаде се могу умањити употребом принципа убрзања (*speedup*) у пакетским комутаторима, што значи да пакетски комутатор ради на *x* пута већој брзини од улазних/излазних линкова чиме је могуће остварити већи број конфигурација пакетског комутатора у јединици времена, а самим тим и проследити већи број пакета са улаза на излазе у јединици времена.

Пакети који се преносе могу бити пакети промењљиве дужине или пакети фиксне дужине. Пакети фиксне дужине се означавају термином ћелије. Показало се да пакетски комутатор много ефикасније ради и да га је знатно једноставније конфигурисати када су пакети фиксне дужине.

Пропусност комутатора одређује ефикасност, односно капацитет комутатора. Рачуна се за случај када су сви улази потпуно оптерећени (непрестано долазе пакети) и сви излази су потпуно оптерећени (пристигли пакети су намењени одговарајућим излазима тако да ниједан излаз није преоптерећен), и циљ је да пропусност комутатора буде што већа.

С обзиром на то да код пакетских комутатора повремено долази до преоптерећења одређених излазних портова, из разлога што је укупан проток пристиглих пакета намењен том одређеном излазном порту већи од брзине тог порта, неопходно је инсталирати бафере у мрежном чвору који ће чувати пакете да не би дошло до њихових непотребних губитака. У наредном делу биће извршена класификација комутатора са становишта позиције бафера са акцентом на бафере на улазним портовима.

2.2. Класификација пакетских комутатора са становишта позиције бафера

Од саме архитектуре мрежног чвора и типа пакетског комутатора који се користи зависи где ће бити позиционирани бафери у мрежном чвору пакетске мреже. Бафери могу да се поставе на улазним портовима, на излазним портовима или у пакетском комутатору, а дозвољене су и међусобне комбинације. Неке од најпознатијих варијанти су:

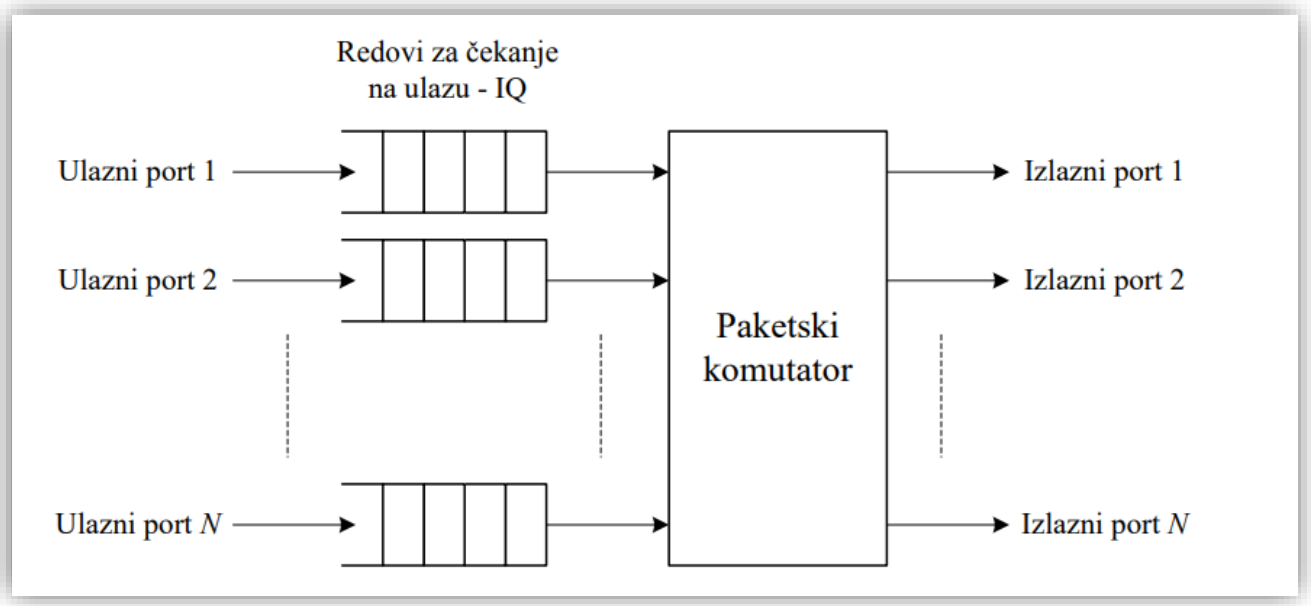
- Бафери на излазним портовима (OQ – *Output Queuing*)
- Бафери на улазним портовима (IQ – *Input Queuing)*
- Бафери на улазним и излазним портовима (CIOQ – *Combined Input and Output Queuing)*
- Заједнички бафер (*Shared Memory Queuing)*
- Бафери у пакетском комутатору (CQ – *Crosspoint Queuing)*

Када се користе комутатори са баферима само на излазним портовима, пакети се са улазних портова прослеђују директно на излазне портове и уписују се у бафере на одговарајућим излазима. Како је могуће да се са више улаза у истом слоту прослеђују пакети за исти излаз, неопходно је обезбедити да се ти пакети прослеђују без одбацивања.

Оно што је за овај рад значајно јесу пакетски комутатори са баферима на улазу, па ће у наредном делу то бити детаљније обрађено.

2.3. Пакетски комутатори са баферима на улазним портовима

Пакети пристигли на улазе пакетског комутатора се смештају у бафере на улазним портовима. Користе се распоређивачи (*scheduler*) који врше упаривање улазних и излазних портова за сваки слот. За упарени пар (*i*,*j*) значи да ће са улазног порта *i* бити прослеђен пакет на излазни порт *j*. На основу тога се у свајом слоту врши конфигурисање пакетског комутатора, тј. повезивање упарених парова улаз/излаз.



Слика 2.3.1. Пакетски комутатор са баферима на улазним портовима.

На слици 2.3.1 је приказана принципска структура комутације пакета у случају када су бафери само на улазним портовима. одговарајућим излазима.

Такође је могуће да комутатор ради са убрзањем *x* па са једног улаза може да се проследи максимално *x* пакета у једном слоту, и на један излаз може да стигне максимално *x* пакета у једном слоту.

Сваки улазни порт има један бафер, па се у току једног слота врши један упис уколико је пристигао пакет на улаз, и врши се једно читање пакета уколико постоји пакет у баферу. Уколико се користи убрзање врши се максимално *x* читања у једном слоту. Мрежни чворови са баферима на улазу су скалабилни и могу да подрже велики број портова. Приликом ове реализације могуће је користити FIFO (*First In First Out*) бафер или бафер са VOQ (*Virtual Output Queuing*) редовима за чекање.

FIFO бафер је једноставнији јер се чувају само пакети и нема додатних информација али његова употреба доводи до HOL(*Head Of Line*) блокаде. У случају када на више улаза у једном слоту дођу пакети за исти излаз, неће моћи сви у том слоту да их проследе, већ ће пакети чекати различит број слотова да би били прослеђени. Како на улазе тих бафера долазе нови бакети, идеално би било да се они могу прослеђивати, али због FIFO правила где се прво морају проследити пакети који су први стигли, тада ће и ти пакети имати неко одређено кашњење. Услед тога може доћи до значајног обарања пропусности пакетског комутатора,

нарочито при великим оптерећењима улазних портова тако да се из тог разлога FIFO бафери више нешто не користе.

Проблем HOL блокаде се избегава коришћењем бафера са VOQ редовима за чекање, који су нешто компликованији за реализацију од FIFO бафера. За разлику од FIFO бафера чији алгоритми за распоређивање раде са N података (N представља број улаза/излаза комутатора), бафери са VOQ редовима за чекање раде са NxN података јер на сваком улазу постоји бафер за сваки излазни порт, па на сваком улазном порту имамо N виртуелних редова за чекање.

Што се тиче упаривања разликују се максимум и максимално упаривање. Максимум упаривање покушава наћи маскималан број упаривања који може да се оствари, док максимално упаривање подразумева инкрементално додавање парова по неком алгоритму све док се нови парови могу додати. У општем случају маскимално упаривање креира мање парова од максимум упаривања, док с друге стране максимум упаривање може да изазове у одређеним саобраћајним случајевима опслуживање токова које није фер, а у екстремним случајевима може чак да доведе и до потпуног неопслуживања појединих токова. [1]

3. СИМУЛАЦИЈА РАСПОРЕЂИВАЊА ПАКЕТА

Оно чиме се овај рад бави јесу пакетски комутатори са баферима са VOQ редовима за чекање који се користе за пренос ћелија, тј. пакета фиксне дужине, где се распоређивање врши на основу приоритета, па ће у овом поглављу бити дати делови кода симулације рађене у програмском језику *C++* као и њихова објашњења, док ће на крају рада, у делу прилога бити доступан и цео код ове симулације.

3.1. Улазни параметри

У симулацији се време једне итерације подудара са трајањем једног слота, колико је трајање и сваког пакета, с обзиром на то да се анализира пренос ћелија. На почетку се дефинишу димензије бафера па је број улаза односно излаза комутатора означен са *N*, и за сваки улаз постоји *N* бафера са редовима за чекање. Број слотова, односно итерација, симулације означен је са *Z*, и у овој симулацији ће износити *Z*=10000. Такође, са *P1* је дефинисано улазно оптерећење које ће бити мењано у симулацији од 0,1 до 0,9. Осим тога на почетку се дефинише и саобраћајни сценарио који можр бити униформни или hot-spot.

```
//broj iteracija simulacije
const int Z = 100000;
//broj ulaza odnosno izlaza paketskog komutatora
const int N = 30;
//ulazno opterecenje
double P1 = 0.5;
```

Слика 3.1.1. Улазни параметри.

3.2. Симулација

Након што су улазни параметри дефинисани, у првом слоту се врши генерисање пакета на свим улазима. Затим, за сваки улаз на ком је заправо изгенерисан пакет, на основу унапред изабраног саобраћајног сценарија се врши избор излаза на који се исти прослеђује. Пакет се смешта у одговарајући ред за чекање у зависности од тога ком је излазу намењен.

У свакој следећој итерацији се генерисање пакета врши након што је завршено прослеђивање пакета који се већ налазе у баферима на излазе. На сваком улазу се бира излаз ка ком се шаље захтев за спајање. Избор је случајан, а тежина избора је пропорционална дужини реда за чекање тог излаза. Када сви улази пошаљу захтеве, излази бирају који захтев прихватају. Излази прихватају захтев од улаза чији је ред за чекање најдужи за тај излаз. Након што је сваки излаз одредио са ког улаза прихвата захтев за спајање, врши се прослеђивање пакета са улаза на излаз у складу са оствареним паровима улаз-излаз.

3.3. Генерисање пакета

```
//generisanje paketa
void generisiPaket() {
    S = ((double)rand() / (RAND_MAX));
    if (S < P1) {
        p = '*';
        broj_generisanih_paketa++;
    }
    else {
        p = NULL;
    }
}
```

Слика 3.3.1. Генерисање пакета.

Креирана је класа *Paket* уз помоћ чије методе *generisiPaket()* се врши генерисање пакета на следећи начин. Генерише се неки случајан број *S* уз помоћ функције *rand()* и величине *RAND_MAX*. Генерисани случајан број се налази у границама између 0 и 1. Уколико је унапред дефинисано улазно оптерећење саобраћаја *P1* веће од случајног броја *S*, пакет ће се генерисати, и глобална променљива *broj_generisanih_paketa* ће се аутоматски инкрементирати за 1. На тај начин се константно чува податак о томе колико је пакета генерисано на улазу од почетка симулације. У супротном, уколико је улазно оптерећење саобраћаја *P1* мање од случајног броја *S*, неће доћи до генерисања пакета на датом улазу у том слоту.

3.3. Саобраћајни сценарио

Првенствено се креира објекат класе *Paket* позивањем конструктора коме се прослеђују параметри о димензијама комутатора, о итерацији у којој се врши генерисање пакета, затим о улазу на ком се генерише пакет, информација о улазном оптерећењу, као и број итерација симулације, након чега се врши генерисање пакета.

```
    Paket(int broj_ulaza_izlaza, int iter_generisanja, int ulaz_generisanja,
double p1, int broj_iteracija) {
        N = broj_ulaza_izlaza;
        iteracija_generisanja = iter_generisanja;
        ulaz = ulaz_generisanja;
        P1 = p1;
        Z = broj_iteracija;
        this->generisiPaket();
    }
```

Слика 3.3.1. Конструктор објекта класе *Paket*.

Уколико је пакет генерисан бира се тип саобраћајног сценарија. У овој симулацији разматрана су два типа саобраћајног сценарија: униформни и hot-spot сценарио, односно на два начина избора излазног порта за пакет који се на почетку генерише.

3.3.1. *Униформни сценарио*

У случају униформног сценарија, вероватноћа да је генерисани пакет намењен неком излазу је подједнака за све излазе у комутатору. Прво се коришћењем if конструкције проверава да ли је пакет генерисан. Уколико јесте променљива *p* неће бити једнака *null* вредности, па с обзиром на то да пакет постоји може се одредити излаз на који треба да буде прослеђен. Генерише се случајан број *P2*, који се затим множи са бројем улаза односно излаза комутатора *N*. Како добијамо реалан број, а потребан нам је излаз који мора бити цео број, коришћењем функције *floor* број заокружујемо на први мањи. Као изузетак, уколико се добије да је *P2=1* а самим тим и *P2*N=N*, у том случају као излаз се бира излаз *N-1* јер се бројање креће од 0, а не од 1, па самим тим имамо улазе/излазе од 0 до *N-1*, а не од 1 до *N*. У супротном, уколико пакет није генерисан, вредност излаза постаје -1, што је неисправна вредност за излаз, а самим тим означава и да не постоји пакет за прослеђивање. Функција *unif()* као повратну вредност враћа вредност променљиве излаз, тј. број излаза на који треба проследити пакет уколико је он генерисан, односно враћа "-1" уколико пакет није генерисан.

```
//uniformni scenario
int unif() {
    //generator saobracaja
    if (p != NULL) {
        P2 = ((double)rand() / (RAND_MAX));
        izlaz = floor(P2 * N);
        if (izlaz == N) izlaz = N - 1;

    }
    else izlaz = -1;
    return izlaz;
}
```

Слика 3.3.1.1. Униформни сценарио.

3.3.2. *Hot-spot сценарио*

У случају *hot-spot* сценарија, повратна вредност променљиве *izlaz* је иста као и код униформног сценарија у случају када пакет није генерисан, што се такође проверава на исти начин у оба случаја.

Карактеристично за овај начин избора излаза за генерисани пакет јесте да у односу на претходни тип, нису сви излази једнако вероватни. За пакет генерисан на улазу *s* вероватноћа прослеђивања на излаз *s* је 0,5 (односно 50%). Вероватноће прослеђивања на излазе који нису једнаки *s* су подједнаке па је преосталих 50% равномерно распоређено међу њима.

Променљивој *P2* се додељује случајна вредност у опсегу од 0 до 1. Променљива *a* чува вредност вероватноће прослеђивања на излаз чији је редни број исти као редни број улаза на ком је пакет генерисан, док је у промељивој *k* смештен који део вероватноће је гарантован осталим излазима.

Доња и горња граница су у првом тренутку једнаке 0. Пролажењем кроз *for* петљу прво доња граница добија вредност горње границе из претходне итерације. Затим, уколико редни број итерације одговара редном броју улаза са ког је дошао пакет горња граница добија вредност доње границе увећане за *a*, а уколико редни број итерације не одговара, тада горња граница добија вредност доње границе увећане за вредност *k*.

За сваку итерацију те петље проверава се да ли случајна вредност *P1* лежи између доње и горње границе. Уколико је одговор потврдан, променљива *izlaz* добија вредност итерације у којој је дошло до поклапања, и генерисани пакет треба проследити на тај излаз.

```
//hot-spot scenario
int hot_spot() {
    //generator saobracaja
    if (p != NULL) {
        P2 = ((double)rand() / (RAND_MAX));
        //verovatnoca izlaza l=j
        double a = 0.5;
        //verovatnoca izlaza l!=j
        double k = 0.5 / (N - 1);

        double donja_granica = 0;
        double gornja_granica = 0;

        for (int l = 0; l < N; l++) {

            donja_granica = gornja_granica;
            if (l == ulaz) gornja_granica = donja_granica + a;
            else gornja_granica = donja_granica + k;

            if (P2 > donja_granica && P2 < gornja_granica) {
                izlaz = l;
            }
        }

    }
    else izlaz = -1;
    return izlaz;
}
```

Слика 3.3.2.1. Hot-spot сценарио.

3.4. Бафер

Након што је на неком улазу генерисан пакет, и у зависности од одабраног саобраћајног сценарија одређен излазни порт на који га треба проследити, врши се његово смештање у бафер. За пакет који је генерисан у *i*-тој итерацији на улазу *j* *data[i][j]* се проверава ком излазу *k* је намењен. Како на сваком улазу имамо онолико редова за чекање колико и излазних портова, пристигли пакет смештамо у онај ред за чекање који одговара излазу коме треба проследити тај пакет.

```
//smestanje u redove za cekanje
for (int k = 0; k < N; k++) {
    if (data[i][j].dohvatiIzlaz() == k) {
        baf[j][k].dodaj(data[i][j]);
    }
}
```

Слика 3.4.1. Смештање у бафер.

За додавање пакета у бафер коришћена је функција *dodaj()* која новопристигли пакет смешта на крај реда за чекање. За реализацију бафера коришћена је *STL (Standard Template Library)* библиотека, и генеричка класа *deque<>* (*double-ended queue)* која заправо омогућава рад са редовима са два краја. Стога је могуће додавати пакет на крај реда приликом доласка новог пакета на улаз, као и узимање пакета са почетка реда у случају када пакет прослеђујемо на излаз, јер комутатор треба да функционише по *FIFO* принципу. [2]

```
void dodaj(const Paket& p) {
    niz.push_back(p);
}
```

Слика 3.4.1. Додавање на крај реда.

3.5. Слање захтева за упаривање

Када у баферу већ постоје пакети, за њихово прослеђивање неопхотно је да се изврши упаривање улаза и излаза где ће доћи до прослеђивања пакета у једном слоту. У првом кораку улазни портови шаљу захтеве за упаривање излазним портовима. На сваком улазу се на случајан начин бира излаз ка ком се шаље захтев за спајање. Тежина избора пропорционална је дужини реда за чекање.

У овом делу код променљивом *r* је означен улазни порт, променљивом *j* излазни порт, а променљивом *i* тренутна итерација. Прва *for* петља пролази кроз све улазне портове и омогућава да се за сваки од њих одреди излазни порт коме се шаље захтев за спајање. У другој *for* петљи се проверава тренутно стање бафера на *r*-том улазном порту, тј. пребројава се колико пакета има у сваком од редова за чекање, као и колико је укупно пакета у свим редовима за чекање на том улазном порту.

Након тога, генерише се случајан број *S* у границама између 0 и 1, и поново се врши избор као и у делу *3.3.2 Hot-spot scenario,* с тим што сада тежински коефицијенти зависе од дужина редова за чекање који су израчунати у другој *for* петљи овог дела кода.

У зависности од тога у којим границама се налази случајна променљива *S*, добија се излаз на који се шаље захтев за слање са улаза за који је прорачун у току.

```
                //SLANJE ZAHTEVA ZA SPAJANJE KA IZLAZU
                //za svaki ulaz ima N mogucih izlaza. svaki izlaz ima red za
cekanje neke duzine
                //ulaz r
                for (int r = 0; r < N; r++) {
                    //broj paketa na ulazu r u iteraciji i
                    broj_paketa[i][r] = 0;
                    zahtev_izlaz[r] = -1;
                    //prebrojavanje paketa u redovima za cekanje
                    for (int j = 0; j < N; j++) {
                        //broj paketa na ulazu r za izlaz j
                        duzina[r][j] = baf[r][j].duzina();
                        broj_paketa[i][r] += duzina[r][j];
                    }

                    double S = ((double)rand() / (RAND_MAX));

                    double donja_granica = 0.0, gornja_granica = 0.0;
                    double k = 1.0 / static_cast<double>(broj_paketa[i][r]);
                    for (int j = 0; j < N; j++) {
                        donja_granica = gornja_granica;
                        gornja_granica = donja_granica + k * duzina[r][j];

                        if (S >= donja_granica && S < gornja_granica) {
                            zahtev_izlaz[r] = j;
                        }
                    }
                }
```

Слика 3.5.1. Одређивање излаза коме се шаље захтев за упаривање.

3.6. Прихватање захтева на излазним портовима

Након што је за сваки улазни порт изабран излазни порт коме се шаље захтев за упаривање у тренутном слоту, узимајући у обзир да постоји могућност да један излазни порт добије захтев за спајање са више улаза, у следећем кораку излази од пристиглих захтева бирају улаз са ког желе да приме пакет. Од пристиглих захтева, излаз прихвата онај улаз у чијем реду за чекање за тај излаз се у том тренутку налази највећи број пакета.

За излаз *o* се на основу претходног бира улаз са ког се прима пакет, и редни број улаза се смешта у променљиву *pom_ulaz.*

Уколико заправо постоји неки улаз који је послао захтев за упаривање посматраном излазу, и након што је извршен избор, врши се прослеђивање пакета за упарени пар улаз-излаз. Такође, инкрементира се вредност променљиве *broj_prosledjenih_paketa* која чува информацију о укупном броју прослеђених пакета од почетка симулације.

```
//svaki izlaz bira ulaz sa kog zeli da primi paket
for (int o = 0; o < N; o++) {
    int duz = 0;
    int pom_ulaz = -1;

    for (int j = 0; j < N; j++) {
        if (zahtev_izlaz[j] == o) {
            if (duz < duzina[j][o]) {
                pom_ulaz = j;
                duz = duzina[j][o];
            }
        }

    }
    if (pom_ulaz != -1) {
        broj_prosledjenih_paketa++;
        izlaz[o] = baf[pom_ulaz][o].uzmi();
        izlaz[o].postaviIteracijuPrijema(i);
        int kasnjenje = izlaz[o].delay();
        suma_kasnjenja += kasnjenje;
        if (kasnjenje > max_kasnjenje)
            max_kasnjenje = kasnjenje;
    }
}
```

Слика 3.6.1. Прихватање захтева на излазним портовима и прослеђивање пакета.

Када се пакет прослеђује, то је у симулацији аналогно уклањању пакета из реда за чекање на улазном порту. Врши се уз помоћ функције *uzmi()* која узима пакет са почетка реда, а самим тим такође испуњава *FIFO* принцип рада, где се прво прослеђују пакети који су први и смештени у бафер.

```
Paket uzmi() {
    if (!prazan()) {
        Paket p = niz.front();
        niz.pop_front();
        return p;
    }
}
```

Слика 3.6.2. Узимање пакета из бафера.

Такође, ажурира се поље објекта класе *Paket* које носи информацију од итерацији у којој је пакет примљен. Додељује му се вредност тренутне итерације. На основу тога могуће је одредити кашњење пакета уз помоћ функције *delay()*.

```
//kasnjenje paketa
int delay() {
    kasnjenje = iteracija_prijema - iteracija_generisanja;
    return kasnjenje;
}
```

Слика 3.6.2. Прорачун кашњења пакета.

Такође, проласком кроз сваку итерацију ажурира се и податак о укупном кашњењу пакета, као и о максималном кашњењу.

3.7. Кашњење пакета и пропусност система

Као битније параметре система пратимо кашњење пакета, при чему издвајамо максимално кашњење и средње кашњење. Такође, битан параметар је и пропусност система.

```
            propusnost = 1.0 * broj_prosledjenih_paketa /
broj_generisanih_paketa;
            srednje_kasnjenje = 1.0 * suma_kasnjenja / broj_prosledjenih_paketa;
            cout <<"PROPUSNOST SISTEMA " << propusnost << endl;
            cout <<"MAKSIMALNO KASNJENJE PAKETA " << max_kasnjenje << endl;
            cout <<"SREDNJE KASNJENJE PAKETA " << srednje_kasnjenje << endl;
```

Слика 3.7.1. Параметри.

3.8. Симулација-излаз

Биће приказан излаз симулације за један пример. Нека број итерација симулације остане *Z=100000*, димензије комутатора *N=8* и улазно оптерећење *P1=0,1*. Као *output* се добија:

```
C:\Users\jt170158d\source\repos\simulacija paketskog komutatora\Debug\simulacija paketskog komutatora...

Ukoliko zelite simulaciju paketskog komutatora ciji je saobracajni scenario uniforman unesite 'unif',
a ukoliko zelite hot-spot saobracajni scenario unesite 'hs'.
hs

PROPUSNOST SISTEMA 0.999931
MAKSIMALNO KASNJENJE PAKETA 7
SREDNJE KASNJENJE SISTEMA 1.04823

Ukoliko zelite simulaciju paketskog komutatora ciji je saobracajni scenario uniforman unesite 'unif',
a ukoliko zelite hot-spot saobracajni scenario unesite 'hs'.
unif

PROPUSNOST SISTEMA 0.999954
MAKSIMALNO KASNJENJE PAKETA 7
SREDNJE KASNJENJE SISTEMA 1.04999

Ukoliko zelite simulaciju paketskog komutatora ciji je saobracajni scenario uniforman unesite 'unif',
a ukoliko zelite hot-spot saobracajni scenario unesite 'hs'.
```

Слика 3.8.1. Симулација.

4. РЕЗУЛТАТИ СИМУЛАЦИЈЕ

У овом поглављу биће представљени резултати симулације за оба типа раније споменутих саобраћајних сценарија: за униформни и *hot-spot* тип саобраћаја. Такође биће узете у обзир различите вредности улазног оптерећења, као и броја улаза/излаза (за *N=8* и *N=16*). Резултати ће прво бити представљени табеларно, а затим и графички.

4.1. Униформни тип саобраћаја

У табели 4.1.1. дат је приказ пропусности, максималног и средњег кашњења пакета у зависности од улазног оптерећења за пакетски комутатор са 8 улаза и 8 излаза.

Табела 4.1.1. Униформни тип саобраћаја за комутатор са 8 улаза/излаза

УЛАЗНО ОПТЕРЕЋЕЊЕ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
ПРОПУСНОСТ[%]	99.9988	99.9988	99.9992	99.9988	99.997	99.9942	93.6898	82.0619	72.9614	65.571
МАКСИМАЛНО КАШЊЕЊЕ	7	10	16	31	59	86	6950	18635	28375	35455
СРЕДЊЕ КАШЊЕЊЕ	1.05438	1.13504	1.27647	1.57403	2.36974	5.70819	3144.68	8978.05	13552.4	17231.4

У табели 4.1.2. дат је приказ пропусности, максималног и средњег кашњења пакета у зависности од улазног оптерећења за пакетски комутатор са 16 улаза и 16 излаза.

Табела 4.1.2. Униформни тип саобраћаја за комутатор са 16 улаза/излаза

УЛАЗНО ОПТЕРЕЋЕЊЕ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
ПРОПУСНОСТ[%]	99.9994	99.9987	99.9992	99.9975	99.9979	99.9889	91.9854	80.4664	71.5155	64.4046
МАКСИМАЛНО КАШЊЕЊЕ	7	11	29	74	89	161	9337	21136	30621	38085
СРЕДЊЕ КАШЊЕЊЕ	1.05816	1.14874	1.3138	1.65071	2.76675	9.16491	4004.54	9768.2	14241.8	17789

Код униформног типа саобраћаја се може закључити да је за мање димензије пакетског комутатора, тј. за комутатор који има мањи број улазних односно излазних портова пропусност већа, а максимално и средње кашњење мањи. За улазно оптерећење до вредности 0.6 је прихватљиво максимално кашњење пакета, а с тим и средње кашњење пакета. Сама пропусност је тад за обе проверене димензије комутатора већа од 99.98%.

4.2.*Hot-spot* тип саобраћаја

У табели 4.2.1. дат је приказ пропусности, максималног и средњег кашњења пакета у зависности од улазног оптерећења за пакетски комутатор са 8 улаза и 8 излаза.

Табела 4.2.1. *Hot-spot* тип саобраћаја за комутатор са 8 улаза/излаза

УЛАЗНО ОПТЕРЕЋЕЊЕ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
ПРОПУСНОСТ	99.9825	99.9869	99.99	99.9843	99.9868	99.9837	95.6505	83.9994	74.8741	67.5912
МАКСИМАЛНО КАШЊЕЊЕ	5	11	18	33	50	106	6543	21774	33548	41867
СРЕДЊЕ КАШЊЕЊЕ	1.0422	1.10902	1.22072	1.42676	1.9613	3.92276	2150.52	7822.35	12253.8	15737.1

У табели 4.2.2. дат је приказ пропусности, максималног и средњег кашњења пакета у зависности од улазног оптерећења за пакетски комутатор са 16 улаза и 16 излаза.

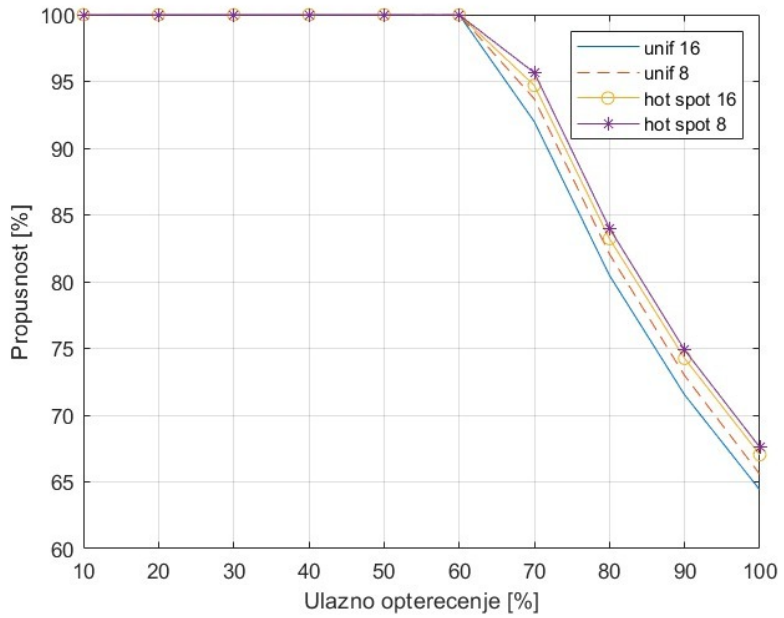
Табела 4.2.2. *Hot-spot* тип саобраћаја за комутатор са 16 улаза/излаза

УЛАЗНО ОПТЕРЕЋЕЊЕ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
ПРОПУСНОСТ	99.9956	99.9912	99.9927	99.9928	99.9915	99.9903	94.7007	83.2169	74.2493	67.037
МАКСИМАЛНО КАШЊЕЊЕ	7	10	22	43	66	178	8635	24239	35216	44486
СРЕДЊЕ КАШЊЕЊЕ	1.04459	1.11393	1.22635	1.45694	2.07101	4.94802	2610.15	8253.4	12552.6	15984.4

Код *hot-spot* типа саобраћаја је ситуација релативно слична као код униформног типа. Максимално кашњење и средње кашњење нису велики за вредности улазног оптерећења испод 0,6. Након тога средње кашњење постаје незадовољавајуће велико а самим тим и пропусност опада.

4.3.Графички приказ резултата симулације

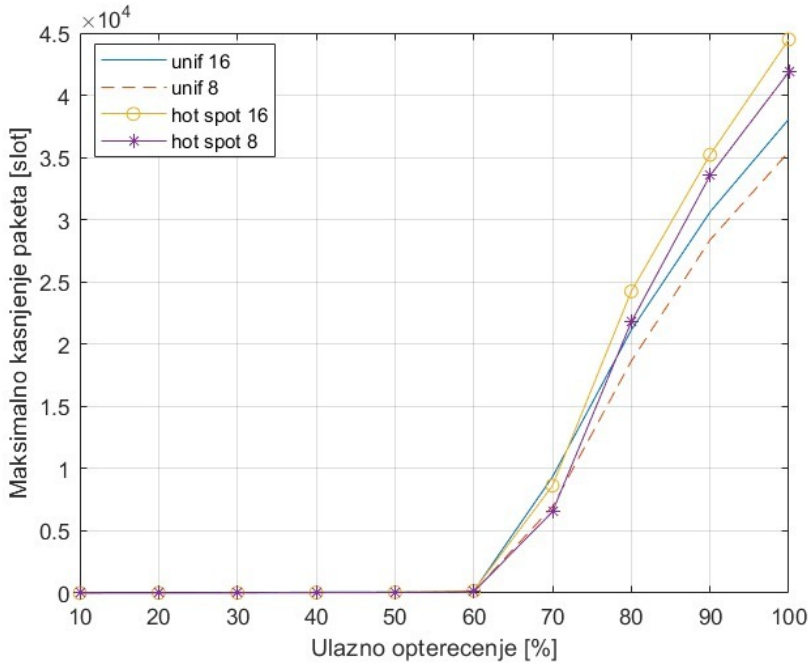
У овом одељку биће графички приказана зависност пропусности и кашњења у зависности од улазног оптерећења комутатора. За улазна оптерећења до 60% одступања су минимална па је на графицима видљива разлика за различите димензије комутатора и за различите типове саобраћаја тек за улазна оптерећења већа од 60%, па ће само тај случај бити коментарисан у овом одељку.



Слика 4.3.1. Зависност пропусности система од улазног оптерећења

На слици 4.3.1. графички је приказана зависност пропусности система од улазног оптерећења за оба начина генерисања саобраћаја, као и за димензије комутатора *N=8* и *N=16*.

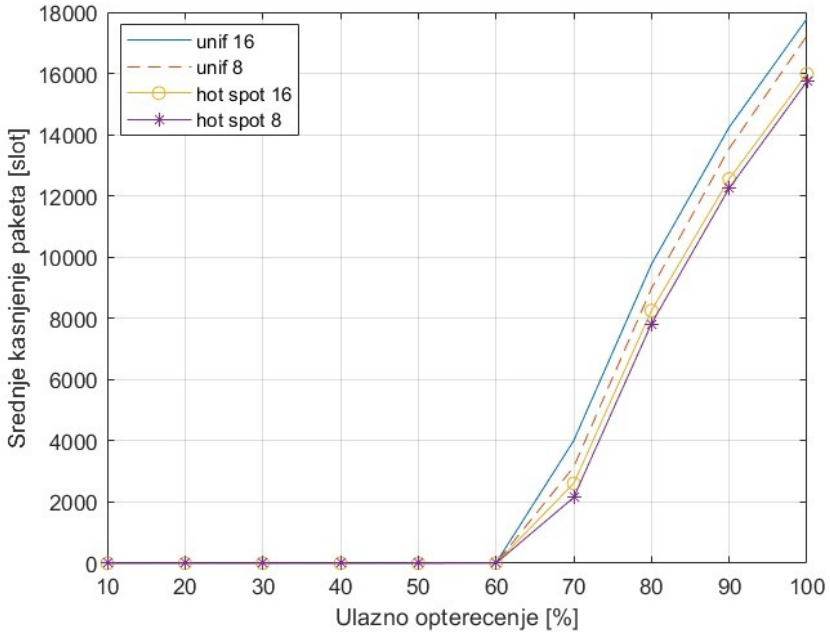
Може се приметити да је пропусност мања код униформног типа генерисања саобраћаја него код *hot-spot* типа. Такође, пропусност је већа за пакетски комутатор мањих димензија.



Слика 4.3.2. Зависност максималног кашњења пакета од улазног оптерећења

На слици 4.3.2. графички је приказана зависност максималног кашњења пакета од улазног оптерећења за оба начина генерисања саобраћаја, као и за димензије комутатора *N=8* и *N=16*.

Максимално кашњење пакета је веће код пакетског комутатора већих димензија, и то за *hot-spot* тип саобраћаја.



Слика 4.3.3. Зависност средњег кашњења пакета од улазног оптерећења

На слици 4.3.3. графички је приказана зависност средњег кашњења пакета од улазног оптерећења за оба начина генерисања саобраћаја, као и за димензије комутатора *N=8* и *N=16*.

Иако је на претходном графику било могуће приметити да је максимално кашњење пакета веће код *hot-spot* типа саобраћаја, са графика на слици 4.3.3 се види да је средње кашњење за тај тип саобраћаја мање у односу на униформни тип. Стога, закључујемо да је *hot-spot* тип саобраћаја погоднији у случају улазног оптерећења које је веће од 60%.

5.Закључак

Постоје различити алгоритми према којима распоређивач смешта пакете у бафере и врши њихово прослеђивање. Оно што је карактеристично за овај алгоритам који врши распоређивање пакета на основу приоритета јесте добра пропусност као и занемарљиво кашњење пакета за улазно оптерећење мање од 60%. За оба типа саобраћајног сценарија, као и за обе димензије комутатора пропусност у том случају опадне за мање од 0,02%.

Међутим, за улазно оптерећење веће од 60%, кашњења пакета се повећавају више него што може да се толерише, а самим тим и пропусност опада. Из табела датих у претходном одељку се може закључити да за улазно оптерећење од 100% пропусност опада и до 36%, што наравно није одговарајуће.

Може се закључити да је овај алгоритам за распоређивање пакета погодан за комутаторе чије улазно оптерећење није веће од 60%. Такође, што су мање вредности улазног оптерећења перформансе система су боље. У погледу пропусности, погодније је користити комутаторе мањих димензија, јер је губитак пакета мањи.

Литература

[1] Зоран Чича, Материјали са предавања из предмета Комутациони системи

[2] Ласло Краус, *Programski jezik C++ sa rešenim zadacima,* Академска мисао, 2015

Списак скраћеница

CIOQ	*Combined Input and Output Queuing*
CQ	*Crosspoint Queuing*
FIFO	*First In First Out*
HOL	*Head Of Line*
IQ	*Input Queuing*
OQ	*Output Queuing*
STL	*Standard Template Library*
VOQ	*Virtual Output Queuing*

СПИСАК СЛИКА

СПИСАК ТАБЕЛА

A. Програмски код

У овом прилогу биће дат комплетан програмски код симулације који се састоји из класа *Paket.h* и *Red.h*, као и главне функције *main.cpp*.

A.1 *Paket.h*

```
#ifndef _paket_h_
#define _paket_h_

#include <stdlib.h>
#include <iostream>

using namespace std;

static int broj_prosledjenih_paketa = 0;
static int broj_generisanih_paketa = 0;
static int max_kasnjenje = 0;
static double suma_kasnjenja = 0;

class Paket {

private:
	//definicija promenljivih
	int iteracija_generisanja;
	int iteracija_prijema;
	int N;
	int Z;
	int ulaz;
	int izlaz = -1;
	double S;
	double P1;
	double P2;
	char p = NULL;
	int kasnjenje = 0;

	//generisanje paketa (generisemo paket ukoliko je slucajan broj S manji od zadatog broja P1
	void generisiPaket() {
		S = ((double)rand() / (RAND_MAX));
		if (S < P1) {
			p = '*';
			broj_generisanih_paketa++;
		}
		else {
			p = NULL;
		}
	}
	//kopiranje paketa
	void kopiraj(const Paket& p) {
```

```
            this->iteracija_generisanja = p.iteracija_generisanja;
            this->iteracija_prijema = p.iteracija_prijema;
            this->N = p.N;
            this->Z = p.Z;
            this->P1 = p.P1;
            this->p = p.p;
            this->ulaz = p.ulaz;
            this->izlaz = p.izlaz;
            this->kasnjenje = p.kasnjenje;
      }

public:

      //konstruktor / vrsi generisanje paketa na ulazu
      Paket() = default;
      Paket(int broj_ulaza_izlaza, int iter_generisanja, int ulaz_generisanja,
double p1, int broj_iteracija) {
            N = broj_ulaza_izlaza;
            iteracija_generisanja = iter_generisanja;
            ulaz = ulaz_generisanja;
            P1 = p1;
            Z = broj_iteracija;
            this->generisiPaket();
      }
      //kopirajuci konstruktor
      Paket(const Paket& l) { kopiraj(l); }
      //premestajuci konstruktor
      Paket(Paket&& l) { kopiraj(l); }
      //kopirajuci operator=
      Paket& operator=(const Paket& l) {
            if (&l != this) { kopiraj(l); }
            return *this;
      }
      //premestajuci operator=
      Paket& operator=(Paket&& l) {
            if (&l != this) { kopiraj(l); }
            return *this;
      }
      //dohvatanje ulaza na kom je generisan paket
      int dohvatiUlaz() {
            return ulaz;
      }
      //dohvatanje izlaza na koji se prosledjuje paket
      int dohvatiIzlaz() {
            return izlaz;
      }
      //saobracajni scenario (odredjujemo na koji izlaz se prosledjuje paket)
      //uniformni scenario
      int unif() {
            //generator saobracaja
            if (p != NULL) {
                  P2 = ((double)rand() / (RAND_MAX));
                  izlaz = floor(P2 * N);
                  if (izlaz == N) izlaz = N - 1;
            }
            else izlaz = -1;
            return izlaz;
      }
      //hot-spot scenario
```

```
	int hot_spot() {
		//generator saobracaja
		if (p != NULL) {
			P2 = ((double)rand() / (RAND_MAX));
			//verovatnoca izlaza l=j
			double a = 0.5;
			//verovatnoca izlaza l!=j
			double k = 0.5 / (N - 1);

			double donja_granica = 0;
			double gornja_granica = 0;

			for (int l = 0; l < N; l++) {

				donja_granica = gornja_granica;
				if (l == ulaz) gornja_granica = donja_granica + a;
				else gornja_granica = donja_granica + k;

				if (P2 > donja_granica && P2 < gornja_granica) {
					izlaz = l;
				}
			}
		}
		else izlaz = -1;
		return izlaz;
	}
	//iteracija u kojoj paket stize na prijem
	void postaviIteracijuPrijema(int x) {
		iteracija_prijema = x;
	}
	//kasnjenje paketa
	int delay() {
		kasnjenje = iteracija_prijema - iteracija_generisanja;
		return kasnjenje;
	}
	//ispis
	friend ostream& operator<<(ostream& it, const Paket& r) {
		if (r.p != NULL)
			it << "Ulaz " << r.ulaz << " izlaz " << r.izlaz << " iteracija
" << r.iteracija_generisanja << endl;
		return it;
	}

};


#endif // !paket.h
```

A.2 *Red.h*

```
#ifndef _red_h_
#define _red_h_

#include "Paket.h"
#include <deque>
```

```
class Red {

public:

	deque<Paket> niz;

	void dodaj(const Paket& p) {
		niz.push_back(p);
	}
	Paket uzmi() {
		if (!prazan()) {
			Paket p = niz.front();
			niz.pop_front();
			return p;
		}
	}
	int duzina() const { return niz.size(); }
	bool prazan() const { return duzina() == 0; }
	friend ostream& operator<<(ostream& it, const Red& r) {
		for (int i = 0; i < r.duzina(); it << r.niz[i++]);
		return it;
	}

};


#endif // !red.h
```

A.3 *main.cpp*

```
#include<iostream>
#include<string>

#include "Red.h"

using namespace std;

int main() {

	srand((unsigned)time(NULL));

	double propusnost;
	double srednje_kasnjenje;


	//broj iteracija simulacije
	const int Z = 100000;
	//broj ulaza odnosno izlaza paketskog komutatora
	const int N = 8;
	//ulazno opterecenje
	double P1 = 0.1;
	//matrica paketa
	Paket data[Z][N];
	Paket izlaz[N];
	Red baf[N][N];
```

```
	//duzina reda za cekanje na i-tom ulazu za j-ti izlaz NxN
	int duzina[N][N] = { 0 };
	//broj paketa na N ulaza
	int broj_paketa[Z][N] = { 0 };
	//izlaz na koji se salje zahtev za slanje sa i-tog ulaza
	int zahtev_izlaz[N];
	string saobracaj;
	while (1) {
		cout << "Ukoliko zelite simulaciju paketskog komutatora ciji je saobracajni scenario uniforman unesite 'unif', a ukoliko zelite hot-spot saobracajni scenario unesite 'hs'." << endl;
		cin >> saobracaj;

		for (int i = 0; i < Z; i++) {
			// PROSLEDJIVANJE PAKETA
			if (i > 0) {
				//SLANJE ZAHTEVA ZA SPAJANJE KA IZLAZU
				//za svaki ulaz ima N mogucih izlaza. svaki izlaz ima red za cekanje neke duzine
				//ulaz r
				for (int r = 0; r < N; r++) {
					//broj paketa na ulazu r
					broj_paketa[i][r] = 0;
					zahtev_izlaz[r] = -1;

					//za izlaz j. moraju 2 for petlje jer se u prvoj broji koliko ukupno imamo paketa na nekom ulazu
					for (int j = 0; j < N; j++) {
						//broj paketa na ulazu i za izlaz j
						duzina[r][j] = baf[r][j].duzina();
						broj_paketa[i][r] += duzina[r][j];
					}

					double S = ((double)rand() / (RAND_MAX));

					double donja_granica = 0.0, gornja_granica = 0.0;
					double k = 1.0 / static_cast<double>(broj_paketa[i][r]);
					for (int j = 0; j < N; j++) {
						//ulaz r
						//broj_paketa[i]
						//duzine paketa duzina[i][j]
						donja_granica = gornja_granica;
						gornja_granica = donja_granica + k * duzina[r][j];

						if (S >= donja_granica && S < gornja_granica) {
							zahtev_izlaz[r] = j;

						}
					}
				}
				//za svaki izlaz se bira ulaz sa kog zeli da primi paket
				for (int o = 0; o < N; o++) {
					int duz = 0;
					int pom_ulaz = -1;

					for (int j = 0; j < N; j++) {
```

```
                                    if (zahtev_izlaz[j] == o) {
                                          if (duz < duzina[j][o]) {
                                                pom_ulaz = j;
                                                duz = duzina[j][o];
                                          }
                                    }

                              }
                              if (pom_ulaz != -1) {
                                    broj_prosledjenih_paketa++;
                                    izlaz[o] = baf[pom_ulaz][o].uzmi();
                                    izlaz[o].postaviIteracijuPrijema(i);
                                    int kasnjenje = izlaz[o].delay();
                                    suma_kasnjenja += kasnjenje;
                                    if (kasnjenje > max_kasnjenje)
                                          max_kasnjenje = kasnjenje;
                              }
                        }
                  }
                  //GENERISANJE PAKETA
                  for (int j = 0; j < N; j++) {
                        //generisemo paket u i-toj iteraciji i j-tom ulazu
                        Paket genericki(N, i, j, P1, Z);
                        if (saobracaj == "unif") genericki.unif();
                        else if (saobracaj == "hs") genericki.hot_spot();
                        data[i][j] = genericki;
                        //paketi su generisani i odredjen je izlaz na koji treba
da se proslede

                        //iteracija- proveravamo sve iteracije do trenutne
                        for (int k = 0; k < N; k++) {
                              if (data[i][j].dohvatiIzlaz() == k) {
                                    //red za cekanje na i-tom ulazu za j-ti
izlaz
                                    baf[j][k].dodaj(data[i][j]);
                              }
                        }
                  }
            }

            if (saobracaj == "hs" || saobracaj == "unif") {
                  propusnost = 1.0 * broj_prosledjenih_paketa /
broj_generisanih_paketa;
                  srednje_kasnjenje = 1.0 * suma_kasnjenja /
broj_prosledjenih_paketa;
                  cout << " \nPROPUSNOST SISTEMA " << propusnost <<  endl;
                  cout << " \nMAKSIMALNO KASNJENJE PAKETA " << max_kasnjenje <<
endl;
                  cout << " \nSREDNJE KASNJENJE SISTEMA " << srednje_kasnjenje
<< endl;
            }
            cout << endl;
      }
      return 0;
}
```