SVEUCILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RACUNARSTVA

DIPLOMSKI RAD br. 2443

Samoadaptivno višekriterijsko usmjeravanje

Stjepan Matijaševic

Zagreb, rujan 2004.

Zahvaljujem se prof. dr. sc. Draganu Jevticu na strucnoj pomoci i savjetima pri izradi ovog diplomskog rada, kao i roditeljima koji su mi pružali potporu tijekom cijelog studija.

Sadržaj

Uvod

Brojnost korisnika i svojstva u telekomunikacijskoj mreži vec danas su na visokoj razini i zahtjeva primjenu sofisticiranih modela za podršku usluge, te za upravljanje mrežnim resursima. Zahtjevi korisnika koji su takoder podložni razlicitim utjecajima, a i primjena klasicnih metoda usmjeravanja mogu uzrokovati neravnomjerno opterecenje cvorova u mreži. To znaci da je dio cvorova u mreži "zagušen", na primjer pozivima ili paketima, dok drugi rade ispod svojih mogucnosti. U takvoj situaciji namece se misao kako bi prirodno bilo prema potrebi rasporedivati opterecenje u mreži. Da bi to jedan algoritam uspio on mora biti "pametan" tj. mora biti u stanju uciti iz interakcije s okolinom. U ovom diplomskom radu prikazan je jedan moguci nacin balansiranja opterecenja u dijelovima paketske mreži. Algoritam implementira svojstvo adaptivnosti, a pripada podrucju strojnog ucenja (engl. *Machine Learning*). Primjenom algoritma nastoji se oponašati elemente ljudskog rezoniranja kroz ucenje temeljeno na pokušajima i pogreškama što cini osnovu odabranog mehanizam.

Balansiranje opterecenja je kljucno za paralelne procese buduci da osigurava dobru iskoristivost kapaciteta procesorskih jedinica. Samoadaptivnost kao kljucno svojstvo u procesu balansiranja opterecenja je prijeko potrebno uvijek kada ponašanje sistema nije predvidivo i kada nikakvo prijašnje znanje o sustavu nije poznato. Izrada ovog zadatka vezana je za aplikaciju koja je osjetljiva na protok paketa od klijenta (davatelja paketa) do servera (obraduje pakete) i na opterecenost servera (svaki paket opterecuje procesor servera za unaprijed odredeni postotak).

Ideja balansiranja sadržana je u promatranju svih servera kao neovisnih agenata *ucenih ojacavanjem* (engl. *Reinforcement Learning*, skraceno RL), koji pokušavaju nauciti koju velicinu podataka trebaju tražiti od klijenta tako da minimiziraju ukupno vrijeme izvodenja paralelnih procesa kao i ukupno opterecenje servera. Buduci da

agenti dijele zajednicki cilj, s teorijskih gledišta možemo promatrati odabrani model kao moguci ishod za rješavanje zadatka koordinacije.

U prvom poglavlju ovog diplomskog govorimo o *ucenju s ojacavanjem*. Ova vrsta strojnog ucenja pripada kontinuiranom strojnom ucenju u kojem RL agent uci iz interakcije s okolinom. Upoznajemo se s dijelovima RL sustava i Q – algoritmom koji se koristi u aplikaciji. Protokoli koji se koriste te njihove karakteristike opisani su u drugom poglavlju. U trecem poglavlju predstavljen je zadatak, njegovo matematicko rješenje, opis mrežne okoline te se upoznajemo s izvedbom modela klijenta i servera. Opis izvedbe programske podrške kao i uputstva za korištenje aplikacije nalaze se u cetvrtom poglavlju. Rezultati provedenih simulacija i zakljucak prikazani su i komentirani u šestom poglavlju.

1. Ucenje s ojacavanjem

Ucenje ojacavanjem je ucenje što raditi – kako bilježiti stanja u akcije – kao i kako maksimizirati numericku nagradu. RL agentu nije unaprijed receno koje akcije treba koristiti, kao što je to u vecini obrazaca strojnog ucenja, nego on treba otkriti koje akcije daju najvecu nagradu. Akcije mogu utjecati ne samo na trenutnu nagradu, nego i na sljedecu akciju, te zbog toga na cijeli skup nagrada. Te dvije karakteristike, pokušaji i pogreške te odgodeno nagradivanje, dvije su najvažnije razlicite osobine RL.

RL nije definiran karakteristicnim ucecim algoritmom nego karakteristicnim zadatkom. Svaki algoritam koji je primjeren rješavanju karakteristicnog zadatka smatramo RL algoritam. Osnovna ideja je da RL agent u interakciji s okolinom ostvari svoj cilj. Ocito je da takav agent mora biti u stanju da "osjeti" stanje okoline do neke velicine i mora biti u stanju odabrati akciju koja utjece na stanje. Agent takoder mora imati cilj vezan za stanje okoline.

Jedan od izazova koji se pojavljuju u RL, a ne u drugim nacinima ucenja je ovisnost izmedu istraživanja i iskorištavanja naucenog. Da bi održali veliki iznos nagrada, RL agent mora izvoditi akcije koje je u prošlosti izvodio i za koje je otkrio da su efikasne u postizanje (vecih) nagrada. Ali da bi agent otkrio takve akcije mora izabrati i akcije koje nikad prije nije izabrao. Agent mora iskoristiti ono što vec zna da bi postizao vece nagrada, ali isto tako mora istraživati da bi u buducnosti mogao izabrati bolje akcije s kojima ce postizati još vece nagrade. Niti istraživanju kao ni iskorištavanju ne možemo pristupiti iskljucivo, a da pri tome ne dode do pogreške. Agent mora isprobati razne akcije i progresivno favorizirati ona koja mu se cine najbolja.

1.1. Dijelovi RL ucenja

Standardni RL model sastoji se od agenta koji je u interakciji s okolinom. Tije kom interakcije agenta s okolinom, agent "osluškuje" okolinu i u odnosu na podražaje koje dobiva od okoline odabire akciju koju ce izvoditi u okolini. Akcija mijenja okolinu i tu promjenu prima agent kao ojacavajuci signal. Cetiri su osnovna dijela RL modela ucenja: strategija, funkcija nagrada, funkcija vrijednosti i model okoline.

1.1.1. Strategija

Strategija (engl. *policy*) definira ponašanje agenta u danom vremenu. Grubo govoreci, strategija je pridodavanje (engl. *mapping*) iz stanja okoline u akciju koja ce biti odabrana u tom stanju. U nekim slucajevima strategija može biti jednostavna funkcija ili tablica s popisom stanja i mogucih akcija. Strategija je jezgra RL agenta s obzirom da je ona dovoljna da odredi njegovo ponašanje.

1.1.2. Funkcija nagrada

Funkcija nagrade (engl. *reward function*) definira cilj RL zadatka. Ona pridodaje zapažena stanja (parove stanje-akcija) okoline u jedan broj, nagradu, koji pokazuje poželjnost tog stanja. Cilj RL agenta je maksimiziranje nagrade koju dobiva. Funkcija nagrade definira koje su akcije dobre, a koje su loše za njega.

1.1.3. Funkcija vrijednosti

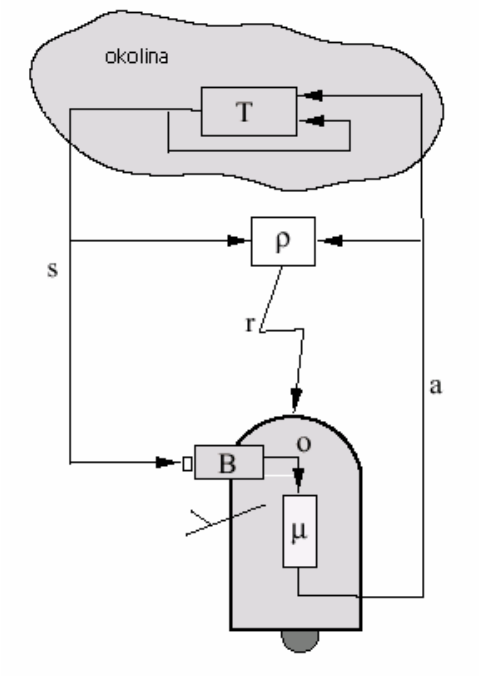
Dok funkcija nagrade (engl. *value function*) pokazuje koja je akcija dobra u pojedinom stanju, funkcija vrijednosti specificira što je dobro "na duge staze". Vrijednost stanja je ukupna vrijednost nagrade koju agent može ocekivati u buducnosti pocevši od pocetnog stanja do krajnjeg stanja. Dok nagrada odreduje trenutnu poželjnost stanja okoline, vrijednost pokazuje poželjnost na duže vrijeme.

1.1.4. Model okoline inteligentnog agenta

Okolina je svijet koji okružuje agenta. Promatra se kao skup (konacan ili beskonacan) stanja zajedno s pravilima kako akcije koje agent izvodi djeluju na promjenu stanja i kako agent biva nagraden za izvodenje akcija. Promjenu stanja okoline može uzrokovati i protjecanje vremena.

1.1.5. Osnovna svojstva ucenja s ojacavanjem

Dakle, *ucenje s ojacavanjem* je proces ucenja u kojem se neki sustav ponaša optimalno u skladu s nekim povratnim vrijednostima iz sustava u nekom periodu vremena. Sistem koji, na pocetku, ne zna koji je odabir akcija najbolji za pojedino stanje. Dobivši odredeni podražaj od okoline u stanju *s(t)* u vremenu *t* agent bira akciju *a(t)* poštivajuci neko pravilo koje nazivamo strategija i oznacavamo s m. Ova akcija je izlaz iz agenta. Ucinak akcije na okolinu ocituje se u vrijednosti ***r(t)*** koji se naziva nagrada.



Sl. 1.1 RL model

Okolina, pod utjecajem neke transformacije, mijenja stanje iz *s(t)* u novo stanje *s(t+1)*.

Gore navedeni zadatak možemo opisati kao Markovljev proces odlucivanja (MDP) kao skup o 4 elementa $\{S,A,T,r\}$ gdje je:

- S – skup stanja
- A – skup akcija
- $T:S \times A \to P(S)$ – definira novo stanje za svaki par stanje-akcija
- $r:S \times A \to R$ – definira numericku vrijednost nagrade koja se dodjeljuje agentu za svaki par stanje-akcija. Pretpostavljamo da je nagrada $r(s,a)$ ogranicena vrijednost s r_{max} za svako stanje i akciju.

Rješenje pronalaženja optimalne strategija su razliciti postupci iz podrucja dinamicnog programiranja. Najpoznatiji postupci iz podrucja RL ucenja su *temporal difference* (TD) i Q-ucenje.

1.2. Q – ucenje

Postupak Q ucenja definira funkciju *Q:S´A®R*. Funkcija $Q(x_t,u_t)$ definira maksimalnu vrijednost nagrade koju agent može ostvariti primjenjujuci akciju x u stanju u. Funkcija Q racuna se po formuli :

$$Q(x_t, u_t) = r(x_t, u_t) + g \max_{u_{t+1}} Q(x_{t+1}, u_{t+1}) \qquad (1)$$

Funkcija *max* vraca maksimalnu vrijednost argumenta uz promjenjivu vrijednost od u.

1.2.1. Postupak Q – ucenja

Rekurzivna definicija funkcije Q omogucuje korištenje iterativnog algoritma za pronalaženje Q funkcije. Vrijednosti Q funkcije moguce je predstaviti dvodimenzionalnom tablicom ciji su stupci stanja x ,a redovi akcije u. Postupak pronalaženja Q funkcije je sljedeci:

Inicijalizirati vrijednosti Q funkcije za svaki par stanje-akcija (x,u) na nulu (ili neku slucajnu vrijednost)

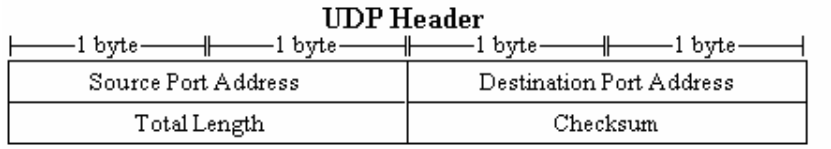
1. Zadati broj iteracija
2. Ponavljati sljedece korake zadani broj puta
    a. trenutno stanje je x
    b. izvesti novu akciju u
    c. novo stanje je x'
    d. primljena nagrada za izvodenje akcije je r
    e. obnoviti vrijednosti $Q(x_t,u_t) = r(x_t,u_t) + g \max_{u_{t+1}} Q(x_{t+1}', u_{t+1}')$
    f. x postaje x'

2. Protokoli, adrese i paketi

2.1. User Datagram Protocol

User datagram protocol (UDP) omogucuje slanje paketa s minimalnim mehanizmom protokola. Protokol je transakcijski orijentiran, a sigurnost isporuke paketa i zaštita od duplikata (istih paketa) nije zajamcena. Aplikacije koje zahtijevaju primanje paketa po redoslijedu slanja i zahtijevaju sigurnost isporuke trebale bi koristiti *Transmission Control Protocol* (TCP).

UDP protokol podrazumijeva korištenje *Internet Protocol-a* (IP) u mrežnom dijelu OSI modela.



Sl. 2.1 UDP zaglavlje

2.1.1. Polja zaglavlja UDP paketa

Source Port Address (16 bita) - predstavlja port aplikacije koja šalje pakete i može se pretpostaviti da predstavlja i port na koji eventualno treba adresirati odgovor u slucaju odsutnosti drugih podataka.

Destination Port Address (16 bita) - adresa aplikacije koja ce primati pakete

Total Length (16 bita) - ukupna velicina paketa u byte-ovima koja ukljucuje velicinu header-a i velicinu podataka

Cheksum (16 bita) - služi za detektiranje pogreške

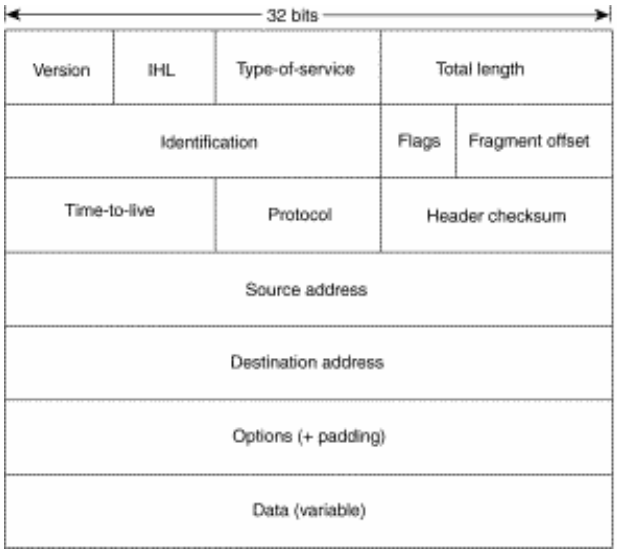
Detaljnije informacije o ovom protokolu potražite u dokumentu RFC 768.

2.2. Internet protokol

Protokol IP je mrežni protokol koji sadrži informacije o adresi i neke kontrolne informacije koje omogucuju usmjeravanja paketa. Zajedno s TCP protokolom predstavlja jezgru Internet protokola. Protokol IP ima dva primarna zadatka:

- mora pružati *connectionless, best effort delivery* datagrama kroz internet mrežu
- fragmentaciju i ponovno sastavljanje paketa

2.2.1. Format IP paketa



Sl. 2.2 Format IP paketa

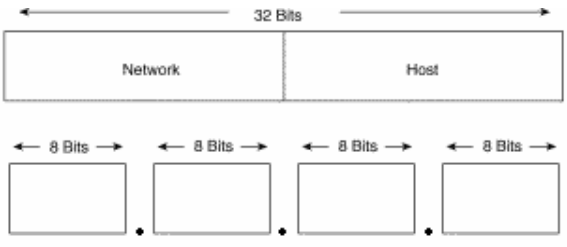
Opis polja IP paketa:

- *Version* – oznacava verziju protokola IP
- *IP Header Length* (IHL) – predstavlja velicinu zaglavlja paketa
- *Type-of-Service* – definira nacin ponašanja višeg sloja OSI modela s paketom i dodjeljuje paketu razne stupnjeve važnosti

- *Total Length* – predstavlja velicinu cijelog paketa u byte-ovima, ukljucuje velicinu podataka i zaglavlja
- *Identification* – predstavlja broj paketa. Ovo polje pomaže prilikom sastavljanja paketa u cjelinu.
- *Flags* – ovo polje sadrži 3 bita od koja zadnja dva (manje važna) kontroliraju fragmentaciju (podjelu velikog paketa na više manjih)
- *Fragment Offset*– oznacava poziciju paketa relativno u odnosu na pocetak originalnog paketa što omogucuje odredišnom IP procesu pravilnu rekonstrukciju paketa.
- *Time-to-Live* – brojaca koji su smanjuje prema nulu kada se paket izbacuje iz mreže. Sprjecava beskonacno kruženje paketa po mreži
- *Protocol* – oznacava koji ce protokol iz višeg sloja OSI modela primiti paket od protokola IP
- *Header Checksum* – pomaže u ocuvanju integriteta IP zaglavlja
- S*ource Address* – adresa pošiljatelja
- *Destination Address* – adresa primatelja
- *Options* – dopušta protokolu IP razne opcije, npr. sigurnost
- *Dana* – podaci

2.2.2. Format IP adrese

32 bitna IP adresa je grupirana u grupe po osam bita medusobno odvojene tockom i predstavljene u decimalnom formatu. Minimalna vrijednost okteta je 0, a maksimalna je 255.



Sl. 2.3 Format IP adrese

2.2.3. Klase IP adrese

IP adresiranje podržava pet razlicitih klasa. To su: A, B, C, D i E. Samo su klase A, B i C dostupne za komercijalnu uporabu. Lijevi (znacajniji) bitovi oznacavaju mrežnu klasu, dok ostali oznacavaju racunalo u mreži.

Opis klasa:

- klasa A – raspon adrese 1.0.0.0 – 126.0.0.0, oblik adrese N.H.H.H[1]
- klasa B – raspon adrese 128.1.0.0 – 191.254.0.0, oblik adrese N.N.H.H
- klasa C – raspon adrese 192.0.1.0 – 223.255.254.0, oblik adrese N.N.N.H
- klasa D – raspon adrese 224.0.0.0 – 239.255.255.255
- klasa E – raspon adrese 240.0.0.0 – 254.255.255.255

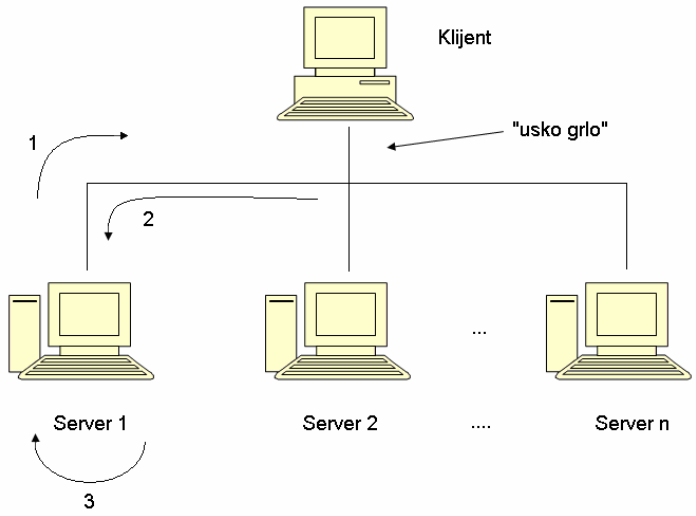
Detaljnije informacije o ovom protokolu potražite u dokumentu RFC 791.

[1] N – oznacava adresu mreže, H – oznacava adresu racunala u mreži

3. Struktura mrežne okoline i rješavanje zadatka

3.1. Model paketske mreže

Na slici 3.1 predstavljen je model u kojem klijent generira pakete i dodjeljuje ih serverima koji ih obraduju. Model se sastoji od jednog klijenta te najmanje jednog servera.



Sl. 3.1 Model paketske mreže

Koraci dodjeljivanja paketa:

1. Server zahtjeva od klijenta na osnovu Q – tablice da mu se pošalje od 0 do m paketa (kasnije u predstavljanom programskom rješenju broj paketa ogranicen je do maksimalno 3 paketa, m = 3)
2. Klijent uzima pakete iz repa i šalje ih serveru na obradu
3. Izracunavaju se nagrade za Q-tablicu, te se ponovo vraca na prvi korak

Usko grlo ovoga modela je veza izmedu klijenta i servera zbog toga, što za slucaj da server zatraži tri paketa na obradu dok klijent nema paketa u repu, klijent ostaje

blokiran za druge servere sve dok ne generira potrebne pakete. U tom slucaju samo jedan server dobiva pakete, a ostali to vrijeme ostaju "bez posla". Ucinkovitije rješenje bi bilo da se model sastoji od tri servera, te sva tri servera zatraže po jedan paket, dakle da se serveri medusobno balansiraju. Zbog toga je u racunanje nagrade uzeto u obzir i vrijeme koje protekne od zahtjeva za paketima pa do njihove isporuke.

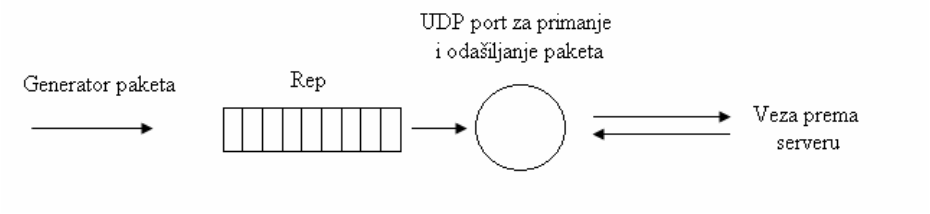
3.2. Klijent

Klijent generira pakete i postavlja ih u rep, te zatim osluškuje na zadanom UDP portu zahtjev za paketima. Zahtjev se sastoji samo od broja paketa koje treba dostaviti, a IP adresa i UDP port servera kojemu se trebaju dostaviti paketi na obradu, klijent sam saznaje iz paketa.

Nakon što klijent dobije zahtjev za paketima oni se uzimaju iz repa te šalju serveru. Ukoliko server traži tri paketa, a u repu ima samo jedan, odmah se šalje taj jedan, a na druge se ceka da prvo dodu u rep i cim dodu šalju se na server.

Ukoliko u repu nema mjesta za dodatni paket, a pokuša se dodati novi paket on se odbacuje, te se bilježi kao odbaceni paket.

Generiranje paketa pocinje s jednolikom razdiobom od 1 paket u 10 sekundi, a intenzitet generiranja smanjuje se svakih 5 sekundi za 5 % sve do 0.2 sekunde te se nakon toga povecava svakih 5 sekundi za 5% do 10 sekundi.



Sl. 3.2 Model klijenta

3.3. Server

Svake sekunde server na osnovu Q - tablice odreduje koliko ce paketa zatražiti od klijenta. Može zatražiti nula paketa, jedan paket, dva ili tri paketa. Nakon što dobije pakete izracunava se nagrada i osvježava Q - tablica. Nagrada se izracunava (u slucaju da je izabrana akcija slanja jednog, dva ili tri paketa) po sljedecoj formuli:

$$nagrada = \frac{trenutnoOpterecenjeServera}{dopustenoOpterecenjeServera} + vrijemeCekanjaPaketa \qquad (2)$$

Vrijeme cekanja paketa predstavlja vrijeme koji je prošlo od zahtjeva za paketima pa do isporuke svih zahtijevanih paketa, a izražava se u sekundama.

U slucaju da je bila izabrana akcija od slanja nula paketa, nagrada se izracunava po formuli:

$$nagrada = 1 - \frac{trenutnoOpterecenjeServera}{dopustenoOpterecenjeServera} \qquad (3)$$

Drugi dio formule (2) je vrijemeCekanjaPaketa koje se dodaje nagradi. Taj podatak kažnjava dugo cekanje na pakete koje se dogada kada npr. server traži tri paketa, a u repu klijenta postoji samo jedan paket. U formuli (3) nema tog dijela jer server zapravo i ne traži pakete od klijenta. Ovom se formulom kažnjava izbor akcije u kojoj se ne traže paketi od klijenta kad je opterecenje servera malo, a povecanjem opterecenja servera kazna postaje sve manja i prelazi u nagradu.

Q – tablica sastoji se od cetiri stupca koji predstavljaju akcije i redova koji predstavljaju stanja, a popunjavaju se prema formuli (1). Broj stanja se zadaje prilikom pokretanja servera, minimalno je jedno stanje, a maksimalno 65536.

U svakom stanju izabire se akcija ciji par (stanje,akcija) ima najmanju vrijednost.

Prilikom izracuna vrijednosti Q, vrijednost faktora umanjenja g (discount faktor) je 0 koji prema [1] omogucuje brzu prilagodbu na promjene okoline.

4. Programska podrška

Programska izvedba zadatka riješena je u programskom jeziku JAVA, a za potrebe ovog programa treba prije njegovog pokretanja imati vec instaliran *java development kit 1.4.0* (jdk1.4.0) ili noviju verziju. Takoder treba povjeriti varijablu okoline (engl. e*nviroment variables*) *JAVA_HOME* da "pokazuje" na mapu (engl. *folder*) u koju je instaliran *java development kit.* Ukoliko takva varijabla ne postoji potrebno ju je dodati.

Aplikacija se nalazi na priloženom CD-u u mapi *DiplomskiProgram*. Da bi je pokrenuli prekopirajte je bilo gdje na lokalni disk vašeg racunala. Ona se sastoji od dva medusobno neovisna dijela – servera koji obraduje pakete i klijetna koji mu daje pakete.

Serverski dio aplikaciji ima mogucnost snimanja svih podataka simulacije, a tu su:

- sve postavke za pokretanje simulacije s prozora Server
- svi podaci potrebni za grafove koje nam nudi aplikacija

Prilikom spremanja podataka na lokalno racunalo predodredena mapa za datoteku s podacima je *save* koja se nalazi u mapi *DiplomskiProgram*. U slucaju da želimo spremiti datoteku u neku drugu mapu možemo to napraviti na isti nacin kao i kod svih windows aplikacija.

Podaci se u datoteku snimaju u xml formatu, a za njegov unos i citanje koristim sljedece java pakete:

- za citanje xml datoteke – xmlParserAPIs.jar
- za spremanje podataka u xml datoteku – xercesImpl.jar

U glavnom prozoru serverskog dijela aplikacije nalaze se polja za unos parametara potrebnih za simulaciju. Da bi se aplikacija obranila od pogrešaka pri unosu podataka (npr. lokalni port mora biti veci od 1024 i manji ili jednak 65536, IP adresa mora biti

u obliku xxx.xxx.xxx.xxx …) koristim regularne izraze (engl. *regular expresions*) koji su se pokazali vrlo efikasnima. Paket koji mi omogucava korištenje regularnih izraza je:

- jakarta-oro-2.0.7.jar

Kod serverskog dijela aplikacije još treba napomenuti da se za iscrtavanje grafova koristi java paket:

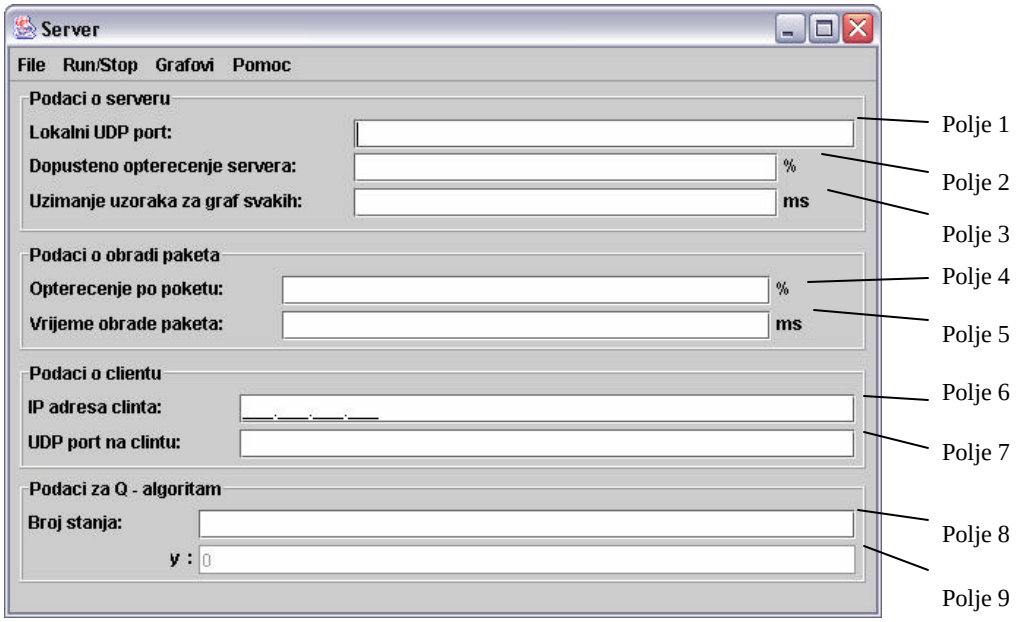
- jfreechart-0.9.18.jar

Svi paketi se mogu naci na Internetu. Dovoljno je samo upisati njihovo ime u Internet pretraživac i dobit cemo nekoliko lokacija s kojih besplatno možemo presnimiti tražene pakete na naše racunalo. Svi paketi potrebni za rad ove aplikacije nalaze se u mapi *lib* koja se nalazi u mapi *DiplomskiProgram*,a krajnji korisnik ne mora biti svjestan da ih koristi jer sve puteve do paketa i klasa (engl. *classpath*) aplikacija sama prilagodava.

Klijentski dio aplikacije puno je jednostavni i jedino što bih kod njega naglasio je da se o ispravnosti unesenih podataka u polja klijentskog prozora takoder brinu regularni izrazi.

4.1. Server

Server se pokrece dvoklikom na datoteku *StartServer.bat* i nakon toga se pojavljuje prozor kao na slici 4.1.



Sl. 4.1 Prozor servera

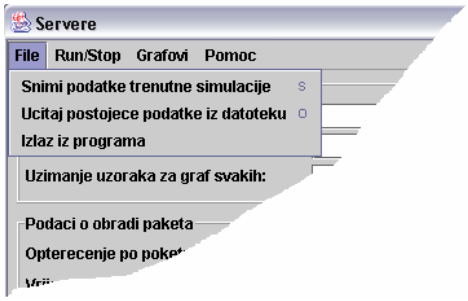
Objašnjenje polje na prozoru servera:

- polje 1 – upisuje se broj UDP porta preko kojeg ce server slati i primati pakete od klijenta, broj UDP porta mora biti u intervalu od 1025 – 65536
- polje 2 – upisuje broj koji oznacuje dopušteno opterecenje servera u postocima, minimalno 1 ,a maksimalno 100%. Broj se upisuje bez oznake postotka.
- polje 3 – server nam nudi graficki prikaz opterecenja servera u vremenu kao u broj odbacenih paketa. Ovdje upisujemo vrijeme u milisekundama, a ono oznacava vrijeme uzimanja uzoraka za graf.

- polje 4 – ovdje upisujmo broj koji oznacava za koliko posto se poveca opterecenje servera s novim paketom koji dode na obradu.
- polje 5 – ovdje upisujemo broj koji oznacava vrijeme obrade paketa u milisekundama
- polje 6 – ovdje upisujemo IP adresu na kojoj se nalazi klijent
- polje 7 – u ovo polje upisujemo broj UDP porta na kojem klijent osluškuje zahtjeve servera za paketima, broj UDP porta krece se u intervalu od 1025 - 65536
- polje 8 – ovdje treba upisati broj koji predstavlja broj stanja u Q – tablici. Mora biti minimalno jedno stanja.
- polje 9 – ovo polje predstavlja vrijednost discoun faktora γ, koji prema [1] treba biti nula. Zbog toga je ovdje onemogucen unos podataka, uvijek je nula.

4.1.1. Padajuci izbornici servera

Klikom na padajuci izbornik "File" dobivamo izbor naredbi kao na slici 4.2.



Sl. 4.2 Padajuci izbornik File

Naredbom "Snimi podatke trenutne simulacije" snimamo u jednu xml datoteku podatke o opterecenju servera, broju odbacenih paketa i broju izabranih akcija kroz cijelo vrijeme trajanja simulacije.

Naredbom "Ucitaj postojece podatke iz datoteke" ucitavamo prethodno snimljene podatke neke simulacije te nakon toga možemo dobiti sve grafove koje program nudi.

Zadnja naredba u ovom izborniku je "Izlaz iz programa".

Klikom na padajuci izbornik "Run/Stop" dobivamo izbor naredbi kao na slici 4.3.

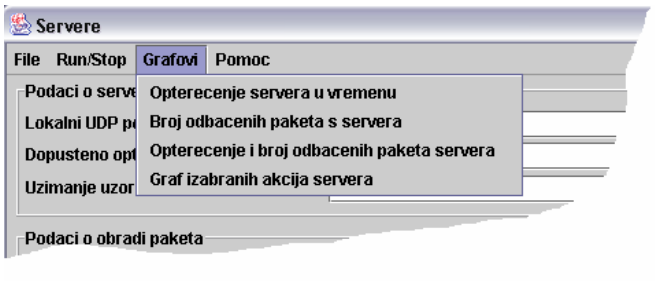
Sl. 4.3 Padajuci izbornik Run/Stop

Klikom na naredbu "Pokreni server" pokrecemo provjeru podataka upisanu u polje 1 do polja 8, a ako su svi podaci dobro upisani simulacija se pokrece.

Klikom na naredbu "Zaustavi server" server se zaustavlja.

Ako je server pokrenut tada možemo samo kliknuti na naredbu "Zaustavi server" i obrnuto, ako server nije pokrenut možemo kliknuti samo na naredbu "Zaustavi server". Time onemogucujemo pokretanje dva servera istovremeno na istom prozoru što bi dovelo do pogrešaka.

Klikom na padajuci izbornik "Grafovi" dobivamo izbor naredbi kao na slici 4.4.

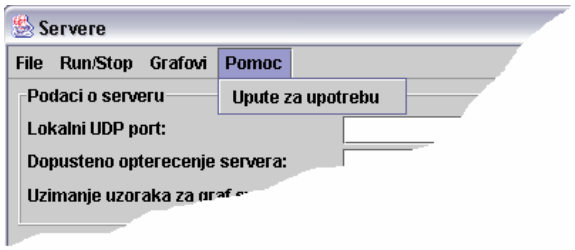


Sl. 4.4 Padajuci izbornik Grafovi

Nude nam se na izbor cetiri moguca grafa:

- opterecenje servera u vremenu – pokazuje nam kako se mijenja opterecenje servera kroz vrijeme trajanja simulacije
- broj odbacenih paketa s servera – pokazuje nam koliko je bilo odbacenih paketa tijekom trajanja simulacije i u kojim trenucima su paketi odbaceni
- opterecenje i broj odbacenih paketa servera – prijašnja dva zajedno
- graf izabranih akcija servera – pokazuje koliko je puta od servera zatraženo slanje jednog paketa, dva paketa, tri paketa i nula paketa.

Zadnji padajuci izbornik je izbornik "Pomoc" koji je prikazan na slici 4.5.



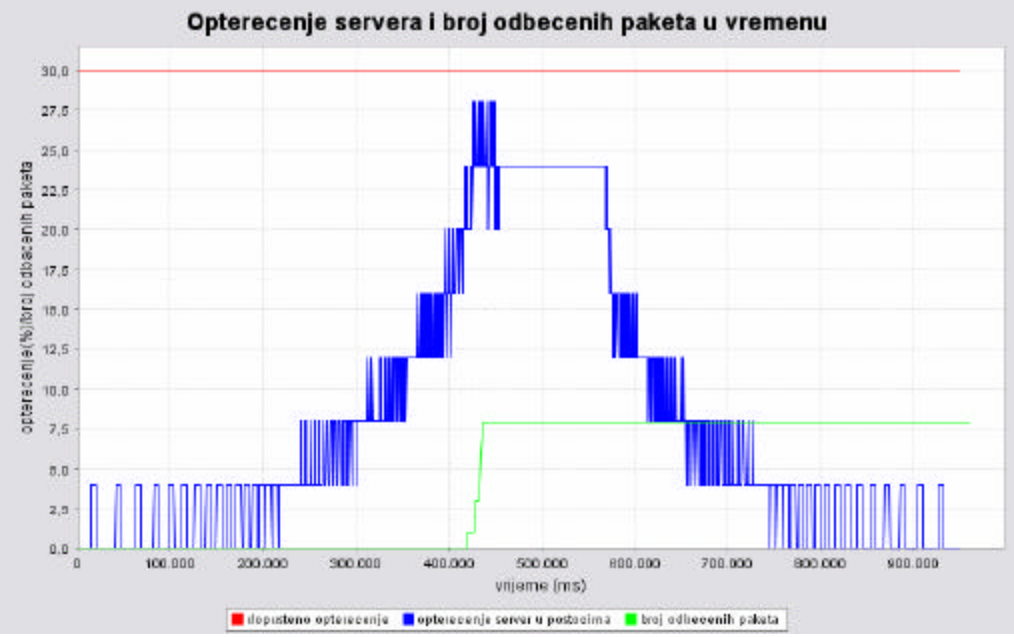
Sl. 4.5 Padajuci izbornik Pomoc

Ovdje nam se nudi samo jedna naredba, a to je "Upute za uporabu". Ako kliknemo na ovu naredbu, buduci da još nema tih uputa, izaci ce nam obavijest koja kaže da trenutno nema dostupnih uputa.

4.1.2. Grafovi servera

Na slici 4.6 prikazan je graf koji prikazuje opterecenje servera i broj odbacenih paketa u vremenu. Na osi *x* nalazi se vrijeme u milisekundama, a na osi *y* nalazi se opterecenje servera u postocima i broj odbacenih paketa. Ovaj jedan graf može se dobiti i kao dva posebna na kojima se posebno prikazuje broj odbacenih paketa na jednom grafu i opterecenje servera na drugom.

Na slici 4.7 prikazan je graf u obliku torte koji nam pokazuje koliko je ukupno puta izabrana akcija slanja jednog, dva, tri ili nula paketa.



Sl. 4.6 Izgled grafa "opterecenje servera i broj odbacenih paketa u vremenu"

Sl. 4.7 Izgled grafa "odabir zahtjeva za paketima"

4.2. Klase servera

Server se sastoji od dva paketa.Prvi je *hr.fer.server* i drugog *hr.fer.server.util.*

Paket hr.fer.server sadrži sljedece klase:

- PacketProcessing.java
- Qalgoritam.java
- Server.java
- ServerWindow.java
- StartServer.java

Paket hr.fer.server.util sadrži sljedece klase:

- SAXHandler.java
- Validator.java

Klasa *ServerStart* pokrece serverski dio aplikacije prikazivanjem prozora koji je definiran klasom *ServerWindow*.

U klasi *Server* imamo osnovne informacije o serveru kao što su vrijeme obrade pojedinog paketa, dopušteno opterecenje servera, opterecenje koje uzrokuje jedan paket u serveru i broj stanja Q tablice. Osim ovih varijabli sadrži i metodu koja izracunava nagradu kojom se osvježava Q – tablica. Klasa *PacketProcessing* predstavlja obradu jednog paket. Ova klasa se izvršava kao zasebna nit, a prilikom pokretanja ove niti prvo se opterecenje servera uvecava za unaprijed definirano opterecenje servera po paketu, zatim "ceka" da prode vrijeme obrade paketa te se smanjuje opterecenje servera. Dakle ova klasa simulira ulazak paketa u server koji uzrokuje povecanje opterecenja servera za unaprijed zadano vrijeme obrade, te njegovim izlaskom iz servera uzrokuje smanjenje opterecenja.

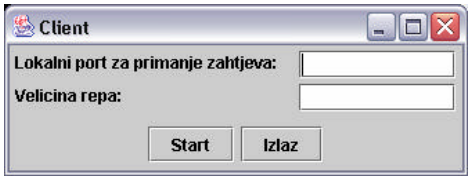
Klasa *QTablica* sadrži polje koje predstavlja Q – tablicu te metode pomocu kojih postavljamo sve vrijednosti Q – tablice na nulu (prilikom pokretanja simulacije) i metode koja nam daje informaciju o akciji koju je najbolje u odredenom trenutku izvesti.

Klasa *SAXHandler* služi za ucitavanje podataka iz xml datoteku u koju su prethodno snimljeni podaci prijašnje simulacije.

Klasa *Validator* služi da provjeravanje upisanih podataka u polja na prozoru servera pomocu regularnih izraza. Na taj se nacin aplikacija "brani" od pogrešaka pri unosu podataka.

4.3. Klijent

Klijent se pokrece dvoklikom na datoteku *StartClient.bat*, a nalazi se u mapi *DiplomskiProgram*. Nakon pokretanja klijenta pojavljuje se prozor kao na slici 4.8



Sl. 4.8 Izgled prozora klijenta

Na prozoru klijenta imamo dva polja za unos podataka. Prvo polje služi za unos broja UDP porta preko kojeg ce klijent primati zahtjeve za paketima. Broj porta ne može biti manji od 1025 niti veci od 65536. Drugo polje je polje u koje upisujemo velicinu repa u kojeg se stavljaju paketi prije slanja serverima na obradu. Rep ne može primati manje od jednog paketa niti više od 65536 paketa.

Trenutni broj paketa koji se nalazi u repu i intenzitet generiranja paketa ispisuje se u *command prompt* prozoru.

4.4. Klase klijenta

Klijent se sastoji od tri paketa: *hr.fer.client.communication* , *hr.fer.client.generator* i *hr.fer.client.rep*

Paket *hr.fer.client.communication* sadrži sljedece klase:

- ClientWindow.java
- Commnication.java
- RequestHandler.java

Paket *hr.fer.client.generator* sadrži klasu *Generator.java*, a paket *hr.fer.client.rep* sadrži klasu *Rep.java*.

Za prikaz prozora klijenta brine se klasa *ClientWindow*. Klasa *Communication* brine se o komunikaciji izmedu servera i klijenta. Ova klasa prima zahtjeve servera i instancira nove niti klase *RequestHandler* u kojima se zahtjevi obraduju. Obrada paketa sastoji se od saznavanja IP adrese servera i UDP porta na serveru na kojeg treba poslati pakete na obradu te koliko paketa treba poslati. Paketi se uzimaju iz repa koji je predstavljen klasom *Rep*. Velicina repa zadaje se prilikom pokretanja klijenta na njegovom prozoru. Pakete u rep postavlja generator koji je predstavljan klasom *Gererator*. U slucaju da u repu nema više mjesta za nove pakete, a generator ipak pokuša postaviti novi paket u rep on se odbacuje.

5. Rezultati simulacije

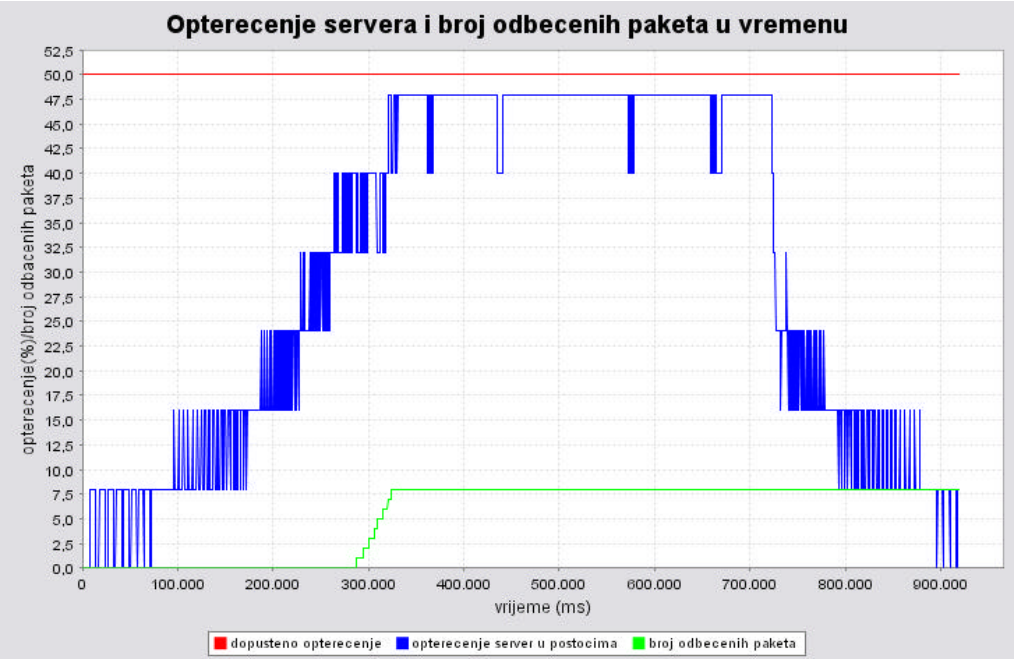
Prove dena su cetiri mjerenja. Prva tri mjerenja provedena su sa svrhom odredivanja broja stanja Q - tablice tako da dobivamo najbolje rezultate. Zajednicko tim trima mjerenjima su:

- dopušteno opterecenje servera 50%
- svaki paket uzrokuje dodatno opterecenje servera za 8%
- vrijeme trajanja obrade svakog paketa je 4 sec.

Razlicite postavke su:

- prvo mjerenje ima jedno stanje Q – tablice
- drugo mjerenje ima dva stanja Q – tablice
- trece mjerenje ima tri stanja Q – tablice

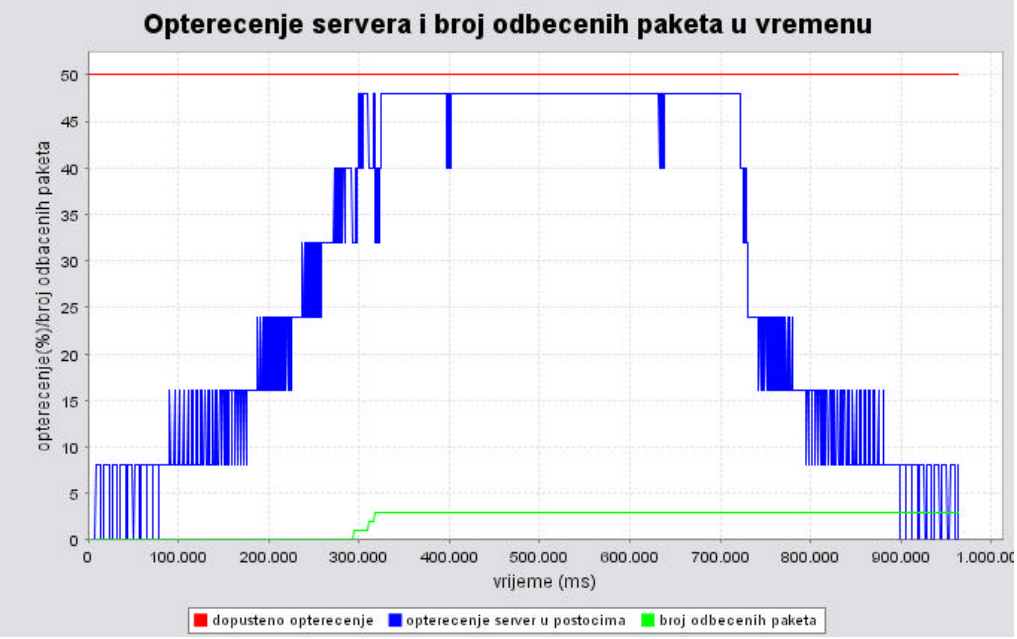
Rezultati prvog mjerenja:



Sl. 5.1

Sl. 5.2

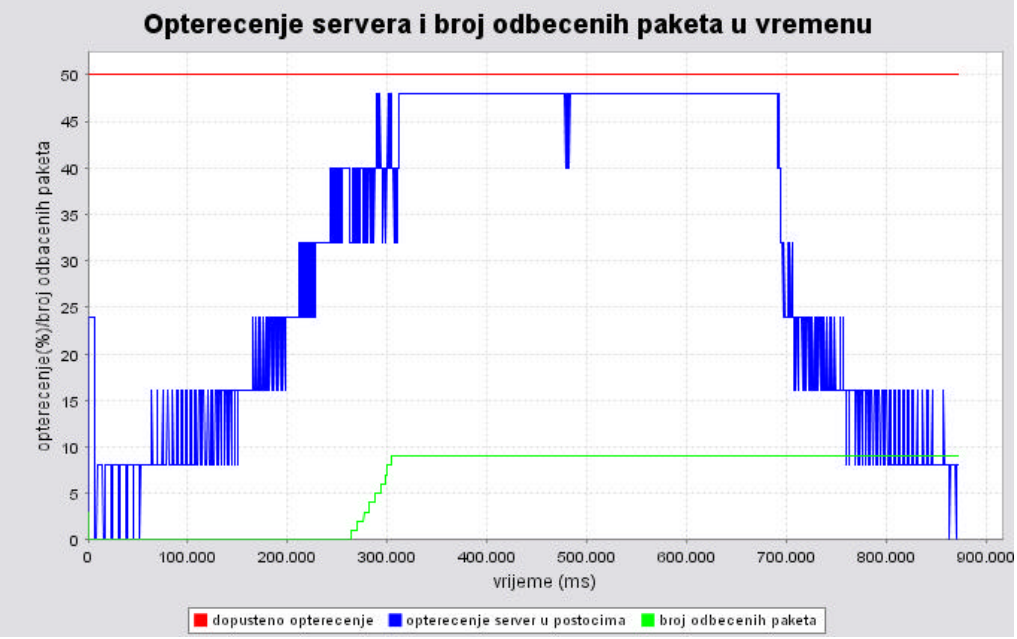
Rezultati drugog mjerenja:



Sl. 5.3

Sl. 5.4

Rezultati treceg mjerenja:



Sl. 5.5

Sl. 5.6

5.1. Rezultati prva tri mjerenja

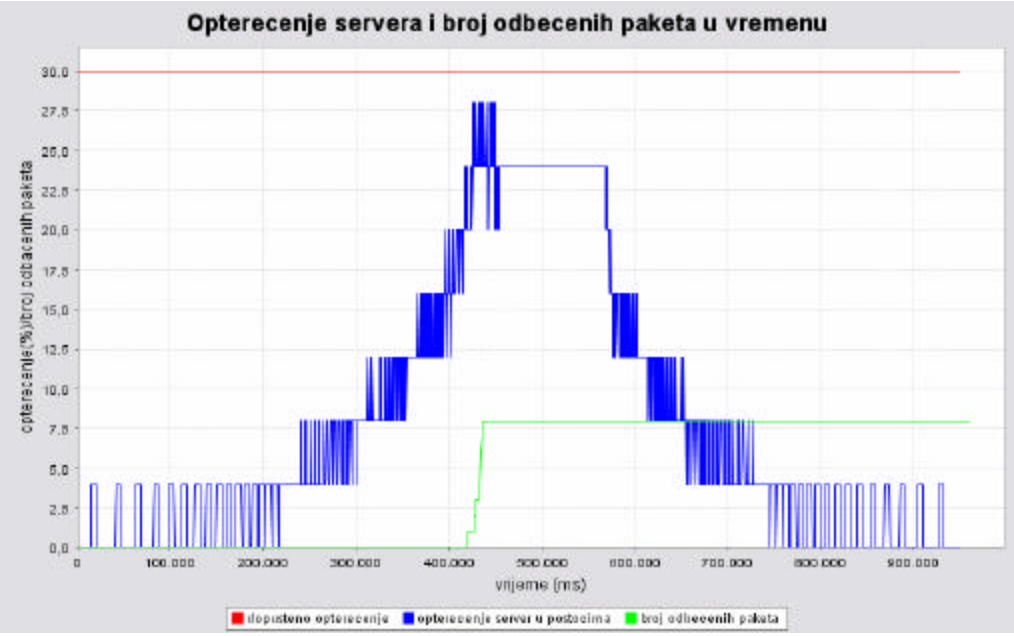
U ova tri mjerenja promatramo utjecaj broja stanja Q – tablice na broj odbacenih paketa. U slucaju jednog stanja Q – tablice broj odbacenih paketa je skoro duplo veci nego kad ubacimo još jedno stanje što uzrokuje smanjenje broja odbacenih paketa. U slucaju povecanja broja stanja na više od dva, kao što je to primjer u trecem mjerenju, broj odbacenih paketa se naglo povecava.

Cetvrta simulacija pokazuje kako se balansira opterecenje izmedu tri servera koja istovremeno traže pakete od jednog klijenta.

Postavke prvog servera:

- dopušteno opterecenje servera 30%
- svaki paket uzrokuje dodatno opterecenje servera za 4%
- vrijeme trajanja obrade svakog paketa je 2 sec.
- dva stanja Q – tablice

Rezultati:



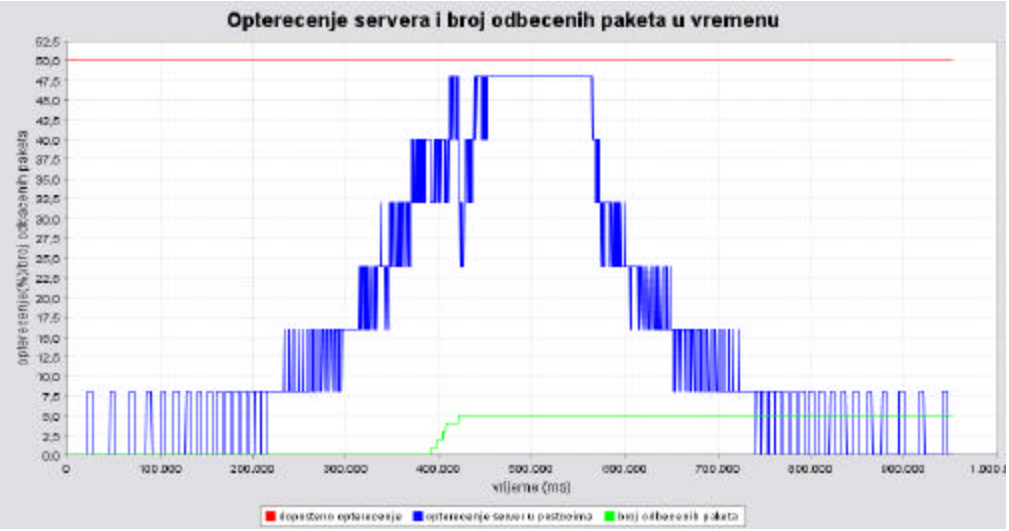
Sl. 5.7

Sl. 5.8

Postavke drugog servera:

- dopušteno opterecenje servera 50%
- svaki paket uzrokuje dodatno opterecenje servera za 8%
- vrijeme trajanja obrade svakog paketa je 4 sec.
- dva stanja Q – tablice

Rezultati:



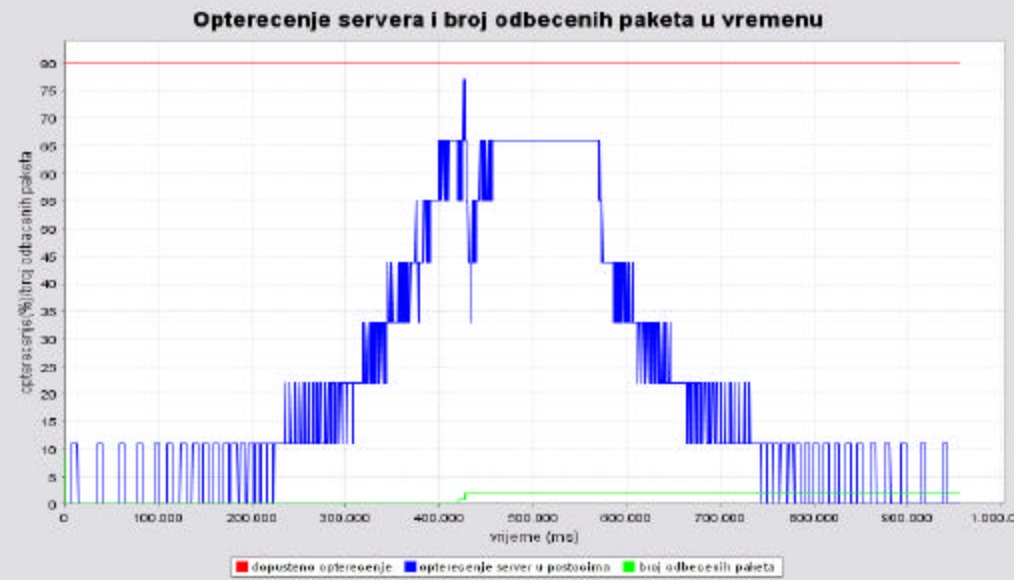
Sl. 5.9

Sl. 5.10

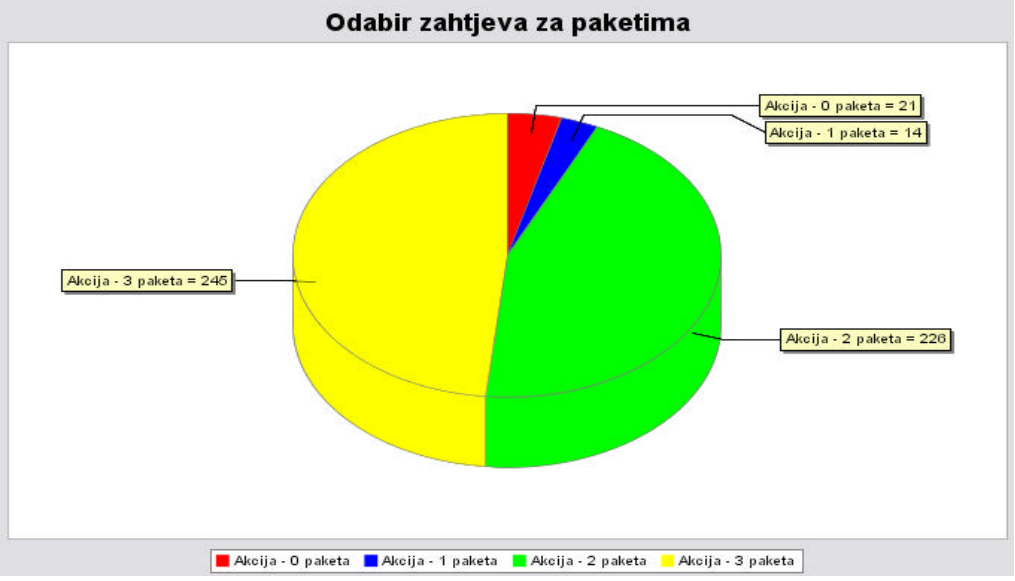
Postavke treceg servera:

- dopušteno opterecenje servera 80%
- svaki paket uzrokuje dodatno opterecenje servera za 11%
- vrijeme trajanja obrade svakog paketa je 11 sec.
- dva stanja Q – tablice

Rezultati:



Sl. 5.11



Sl. 5.12

5.2. Rezultati cetvrtog mjerenja

Kroz ova mjerenja možemo lijepo vidjeti da balansiranje opterecenja servera doista funkcionira. Tako na primjer možemo vidjeti da u 300 – totoj sekundi simulacije opterecenja svakog servera iznose:

- opterecenje prvog servera - 26,6% od dozvoljenog opterecenja
- opterecenje drugog servera – 32% od dozvoljenog opterecenja
- opterecenje treceg servera – 27,5% od dozvoljenog opterecenja

Dakle sva opterecenja se nalaze na približno istoj vrijednosti.

6. Zakljucak

Izradujuci ovaj diplomski rad proucavana je mogucnost adaptivnog balansiranja opterecenja.

Kao model odabran je sustav s tri poslužitelja, tri agenta i jednim izvorištem. Promet je generiran jednolikom razdiobom, a postojala je jedna klasa prometa. Programska realizacija dopušta n poslužitelja i n agenata (n > 0) te jedno izvorište, a pri povecanju broja poslužitelja vrijeme testiranja povecava se n-struko.

Testiranja su izvedeno za model od jednog poslužitelja, jednog agenta i jednog izvorišta te za model od tri poslužitelja, tri agenta i jednim izvorištem. U svakom modelu agenti su smješteni na strani poslužitelja, a buduci da nisu medusobno povezani nisu "svjesni" jedan drugog.

Testiranje s jednim poslužiteljem provedeno je s ciljem odredivanja velicine tj. broja stanja Q – tablice. Ovisno o broju stanja poslužitelj se brže ili sporije prilagodava intenzitetu nailazaka paketa.

Testiranje u drugom modelu prikazuje sposobnost agenata u balansiranju opterecenja izmedu tri poslužitelja. U prikazu rezultata ovog testiranja vidimo da su agenti sposobni medusobno balansirati opterecenje iako nisu "svjesni" medusobnog postojanja.

Za optimalno djelovanje agenta potrebno je cekati od 3 do 16 koraka simulacije, gdje jedan korak odgovara vremenskom intervalu od 1 sekunde. Optimalno vrijeme djelovanja agenta izravno ovisi o kolicini paketa koje agent može tražiti od izvorišta te velicini tj. broju stanja Q – tablice.

Najveci prostor poboljšanjima kao i najveci zadatak ovog algoritma leže u sustavu nagradivanja. Formula pomocu koje izracunavamo nagradu u ovoj aplikaciji uzima u obzir opterecenost poslužitelja i vrijeme koje protekne od zahtjeva za paketima pa do njihove isporuke. To vrijeme predstavlja vrijeme blokiranja ostalih poslužitelja (za to vrijeme ostali poslužitelji ne mogu tražiti pakete od izvora).

7. Literatura

[1] Johan Parent, *Adaptive Load Balancing of Parallel Applications with ReinforcementLearning on Heterogeneous Networks*

[2] Johan Parent, Katja Verbeeck, Ann Nowe, Kris Steenhaut, *Adaptive Load Balancing of Parallel Applications with Multi-Agent Reinforcement Learning on Heterogeneous Systems*

[3] Christian R. Shelton, *Balancing Multiple Sources of Reward in Reinforcement Learning*

[4] Mance E. Harmon, Stephanie S. Harmon, *Reinforcement Learning: A Tutorial*

[5] Leoanid Peshkin, *Reinforcement Learning by Policy Search*

[6] *Computing in Cognitive Science*, www.indiana.edu/~q320/indeks

8. Prilog

8.1. Klasa ClientWindow.java

```
package hr.fer.client.comunication;

import hr.fer.server.util.Validator;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;


public class ClientWindow extends JFrame {
	BorderLayout borderLayout1 = new BorderLayout();
	JPanel osnovniPanel = new JPanel();
	GridBagLayout gridBagLayout1 = new GridBagLayout();
	JPanel jPanel1 = new JPanel();
	JPanel jPanel2 = new JPanel();
	JLabel jLabel1 = new JLabel();

	public static JTextField lokalniUdpPortF = new JTextField();
	JLabel jLabel2 = new JLabel();
	public static JTextField velicinaRepaF = new JTextField();
	GridBagLayout gridBagLayout2 = new GridBagLayout();
	GridLayout gridLayout1 = new GridLayout(1,1);
	JPanel jPanel3 = new JPanel();
	JPanel jPanel4 = new JPanel();
	JButton startButton = new JButton();

	JButton exitButton = new JButton();
	JLabel jLabel5 = new JLabel();

	JLabel jLabel6 = new JLabel();
	public static ClientWindow w = null;

	Thread comm = null;
	Validator v = new Validator();

	public ClientWindow() {

		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		try {
```

```
                jbInit();
            } catch (Exception e) {
                e.printStackTrace();
            }
            this.setTitle("Client");
            this.setSize(335, 125);
            Dimension desktopSize =
Toolkit.getDefaultToolkit().getScreenSize();
            this.setLocation(desktopSize.width / 2  - 165, desktopSize.height / 2 -
62);
        }
        private void jbInit() throws Exception {
            jPanel1.setLayout(gridBagLayout2);
            jLabel2.setText("Velicina repa:");
            jLabel1.setTe xt("Lokalni port za primanje zahtjeva:");
            this.getContentPane().setLayout(borderLayout1);
            osnovniPanel.setLayout(gridBagLayout1);
            jPanel2.setLayout(gridLayout1);
            startButton.setText("Start");
            startButton.addActionListener(new java.awt.event.Actio nListener() {
                public void actionPerformed(ActionEvent e) {
                    startButton_actionPerformed(e);
                }
            });

            exitButton.setText("Izlaz");
            exitButton.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    exitButton_actionPerformed(e);
                }
            });
            this.getContentPane().add(osnovniPanel, BorderLayout.CENTER);
            osnovniPanel.add(jPanel1, new GridBagConstraints(0, 0, 1, 1, 1.0,
0.0, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new
Insets(2, 2, 2, 2), 0, 0));
            jPanel1.add(jLabel1, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0,
GridBagConstraints.NORTHWEST, GridBagConstraints.NONE, new Insets(2, 2, 2,
2), 0, 0));
            jPanel1.add(lokalniUdpPortF, new GridBagConstraints(1, 0, 1, 1, 0.9,
0.0, GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL, new
Insets(2, 2, 2, 2), 0, 0));
            jPanel1.add(jLabel2, new GridBagConstraints(0, 1, 1, 1, 0.1, 0.0,
GridBagConstraints.NORTHWEST, GridBagConstraints.NONE, new Insets(2, 2, 2,
2), 0, 0));
            jPanel1.add(velicinaRepaF, new GridBagConstraints(1, 1, 1, 1, 0.9,
0.0, GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL, new
Insets(2, 2, 2, 2), 0, 0));
```

```
		osnovniPanel.add(jPanel2, new GridBagConstraints(0, 1, 1, 1, 1.0,
1.0, GridBagConstraints.NORTHWEST, GridBagConstraints.HORIZONTAL, new
Insets(2, 2, 0, 2), 0, 0));
		jPanel2.add(jPanel4, null);
		jPanel4.add(startButton, null);
		jPanel4.add(exitButton, null);
	}

	void startButton_actionPerformed(ActionEvent e) {
		String temp = null;

		temp = lokalniUdpPortF.getText();
		if (v.doMatch(temp, "[0123456789]+")) {
			if (Integer.parseInt(temp) <= 1024 || Integer.parseInt(temp) >=
65536) {
				JOptionPane.showMessageDialog(this, "Lokalni UDP
port mora biti u rasponu od 1024 - 65536", "Opis greske",
JOptionPane.ERROR_MESSAGE);
				return;
			}
		} else {
			JOptionPane.showMessageDialog(this, "Lokalni UDP port
mora biti u rasponu od 1024 - 65536", "Opis greske",
JOptionPane.ERROR_MESSAGE);
			return;
		}

		temp = velicinaRepaF.getText();
		if (v.doMatch(temp, "[0123456789]+")) {
			if (Integer.parseInt(temp) <= 1 || Integer.parseInt(temp) >=
65536) {
				JOptionPane.showMessageDialog(this, "Velicina repa
mora biti od 1 - 65536", "Opis greske", JOptionPane.ERROR_MESSAGE);
				return;
			}
		} else {
			JOptionP ane.showMessageDialog(this, "Velicina repa mora
biti od 1 - 65536", "Opis greske", JOptionPane.ERROR_MESSAGE);
			return;
		}

		startButton.setEnabled(false);
		comm = (Thread) new Communication();
		comm.start();
	}

	void exitButton_actionPerformed(ActionEvent e) {
		System.exit(0);
```

```
	}

	public static int getPort() {
		return Integer.parseInt(lokalniUdpPortF.getText());
	}

	public static void main(String[] args) {
		w = new ClientWindow();
		w.setVisible(true);

	}

	public static int getVelicinaRepa() {
		return Integer.parseInt(velicinaRepaF.getText());
	}
}
```

8.2. Klasa Communication.java

```
package hr.fer.client.comunication;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;

import hr.fer.client.generator.Generator;
import hr.fer.client.rep.Rep;

/**
 * @author Stjepan Matijasevic
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class Communication extends Thread{

	private static int port = 0;
	private static Rep rep = null;
	public static int velicinaRepa = 0;
	Thread generator = null;
	public Communication() {
	}

	public void run() {
		rep = new Rep(ClientWindow.getVelicinaRepa());
		port = ClientWindow.getPort();
		byte[] b = new byte[1024];
		generator = (Thread) new Generator(rep);
		generator.start();
		DatagramSocket socket = null;
		try {
			socket = new DatagramSocket(port);
		} catch (SocketException e1) {

		while (true) {
			try {
				DatagramPacket paket = new DatagramPacket(b,
1024);
				socket.receive(paket);
				Thread t = (Thread) new RequestHandler(paket,rep);
				t.start();
```

```
                    } catch (SocketException e) {
                        e.printStackTrace();
                    } catch (IOException e)  {
                        e.printStackTrace();
                    }
                }
            }
        }
```

8.3. Klasa RequestHandler.java

```
package hr.fer.client.comunication;

import hr.fer.client.rep.Rep;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.Ine tAddress;
import java.net.SocketException;

/**
 * @author Stjepan Matijasevic
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class RequestHandler extends Thread{

	private DatagramPacket cPaket = null;

	private DatagramPacket sPacket = null;
	private DatagramSocket toClient = null;

	private int cPort = 0;
	private InetAddress ipAdresa = null;

	private Rep rep = null;

	public RequestHandler(DatagramPacket p,Rep r){
		cPaket = p;
		rep = r;
	}

	public void obradaZahtjeva(){
		String sp = new String(cPaket.getData());
		String sadrzajPaketa = sp.substring(0,1);
		cPort = cPaket.getPort();
		ipAdresa = cPaket.getAddress();

		sendPacketToServer( Integer.parseInt(sadrzajPaketa) );

	}
```

```
	public void sendPacketToServer(int brojPaketaZaSlanje){

		int i = 0;
		byte[] buffer = new byte[1024];
		try {
			toClient = new DatagramSocket();
		} catch (SocketException e) {

			e.printStackTrace();
		}

		while(i < brojPaketaZaSlanje){

			buffer = rep.getPackage().getBytes();
			sPacket = new
DatagramPacket(buffer,buffer.length,ipAdresa,cPort);
			try {
				toClient.send(sPacket);
			} catch (IOException e1) {
				e1.printStackTrace();
			}
			i++;
		}
	}

	public void run(){
		obradaZahtjeva();
	}
}
```

8.4. Klasa Generator.java

```
package hr.fer.client.generator;

import hr.fer.client.rep.Rep;

/**
 * @author Stjepan Matijasevic
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class Generator extends Thread{
	public Rep rep = null;

	// Konstantni period generiranja poruka odredjenim intezitetom
	private long konstantiPeriod = 5000;
	private long vrijemeIzmedjuGeneriranjaPaketa = Math.round(10000);
	private long vrijemeIzmedjuGeneriranjaPaketaFix = Math.round(10000);

	private long vrijeme = 0;
	private long tempVrijeme = 0;

	private long brojGeneriranihPaketa = 0;

	private String paket = null;
	private boolean smanjivanjeVremena = true;

	/**
	 * Konstruktor klase Generator
	 * @param r - predstavlja rep u kojeg generira pakete
	 */
	public Generator(Rep r){
		rep = r;
	}


	public void razdioba(){
		vrijeme = System.currentTimeMillis();
		while(true){

			staviURep();
			tempVrijeme = System.currentTimeMillis();
```

```
                    if( (vrijeme + konstantiPeriod) < System.currentTimeMillis()
){
                            vrijeme = System.currentTimeMillis();
                            if(smanjivanjeVremena == true){
                                    vrijemeIzmedjuGeneriranjaPaketa =
vrijemeIzmedjuGeneriranjaPaketa - Math.round(vrijemeIzmedjuGeneriranjaPaketa
*0.05);
                                    System.out.println("Vrijeme generiranja
paketa:"+vrijemeIzmedjuGeneriranjaPaketa);
                            }else{
                                    vrijemeIzmedjuGeneriranjaPaketa =
vrijemeIzmedjuGeneriranjaP aketa + Math.round(vrijemeIzmedjuGeneriranjaPaketa
*0.05);
                                    System.out.println("Vrijeme generiranja
paketa:"+vrijemeIzmedjuGeneriranjaPaketa);
                            }

                            if((smanjivanjeVremena == true) &&
(vrijemeIzmedjuGeneriranjaPaketa <= 200)){
                                    smanjivanjeVremena = false;
                            }

                            if((smanjivanjeVremena == false) &&
(vrijemeIzmedjuGeneriranjaPaketa >= vrijemeIzmedjuGeneriranjaPaketaFix)){
                                    smanjivanjeVremena = true;
                            }



                    }

                    // ceka da prodje vrijeme izmedju generiranja paketa pa
generira novi paket

                    long tt = System.currentTimeMillis() - tempVrijeme;
                    try {
                            if(vrijemeIzmedjuGeneriranjaPaketa - tt > 0){
                                    sleep(vrijemeIzmedjuGeneriranjaPaketa - tt);
                            }

                    } catch (InterruptedException e) {
                            e.printStackTrace();
                    }


            }
```

```
        }

        public  void staviURep(){
                brojGeneriranihPaketa++;
                rep.putPackage("Paket br. :"+brojGeneriranihPaketa);

        }

        public void run(){
                razdioba();
        }

}
```

8.5. Klasa Rep.java

```
package hr.fer.client.rep;

import java.util.ArrayList;

/**
 * @author Stjepan Matijasevic
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class Rep {

	public static ArrayList rep = null;
	public static int pocetak;
	public static int kraj;
	public static int velicinaRepa = 0;
	public static long brojOdbacenihPaketa;

	public Rep(int brojElemenataRepa) {
		velicinaRepa = brojElemenataRepa;

		rep = new ArrayList(velicinaRepa);
		pocetak = 0;
		kraj = 0;
		brojOdbacenihPaketa = 0;
	}

	/**
	 * @return int - trenutni broj paketa u repu
	 */
	public int getTrenutniBrojPaketaURepu() {
		int trenutnoPaketa = 0;

		if (((pocetak + 1) % velicinaRepa) == (kraj % velicinaRepa)) {
			trenutnoPaketa = 0;
		} else {
			if (((kraj + 1) % velicinaRepa) == (pocetak % velicinaRepa)) {
				trenutnoPaketa = velicinaRepa;
			} else {
				if (((kraj % velicinaRepa) - (pocetak % velicinaRepa))
< 0) {
					trenutnoPaketa = (pocetak % velicinaRepa) -
(kraj % velicinaRepa);
```

```
                        } else {
                                trenutnoPaketa = (kraj % velicinaRepa) -
(pocetak % velicinaRepa);
                        }
                }
        }
        System.out.println("Broj paketa u repu:" + trenutnoPaketa);
        return trenutnoPaketa;
    }

    public void putPackage(String paket) {
        if (getTrenutniBrojPaketaURepu() <= velicinaRepa) {
            rep.add((kraj % velicinaRepa), paket);
            if ((kraj + 1) % velicinaRepa == (pocetak % velicinaRepa)) {
                brojOdbacenihPaketa++;
                System.out.println("Broj odbacenih paketa:" +
brojOdbacenihPaketa);
            } else {
                kraj++;
                kraj = kraj % velicinaRepa;
            }

        } else {
            brojOdbacenihPaketa++;
            System.out.println("Broj odbacenih paketa:" +
brojOdbacenihPaketa);
        }

    }

    public synchronized String getPackage() {

        if ((pocetak + 1) % velicinaRepa == kraj % velicinaRepa) {
            while ((pocetak + 1) % velicinaRepa == kraj % velicinaRepa)
{
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
        String tempPaket = (String) rep.get(pocetak % velicinaRepa);
        pocetak++;
        pocetak = pocetak % velicinaRepa;
        notifyAll();
        return tempPaket;
    }
```

}

8.6. Klasa StartServer.java

```
package hr.fer.server;

public class StartServer {

  public StartServer() {
  }
  public static void main(String[] args) {

    ServerWindow s = new ServerWindow();
    s.setVisible(true);
  }
}
```

8.7. Klasa ServerWindow.java

```
package hr.fer.server;

import hr.fer.server.util.SAXHandler;
import hr.fer.server.util.Validator;

import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.text.MaskFormatter;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.apache.xerces.dom.DocumentImpl;
import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;

import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.DefaultPieDataset;
import org.jfree.data.XYDataItem;
import org.jfree.data.XYDataset;
import org.jfree.data.XYSeries;
import org.jfree.data.XYSeriesCollection;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

import org.xml.sax.helpers.DefaultHandler;

import java.awt.event.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;
```

```
import java.text.ParseException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class ServerWindow extends JFrame {

	public static float prozorOpterecenje = 0;
	public static float prozorOpterecenjePoPaketu = 0;
	public static long prozorVrijemeObradePaketa = 0;
	public static float prozorDopustenoOpterecenje = 0;
	public static int prozorBrojStanja = 0;
	public static String prozorIpClienta = null;
	public static int prozorClientPort = 0;
	public static boolean podaciSProzoraOK = false;
	public static int lokalniUdpPort = 0;
	public static long vrijemeuzorkovanjaZaGraf = 0;
	public static int brojOdbacenih = 0;

	private Thread t = null;
	private Thread monitoring = null;
	public double[] akcijeList = null;
	public static ArrayList opterecenjeList = null;

	public static ArrayList brojOdbacenihList = null;

	// za graf
	public static XYSeries serijaPodatakaOpterecenjeServera = null;
	public static XYSeries serijaPodatakaBrojOdbacenihPaketa = null;
	public static XYSeries serijaDopustenoOpterecenje = null;
	public static DefaultPieDataset serijaPodatakaAkcija = null;
	public static JFreeChart grafAkcija = null;

	final JFileChooser fc = new JFileChooser(".\\save");

	Object[] tipkaOK = { "OK" };

	BorderLayout borderLayout1 = new BorderLayout();
	JPanel centralniPanel = new JPanel();
	GridBagLayout gridBagLayout1 = new GridBagLayout();
	JPanel jPanel1 = new JPanel();
	JPanel jPanel2 = new JPanel();
	JPanel jPanel3 = new JPanel();
	JPanel jPanel4 = new JPanel();
	GridBagLayout gridBagLayout2 = new GridBagLayout();
	JButton startB = new JButton();
	JButton prikazGrafovaB = new JButton();
	JButton StopB = new JButton();
```

```
        TitledBorder titledBorder1;
        TitledBorder titledBorder2;
        TitledBorde r titledBorder3;
        GridBagLayout gridBagLayout3 = new GridBagLayout();
        JLabel jLabel1 = new JLabel();
        public static JTextField lokalniUdpF = new JTextField();
        JLabel jLabel2 = new JLabel();
        public static JTextField dopustanoOptServeraF = new JTextField( );
        JLabel jLabel3 = new JLabel();
        MaskFormatter ipMask = null;
        public static JFormattedTextField iPAdresaClientaF = null;
        GridBagLayout gridBagLayout4 = new GridBagLayout();
        GridBagLayout gridBagLayout5 = new GridBagLayout();
        JLabel jLabel4 = new JLabel();
        public static JTextField opterecenjePoPaketuF = new JTextField();
        JLabel jLabel5 = new JLabel();
        public static JTextField vrijemeObradePaketaF = new JTextField();
        JPanel jPanel5 = new JPanel();
        TitledBorder titledBorder4;
        JLabel jLabel6 = new JLabel();
        public static JTextField brojStanjaF = new JTextField();
        JLabel jLabel7 = new JLabel();
        JTextField gamaF = new JTextField();

        JLabel uzimanjeUzorakaZaGrafL = new JLabel("Uzimanje uzoraka za graf
svakih:");
        public static JTextField uzimanjeUzorakaZaGrafF = new JTextField();
        GridBagLayout gridBagLayout6 = new GridBagLayout();
        JMenuBar jMenuBar1 = new JMenuBar();
        JMenu jMenu1 = new JMenu();
        JMenu jMenu2 = new JMenu();
        JMenuItem exitMI = new JMenuItem("Izlaz iz programa");
        JMenuItem snimiPodatkeUFileMI = new JMenuItem();
        JMenuItem pokreniServerMI = new JMenuItem();
        JMenuItem zaustaviServerMI = new JMenuItem();
        JMenu jMenu3 = new JMenu();
        JMenuItem ucitajDatotekuMI = new JMenuItem();
        JMenuItem grafOpterecenjeMI = new JMenuItem();
        JMenuItem grafBrojOdbacenihMI = new JMenuItem();
        JMenuItem grafAkcijeMI = new JMenuItem("Graf izabranih akcija servera");
        JMenuItem grafZAjednoMI = new JMenuItem();
        JMenu pomocM = new JMenu();
        JMenuItem uputeZaUporabuMI = new JMenuItem();
        JLabel jLabel8 = new JLabel();
        public static JTextField udpPortNaClientuF = new JTextField();

        public ServerWindow() {
                try {
```

```
                ipMask = new MaskFormatter("###.###.###.###");
                ipMask.setValidCharacters("0123456789");
                ipMask.setPlaceholderCharacter('_');
                iPAdresaClientaF = new JFormattedTextField(ipMask);
            } catch (ParseException e1) {
                e1.printStackTrace();
            }
            try {
                jbInit();
            } catch (Exception e) {
                e.printStackTrace();
            }
            pokreniServerMI.setEnabled(true);
            zaustaviServerMI.setEnabled(false);
            Dimension desktopSize =
Toolkit.getDefaultToolkit().getScreenSize();
            this.setSize(600, 420);
            this.setLocation(desktopSize.width / 2  - 320, desktopSize.height / 2 -
240);
        }
        public ArrayList getOpterecenjeList() {
            return opterecenjeList;
        }
        private void jbInit() throws Exception {
            titledBorder1 = new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(156, 156, 158)),
"Podaci o serveru");
            titledBorder2 = new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED, Color.white, new  Color(156, 156, 158)),
"Podaci o obradi paketa");
            titledBorder3 = new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(156, 156, 158)),
"Podaci o clientu");
            titledBorder4 = new TitledBorder(new
EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(156, 156, 158)),
"Podaci za Q - algoritam");
            this.getContentPane().setLayout(borderLayout1);
            centralniPanel.setLayout(gridBagLayout1);
            jPanel4.setLayout(gridBagLayout2);
            startB.setText("Start");
            prikazGrafovaB.setText("Prikaz grafova");
            StopB.setText("Stop");
            jPanel1.setBorder(titledBorder1);
            jPanel1.setLayout(gridBagLayout3);
            jPanel2.setBorder(titledBorder2);
            jPanel2.setLayout(gridBagLayout4);
            jPanel3.setBorder(titledBorder3);
            jPanel3.setLayout(gridBagLayout 5);
```

```
                jLabel1.setText("Lokalni UDP port:");
                jLabel2.setText("Dopusteno opterecenje servera:");
                jLabel3.setText("IP adresa clinta:");
                jLabel4.setText("Opterecenje po poketu:");
                jLabel5.setText("Vrijeme obrade paketa:");

                jPanel5.setBorder(titledBorder4);
                jPanel5.setLayout(gridBagLayout6);
                jLabel6.setText("Broj stanja:");
                jLabel7.setText("y  :");
                jMenu1.setMnemonic('0');
                jMenu1.setRolloverEnabled(true);
                jMenu1.setText("File");
                jMenu2.setText("Run/Stop");
                snimiPodatkeUFileMI.setText("Snimi podatke trenutne simulacije");

        snimiPodatkeUFileMI.setAccelerator(javax.swing.KeyStroke.getKeyStroke(8
3, 0, false));

                snimiPodatkeUFileMI.addActionListener(new
java.awt.event.ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                snimiPodatkeUFileMI_actionPerformed(e);
                        }
                });
                pokreniServerMI.setText("Pokreni server");

        pokreniServerMI.setAccelerator(javax.swing.KeyStroke.getKeyStroke(82, 0,
false));

                pokreniServerMI.addActionListener(new
java.awt.event.ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                pokreniServerMI_actionPerformed(e);
                        }
                });

                exitMI.addActionListener(new java.awt.event.ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                System.exit(0);
                        }
                });

                zaustaviServerMI.setText("Zaustavi server");

        zaustaviServerMI.setAccelerator(javax.swing.KeyStroke.getKeyStroke(69, 0,
false));
```

```
                zaustaviServerMI.addActionListener(new
java.awt.event.ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                zaustaviServerMI_actionPerformed(e);
                        }
                });
                jMenu3.setText("Grafovi");
                ucitajDatotekuMI.setText("Ucitaj postojece podatke iz datoteku");

        ucitajDatotekuMI.setAccelerator(javax.swing.KeyStroke.getKeyStroke(79, 0,
false));
                ucitajDatotekuMI.addActionListe ner(new
java.awt.event.ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                ucitajDatotekuMI_actionPerformed(e);
                        }
                });
                grafOpterecenjeMI.setText("Opterecenje servera u vremenu");
                grafOpterecenjeMI.addActionListener(new
java.awt.event.ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                grafOpterecenjeMI_actionPerformed(e);
                        }
                });
                grafBrojOdbacenihMI.setText("Broj odbacenih paketa s servera");
                grafBrojOdbacenihMI.addActionListener(new
java.awt.event.ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                grafBrojOdbacenihMI_actionPerformed(e);
                        }
                });
                grafZAjednoMI.setText("Opterecenje i broj odbacenih paketa
servera");
                grafZAjednoMI.addActionListener(new
java.awt.event.ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                grafZAjednoMI_actionPerformed(e);
                        }
                });

                grafAkcijeMI.addActionListener(new java.awt.event.ActionListener()
{
                        public void actionPerformed(ActionEvent e) {
                                grafAkcijaMI_actionPerformed(e);
                        }
                });

                pomocM.setText("Pomoc");
```

```
                this.setDefaultCloseOperation(3);
                this.setJMenuBar(jMenuBar1);
                this.setTitle("Server");
                uputeZaUporabuMI.setActionCommand("Upute za upotrebu");
                uputeZaUporabuMI.setText("Upute za upotrebu");
                uputeZaUporabuMI.addActionListener(new
java.awt.event.ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                uputeZaUporabuMI_actionPerformed(e);
                        }
                });
                jLabel8.setText("UDP port na clintu:");
                this.getContentPane().add(centralniPanel, BorderLayout.CENTER);
                centralniPanel.add(jPanel1, new GridBagConstraints(0, 1, 1, 1, 1.0,
0.0, GridBagConstraints.NORTHWEST, GridBagConstraints.HORIZONTAL, new
Insets(2, 5, 2, 5), 1, 1));

                jPanel1.add(jLabel1, new GridBagConstraints(0, 0, 1, 1, 0.1, 0.0,
GridBagConstraints.NORTHWEST, GridBagConstraints.NONE, new Insets(2, 2, 5,
2), 0, 0));
                jPanel1.add(lokalniUdpF, new GridBagConstraints(1, 0, 2, 2, 0.7, 0.0,
GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL, new
Insets(2, 2, 5, 2), 0, 0));

                jPanel1.add(jLabel2, new GridBagConstraints(0, 1, 1, 2, 0.1, 0.0,
GridBagConstraints.NORTHWEST, GridBagConstraints.NONE, new Insets(2, 2, 2,
2), 0, 0));
                jPanel1.add(dopustanoOptServeraF, new GridBagConstraints(1, 1, 1,
1, 0.9, 0.0, GridBagConstraints.NORTHEAST,GridBagConstraints.HORIZONTAL,
new Insets(2, 2, 2, 2), 0, 0));
                jPanel1.add(new JLabel("%"), new GridBagConstraints(2, 1, 1, 1, 0.1,
0.0, GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL, new
Insets(2, 2, 2, 2), 0, 0));

                jPanel1.add(uzimanjeUzorakaZaGrafL, new GridBagConstraints(0, 2,
1, 2, 0.1, 0.0, GridBagConstraints.NORTHWEST, GridBagConstraints.NONE, new
Insets(2, 2, 5, 2), 0, 0));
                jPanel1.add(uzimanjeUzorakaZaGrafF, new GridBagConstraints(1, 2,
1, 1, 0.8, 0.0, GridBagConstraints.NORTHEAST,
GridBagConstraints.HORIZONTAL, new Insets(2, 2, 5, 2), 0, 0));
                jPanel1.add(new JLabel("ms"), new GridBagConstraints(2, 2, 1, 1,
0.1, 0.0, GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL,
new Insets(2, 2, 2, 2), 0, 0));

                centralniPanel.add(jPanel2, new GridBagConstraints(0, 2, 1, 1, 1.0,
0.0, GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL, new Insets(2,
5, 2, 5), 1, 1));
```

```
        jPanel2.add(jLabel4, new GridBagConstraints(0, 0, 1, 1, 0.1, 0.0,
GridBagConstraints.NORTHWEST, GridBagConstraints.NONE, new Insets(2, 2, 2,
2), 0, 0));
        jPanel2.add(opterecenjePoPaketuF, new GridBagConstraints(1, 0, 1,
1, 0.9, 0.0, GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL,
new Insets(2, 2, 2, 2), 0, 0));
        jPanel2.add(new JLabel("%"), new GridBagConstraints(2, 0, 1, 1, 0.1,
0.0, GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL, new
Insets(2, 2, 2, 2), 0, 0));

        jPanel2.add(jLabel5, new GridBagConstraints(0, 1, 1, 1, 0.1, 0.0,
GridBagConstraints.NORTHWEST, GridBagConstraints.NONE, new Insets(2, 2, 2,
2), 0, 0));
        jPanel2.add(vrijemeObradePaketaF, new GridBagConstraints(1, 1, 1,
1, 0.9, 0.0, GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL,
new Insets(2, 2, 2, 2), 0, 0));
        jPanel2.add(new JLabel("ms"), new GridBagConstraints(2, 1, 1, 1,
0.1, 0.0, GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL,
new Insets(2, 2, 2, 2), 0, 0));

        centralniPanel.add(jPanel3, new GridBagConstraints(0, 3, 1, 1, 1.0,
0.0, GridBagConstraints.NORTHWEST, GridBagConstraints.HORIZONTAL, new
Insets(2, 5, 0, 5), 1, 1));
        jPanel3.add(jLabel3, new GridBagConstraints(0, 0, 1, 1, 0.1, 0.0,
GridBagConstraints.NORTHWEST, GridBagConstraints.NONE, new Insets(2, 2, 2,
2), 0, 0));
        jPanel3.add(iPAdresaClientaF, new GridBagConstraints(1, 0, 1, 1,
0.9, 0.0, GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL,
new Insets(2, 2, 2, 2), 0, 0));
        jPanel3.add(jLabel8, new GridBagConstraints(0, 1, 1, 1, 0.1, 0.0,
GridBagConstraints.NORTHWEST, GridBagConstraints.NONE, new Insets(2, 2, 2,
2), 0, 0));
        jPanel3.add(udpPortNaClientuF, new GridBagConstraints(1, 1, 1, 1,
0.9, 0.0, GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL,
new Insets(2, 2, 2, 2), 0, 0));

        centralniPanel.add(jPanel5, new GridBagConstraints(0, 4, 1, 1, 1.0,
1.0, GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL, new
Insets(2, 5, 2, 5), 0, 0));
        jPanel5.add(jLabel6, new GridBagConstraints(0, 0, 1, 1, 0.1, 0.0,
GridBagConstraints.NORTHWEST, GridBagConstraints.NONE, new Insets(2, 2, 2,
2), 0, 0));
        jPanel5.add(brojStanjaF, new GridBagConstraints(1, 0, 1, 1, 0.9, 0.0,
GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL, new
Insets(2, 2, 2, 2), 0, 0));
        jPanel5.add(jLabel7, new GridBagConstraints(0, 1, 1, 1, 0.1, 0.0,
GridBagConstraints.NORTHEAST, GridBagConstraints.NONE, new Insets(2, 2, 2,
2), 0, 0));
```

```
                jPanel5.add(gamaF, new GridBagConstraints(1, 1, 1, 1, 0.9, 0.0,
GridBagConstraints.NORTHEAST, GridBagConstraints.HORIZONTAL, new
Insets(2, 2, 2, 2), 0, 0));
                gamaF.setText("0");
                gamaF.setEnabled(false);
                //this.getContentPane().add(jPanel4, BorderLayout.EAST);
                //jPanel4.add(startB, new GridBagConstraints(0, 0, 1, 1, 1.0, 0.0,
GridBagConstraints.NORTH, GridBagConstraints.HORIZONTAL, new Insets(2, 5,
2, 5), 0, 0));
                //jPanel4.add(prikazGrafovaB, new GridBagConstraints(0, 1, 1, 1, 1.0,
0.0, GridBagConstraints.NORTH, GridBagConstraints.HORIZONTAL, new
Insets(2, 5, 2, 5), 1, 1));
                //jPanel4.add(StopB, new GridBagConstraints(0, 2, 1, 1, 1.0, 1.0,
GridBagConstraints.NORTHWEST, GridBagConstraints.HORIZONTAL,new
Insets(2, 5, 2, 5), 0, 0));
                jMenuBar1.add(jMenu1);
                jMenuBar1.add(jMenu2);
                jMenuBar1.add(jMenu3);
                jMenuBar1.add(pomocM);
                jMenu1.add(snimiPodatkeUFileMI);
                jMenu1.add(ucitajDatotekuMI);
                jMenu1.add(exitMI);
                ;

                jMenu2.add(pokreniServerMI);
                jMenu2.add(zaustaviServerMI);
                jMenu3.add(grafOpterecenjeMI);
                jMenu3.add(grafBrojOdbacenihMI);
                jMenu3.add(grafZAjednoMI);
                jMenu3.add(grafAkcijeMI);

                pomocM.add(uputeZaUporabuMI);
        }

        void snimiPodatkeUFileMI_actionPerformed(ActionEvent e) {
                fc.setDialogTitle("Snimanje podataka iz simulacije u xml file");
                int save = fc.showSaveDialog(ServerWindow.this);

                if (save == JFileChooser.APPROVE_OPTION) {

                        File newFile = fc.getSelectedFile();

                        Element elementOpterecenje = null;
                        Element elementSerije = null;
                        Element elementAkcija = null;
                        Element elementDopustenoOpt = null;
                        Element elementParametriSimulacije = null;
```

```
                    Element elementOdbaceni = null;

                    Node node = null;
                    Document xmldoc = new DocumentImpl();

                    Element root =  xmldoc.createElement("SERVER_PODACI");

                    List lista = serijaPodatakaOpterecenjeServera.getItems();
                    Iterator ite = lista.iterator();
                    long brojac = 0;

                    elementOpterecenje = xmldoc.createElementNS(null,
"OPTERECENJE");
                    elementOdbaceni = xmldoc.createElementNS(null,
"ODBACENI");
                    elementAkcija = xmldoc.createElementNS(null, "AKCIJE");
                    elementDopustenoOpt =
xmldoc.createElementNS(null,"DOPUSTENO_OPTERECENJE");
                    elementParametriSimulacije =
xmldoc.createElementNS(null,"PARAMETRI_SIMULACIJE");

                    root.appendChild(elementOpterecenje);
                    root.appendChild(elementOdbaceni);
                    root.appendChild(elementAkcija);
                    root.appendChild(elementDopustenoOpt);
                    root.appendChild(elementParametriSimulacije);

                    while (ite.hasNext()) {
                        XYDataItem serija = (XYDataItem) ite.next();

                        elementSerije = xmldoc.createElementNS(null,
"SerijaOpterecenja");
                        elementSerije.setAttributeNS(null, "X", serija.getX() +
"");
                        elementSerije.setAttributeNS(null, "Y", serija.getY() +
"");

                        elementOpterecenje.appendChild(elementSerije);
                        brojac++;

                    }

                    List listaOdbacenihPaketa =
serijaPodatakaBrojOdbacenihPaketa.getItems();
                    ite = listaOdbacenihPaketa.iterator();
                    brojac = 0;
                    while (ite.hasNext()) {
                        XYDataItem serija = (XYDataItem) ite.next();
```

```
                        elementSerije = xmldoc.createElementNS(null,
"SerijaOdbacenih");
                        elementSerije.setAttributeNS(null, "X", serija.getX() +
"");
                        elementSerije.setAttributeNS(null, "Y", serija.getY() +
"");

                        elementOdbaceni.appendChild(elementSerije);

                        brojac++;

                }

                elementSerije = xmldoc.createElementNS(null,
"IzabraneAkcije");
                elementSerije.setAttributeNS(null, "Akcija0", "" +
serijaPodatakaAkcija.getValue(0));
                elementSerije.setAttributeNS(null, "Akcija1", "" +
serijaPodatakaAkcija.ge tValue(1));
                elementSerije.setAttributeNS(null, "Akcija2", "" +
serijaPodatakaAkcija.getValue(2));
                elementSerije.setAttributeNS(null, "Akcija3", "" +
serijaPodatakaAkcija.getValue(3));
                elementAkcija.appendChild(elementSerije);

                List listaDopustOpt = serijaDopustenoOpterecenje.getItems();
                ite = listaDopustOpt.iterator();
                brojac = 0;
                while (ite.hasNext()) {
                        XYDataItem serija = (XYDataItem) ite.next();

                        elementSerije = xmldoc.createElementNS(null,
"DopustenoOpt");
                        elementSerije.setAttributeNS(null, "X", serija.getX() +
"");
                        elementSerije.setAttributeNS(null, "Y", serija.getY() +
"");

                        elementDopustenoOpt.appendChild(elementSerije);

                        brojac++;

                }
                Element lokalniUdpPort =
xmldoc.createElementNS(null,"LokalniUdpPort");
                Element dopustenoOpterecenjeServera =
xmldoc.createElementNS(null,"DopustenoOpterecenjeServera");
```

```
                    Element vrijemeUzorkovanjaZaGraf =
xmldoc.createElementNS(null,"VrijemeUzorkovanjaZaGraf");
                    Element opterecenjePoPaketu =
xmldoc.createElementNS(null,"OpterecenjePoPaketu");
                    Element vrijemeObradePaketa =
xmldoc.createElementNS(null,"VrijemeObradePaketa");
                    Element iPAdresaKlijenta =
xmldoc.createElementNS(null,"IPAdresaKlijenta");
                    Element udpPortKlijenta =
xmldoc.createElementNS(null,"UdpPortKlijenta");
                    Element brojStanjaQTablice =
xmldoc.createElementNS(null,"BrojStanjaQTablice");


        lokalniUdpPort.setAttributeNS(null,"brojPorta",lokalniUdpF.getText());

        dopustenoOpterecenjeServera.setAttributeNS(null,"iznos",dopustanoOptServ
eraF.getText());

        vrijemeUzorkovanjaZaGraf.setAttributeNS(null,"vrijeme",uzimanjeUzorakaZ
aGrafF.getText());

        opterecenjePoPaketu.setAttributeNS(null,"iznos",opterecenjePoPaketuF.getT
ext());

        vrijemeObradePaketa.setAttributeNS(null,"vrijeme",vrijemeObradePaketaF.g
etText());

        iPAdresaKlijenta.setAttributeNS(null,"adresa",iPAdresaClientaF.getText());

        udpPortKlijenta.setAttributeNS(null,"brojPorta",udpPortNaClientuF.getText(
));

        brojStanjaQTablice.setAttributeNS(null,"iznos",brojStanjaF.getText());

                    elementParametriSimulacije.appendChild(lokalniUdpPort);

        elementParametriSimulacije.appendChild(dopustenoOpterecenjeServera);

        elementParametriSimulacije.appendChild(vrijemeUzorkovanjaZaGraf);

        elementParametriSimulacije.appendChild(opterecenjePoPaketu);

        elementParametriSimulacije.appendChild(vrijemeObradePaketa);
                    elementParametriSimulacije.appendChild(iPAdresaKlijenta);
                    elementParametriSimulacije.appendChild(udpPortKlijenta);

        elementParametriSimulacije.appendChild(brojStanjaQTablice);
```

```
                    xmldoc.appendChild(root);

                    try {

                            FileOutputStream fos = new
FileOutputStream(newFile.getPath());

                            OutputFormat of = new OutputFormat("XML", "ISO-
8859-2", true);
                            of.setIndent(1);

                            XMLSerializer serializer = new XMLSerializer(fos,
of);

                            serializer.asDOMSerializer();
                            serializer.serialize(xmldoc.getDocumentElement());
                            fos.close();

                    } catch (FileNotFoundException eer) {
                            int greska = JOptionPane.showOptionDialog(this,
"Doslo je do greske prilikom snimanja. Molimo probajte ponovno.", "Obavijest",
JOptionPane.YES_OPTION, JOptionPane.ERROR_MESSAGE, null, tipkaOK,
tipkaOK[0]);

                            //
        System.out.println("Problemi s file-om");
                    } catch (IOException eerr) {
                            int greska = JOptionPane.showOptionDialog(this,
"Doslo je do greske prilikom snimanja. Molimo probajte ponovno.", "Obavijest",
JOptionPane.YES_OPTION, JOptionPane.ERROR_MESSAGE, null, tipkaOK,
tipkaOK[0]);

                            //                              System.out.println("IO
Exception");
                    }

            }

        }

        void ucitajDatotekuMI_actionPerformed(ActionEvent e) {
                fc.setDialogTitle("Otvaranje snimljenih podataka");

                int returnVal = fc.showOpenDialog(ServerWindow.this);

                if (returnVal == JFileChooser.APPROVE_OPTION) {
```

```
                        File file = fc.getSelectedFile();

                        DefaultHandler handler = new SAXHandler();
                        SAXParserFactory factory =
SAXParserFactory.newInstance();

                        serijaPodatakaOpterecenjeServera = new
XYSeries("opterecenje server u postocima");
                        serijaPodatakaBrojOdbacenihPaketa = new XYSeries("broj
odbecenih paketa");
                        serijaDopustenoOpterecenje = new XYSeries("dopusteno
opterecenje");

                        try {
                                SAXParser saxParser = factory.newSAXParser();
                                saxParser.parse(file, handler);

                        } catch (Exception p) {
                                int greska = JOptionPane.showOptionDialog(this,
"Doslo je do greske prilikom otvaranja file-a. Molimo probajte ponovno.",
"Obavijest", JOptionPane.YES_OPTION, JOptionPane.ERROR_MESSAGE, null,
tipkaOK, tipkaOK[0]);
                        }

                }

        }

        void pokreniServerMI_actionPerformed(ActionEvent e) {


                getDataFromForm();
                if (ServerWindow.podaciSProzoraOK == true) {
                        t = (Thread) new RunServer();
                        monitoring = (Thread) new Monitoring();
                        t.start();
                        monitoring.start();
                        pokreniServerMI.setEnabled(false);
                        zaustaviServerMI.setEnabled(true);
                }

        }

        void zaustaviServerMI_actionPerformed(ActionEvent e) {
                pokreniServerMI.setEnabled(true);
                zaustaviServerMI.setEnabled(false);
                t.stop();
                monitoring.stop();
```

```
    }

    void grafOpterecenjeMI_actionPerformed(ActionEvent e) {


        XYSeriesCollection coll = new XYSeriesCollection();
        coll.addSeries(serijaDopustenoOpterecenje);
        coll.addSeries(serijaPodatakaOpterecenjeServera);

        JFreeChart graf = ChartFactory.createXYLineChart("Opterecenje
servera u vremenu", "vrijeme (ms)", "opterecenje servera (%)", coll,
PlotOrientation.VERTICAL, true, false, false);

        ChartFrame grafFrame = new ChartFrame("Prikaz grafa -  opterecenje
servera u vremenu", graf, true);
        grafFrame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        grafFrame.setSize(640, 480);
        grafFrame.setVisible(true);
    }

    void grafBrojOdbacenihMI_actionPerformed(ActionEvent e) {

        XYDataset dataset = new
XYSeriesCollection(serijaPodatakaBrojOdbacenihPaketa);
        JFreeChart graf = ChartFactory.createXYLineChart("Broj odbacenih
paketa u vremenu", "vrijeme (ms)", "broj odbacenih paketa", dataset,
PlotOrientation.VERTICAL, true, false, false);

        ChartFrame grafFrame = new ChartFrame("Prikaz grafa -  broj
odbacenih paketa u vremenu", graf, true);
        grafFrame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        grafFrame.setSize(640, 480);
        grafFrame.setVisible(true);

    }

    void grafAkcijaMI_actionPerformed(ActionEvent e) {

        ChartFrame grafFrame = new ChartFrame("Prikaz grafa -  odabir
zahtjeva za paketima", grafAkcija, true);
        grafFrame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        grafFrame.setSize(640, 480);
        grafFrame.setVisible(true);
    }

    void grafZAjednoMI_actionPerformed(ActionEvent e) {
```

```
                XYSeriesCollection coll = new XYSeriesCollection();
                coll.addSeries(serijaDopustenoOpterecenje);
                coll.addSeries(serijaPodatakaOpterecenjeServera);
                coll.addSeries(serijaPodatakaBrojOdbacenihPaketa);

                JFreeChart graf = ChartFactory.createXYLineChart("Opterecenje
servera i broj odbecenih paketa u vremenu", "vrijeme (ms)", "opterecenje(%)/broj
odbacenih paketa", coll, PlotOrientation.VERTICAL, true, false, false);
                ChartPanel grafPanel = new ChartPanel(graf, true);
                ChartPanel grafAkcijePanel = new ChartPanel(grafAkcija, true);

                JFrame grafFrame = new JFrame("Prikaz grafa -  opterecenja servera i
broj odbacenih paketa u vremenu");

        grafFrame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);

                JPanel osnovni = new JPanel(new BorderLayout());
                GridBagLayout gbLayout = new GridBagLayout();
                GridBagConstraints c = new GridBagConstraints();
                grafFrame.getContentPane().add(osnovni);
                JScrollPane scrollGraf = new JScrollPane(grafPanel);
                JScrollPane scrollAkcijeGraf = new JScrollPane(grafAkcijePanel);
                JPanel centralniP = new JPanel(new GridLayout(1, 1));
                centralniP.add(scrollGraf);
                osnovni.add(centralniP, BorderLayout.CENTER);
                grafFrame.setSize(640, 480);
                grafFrame.setVisible(true);

        }

        void uputeZaUporabuMI_actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(this, "Upute za koristenje programa
nisu dostupne.", "Pomoc", JOptionPane.INFORMATION_MESSAGE);
        }

        //dohvat podataka s forme
        public void getDataFromForm() {
                Validator v = new Validator();
                String temp = null;

                ServerWindow.podaciSProzoraOK = true;

                //lokalni udp port
                temp = lokalniUdpF.getText();
                if (v.doMatch(temp, "[0123456789]+")) {
                        ServerWindow.lokalniUdpPort = Integer.parseInt(temp);
                        if (ServerWindow.lokalniUdpPort <= 1024 ||
ServerWindow.lokalniUdpPort >= 65536) {
```

```
                                JOptionPane.showMessageDialog(this, "Lokalni UDP
port mora biti u rasponu od 1024 - 65536", "Opis greske",
JOptionPane.ERROR_MESSA GE);
                                ServerWindow.podaciSProzoraOK = false;
                                return;
                        }
                } else {
                        JOptionPane.showMessageDialog(this, "Lokalni UDP port
mora biti u rasponu od 1024 - 65536", "Opis greske",
JOptionPane.ERROR_MESSAGE);
                        ServerWindow.podaciSProzoraOK = false;
                        return;
                }

                //dopusteno opterecenje servera
                temp = dopustanoOptServeraF.getText();
                if (v.doMatch(temp, "[0123456789]+")) {
                        ServerWindow.prozorDopustenoOpterecenje =
Integer.parseInt(temp);
                        if (ServerWindow.prozorDopustenoOpterecenje <= 0 ||
ServerWindow.prozorDopustenoOpterecenje >= 101) {
                                JOptionPane.showMessageDialog(this, "Dopusteno
opterecenje servera mora biti u raspon od 1-100", "Opis greske",
JOptionPane.ERROR_MESSAGE);
                                ServerWindow.podaciSProzoraOK = false;
                                return;
                        }
                } else {
                        JOptionPane.showMessageDialog(this, "Dopusteno
opterecenje servera mora biti u raspon od 1-100", "Opis greske",
JOptionPane.ERROR_MESSAGE);
                        ServerWindow.podaciSProzoraOK = false;
                        return;
                }

                //opterecenje po paketu
                temp = opterecenjePoPaketuF.getText();
                if (v.doMatch(temp, "[0123456789]+")) {
                        ServerWindow.prozorOpterecenjePoPaketu =
Float.parseFloat(temp);
                        if (ServerWindow.prozorOpterecenjePoPaketu <= (float) 0 ||
ServerWindow.prozorOpterecenjePoPaketu >= (float) 101) {
                                JOptionPane.showMessageDialog(this, "Dopusteno
opterecenje po paketu mora biti u raspon od 1-100", "Opis greske",
JOptionPane.ERROR_MESSAGE);
                                ServerWindow.podaciSProzoraOK = false;
                                return;
                        }
```

```
                } else {
                        JOptionPane.showMessageDialog(this, "Dopusteno
opterecenje po paketu mora biti u raspon od 1-100", "Opis greske",
JOptionPane.ERROR_MESSAGE);
                        ServerWindow.podaciSProzoraOK = false;
                        return;
                }
                //uzorkovanje za graf
                temp = uzimanjeUzorakaZaGrafF.getText();
                try {
                        ServerWindow.vrijemeuzorkovanjaZaGraf =
Long.parseLong(temp);
                } catch (Exception e) {
                        JOptionPane.showMessageDialog(this, "U polje broj stanja
upisite samo brojke", "Opis greske", JOptionPane.ERROR_MESSAGE);
                        ServerWindow.podaciSProzoraOK = false;
                        return;
                }

                //vrijeme obrade po poaketu
                temp = vrijemeObradePaketaF.getText();
                try {
                        ServerWindow.prozorVrijemeObradePaketa =
Long.parseLong(temp);
                } catch (Exception e) {
                        JOptionPane.showMessageDialog(this, "Vrijeme odradu
upisuje se samo kao brojke", "Opis greske", JOptionPane.ERROR_MESSAGE);
                        ServerWindow.podaciSProzoraOK = false;
                        return;
                }

                //ip adresa clienta
                temp = iPAdresaClientaF.getText();
                if (v.doMatch(temp, "[0-2][0-9][0-9].[0-2][0-9][0-9].[0-2][0-9][0-
9].[0-2][0-9][0-9]")) {
                        ServerWindow.prozorIpClienta = temp;
                } else {
                        JOptionPane.showMessageDialog(this, "IP adresa clienta
upisuje se u obliku ###.###.###.###", "Opis greske",
JOptionPane.ERROR_MESSAGE);
                        ServerWindow.podaciSProzoraOK = false;
                        return;
                }

                //udp port clienta
                temp = udpPortNaClientuF.getText();
                if (v.doMatch(temp, "[0123456789]+")) {
                        ServerWindow.prozorClientPort = Integer.parseInt(temp);
```

```
                        if (ServerWindow.prozorClientPort <= 1024 ||
ServerWindow.prozorClientPort >= 65536) {
                                JOptionPane.showMess ageDialog(this, "UDP port
clienta mora biti u rasponu od 1024 - 65536", "Opis greske",
JOptionPane.ERROR_MESSAGE);
                                ServerWindow.podaciSProzoraOK = false;
                                return;
                        }
                } else {
                        JOptionPane.showMessageDialog(this, "UDP port clienta
mora biti u ʀsponu od 1024 - 65536", "Opis greske",
JOptionPane.ERROR_MESSAGE);
                        ServerWindow.podaciSProzoraOK = false;
                        return;
                }

                //broj stanja q tablice
                temp = brojStanjaF.getText();
                try {
                        ServerWindow.prozorBrojStanja = Integer.parseInt(temp);
                } catch (Exception e) {
                        JOptionPane.showMessageDialog(this, "U polje broj stanja
upisite samo brojke", "Opis greske", JOptionPane.ERROR_MESSAGE);
                        ServerWindow.podaciSProzoraOK = false;
                        return;
                }

        }

        public void odabranaAkcijaNulaPaketa() {
                double d = akcijeList[0];
                d = d + 1;
                akcijeList[0] = d;
                serijaPodatakaAkcija.setValue("Akcija - 0 paketa", akcijeList[0]);
                grafAkcija.fireChartChanged();

        }
        public void odabranaAkcijaJedanPaketa() {
                double d = akcijeList[1];
                d = d + 1;
                akcijeList[1] = d;
                serijaPodatakaAkcija.setValue("Akcija - 1 paketa", akcijeList[1]);
                grafAkcija.fireChartChanged();

        }

        public void odabranaAkcijaDvaPaketa() {
                double d = akcijeList[2];
```

```
                d = d + 1;
                akcijeList[2] = d;
                serijaPodatakaAkcija.setValue("Akcija - 2 paketa", akcijeList[2]);
                grafAkcija.fireChartChanged();

        }
        public void odabranaAkcijaTriPaketa() {
                double d = akcijeList[3];
                d = d + 1;
                akcijeList[3] = d;
                serijaPodatakaAkcija.setValue("Akcija - 3 paketa", akcijeList[3]);
                grafAkcija. fireChartChanged();

        }

        public void zabiljeziBrojOdbecenih() {
                ServerWindow.brojOdbacenihList.add(new
Integer(ServerWindow.brojOdbacenih));
                }

        public void zabiljeziTrenutnoOpterecenje() {
                ServerWindow.opterecenjeList.add(new Float(Server.opterecenje));
        }

        public JFrame getMe() {
                return this;
        }

        //biljezi podatke potrebne za grafove
        class Monitoring extends Thread {

                public void run() {
                        serijaPodatakaOpterecenjeServera = new
XYSeries("opterecenje server u postocima");
                        serijaPodatakaBrojOdbacenihPaketa = new XYSeries("broj
odbecenih paketa");
                        serijaDopustenoOpterecenje = new XYSeries("dopusteno
opterecenje");

                        long time = 0;
                        long brojac = 0;
                        while (true) {
                                brojac++;
                                time = System.currentTimeMillis();

                                serijaPodatakaOpterecenjeServera.add(((double)
ServerWindow.vrijemeuzorkovanjaZaGraf * brojac), (double) Server.opterecenje);
```

```
                        serijaPodatakaBrojOdbacenihPaketa.add(((double)
ServerWindow.vrijemeuzorkovanjaZaGraf * brojac), (double)
ServerWindow.brojOdbacenih);
                        serijaDopustenoOpterecenje.add(((double)
ServerWindow.vrijemeuzorkovanjaZaGraf * brojac), (double)
ServerWindow.prozorDopustenoOpterecenje);

                        serijaPodatakaOpterecenjeServera.fireSeriesChanged();

        serijaPodatakaBrojOdbacenihPaketa.fireSeriesChanged();
                        try {
                                long g = (System.currentTimeMillis() - time);
                                if (g < (long) 0){
                                        g = 0;
                                }

                                if(ServerWindow.vrijemeuzorkovanjaZaGraf - g
> 0){

        Thread.sleep(ServerWindow.vrijemeuzorkovanjaZaGraf - g);
                                }

                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }
        }

        //klasa koja pokrece server
        class RunServer extends Thread {

                public void run() {
                        serijaPodatakaAkcija = new DefaultPieDataset();
                        grafAkcija = ChartFactory.createPie 3DChart("Odabir zahtjeva
za paketima", serijaPodatakaAkcija, true, false, false);
                        ServerWindow.opterecenjeList = new ArrayList();
                        ServerWindow.brojOdbacenihList = new ArrayList();
                        ServerWindow.brojOdbacenih = 0;
                        akcijeList = new double[4];

                        if (ServerWindow.podaciSProzoraOK == true) {

                                Server server = new
Server(prozorOpterecenjePoPaketu, prozorVrijemeObradePaketa,
prozorDopustenoOpterecenje, prozorBrojStanja, prozorIpClienta, prozorClientPort);
                        Qalgoritam q = new Qalgoritam();
```

```
                        byte[] buffer = new byte[1024];
                        DatagramPacket paketForClient = null;
                        DatagramPacket fromClient = new
DatagramPacket(buffer, buffer.length);
                        String zahtjevaniBrojPaketa = null;
                        DatagramSocket socket = null;
                        float novoOpterecenje = 0;
                        boolean ovelload = false;
                        long odbaceni = 0;
                        try {

                                socket = new
DatagramSocket(ServerWindow.lokalniUdpPort);
                                int brPaketa = 0;
                                long vrijemeSlanjaPaketa = 0;
                                while (true) {
                                        // zatrazimo od clienta da na posalje
odredjeni broj paketa
                                        //brPaketa = q.odabirAkcije();
                                        brPaketa = q.newOdabirAkcije();
                                        if (brPaketa == 0) {
                                                odabranaAkcijaNulaPaketa();
                                        } else {
                                                if (brPaketa == 1) {

        odabranaAkcijaJedanPaketa();
                                                } else {
                                                        if (brPaketa == 2) {

        odabranaAkcijaDvaPaketa();
                                                        } else {

        odabranaAkcijaTriPaketa();
                                                        }
                                                }
                                        }
                                        zahtjevaniBrojPaketa =
Integer.toString(brPaketa);
                                        buffer =
zahtjevaniBrojPaketa.getBytes();

                                        paketForClient = new
DatagramPacket(buffer, buffer.length,
InetAddress.getByName(iPAdresaClientaF.getText()),
ServerWindow.prozorClientPort);

                                        if (brPaketa != 0) {
```

```
                                                vrijemeSlanjaPaketa =
System.currentTimeMillis();
                                                socket.send(paketForClient);



                                                for (int i = 0; i < brPaketa; i++) {

        socket.receive(fromClient);
                                                        if (Server.opterecenje +
brPaketa * Server.opterecenjePoPaketu > Server.dopustenoOpterecenje) {
                                                                ovelload = true;
                                                        } else {
                                                                Thread t =
(Thread) new PacketProcessing(fromClient);
                                                                t.start();
                                                        }
                                                }
                                        }
                                        float nagrad = 0;
                                        if (ovelload == false) {
                                                nagrad =
Server.newNagrada(Server.opterecenje, brPaketa, q, vrijemeSlanjaPaketa,
System.currentTimeMillis());

        q.q[Server.trenutnoStanje][brPaketa] = nagrad;


                                        } else {

        ServerWindow.brojOdbacenih++;

        q.q[Server.trenutnoStanje][brPaketa] = (float) 1.0;

                                                nagrad = (float) 1.0;

                                        }
                                        ovelload = false;
                                        Server.trenutnoStanje++;
                                        Server.trenutnoStanje =
Server.trenutnoStanje % Server.brojStanja;
                                        q.trenutnoStanje =
Server.trenutnoStanje;



                                        try {
                                                if (brPaketa == 0) {
```

```
                                        Thread.sleep(1000);
                                    } else {
                                        long t = 1000 -
(System.currentTimeMillis() - vrijemeSlanjaPaketa);
                                        if (t > 0) {
                                            sleep(t);
                                        }
                                    }
                                } catch (InterruptedException e1) {
                                    e1.printStackTrace();
                                }
                            }
                        } catch (SocketException ee) {

                            int greska =
JOptionPane.showOptionDialog(getMe(), "Dogodila se greska vezana uz UDP port,
vjerojatno je zauzet. Molimo probajte neki drugi port.", "Obavijest",
JOptionPane.YES_OPTION, JOptionPane.ERROR_MESSAGE, null, tipkaOK,
tipkaOK[0]);

                        } catch (UnknownHostException eee) {
                            int greska =
JOptionPane.showOptionDialog(getMe(), "Nepoznata ip adresa, probajte ponovno",
"Obavijest", JOptionPane.YES_OPTION, JOptionPane.ERROR_MESSAGE, null,
tipkaOK, tipkaOK[0]);

                        } catch (IOException eeee) {
                            int greska =
JOptionPane.showOptionDialog(getMe(), "IOException, greska prilikom slanja i
dobivanja paketa", "Obavijest", JOptionPane.YES_OPTION,
JOptionPane.ERROR_MESSAGE, null, tipkaOK, tipkaOK[0]);
                        }
                    }
                }
            }
        }
```

8.8. Klasa Server.java

```
package hr.fer.server;

/**
 * @author Stjepan Matijasevic
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class Server {

	public static float opterecenje = 0;
	public static float opterecenjePoPaketu = 0;
	public static long vrijemeObradePaketa = 0;
	public static float dopustenoOpterecenje = 0;
	public static int trenutnoStanje = 0;
	public static int brojStanja = 0;


	public Server(float optPoPaketu, long vrijemeObradePaketa, float
dopustenoOpt, int brojStanjaQTablice, String ipClienta,int portClienta) {

	Server.opterecenjePoPaketu = optPoPaketu;
	Server.vrijemeObradePaketa = vrijemeObradePaketa;
	Server.dopustenoOpterecenje = dopustenoOpt;
	Server.brojStanja = brojStanjaQTablice;
	}

	public static float newNagrada(float opterecenjeNakonDodatnihPaketa, int
brPaketa, Qalgoritam qq, long t1, long t2) {

		if (brPaketa == 0) {
			float r = ((float) 1.0) - (float)
(opterecenjeNakonDodatnihPaketa / dopustenoOpterecenje);
			return r;
		}

		float r = opterecenjeNakonDodatnihPaketa / dopustenoOpterecenje +
((float) (t2 - t1) / 10000);
		return r;

	}
}
```

8.9. Klasa Qalgoritam.java

```
package hr.fer.server;

import java.util.ArrayList;
import java.util.Random;

/**
 * @author Stjepan Matijasevic
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class Qalgoritam {
	public static ArrayList qTablica = null;

	public int trenutnoStanje = 0;
	public float q[][] = new float[Server.brojStanja][4];
	public boolean f = true;

	public Qalgoritam() {

		clearQTable();
	}


	public void clearQTable() {
		int p = Server.brojStanja;
		if (p == 0)
			p = 1;
		for (int i = 0; i < p; i++) {
			for (int j = 0; j < 4; j++) {
				q[i][j] = 0;
			}
		}
	}

	//vraca trenutnu q-tablicu
	public float[][] getNewQTable() {
		return q;
	}

	public int newOdabirAkcije() {
		float nula = q[trenutnoStanje][0];
		float prvi = q[trenutnoStanje][1];
```

```
                float drugi = q[trenutnoStanje][2];
                float treci = q[trenutnoStanje][3];

                if ((nula == prvi) && (nula == drugi) && (nula == treci)) {

                        Random r = new Random();
                        int akcija = r.nextInt(4);
                        return akcija;
                } else {
                        int indexNajmanjeg = 0;
                        float min = q[trenutnoStanje][0];
                        float temp = (float) 0.0;

                        for (int i = 1; i < 4; i++) {
                                temp = q[trenutnoStanje][i];
                                if (min >= temp) {
                                        min = temp;
                                        indexNajmanjeg = i;
                                }
                        }
                        return indexNajmanjeg;
                }
        }
}
```

8.10. Klasa PacketProcessing.java

```
package hr.fer.server;

import java.net.DatagramPacket;

/**
 * @author Stjepan Matijasevic
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class PacketProcessing extends Thread {

	DatagramPacket paket = null;
	byte[] buffer = new byte[1024];

	public PacketProcessing(DatagramPacket p) {
		paket = p;

	}

	public void obrada() {
		Server.opterecenje = Server.opterecenje +
Server.opterecenjePoPaketu;

		try {
			long time = System.currentTimeMillis();
			sleep((long)Server.vrijemeObradePaketa);

		} catch (InterruptedException e) {
		}
		Server.opterecenje = Server.opterecenje -
Server.opterecenjePoPaketu;
	}
	public void run() {

		obrada();
	}
}
```