ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

IMPLEMENTACIJA ALGORITMA ZA NALAŽENJE PUTA U OPTIČKOJ MREŽI BEZ MOGUĆNOSTI KONVERTOVANJA TALASNE DUŽINE

– Diplomski rad –

Kandidat:

Ivan Radivojević 2010/268

Mentor:

doc. dr Zoran Čiča

Beograd, Oktobar 2014.

SADRŽAJ

1. UVOD

Razvoj telekomunikacionih tehnologija je glavni uzrok za upotrebu širokog spektra telekomunikacionih servisa među ogromnim brojem korisnika koji se iz godine u godinu progresivno povećavao. I upravo je dalji razvoj telekomunikacionih tehnologija mogao zasititi narastajuću potrebu za većim kapacitetima, odnosno potrebu za neometanim pružanjem postojećih servisa starim i novim korisnicima, kao i uvođenje novih servisa koji bi privukli još veći broj novih korisnika. Tehnologija koja se dominantno nametnula je omogućila i štaviše nadmašila zahteve koji su pred nju postavljeni. Reč je o optičkim komunikacijama.

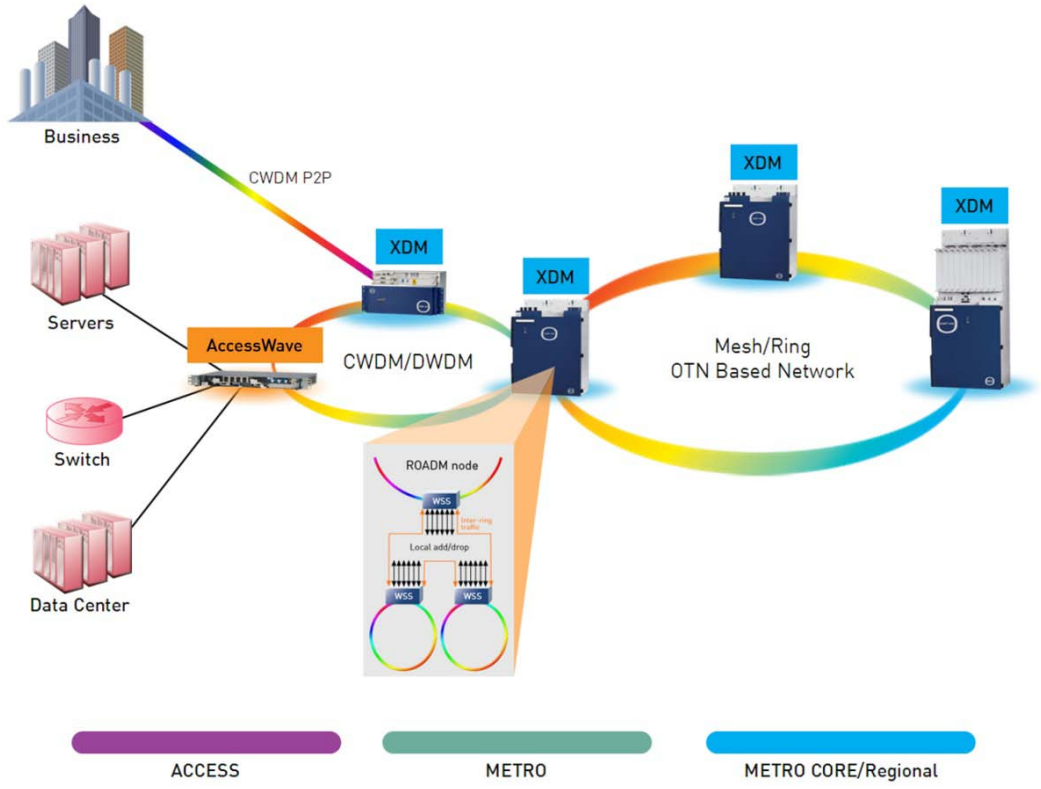
Optičke komunikacije dovele su do revolucije u oblasti telekomunikacionih tehnologija omogućavajući ogromne kapacitete po niskoj ceni instalacije i ugradnje. Prednost optičkih komunikacija se najviše ogleda u medijumu za prenos korisničkog signala. Optička vlakna, koja su sačinjena od jezga (staklo) i omotača (poliuretan), bez obzira što je reč o posebno izrađenoj vrsti stakla, daleko su jeftinija za proizvodnju i ugradnju u odnosu na cenu proizvodnje i ugradnje bakarnih vodova koji bi omogućili ekvivalentni kapacitet, a koji su do komercijalizacije tehnologije optičkih vlakana držali primat kao medijum za prenos u pristupnim i transportnim mrežama. Pored cene koja je dominantan faktor odlučivanja još neke od prednosti su: veliki protoci, mali gubici u medijumu za prenos, otpornost na ometanje i mali stepen interferencije sa drugim komunikacionim sistemima, kod multimodnih vlakana prenos većeg broja nezavisnih modova. Sve je to dovelo do prakse da gde god je moguće postaviti optički link on ima prednost u odnosu na bakarni provodnik ili radio link. Visoki kapaciteti koje su optički sistemi doneli zaslužni su za širokopojasni pristup internetu i izuzetan razvoj transportnih mreža.

Ovaj rad se bavi konkretno primenom optičkih komunikacija u transportnim mrežama. Optičke transportne mreže OTN (*Optical Transport Network*) su najpre implementirale WDM (*Wavelength-Division Multiplexing*) tehniku multipleksiranja, a naknadno shodno zahtevima i TDM (*Time-Division Muliplexing*) tehniku, obzirom da su OTN mreže potpadale pod SDH (*Synchronous Digital Hierarchy*) hijerarhiju koja je standardizovala TDM. Naime WDM tehnika je omogućila da se dobije ukupan porast kapaciteta linka, a da se protok ne poveća preterano ni na jednoj talasnoj dužini. Prednost postaje jasna kada se zna da ako se želi postići kapacitet od 400 Gb/s, bolje upotrebiti WDM tehniku da se prenosi po 10 Gb/s na četrdeset talasnih dužina, nego da se koristi jedna talasna dužina preko koje se prenosi 400 Gb/s [1].

U poglavlju 2 ovog rada biće dat opis problema koji se tiče pronalaženja slobodnog puta u mrežama različite topologije, nalaženja slobodne talasne dužine na putu i njenog zauzimanja, kao i pokušaja ostvarivanja dvosmerne veze. U poglavlju 3 ovog rada biće dat opis algoritma za nalaženje puta duž koga se koristi jedna talasna dužina, a u poglavlju 4. način kreiranja i izbora određene topologije kao i simulacioni rezultati za primere dinamičke uspostave i raskida veza. U prilogu će biti dat celokupan programski kod.

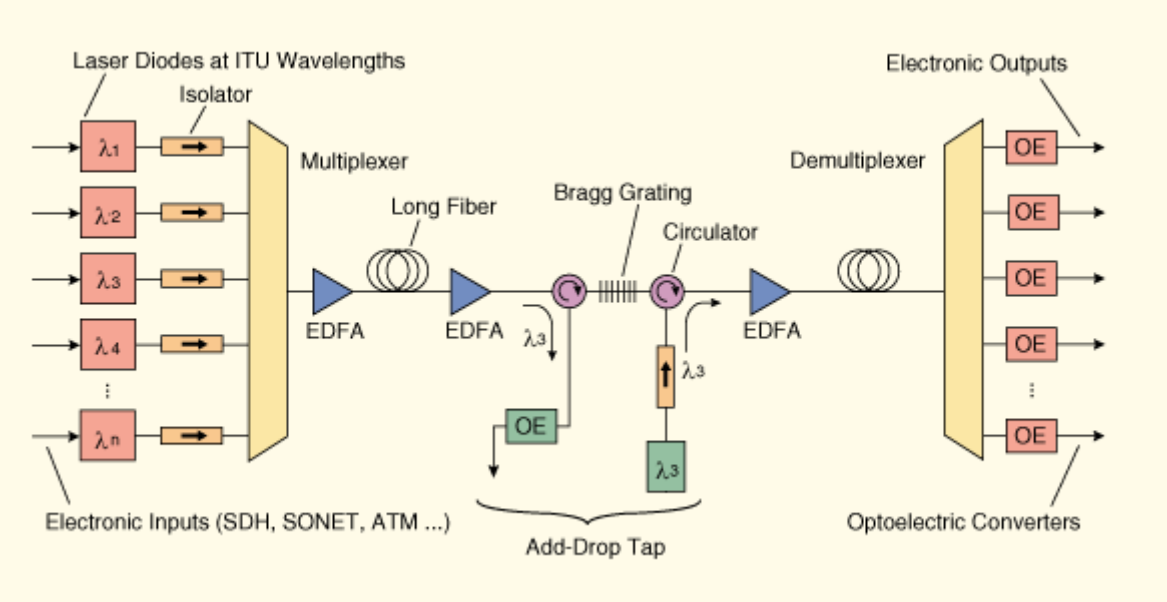
2. OPIS PROBLEMA

Optičko umrežavanje predstavlja vrstu komunikacije koja koristi optičke signale da prenese informaciju između različitih tipova mrežnih čvorova jedne telekomunikacione mreže. Ovaj tip umrežavanja koristi se u LAN (*Local Area Network*) mrežama ograničenog dometa ili u WAN (*Wide Area Network*) mrežama velikog dometa, tako da daljine koje optičke mreže pokrivaju variraju od gradskih i međugradskih, preko regionalnih i nacionalnih do internacionalnih i transokeanskih distanci. Najpre treba napomenuti da kada se kaže da se u optičkim mrežama koristi optički signal, to ne predstavlja tačnu definiciju. Naime, signali koji se prenose u optičkim mrežama zauzimaju deo spektra elektromagnetnih talasa koji je jako blizu delu spektra vidljive svetlosti, ali ljudsko oko nije sposobno da registruje takve signale jer spada u domen infracrvene svetlosti. Prvobitni optički sistemi su radili u domenu vidljive svetlosti, ali se kasnije prešlo na više učestanosti zbog brojnih prednosti. Tipične talasne dužine (talasne dužine se koriste kao parametar umesto frekvencije u optičkim komunikacijama) o kojima govorimo su oko 850, 1300 i 1550 nanometara. Osnovni elementi u optičkoj mreži su: izvori optičkog signala (laseri i LED diode), fiber optički kablovi kao medijum za prenos, pojačavači i regeneratori signala, i detektori optičkog signala (fotodioda). Kao osnovna tehnika multipleksiranja koristi se WDM. Obzirom da optički sistemi omogućavaju ekstremno visok propusni opseg, oni su tehnologija koja je dominantno zaslužna za implementaciju širokopojasnih telekomunikacionih sistema, i pružanje servisa koji su bazirani na tim sistemima [2].



Slika 2.1. Primer optičke mreže

Multipleksiranje po talasnoj dužini (WDM) je metod kombinovanja više signala na bazi laserskog elektromagnetnog talasa na različitim infracrvenim talasnim dužinama za potrebe transmisije kroz fiber optički medijum. Svaki laser je modulisan od strane nezavisnog skupa signala. WDM je tehnika slična tehnici frekvencijskog multipleksiranja FDM (*Freqency-Division Multiplexing*). FDM se odnosi RF (*Radio Frequency*) deo elektromagnetnog spektra, a WDM se odnosi na IR (*Infra Red*) deo elektromagnetnog spektra. Svaki IR kanal nosi nekoliko RF signala kombinovanih putem FDM ili TDM tehnike. Svaki multipleksirani IR kanal (na nekoj talasnoj dužini) se demultipleksira i razdvaja u originalne signale na prijemnoj strani optičkog linka. Korišćenje FDM ili TDM tehnike u svakom IR kanalu u kombinaciji sa korišćenjem WDM i nekoliko IR kanala dovodi do mogućnosti da podaci različitih formata na različitim brzinama mogu biti preneti istovremeno putem jednog optičkog vlakna. U ranim WDM sistemima korišćena su dva IR kanala po jednom optičkom vlaknu. Na prijemu je korišćen *dichroic* filter (dve talasne dužine) sa odsecanjem približno na srednjoj vrednosti između dve talasne dužine korišćenih kanala. Uskoro je postalo jasno da daleko više od dva kanala može biti multipleksirano/demultipleksirano upotrebom kaskadno vezanih *dichroic* filtara. To je bio uvod u razvoj i implementaciju CWDM (*Coarse Wavelength-Division Multiplexing*) i DWDM (*Dense Wavelength-Division Multiplexing*) tehnika. U CDWM se obično koristi 8 kanala, a može ih biti i do 18. U DWDM ih može biti daleko više.



Slika 2.2. Šematski prikaz DWDM tehnike

Kako svaki kanal nosi sopstveni set multipleksiranih RF signala teoretski je moguće preneti mešovite podatke putem jednog optičkog vlakna na efektivnim brzinama reda veličine nekoliko stotina gigabita po sekundi. Upotreba WDM tehnike umnožava sa velikim faktorom efektivni propusni opseg fiber optičkog komunikacionog sistema, ali isplativost sistema mora konstantno biti evaluirana protiv alternativnog rešenja korišćenja više optičkih vlakana spregnutih u optički kabl. Mana WDM tehnike ogleda se u ograničenjima vezanim za upotrebu velikog broja talasnih dužina, kao i nemogućnosti postizanja velikih dometa jer se prenošena energija troši na čvorovima pa se deo energije koji stiže na odredište drastično smanjuje kako se broj čvorova u mreži povećava. U prilog WDM tehnici ide rešenje koje se ogleda u upotrebi regeneratorskog uređaja nazvanog erbijumski pojačavač koji omogućava da WDM tehnika dugoročno bude isplativo rešenje [3].

Ovaj rad se bavi problematikom nalaženja puta u optičkoj mreži u kojoj čvorovi nemaju mogućnost konvertovanja talasne dužine. Za čvor se kaže da ima mogućnost konverzije talasne dužine ukoliko je u stanju da pomera talasnu dužinu prenošenog optičkog signala. Konvertor

talasne dužine je optički uređaj koji prebacuje jednu talasnu dužinu u drugu talasnu dužinu. Prednost konverzije se ogleda u tome da mreža u kojoj postoji mogućnost konvertovanja podržava više tokova, a nedostaci se ogledaju u problemima vezanim za složenost implementacije kao i cenu ovog rešenja. Kao kompromisno rešenje ponuđene su WDM optičke mreže sa limitiranom mogućnošću konverzije talasne dužine. Topologije koje će biti obrađene u ovom radu i koje inicijalno predstavljaju uzorak za ispitivanje i implementaciju RWA (*Route and Wavelength Assignment*) algoritama su topologije zvezde i prstena. Eventualno ako algoritmi budu zadovoljili kriterijume koji su pred njih postavljeni biće primenjeni na generalne mrežne topologije.

Postavka problema obrađenog u ovom radu je sledeća: treba kreirati algoritam za nalaženje puta u optičkoj mreži od N čvorova i L linkova, koji će imati istu slobodnu talasnu dužinu na svim deonicama puta od izvorišnog do odredišnog čvora. Svaki čvor će raspolagati sa K talasnih dužina, a između svaka dva čvora moraju postojati bar dva puta. Algoritam kreiran na osnovu datih uslova biće predstavljen u poglavlju 3 ovog rada. Zatim treba sprovesti simulacije dinamičke uspostave i raskida veza i komentarisati dobijene rezultate. Dinamička uspostava i raskid veza se mogu izvršiti u svakom vremenskom slotu, a na osnovu predefinisanih vrednosti verovatnoće uspostave P1 (za uspostavu veza) i verovatnoće raskida P2 (za raskid veza). Algoritam iz poglavlja 3 će biti upotrebljen u simulaciji, odnosno prilikom pokušaja uspostava veza, za vezu će se smatrati slobodan link između dva čvora na istoj talasnoj dužini duž svih deonica puta. Za rezultate ove simulacije treba prikazati broj uspešno uspostavljenih veza, broj neuspešno uspostavljenih veza i odnos uspešno/neuspešno uspostavljenih veza, što će davati verovatnoću blokiranja veze. Proces simulacije, obrada simulacionih rezultata, kao i komentar istih biće dat u okviru poglavlja 4

3. OPIS ALGORITMA ZA NALAŽENJE PUTA

Algoritam za nalaženje puta između dva čvora tako da taj put ima slobodnu talasnu dužinu na svim svoj deonicama, napisan je u programskom jeziku C++. U ovom poglavlju ćemo predstaviti i objasniti ključne delove programskog koda koji se odnose na realizaciju ovog algoritma. Najpre napomenimo da je algoritam za pronalaženje slobodne putanje kreiran za topologiju zvezde i topologiju prstena. Bez obzira na topologiju algoritam za nalaženje puta ostaje isti.

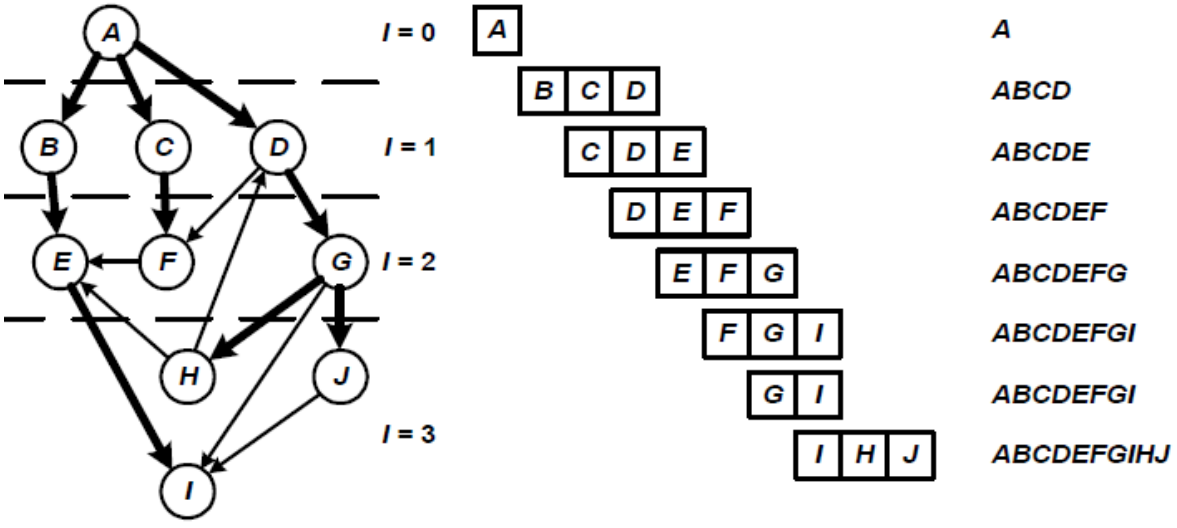
Da bi bolje pojasnili algoritam ovde ćemo dati delove programskog koda fajla ’Mreza.cpp’ kao i komentare na najbitnije linije koda vezane za algoritam. Još jedna napomena: u okviru fajla ’Mreza.cpp’ se pored kreiranih funkcija za potrebe nalaženja slobodnog puta nalaze integrisane i funkcije za kreiranje topologije i dinamičku uspostavu i raskid veza, kao i deo koda za sprovođenje simulacije. Ključna funkcija za kreiranje algoritma je funkcija ’*prohodnost*’.

```
int Mreza::prohodnost(int id1,int id2)
```

Ova funkcija vraća vrednost talasne dužine na kojoj je moguće uspostaviti vezu između čvora 1 i čvora 2 ili -1 ukoliko nije moguće ostvariti vezu ni na jednoj talasnoj dužini između ova dva čvora. Čvorovi su definisani promenljivim *id1* i *id2* tipa *integer*.

Algoritam ove funkcije je zapravo modifikovani algoritam obilaska grafa po širini. Strategija ovog algoritma se sastoji od sledećih koraka:

- poseti početni čvor
- poseti njegove susede
- poseti njihove neposećene susede istim redom



Slika 3.1. Primer obilaska grafa po širini

Posete su u „talasima“ po nivoima udaljenosti, a koristi se neprioritetni red za čekanje i vektor posećenosti. Modifikacija se sastoji u tome što se obilazak grafa izvršava više puta, za svaku talasnu dužinu ponaosob i prekida se ukoliko se pre obilaska celog grafa, u ovom slučaju mreže utvrdi mogućnost konektovanja [4].

Prvi deo algoritma obezbeđuje i inicijalizuje podatke struktura:

```
int Mreza::prohodnost(int id1,int id2)
{
	Cvor *c1=0, *c2=0, *pom=0;;
	c1 = &pronadjiC(id1);
	c2 = &pronadjiC(id2);

	Red *red=new Red();
```

Kreće sa pretragom mreže po talasnim dužinama, mreža se posebno obilazi za svaku talasnu dužinu:

```
	/* ispitivanje za svaku frekvenciju */
	int *provereni=new int[brC];
	for(int i=0; i<opseg; i++)
	{
		red->dodaj(id1);
		provereni[0]=id1; //interni spisak cvorova koji su vec obradjeni
		for (int loc=1; loc<brC; provereni[loc++] =- 1);
```

Bitne strukture koje se ovde inicijalizuju su:

1) red koji će čuvati redosled provere čvorova počevši od čvora *id1*
2) spisak čvorova koje je algoritam već obišao za ovu talasnu dužinu takođe odmah ubacivši čvor *id1*

Sve dok ne naiđe na prvu talasnu dužinu za koju se utvrdi da je moguće uspostaviti vezu od čvora *id1* do čvora *id2*; odnosno sve dok se red koji čuva redosled provere ne isprazni:

```
		while(!red->prazan())
		{
			int pt=red->dohvati();
			pom=&pronadjiC(pt);
			/*provera da li se cvor koji se trenutno obradjuje granici sa
	cvorom koji je destinacija */
```

Ukoliko čvor *id2* nije prvi sused čvoru kojeg obrađujemo (za prvi prolaz čvor *id1*):

```
			if(!(pom->povezan(c2)))
			{
				int j=0;
				/* sve dok cvor koji se trenutno obradjuje ima suseda */
				while(pom->susedni(j)!=-1)
				{
					int k=0;
					bool ind=true;
				/* indikator da sused cvora nije do sad bio posecen */
					while((provereni[k] != -1) && (ind))
					{
						if(pom->susedni(j)==provereni[k])
						ind =false;
						k++;
```

```
                                }
                                if(ind)
                                {
/* ako je frekvencija slobodna stavi cvor u spisak cvorova za dalju proveru */
                                        if(pom->proveraVeze(j,i))
                                        {
                                                pt=pom->susedni(j);
                                                red->dodaj(pt);
                                                provereni[k]=pt;
                                        }
                                }
                                j++;
                        }
                }
```

Ukoliko čvor *id2* jeste sused čvora koji trenutno obrađujemo:

```
                else
                {
                        int j=pom->povezanNa(c2);
                        if(pom->proveraVeze(j,i))
                        {
                                delete red;
                                delete provereni;
                                return i;
                                // vracamo frekvenciju ukoliko je slobodna
                        }
                }
        } // praznjenje reda priprema za prolazak ispitivanja za sledecu
frekvenciju
```

Ustanovljena je mogućnost uspostavljanja putanje i detektovana je talasna dužina koja to omogućava. Brišu se strukture i kao vrednost funkcije vraća se vrednost talasne dužine i time se algoritam prekida. Ukoliko je talasna dužina zauzeta nastavlja se sa pretragom po redosledu ne bi li se došlo do drugog čvora koji je povezan sa čvorom *id2*.

```
        while(!red->prazan())
        {
                int a=red->dohvati();
        }
}
```

Ukoliko je algoritam stigao do ove faze ustanovljeno je da su svi mogući putevi na svim frekvencijama između čvorova *id* i *id2* neprohodni, oslobađaju se zauzeti resursi podataka i vraća se -1 kao indikator da trenutno nije moguće uspostaviti vezu između dva zadata čvora. Napomenimo da je u kodu korišćen termin frekvencija umesto talasna dužina zbog kraćeg pisanja.

```
delete red;
delete provereni;
return -1;
```

Dajmo jedan prost primer koji će uprošćeno pokazati na koji način radi opisana funkcija.

Neka je data prosta mreža u kojoj su neki linkovi na izabranoj talasnoj dužini zauzeti. Uzmimo nasumično iz date mreže dva čvora između kojih želimo pronaći slobodan put na toj talasnoj dužini. Proveru počinjemo od prvog čvora koji treba povezati. Smeštamo ga u listu koju možemo nazvati „Provereni“. Za čvor koji se nalazi u ovoj listi ispitujemo dostupnost linkova prema drugim čvorovima za traženu talasnu dužinu. Neka na primer ovaj čvor ima slobodne linkove prema neka dva čvora. Ta dva čvora se smeštaju u listu „Provereni“ kao i u pomoćnu listu

koju možemo nazvati „Susedi“. Sad krenemo redom proveru po članovima ove pomoćne liste. Za jedan od čvorova ispitujemo dostupnost linkova na datoj talasnoj dužini prema čvorovima koji su povezani sa tekućim čvorom iz pomoćne liste. Ako pronađemo link prema čvoru koji se već nalazi u listi „Provereni“ ne radimo ništa. Ako ne pronađemo slobodne linkove prema drugim čvorovima takođe ne radimo ništa već brišemo taj čvor iz liste „Susedi“. Međutim, ako pronađemo slobodne linkove na datoj talasnoj dužini prema nekom novom čvoru tada smeštamo taj novi čvor u obe liste i brišemo onaj tekući čvor iz pomoćne liste „Susedi“ i prelazimo na sledeći čvor u toj listi. Postupak ponavljamo sve dok se i čvor koji je cilj (drugi od dva čvora koji su zadati da se između njih proveri dostupnost na zadatoj talasnoj dužini duž celog puta) ne nađe u listi „Provereni“. U slučaju da nema slobodnog puta do tog čvora program će javiti da nije moguće ostvariti traženu konekciju. Algoritam je realizovan tako da u slučaju da postoji više mogućih putanja biće izabrana ona najkraća. Izbor najkraće putanje biće postignut zahvaljujući funkcijama ’*najblizi*’ i ’*konektuj*’ koje ćemo objasniti.

Sledeća funkcija bitna za nalaženje puta između dva čvora je funkcija ’*najblizi*’. Ova funkcija traži najbliži čvor čvoru *id2* na putanji između čvorova *id1* i *id2* i talasnoj dužini *f* , i vraća id čvora koji je povezan sa *id2* i ima slobodnu talasnu dužinu *f*. Ova funkcija je zapravo izmenjena funkcija ’*prohodnost*’. U simulaciji se koristi isključivo nakon dobijene talasne dužine iz funkcije ’*prohodnost*’ za isti par čvorova *id1* i *id2*, a u sklopu funkcije ’*konektuj*’ koja će zauzimati talasnu dužinu na putu.

```
int Mreza::najblizi(int id1,int id2,int f)
{
	Cvor *c1=0, *c2=0, *pom=0;;
	c1 = &pronadjiC(id1);
	c2 = &pronadjiC(id2);

	Red *red=new Red();

	int *provereni=new int[brC];
	//interni spisak cvorova koji su vec obradjeni

	red->dodaj(id1);
	for (int loc=0; loc<brC;loc++) provereni[loc]=-1;
	provereni[0]=id1;
	/* sve dok ima potencijalnih cvorova koji su dostupni za frekvenciju koja
se ispituje */
```

Odnosno sve dok se red koji čuva redosled provere čvorova ne isprazni:

```
	while(!red->prazan())
	{
		int pt=red->dohvati();
		pom=&pronadjiC(pt);

		int j=0;
		/* sve dok cvor koji se trenutno obradjuje ima suseda */
		while(pom->susedni(j)!=-1)
		{
		/* da li se cvor koji se trenutno obradjuje granici sa cvorom koji
je destinacija */
			if(pom->povezan(c2))
			{
				int u=pom->povezanNa(c2);
				//ako je cvor destinacije sused cvoru koji se trenutno
obradjuje proveravamo dostupnost na trazenoj frekvenciji
```

Dakle, ako je čvor *id2* sused čvoru koji se trenutno obrađuje i ukoliko je talasna dužina *f* slobodna između njih, ustanovljena je mogućnost uspostavljanja putanje na traženoj talasnoj dužini, tada posmatrani čvor na svim linkovima oko sebe zauzima tu talasnu dužinu. Brišu se strukture i kao vrednost funkcije vraća se identitet čvora koji trenutno obrađujemo, a time i algoritam izlazi iz funkcije. Ukoliko je talasna dužina zauzeta nastavlja se sa pretragom po redosledu ne bi li se došlo do drugog čvora koji je povezan sa čvorom *id2*.

```
                        if(pom->proveraVeze(u,f))
                        {
                              delete red;
                              delete provereni;
                              return pom->identitet();
// vracamo id cvora koji ima direktnu vezu sa destinacijom
                        }
                  }
                  int k=0;
                  bool ind=true;
                  while((provereni[k] != -1) && (ind))
                  {
                        if(pom->susedni(j)==provereni[k])ind =false;
                        k++;
                  }
                  /* sused cvora nije do sad bio posecen */
                  if(ind)
                  {
                        if(pom->prvoeraVeze(j,f))
                        {
                              pt=pom->susedni(j);/* ukoliko je frekvencija
slobodna stavi cvor u spisak cvorova za dalju proveru */
                              red->dodaj(pt);
                              provereni[k]=pt;
                        }
                  }
                  j++;
            }
      }
      delete red;
      delete provereni;
      return -1;
```

U poslednjem delu funkcije brišu se strukture i algoritam vraća vrednost -1 kao rezultat neuspešnog konektovanja. Algoritam pak, nikada ne dolazi do ove faze jer je prethodnim (’*prohodnost*’) utvrđeno da je konekcija moguća.

Kao što ćemo videti, u funkciji ’*konektuj*’ algoritam se poziva i po više puta uzastopno jer kao vrednost vraća identitet čvora koji je sused čvoru *id2*, a po ideji da tako čvor po čvor dobijemo kompletnu putanju između početnog i krajnjeg čvora, sve dok se kao vrednost ne vrati id čvora koji je sused čvoru *id1*, nakon čega će se algoritam pozvati poslednji put, za taj zadati par *id1* i *id2*. Funkcija ’*konektuj*’ je funkcija koja ima zadatak da zauzima talasnu dužinu između zadatih čvorova i predstavlja poslednji deo algoritma za nalaženje puta. Pogledajmo kako izgleda ta funkcija.

```
void Mreza::konektuj(int c1, int c2, int f)
{
      Cvor *pom1=0, *pom2=0;
      Red *r=new Red;
      int *put=new int[brC];
      for(int i=0; i<brC; i++)
```

```
	{
		put[i]=-1;
	}
	int p=c2;
	int p2=-1;
	int count=0;
```

Sve dok ne dođemo do početnog čvora izvršava se naredna petlja. Za prvi prolaz *p* je čvor destinacije, a za ostale prolaze *p* je najbliži čvor za koji smo već ostvarili deo putanje od krajnjeg do početnog čvora.

```
	while(p2!=c1)
	{
		p2=najblizi(c1,p,f);
		if(p2!=-1)
```

U sledećem zaglavlju redom su date naredbe koje obavljaju sledeće:

1) uzimamo sam čvor koji smo odabrali kao jedan deo putanje između dva čvora
2) oko tog čvora na svim linkovima zauzimamo talasne dužine kako bi izbegli dupli prolaz
3) id čvora ubacujemo u ono što će nam biti putanja
4) menjamo parametar čvora destinacije za sledeći prolaz
5) inkrementiramo brojač koji služi za indeksiranje članova reda *put*

```
			{
				pom1=&pronadjiC(p);
				pom1->zauzmiF(f);
				put[count]=p2;
				p=p2;
				count++;
			}
		}
		count--;

		// zauzimanje frekfencije na linkovima izmedju cvorova od polaznog cvora
do destinacionog cvora
		pom1=&pronadjiC(c1);
		pom1->zauzmiF(f);
		int i=0;
		while(i<count)
		{
			pom1=&pronadjiC(put[i]);
			pom1->zauzmiF(f);
			i++;
		}
```

Sve dok je brojač čvorova na putanji između dva čvora koja povezujemo, veći od nule, tada ubacujemo u privremenu strukturu tipa red, id čvora koji smo prethodno obeležili kao čvor za putanju i smanjujemo brojač čvorova. Napomena: čvorovi su sačuvani u suprotnom redosledu!

```
		while(count>0)
		{
			r->dodaj(put[count-1]);
			count--;
		}
		DodajVezu(r,c1,c2,f);
}
```

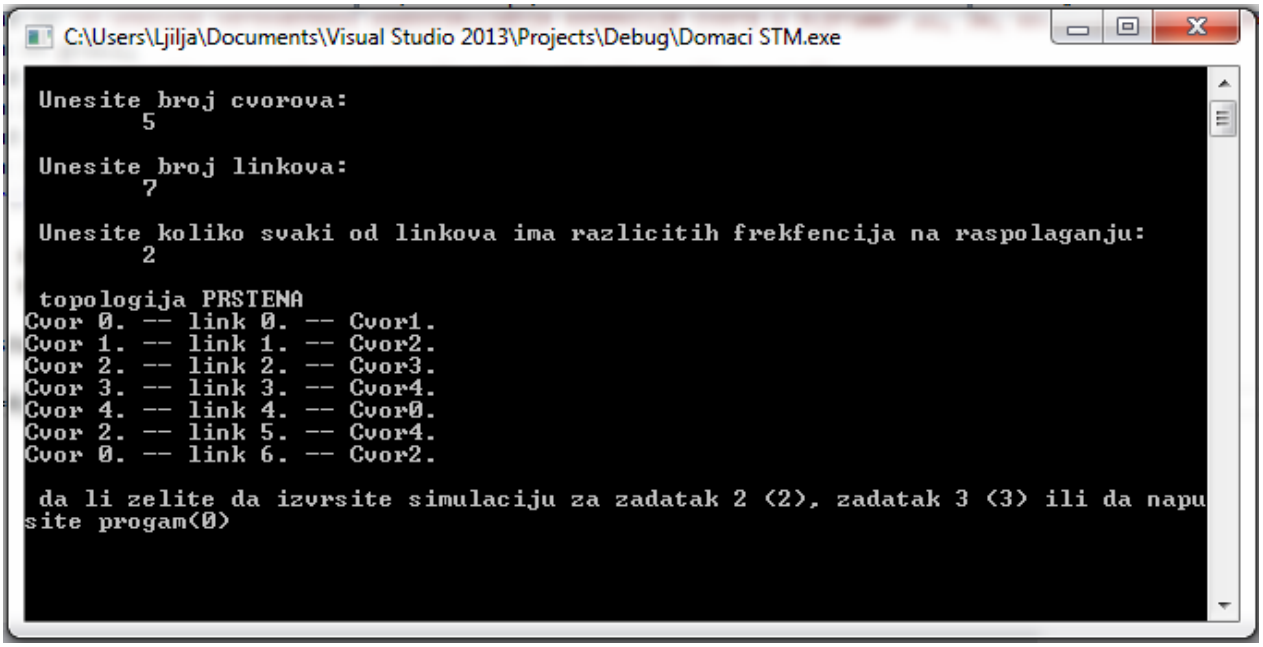
Na kraju se taj red, *id* početnog čvora, *id* krajnjeg čvora i talasna dužina *f* smeštaju u listu strukture koju smo nazvali ’veza’ [5].

4. SIMULACIONI REZULTATI

U ovom poglavlju koje se odnosi na simulacione rezultate i komentare istih najpre ćemo dati kratak pregled definisanja topologije mreže, uslova izbora tih topologija, kao i primere za neke vrednosti parametara. Kao razvojno okruženje za sprovođenje simulacije, za potrebe ovog rada korišćen je softver Microsoft Visual Studio 2013. Odmah po unosu broja linkova i boja čvorova, algoritam bira između dve topologije, prstenaste i zvezdaste i odlučuje se za jednu a na osnovu uslova:

```
if (brL > brC * (log(brC)/log(3))) // uslov biranja topologije
      {
      zvedasta();
      }
      else
      {
      prstenasta();
      }
```

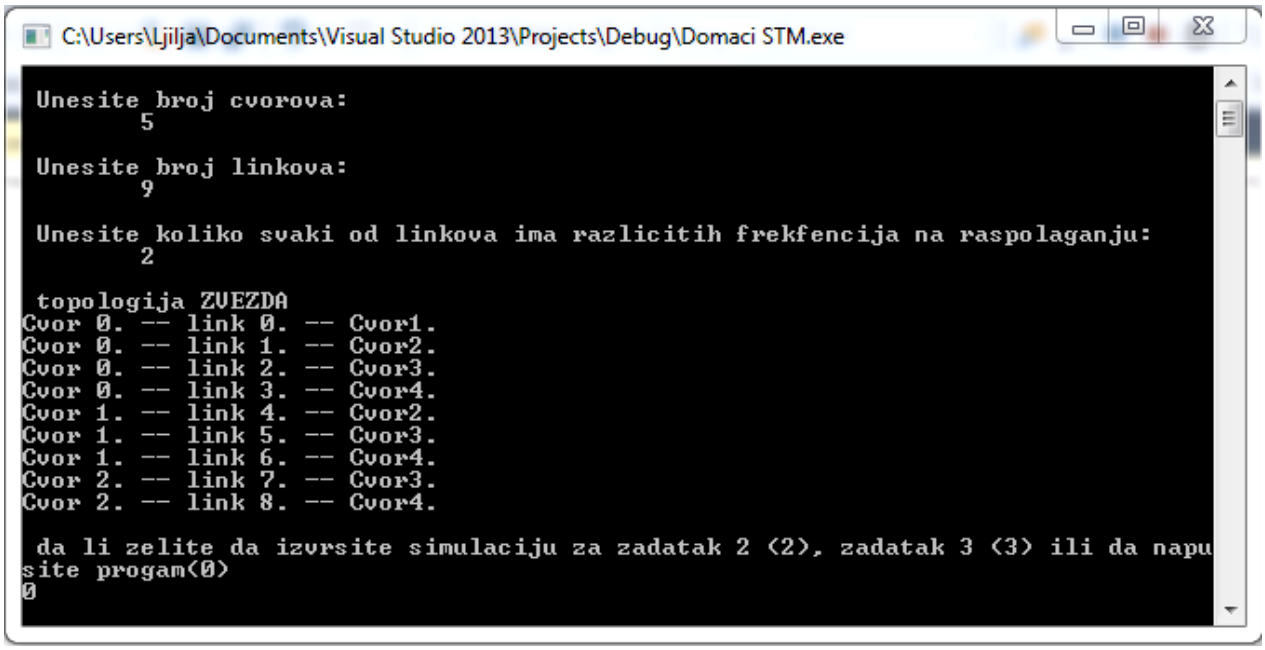
gde je *brL* broj linkova, a *brC* broj čvorova. Dakle, ako je ispunjen uslov u naveden u kodu iznad, mreža je zvezdaste topologije, a ako nije onda je prstenaste topologije.



Slika 4.1. Kreiranje topologije prstena na osnovu zadatih parametara

Pri kreiranju prstenaste topologije linkovima se prvo povezuju svi susedni čvorovi, a zatim se svi preostali linkovi na slučajan način raspoređuju između nepovezanih čvorova. Kod kreiranja zvezdaste topologije postupak je sledeći: prvi čvor se povezuje sa svi čvorovima, zatim drugi čvor sa svim čvorovima i sve tako dok ne poveze sve zadate linkove. Minimalan broj linkova određen je uslovom da broj linkova mora biti veći ili jednak broju čvorova. Ukoliko korisnik ne unese dovoljan broj linkova program će obavestiti korisnika da je uneo neispravnu vrednost, i automatski

će popuniti broj linkova do minimalnog potrebnog broja. Slično, maksimalni boj linkova mora biti jednak sumi: broj čvorova – i, gde je i brojač koji ide od 1 do broja čvorova.



Slika 4.2. Kreiranje topologije zvezde na osnovu zadatih parametara

Sada sledi niz simulacija koje za cilj imaju da pokažu uticaj određenih parametara na ishod simulacije. Najpre ćemo definisati najvažnije parametre. Prvu grupu čine parametri koje definiše korisnik i koji predstavljaju ulazne parametre simulacije:

- N – broj čvorova
- L – broj linkova
- K – broj talasnih dužina
- P1 – verovatnoća uspostave veze
- P2 – verovatnoća raskida veze (vrednosti parametara P1 i P2 se prilikom iniciranja simulacije unose u procentima)
- T – broj vremenskih slotova za koje se izvršava simulacija

Drugu grupu parametara čine izlazni parametri simulacije kao i parametri izvedeni iz izlaznih parametara simulacije, a koji služe za procenu kvaliteta različitih modela:

- BrK – ukupan broj pokušaja konektovanja
- UsK – broj uspešnih konektovanja
- NeK – broj neuspešnih konektovnja
- A – verovatnoća propusnosti (broj uspešnih konektovanja/broj ukupnih pokušaja)
- B – verovatnoća blokade (broj neuspešnih konektovanja/broj ukupnih pokušaja).
- F – faktor uspešnosti, odnos usešna/neuspešna konektovanja

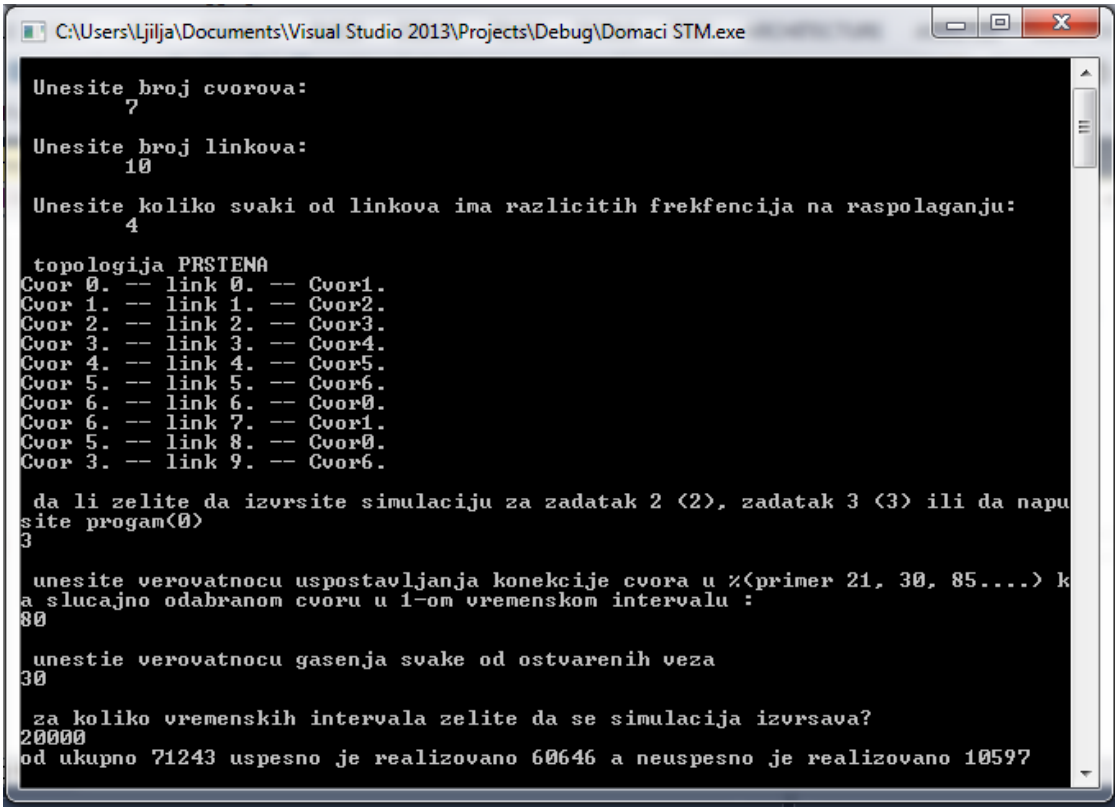
U narednoj tabeli su dati rezultati simulacije za jednu topologiju. Najpre ćemo komentarisati izlazne i izvedene parametre u zavisnosti od promene ostalih parametara (onih koji ne definišu topologiju), a zatim na osnovu izvedenih rezultata dati ocenu performansi mreže.

Slučaj 1. Parametri definisani za ovaj slučaj su: **N=8, L=10, K=4, P1=80%, P2=30%, T=10000**

Parametri topologije: N=8, L=10 ⟶ Topologija prstena									
T=10000	P1=80%, P2=30%			K=4, P2=30%			K=4, P1=80%		
	K=2	K=4	K=8	P1=30	P1=50	P1=70	P2=20	P2=50	P2=80
BrK	52703	45740	40836	20467	31182	41276	40500	52111	64024
UsK	19088	31138	40309	17109	24540	29276	25213	40529	219
NeK	33615	14602	527	3358	6642	12000	15287	11582	63805
A	**0.362**	**0.68**	**0.987**	**0.836**	**0.787**	**0.71**	**0.622**	**0.777**	**0.0034**
B	0.638	0.32	0.013	0.164	0.213	0.29	0.378	0.223	0.9966
F	0.568	2.13	76.5	5.095	3.7	2.72	1.65	3.5	0.0034

Tabela 4.1. Uticaj parametara u topologiji sa malim brojem čvorova i malim brojem linkova

Komentar: Na osnovu rezultata datih u gornjoj tabeli možemo zaključiti nekoliko stvari. U maloj mreži topologije prstena koja ima dva dodatna linka, vidimo da ne postoji puno mogućnosti za uspostavljanje veza, što se direktno odražava na propusnost mreže na negativan način. Primećujemo da povećanje broja talasnih dužina u sistemu vodi drastičnom povećanju propusnosti što je očekivani rezultat, a taj efekat je utoliko upečatljiviji obzirom na datu topologiju mreže u ovom slučaju. Kada je reč o uticaju parametra verovatnoće uspostave veze primećujemo da što je verovatnoća uspostave veća to je veća zauzetost veza, odnosno veća je mogućnost blokade što je direktno vezano sa manjom propusnošću sistema. U ovom primeru nasuprot očekivanjima da će porast verovatnoće raskida veza dovesti do veće propusnosti, dogodilo se suprotno: nakon pada za P2=50%, za P2=80% porasla je verovatnoća blokade. Objašnjenje za ovaj paradoks potencijalno leži u činjenici da visoke vrednosti verovatnoća P1 i P2 izazivaju prevelik broj kratkih veza pa često dolazi do talasa uspostave velikog broja veza u istom momentu što dovodi i do povećanogbroja neuspeha tj. veće verovatnoće blokade. Naravno, bilo bi potrebno dodatno ispitivanje ovog slučaja, ali to nije rađeno u ovoj tezi pošto to nije bio primarni cilj istraživanja. Konačno na osnovu svega zaključenog ovakva mreža bi generalno imala loše performanse u komercijalnom širokopojasnom sistemu, ali bi uz ulaganja (koja bi omogućila veliki broj talasnih dužina u WDM sistemu) dala dobre rezultate u malim okruženjima kao što su školski centri, mala i srednja preduzeća, gde se očekuju veliki protoci na malim dometima. Na slici 4.3 su dati primeri ispisa u konzoli za pojedina testiranja slučaja 1.



Slika 4.3. Sprovođenje dve simulacije u kojima su jedino vrednosti parametra P2 različite

Slučaj 2. Parametri definisani za ovaj slučaj su: **N=8, L=18, K=4, P1=80%, P2=30%, T=10000**

Parametri topologije: N=8, L=18 ⟶ Topologija zvezde									
T=10000	P1=80%, P2=30%			K=4, P2=30%			K=4, P1=80%		
	K=2	K=4	K=8	P1=30	P1=50	P1=70	P2=20	P2=50	P2=80
BrK	47367	41741	40693	19854	30066	38194	34653	50582	61826
UsK	28171	38732	40691	19559	28885	35803	31572	48206	18766
NeK	19196	3009	2	295	1181	2391	3081	2376	43060
A	**0.595**	**0.928**	**≈1**	**0.985**	**0.96**	**0.937**	**0.91**	**0.953**	**0.3**
B	0.405	0.072	≈0	0.015	0.04	0.063	0.09	0.047	0.7
F	1.47	12.87	20347	66.3	24.44	≈15	10.1	20.3	0.42

Tabela 4.2. Uticaj parametara u topologiji sa malim brojem čvorova i velikim brojem linkova

Slučaj 3. Parametri definisani za ovaj slučaj su: **N=25, L=40, K=4, P1=80%, P2=30%, T=20000**

Parametri topologije: N=25, L=40 ⟶ Topologija zvezde									
T=20000	P1=80%, P2=30%			K=4, P2=30%			K=4, P1=80%		
	K=2	K=4	K=8	P1=30	P1=50	P1=70	P2=20	P2=50	P2=80
BrK	382295	367685	350292	143268	234873	323891	353457	380954	390270
UsK	106106	190289	302081	114480	153224	179826	170766	226928	280678
NeK	276189	177396	48211	28788	81649	144065	182691	154026	109592
A	**0.2775**	**0.517**	**0.862**	**0.8**	**0.652**	**0.555**	**0.483**	**0.6**	**0.72**
B	0.7225	0.483	0.138	0.2	0.348	0.445	0.517	0.4	0.28
F	0.384	1.07	6.26	4	1.87	1.25	0.934	1.5	2.57

Tabela 4.3. Uticaj parametara u topologiji sa velikim brojem čvorova i velikim brojem linkova

Što se tiče Slučaja 2 uočavamo bitnu razliku kada je reč o propusnosti u odnosu na Slučaj 1. Porast propusnosti direktna je posledica drugačije konfiguracije mreže, obzirom da su i u Slučaju 1 i u Slučaju 2 korišćeni isti ulazni parametri. Ključni faktor u ovoj promeni igra daleko veći broj linkova kojih ima gotovo duplo više nego što je to bio slučaj u Slučaju 1. Značajno veća propusnost favorizuje mreže koje imaju znatno veći broj linkova nego što imaju čvorova. U Slučaju 2 očigledno je da povećanje broja talasnih dužina dovodi do povećanja propusnosti i smanjenja verovatnoće blokade u znatno bržoj meri nego što je to bio slučaj sa Slučajem 1. Primetno je blago smanjenje propusnosti za povećavanje procenta uspešne uspostave veze, a to je takođe posledica konfiguracije mreže. Efekat je ublažen upravo zbog velikog broja linkova. Kada je reč o promeni parametra verovatnoće raskida veze koji je ključan za rasterećenje mreže uočava se sličnost sa Slučajem 2. Pri velikim vrednostima verovatnoće uspostave veze, a za male i srednje vrednosti verovatnoće raskida veze, ukoliko ona raste rasterećuje se mreža i propusnost raste. Ali u slučaju velike verovatnoće uspostave i velike verovatnoće raskida istovremeno, postoji drastičan pad propusnosti iz istih razloga navedenih za Slučaj 1. U prilog tome govori podatak iz simulacije u Slučaju 2 da ako je P1=30% i P2=30% propusnost iznosi A=98.5%, a ako su vrednosti parametara P1=80% i P2=80% propusnost iznosi svega A=30%. Obim opterećenja mreže postaje jasan kada se primeti da je u slučaju P1=P2=30% broj veza bio BrK=19854, a u slučaju P1=P2=80% broj veza je bio 61826 što je gotovo tri puta više pa je srazmerno tome propusnost opala približno tri puta.

Slučaj 3 je dao bolji uvid u složenost ovakvih sistema. Direktan zaključak koji se nameće je da se veće mreže (sa većim brojem čvorova i linkova) bolje nose sa zagušenjem nego što to čine manje mreže. To se zaključuje iz činjenice da je u Slučaju 3 za verovatnoće P1=P2=80% propusnost povećana, a ne smanjena nakon povećanja verovatnoće raskida veza, kao što je to bio slučaj u prethodna dva slučaja. Razlika u obradi je velika: 61826 veza u Slučaju 2 i 390270 veza u Slučaju 3. Takođe pokazuje se da mreže srednjeg obima (nazovimo tako mrežu iz Slučaja 2) zahtevaju znatno više talasnih dužina da bi održale propusnost na visokom nivou, nego što zahtevaju mreže malog obima. Takođe, kao i u slučaju malih mreža, u slučaju srednjih mreža zakonitost da se propusnost povećava kako porastom verovatnoće raskidanja tako i smanjenjem verovatnoće uspostavljanja, postaje dosta izraženija.

Iz Slučaja 2 i Slučaja 3 izvodi se jednostavan zaključak da su mrežne topologije sa većim brojem linkova daleko bolje rešenje po pitanju propusnosti i smanjenju verovatnoće blokade nego što je to slučaj sa mrežnim topologijama sa manjim brojem linkova. Mana je takođe više nego očigledna, veliki broj linkova je skupa investicija i za instalaciju i za održavanje treba izdvojiti značajna sredstva, međutim, kao što smo već pre napomenuli tehnike DWDM pristupa se svakodnevno unapređuju i poboljšavaju.

5. ZAKLJUČAK

Algoritam predstavljen u ovom radu daje zadovoljavajuće rezultate pri simulacijama mreža koje imaju manji broj čvorova i linkova. U toku izrade ovog rada autor je sproveo veliki broj simulacija čiji rezultati ovde ne mogu biti ukratko predstavljeni. Stoga će u ovom poglavlju biti najviše reči o načinu ponašanja algoritma i oceni njegovih performansi.

Što se tiče malih mreža u kojima se broj čvorova i linkova kreće oko 20 algoritam ne pokazuje znake slabosti. U poglavlju 4 je u Slučajevima 1 i 2 simulacija sprovođena u 10000 vremenskih slotova. Vreme rada simulacije nije prelazilo 5 sekundi. Takođe, program, kada je u pitanju mala mreža od oko desetak čvorova i od 10-20 linkova, sprovodi simulaciju i za par stotina hiljada slotova na nivou minute. Primetna je razlika u rezultatima za simulacije koje se izvode u par stotina hiljada slotova od onih datih u tabelama 4.1. i 4.2. ali ona nije drastična. Kod malih mreža varira od 10E-2 (ređe) do 10E-3 (češće) stepena što je zanemarljiva razlika.

Kada su u pitanju mreže srednjeg obima (do 50 čvorova i 80 linkova) trajanja simulacije ukoliko broj talasnih dužina nije dvocifren i u slučaju 20000 vremenskih slotova, kreću se u rasponu od 1-3 minuta. Za ove mreže može se koristiti i uzorak od 50000 vremenskih slotova, i ako broj talasnih dužina nije dvocifren trajanja simulacije će u najvećem broju slučajeva biti oko 5 minuta što je prihvatljiva vrednost. Program se u razumnom vremenu od desetak minuta nosi i sa velikom mrežom od 200 čvorova i 300 linkova, ali ukoliko je broj talasnih dužina 4, a broj vremenskih slotova 10000. Tako da program odnosno algoritam ima svojevrsnu manu – nije dovoljno brz za testiranja velikih mreža (par stotina čvorova i linkova).

Rešenje ovog problema leži u dodatnoj optimizaciji realizovanog algoritma u cilju bolje iskorišćenosti resursa koje koristi, a što bi dovelo do razumnog vremena trajanja simulacije i za neke složenije mrežne konfiguracije sa ozbiljnijim brojem mrežnih elemenata (čvorova, linkova i talasnih dužina). Na kraju napomenimo da osnovna upotrebna vrednost ovog algoritma leži u njegovoj sposobnosti da studentima na malom modelu pruži bolji uvid u način funkcionisanja i strukturu ponašanja optičkih mreža.

LITERATURA

[1] Dr Zoran Čiča – Materijali sa predavanja Širokopojasne telekomunikacione mreže

[2] Wikipedia – Optical Networks

[3] http://searchnetworking.techtarget.com/definition/wavelength-division-multiplexing

[4] Milo V. Tomašević – Algoritmi i strukture podataka

[5] http://www.learncpp.com/

A. PROGRAMSKI KOD

A.1. Mreza.cpp

```
#include "Mreza.h"
#include <cmath>
#include <cstdlib>

Mreza::Mreza(int bc, int bl, int ops)
{
	listaL = 0;
	listaC=poslC=0;
	brC=bc;
	brL=bl;
	opseg=ops;
	uspK=neK=0;                                  // brojac uspesnih i neuspesnih
konekcija
	veze=0;
	Cvor *c=new Cvor(0, this);                   // pravi prvi cvor
	ElemC *e=new ElemC(c);
	listaC=poslC=e;                              // ubacuje inicijalni cvor u
listu
	for(int i=1;i<brC;i++)                       // popunjava listu cvorova
	{
		c=new Cvor(i, this);
		e=new ElemC(c);
		poslC->sled=e;
		poslC=poslC->sled;
	}

	if (brL > brC * (log(brC)/log(3)))               // uslov biranja
topologije
	{
		zvedasta();
	}
	else
	{
		prstenasta();
	}
}
Cvor& Mreza::pronadjiC(int id)
{
	ElemC *tekC=listaC;
	while(tekC->p->identitet()!=id)              //postavi tek na cvor
identifikacionog broja, id
		tekC=tekC->sled;
	Cvor *a;
	a=tekC->p;
	return *a;
}
void Mreza::zvedasta()
{
```

```
	cout<<"\n topologija ZVEZDA \n";
	Cvor *c1=0, *c2=0;
	int povez=0;
	// brojac do sada ostvarnih povezivanja
	for(int i=0;(i<brC-1)&&(povez<brL);i++)
	{
		c1 = &pronadjiC(i);
		for(int j=i+1;(j<brC)&&(povez<brL);j++)
		{
			c2 = &pronadjiC(j);
			c1->dodaj(c2, opseg);
			povez++;
		}
	}
}
void Mreza::prstenasta()
{
	cout<<"\n topologija PRSTENA \n";
	Cvor *c1=0, *c2=0;
	int povez=0;                                                    //
brojac do sada ostvarnih povezivanja
	while(povez<brC)                                                //
Inicijalno povezuje sve cvorove u krug (susedi su mu id cvora +/-1)
	{
		c1 = &pronadjiC(povez % brC);
		c2 = &pronadjiC((povez+1)% brC);
		c1->dodaj(c2, opseg);
		povez++;
	}
	// Ukoliko ima preostalih linkova slobodnih dodljuje dva slucajna cvora
koja do sad nisu povezani
	while(povez<brL)
	{
		int a= rand() % brC;
		int b= rand() % brC;
		if (a!=b)
		{
			c1=&pronadjiC(a);
			c2=&pronadjiC(b);
			if(!(c1->povezan(c2)))        // ukoliko ne postoji veza
izmedju cvorva c1 i c2
			{
				c1->dodaj(c2, opseg);
				povez++;
			}
		}
	}
}
void Mreza::DodajLink(Link* l)
{
	ElemL* novi = new ElemL(l);
	novi->sled = listaL;
	listaL = novi;
}
void Mreza::zauzetost(int proc)
{
	ElemC *tekC=listaC;
	Cvor *pom=0;
	for(int i=0;i<brC;i++)
```

```
	{
		if(rand() % 100 < proc)
		{
			int f=rand()% opseg;
			tekC->p->zauzmiF(f);
		}
		tekC=tekC->sled;
	}
}
int Mreza::prohodnost(int id1,int id2)
{
	Cvor *c1=0, *c2=0, *pom=0;;
	c1 = &pronadjiC(id1);
	c2 = &pronadjiC(id2);

	Red *red=new Red();

	int *provereni=new int[brC];
	/* ispitivanje za svaku frekvenciju */
	for(int i=0; i<opseg; i++)
	{
		red->dodaj(id1);
		provereni[0]=id1;			//interni spisak cvorova koji su
vec obradjeni
		for (int loc=1; loc<brC; provereni[loc++] =- 1);
		/* sve dok ima potencijalnih cvorova koji su dostupni za frekvenciju
koja se ispituje */
		while(!red->prazan())
		{
			int pt=red->dohvati();
			pom=&pronadjiC(pt);
			/* da li se cvor koji se trenutno obradjuje granici sa cvorom
koji je destinacija */
			if(!(pom->povezan(c2)))
			{
				int j=0;
				/* sve dok cvor koji se trenutno obradjuje ima suseda */
				while(pom->susedni(j)!=-1)
				{
					int k=0;
					bool ind=true;
					/* sused cvora nije do sad bio posecen */
					while((provereni[k] != -1) && (ind))
					{
						if(pom->susedni(j)==provereni[k])ind =false;
						k++;
					}
					if(ind)
					{
						/* ukoliko je frekvencija slobodna stavi
cvor u spisak cvorova za dalju proveru */
						if(pom->prvoeraVeze(j,i))
						{
							pt=pom->susedni(j);
							red->dodaj(pt);
							provereni[k]=pt;
						}
					}
					j++;
```

```
                        }
                  }
                  //ako je cvor destinacije sused trenutno obrajdivanom cvoru
proveravamo dostupnost na trazenoj frekvenciji
                  else
                  {
                        int j=pom->povezanNa(c2);
                        if(pom->prvoeraVeze(j,i))
                        {
                              delete red;
                              delete provereni;
                              return i; // vracamo frekvenciju ukoliko je
slobodna
                        }
                  }
            }
            // praznjenje reda priprema za prolazak ispitivanja za sledecu
frekvenciju
            while(!red->prazan())
            {
                  int a=red->dohvati();
            }
      }
      // nije moguce ostvariti vezu izmedju ova dva cvora ni na jednoj
frekvenciji
      delete red;
      delete provereni;
      return -1;
}
int Mreza::najblizi(int id1,int id2,int f)
{
      Cvor *c1=0, *c2=0, *pom=0;;
      c1 = &pronadjiC(id1);
      c2 = &pronadjiC(id2);

      Red *red=new Red();

      int *provereni=new int[brC];  //interni spisak cvorova koji su vec
obradjeni

      red->dodaj(id1);
      for (int loc=0; loc<brC;loc++) provereni[loc]=-1;
      provereni[0]=id1;
      /* sve dok ima potencijalnih cvorova koji su dostupni za frekvenciju koja
se ispituje */
      while(!red->prazan())
      {
            int pt=red->dohvati();
            pom=&pronadjiC(pt);

            int j=0;
            /* sve dok cvor koji se trenutno obradjuje ima suseda */
            while(pom->susedni(j)!=-1)
            {
            /* da li se cvor koji se trenutno obradjuje granici sa cvorom koji
je destinacija */
                  if(pom->povezan(c2))
                  {
                        int u=pom->povezanNa(c2);
```

```
                        //ako je cvor destinacije sused trenutno obrajdivanom
cvoru proveravamo dostupnost na trazenoj frekvenciji
                        if(pom->prvoeraVeze(u,f))
                        {
                              delete red;
                              delete provereni;
                              return pom->identitet();// vracamo id cvora koji
ima direktnu vezu sa destinacijom
                        }
                  }
                  int k=0;
                  bool ind=true;
                  while((provereni[k] != -1) && (ind))
                  {
                        if(pom->susedni(j)==provereni[k])ind =false;
                        k++;
                  }
                  /* sused cvora nije do sad bio posecen */
                  if(ind)
                  {
                        if(pom->prvoeraVeze(j,f))
                        {
                              pt=pom->susedni(j);/* ukoliko je frekvencija
slobodna stavi cvor u spisak cvorova za dalju proveru */
                              red->dodaj(pt);
                              provereni[k]=pt;
                        }
                  }
                  j++;
            }
      }
      delete red;
      delete provereni;
      return -1;// ne postoji cvor koji ima direktnu vezu sa destinacijom i
slobodan je na frekvenciji f
}
void Mreza::zauzmi(int proc)
{
      ElemV *temp=0;
      for(int i=0;i<brC;i++)
      {
            temp=veze;
            int povez = rand() % 100;
            if(povez<proc)
            {
                  while(1)
                  {
                        povez=rand() % brC;
                        if(povez!=i)
                        {
                              break;
                        }
                  }
                  bool ind=true;
                  // proveravamo da li zeljeni par za povezivanje vec poseduje
ostvarenu vezu
                  while(temp && ind)
                  {
                        if(temp->c1==i || temp->c1==povez)
```

```
                        {
                              if(temp->c2==i || temp->c2==povez)
                              {
                                    ind=false;
                              }
                        }
                        temp=temp->sled;
                  }
                  if(ind)
                  {
                        // odmah proveravamo da li je put izmedju cvorova
slobodan na nekoj frekfenciji
                        int mogucaf=prohodnost(i, povez);
                        if(mogucaf!=-1)
                        {
                              uspK++;
                              konektuj(i,povez,mogucaf);
                        }
                        else
                        {
                              neK++;
                        }
                  }
            }
      }
}
void Mreza::DodajVezu(Red *r, int a, int b, int f)
{
      ElemV *novi = new ElemV(r);
      novi->postavi(a,b,f);
      novi->sled = veze;
      veze = novi;
}
void Mreza::konektuj(int c1, int c2, int f)
{
      Cvor *pom1=0, *pom2=0;
      Red *r=new Red;
      int *put=new int[brC];
      for(int i=0; i<brC; i++)
      {
            put[i]=-1;
      }
      int p=c2;
      int p2=-1;
      int count=0;
      //pravi putanju izmedju cvorova c1 i c2
      while(p2!=c1)
      {
            p2=najblizi(c1,p,f);
            if(p2!=-1)
            {
                  pom1=&pronadjiC(p);
                  pom1->zauzmiF(f);
                  put[count]=p2;
                  p=p2;
                  count++;
            }
      }
      count--;
```

```
	// zauzimanje frekfencije na linkovima izmedju cvorova od polaznog cvora
do destinacionog cvora
	pom1=&pronadjiC(c1);
	pom1->zauzmiF(f);
	int i=0;
	while(i<count)
	{
		pom1=&pronadjiC(put[i]);
		pom1->zauzmiF(f);
		i++;
	}
	while(count>0)
	{
		r->dodaj(put[count-1]);
		count--;
	}
	DodajVezu(r,c1,c2,f);
}
void Mreza::PrekidVeza(int proc)
{
	ElemV *temp=veze;
	while(temp)
	{
		int a=rand() % 100;
		if(a<=proc)
		{
			if(temp==veze)
			{
				if(veze->sled)
				{
					veze=veze->sled;
					oslobodi(temp);
					temp=veze;
				}
				else
				{
					temp=veze=0;
				}
			}
			else
			{

				ElemV *pom=veze;
				while(pom->sled!=temp)
				{
					pom=pom->sled;
				}
				pom->sled=temp->sled;
				oslobodi(temp);
				temp=pom->sled;
			}
		}
		else
		{
			temp=temp->sled;
		}
	}
}
```

```
void Mreza::oslobodi(ElemV *v)
{
	Cvor *pom=0;
	pom = &pronadjiC(v->c1);
	pom->oslobodiF(v->f);
	while(!v->r->prazan())
	{
		pom = &pronadjiC(v->r->dohvati());
		pom->oslobodiF(v->f);
	}
	pom = &pronadjiC(v->c2);
	pom->oslobodiF(v->f);
	delete v;
}
void Mreza::statistika()
{
	cout<<"od ukupno "<< uspK+neK <<" uspesno je realizovano "<<uspK<<" a
neuspesno je realizovano "<<neK<<" \n";
}
Mreza::~Mreza(void)
{
	while(listaC)			// obrisi sve cvorove u mrezi
	{
		ElemC *tekC=listaC;
		listaC=listaC->sled;
		delete tekC;
	}

	while(listaL)			// obrisi sve linkove u mrezi
	{
		ElemL* temp = listaL;
		listaL=listaL->sled;
		delete temp;
	}
	while(veze)			// brise sve uspostavljene veze
	{
		ElemV *temp = veze;
		veze=veze->sled;
		delete temp;
	}
}
int main()
{
	int brC, brL, opsK;
	while(1)                                                        //
Proces unosenja broja cvorova sa zadatim uslovima
	{
		cout <<"\n Unesite broj cvorova:\n  ";
		cin >> brC;
		if(brC>=3)break;else
		{
			cout << "\n\n Broj cvorova mora da bude veci od 3 \n";
		}
	}
	// Proces unosenja broja linkova
	int Lmax=0;                                                     //
	Maksimalni broj linkova u zavisnosti od broja cvorova
	for(int i=1;i<=brC;i++)Lmax+=brC-i;                    //    Racunanje
maksimalnog broja linkova
```

```
	cout <<"\n Unesite broj linkova:\n  ";
	cin >> brL;
	//    Minimalni broj linkova za bilo koji broj zadatih cvorova jeste broj cvorova
	if(brC>brL)
	{
		cout << "Broj linkova je povecan na minimalnih :"<<brC<<"\n";
		brL=brC;
	}
	if(brL>Lmax)
	{
		cout << "\n Broj linkova je smanjen na maksimalnih  :"<<Lmax<<"\n ";
		brL=Lmax;
	}
	// Unosenje broja opsega na linkovima
	cout << "\n Unesite koliko svaki od linkova ima razlicitih frekfencija na raspolaganju: \n  ";
	cin >> opsK;
	if(opsK<=0)
	{
		cout << "\n Broj kanala je automatski povecan na 1\n";
		opsK=1;
	}
	Mreza m(brC,brL,opsK);
	int opcija=1;
	while(opcija)
	{
		cout <<"\n da li zelite da izvrsite simulaciju za zadatak 2 (2), zadatak 3 (3) ili da napusite progam(0) \n";
		cin >> opcija;
		if(opcija==2)
		{
			int proc1=0;
			int ukupno=0,uspesno=0;
			cout <<"\n unesite verovatnocu zauzetosti cvora u %(primer 21, 30, 85....) u 1-om vremenskom intervalu :\n";
			cin >> proc1;
			m.zauzetost(proc1);
			for(int i=0;i<brC-1;i++)
			{
				for(int j=i+1;j<brC;j++)
			{
			ukupno++;
			if(m.prohodnost(i,j)!=-1)uspesno++;
			}
		}
		cout <<"\n od maksimalnih :"<<ukupno<<"zahteva za komunikaciju medju cvorovima, moguce je ostvariti :"<<uspesno<<" konekcija medju razlicitim cvorovima za slucaj kada je verovatnoca zauzetosti svakog od covorova:"<<proc1<<"%. \n";
		int c1=0, c2=0;
		cout <<"\n unesite scvor 1 \n";
		cin >>c1;
		cout <<"\n unesite scvor 2 \n";
		cin >>c2;
		if(c1==c2)cout<<"\n cvorovi 1 i 2 moraju biti razliciti \n";
		else
		if(m.prohodnost(c1,c2)!=-1)
		{
```

```
                cout<<"\n uspesno konektovanje izmedju"<<c1<<" i "<<c2<<" \n";
            }
            else
            {
                cout <<"\n nije moguce uspostaviti vezu izmedju ova dva
cvora\n";
            }
            }
            if(opcija==3)
            {
                int proc1=0;
                int proc2=0;
                int vreme;
                int ukupno=0,uspesno=0;
                cout <<"\n unesite verovatnocu uspostavljanja konekcije cvora
u %(primer 21, 30, 85....) ka slucajno odabranom cvoru u 1-om vremenskom
intervalu :\n";
                cin >> proc1;
                cout <<"\n unestie verovatnocu gasenja svake od ostvarenih
veza\n";
                cin >> proc2;
                cout <<"\n za koliko vremenskih intervala zelite da se
simulacija izvrsava? \n";
                cin >> vreme;
                for(int i=0;i<vreme;i++)
                {
                    m.zauzmi(proc1);
                    m.PrekidVeza(proc2);
                }
                m.statistika();
            }
            opcija=0;
      }
}
```

A.2. Red.cpp

```
#include "Red.h"


Red::Red(void)   //formira prazan red
{
	posl=prvi=0;
}
int Red::prazan()
{
	if(prvi!=0)                                          // ako pokazivac na
prvi element nije 0 onda red ima sadrzaj
	{
		return 0;
	}
	else
	{
		return 1;
	}
}
void Red::dodaj(int c)
{
	if(prazan())                                         // ispitivanje da li
je ovo prvi element koji se dodaje u red
	{
		prvi=posl= new Elem(c);                    // na prvo mesto se dodje
novi element, ujedno to je i poslednje mesto tako da odmah inicijalizujemo
vrednost
	}else
	{
		posl->sled=new Elem(c);                    // na poslednje mesto se
stavlja novi element, posto je red vec bio formiran
		posl=posl->sled;
	}
}
int Red::dohvati()
{
	int c;
	if(!prazan())
	{
		Elem *pom=prvi;
		c=pom->c;               // izvlacimo vrednost sa prvog elementa
		prvi=prvi->sled;  // pomeramo pokazivac koji treba da pokazuje na
prvi element
		delete pom;
		pom=0;
		return c;   // vracamo vrednost koju je do sada prvi element imao
	}
}
Red::~Red(void)                                          // brisanje reda
{
	while(prvi)
	{
		Elem* pom=prvi->sled;
		delete prvi;
		prvi=pom;
	}
}
```

A.3. Link.cpp

```
#include "Link.h"

int Link::uID=0;


Link::Link(int a, int b, int op)                                    // id cvora sa
jedne strane linka, id cvora sa druge strane linka i broj frekfencija
{
	c1=a;
	c2=b;
	opseg=op;
	id=uID++;
	k = new int[op];                                                // pravi
niz koji je velicine broja frekvencija
	cout<<"Cvor "<<a<<". -- link "<<id<<". -- Cvor"<<b<<".\n";        // ispis
topologije
	for(int i=0; i < op ; i++)                                      // sve
frekvencije postavlja na 0- slobodnu vrednost
	{
		k[i] = 0;
	}
}
int Link::sused(const int c)
{
	if (c == c1)
	{
		return c2;
	}
	return c1;
}
bool Link::frekvencija(int f)
{
	if (k[f] == 0)
	{
		return true;
	}
	else
	{
		return false;
	}
}
void Link::zuzmiF(int f)
{
	k[f]=1;
}
void Link::oslobodiF(int f)
{
	k[f]=0;
}
Link::~Link(void)
{
	delete [] k;
}
```

A.4. Cvor.cpp

```
#include "Cvor.h"
#include "Mreza.h"


Cvor::Cvor(int ident, Mreza* m)
{
	prvi=tek=posl=0;                          // prilikom samog kreiranja
cvora on sam nema veze ni sa jednim drugim cvorom
	id=ident;
	mreza = m;
}
int Cvor::identitet()
{
	return id;
}
int Cvor::susedni(int i)
{
	int j=0;
	tek=prvi;
	while(j<i && tek)
	{
		j++;
		tek=tek->sled;
	}
	if(tek)
	{
		return tek->l->sused(id);           // poziva id suseda na tekucem
linku (linku rednog broja i)
	}
	return -1;
}
bool Cvor::prvoeraVeze(int i, int f)      // vraca true ako je konekcija na i-
toj vezi i frekvenciji f moguca
{
	int a=susedni(i);
	if(a!=-1)
	{
		if(tek->l->frekvencija(f)) return true;
	}
	return false;
}
void Cvor::dodaj(Link& link)
{
	if(prvi!=0)
	{
		posl->sled=new ElemL(link);                                    //
dodaje link koji mu je prosledjen u svoju listu na poslednje mesto
		posl=posl->sled;
	}else
	{
		prvi=new ElemL(link);                          // dodaje link koji
mu je proslednjen na prvo mesto u listi
		posl=prvi;
	}
}
void Cvor::dodaj(Cvor *c, int k)
{
```

```
	if(prvi)
	{
		Link* nov = new Link(id,c->identitet(), k);		// pravi se novi link prosledivsi mu kao argumente svoj id i id cvora koji se povezuje preko tog linka
		mreza->DodajLink(nov);
		c->dodaj(*nov);		// prosledjuje drugom cvoru kreirani link kako bi i drugi cvor dodao ovaj u listu veza
		posl->sled=new ElemL(*nov);		// postavlja novi link na poslednje mesto u listi linkova
		posl=posl->sled;		// pomera pokazivac na poslednji link
	}
	else
	{
		Link* nov = new Link(id, c->identitet(), k);
		c->dodaj(*nov);
		mreza->DodajLink(nov);
		prvi=new ElemL(*nov);		// razlika je samo u toliko sto prvi pokazuje na novi link
		posl=prvi;		// stavlja pokazivace prvi i poslednji na isti elemt (jedini u listi)
	}
}
void Cvor::skidaj(Link& link)
{
	ElemL* prethodni = 0;
	ElemL* temp = prvi;

	while (temp != 0)
	{
		if (temp->l == &link)
		{
			if (prethodni == 0)
			{
				// Ako je prvi link taj koji treba da se ukine, samo se prebaci pokazivac na sledeci
				prvi = temp->sled;
			}
			else
			{
				prethodni->sled = temp->sled;
			}

			return;
		}
		else
		{
			prethodni = temp;
			temp = temp->sled;
		}
	}
}
bool Cvor::povezan(Cvor *c)
{
	tek=prvi;
	while(tek)
	{
		if(tek->l->sused(id)==c->identitet())
```

```
            {
                        return true;
      // ukoliko je pronadjen cvor sa trazenim identitetom vracamo true vrednost
            }
            tek=tek->sled;
      }
      return false;
      // trazenog cvora nema u listi tako da vracamo false vrednsot
}
void Cvor::zauzmiF(int f)
{
      int i=0;
      while(susedni(i)!=-1)
      {
            tek->l->zuzmiF(f);
            i++;
      }
}
void Cvor::oslobodiF(int f)
{
      int i=0;
      while(susedni(i)!=-1)
      {
            tek->l->oslobodiF(f);
            i++;
      }
}
int Cvor::povezanNa(Cvor *c)
{
      int i=0;
      if(!(this->povezan(c)))                                          //
ispituje da li su cvor i trazeni cvor povezani
      {
            return -1;                                                       //
ukoliko nisu vrati -1
      }
      else                                                             //
ukoliko su povezani
      {
            tek=prvi;
            while(this->susedni(i)!=c->identitet())              //za svaki
link cvora koji trazi vezu proveravamo da li smo nasli trazenu vezu
            {
                  i++;
            }
            return i;                                                        //
vracamo redni broj linka koji povezuje ova dva cvora za cvor koji je pozvao
proveru
      }
}
Cvor::~Cvor(void)
{
}
```

A.5. Mreza.h

```
#pragma once
#include "Cvor.h"
#include "Red.h"
#include <iostream>

using namespace std;

class Mreza
{
	struct ElemC	//pomocna sturktura za cuvanje spiska cvorova
{
	Cvor *p;
	ElemC *sled;

	ElemC(Cvor *c)
	{
		p = c;
		sled = 0;
	}
	~ElemC()
	{
		delete p;
		p = 0;
	}
};

struct ElemL	//pomocna sturktura za cuvanje spiska linkova
{
	Link *p;
	ElemL *sled;

	ElemL(Link *l)
	{
		p = l;
		sled = 0;
	}
	~ElemL()
	{
		delete p;
		p = 0;
	}
};
struct ElemV	//pomocna sturktura za cuvanje spiska ostvarenih veza
{
	Red *r;
	int c1, c2, f;
	ElemV *sled;

	ElemV(Red *st)
	{
		c1=c2=f=0;
		r = st;
		sled = 0;
	}
	void postavi(int poc,int kraj, int kanal)
	{
		c1=poc;
```

```
            c2=kraj;
            f=kanal;
      }
      ~ElemV()
      {
            delete r;
            r = 0;
      }
};
      ElemC *listaC, *poslC;  // prvi i poslednji cvor u spisku cvorova

      ElemL *listaL;          // pokazivac na prvu u spisku linkova

      ElemV *veze;            // pokazivac na prvu u spusku ostvarenih veza
      int brC, brL, opseg, uspK, neK;           // broj cvorova, broj linkova,
broj razlicitih frekfencija na linkovima, uspeno ostvarene konekcije i
// neuspesno ostvarene konekcije za koje je poslat zahtev

public:
      Mreza(int bc, int bl, int ops);           // inicijalizuje mrezu tj. pravi
cvorove i linkove u skladu sa zadatim brojem i odabira porgodnu topologiju

      Cvor& pronadjiC(int id);                  // Pronalazi cvor sa zadatim
identitetom i vraca njegovu adresu (referencu)

      void zvedasta();  // Povezuje linkove i cvorove u zvezdastu topologiju

      void prstenasta(); // Povezuje linkove i cvorove u prstenastu topologiju

      void DodajLink(Link* l); // dodaje vec postojaci link u spisak linkova

      int prohodnost(int id1, int id2);   // vraca frekfenciju na kojoj je
moguce uspostaviti vezu izmedju cvora 1 i cvora 2 ili -1 ukoliko
// ne moze da se ostvari veza ni na jednoj frekfenciji izmedju ova dva cvora
// koristi se samo za testiranje funkcije prohodnost u drugom zadatku
      void zauzetost(int proc); // za sve cvorove u topologiji proverava
vrednoscu p1 zauzima slucajno odabranu frekvenciju
      void zauzmi(int proc);  // obilazi sve cvorove u mrezi i za svaki
verovatnocom p1 pokusava da uspostavi konekciju sa slucajno odabranim cvorem
// ukoliko ova dva cvora vec nisu u vezi
      void DodajVezu(Red *r, int a, int b, int f);// dodaje uspesno ostvarenu
vezu u spisak do sad ostvarenih veza
      void PrekidVeza(int proc);                // prolazi kroz sve ostvarene
veze i u skladu sa verovatnocom P2 odlucuje da li veza treba da bude prekinuta
      void Mreza::oslobodi(ElemV *v);           // omogucuje uklanjanje veze
      void konektuj(int c1, int c2, int f);     // obezbedjuje zauzetos
frekfencije f na putu izmedju cvorova c1 i c2
//    void putanja(int c1, int c2, int f,int *put);   // pravi spisak cvorova
koji omogucavaju vezu izmedju cvorova c1 i c2 na frekfenciji f
      void statistika();                                // ispisuje broj uspesno
realizovanih konekcija i broj neuspesno realizovanih konekcija
      int najblizi(int id1, int id2, int f);  // trazi najblizi cvor cvoru id2 i
na putanji id1 id2 i frekvenciji f, i vraca id cvora koji je povezan sa id2 i
// ima slobodnu frekvenciju f

      ~Mreza(void);
};
```

A.6. Red.h

```
#pragma once
#pragma once
using namespace std;

class Red
{
	struct Elem{ //struktura liste za red
	int c;
	Elem *sled;
	Elem(int c1){
		c=c1;
		sled=0;
	}
	~Elem()
	{
		sled=0;
	}
	};
	Elem *prvi, *posl;
public:
	Red(void);
	void dodaj(int c);                                  // dodaje se novi element
u listu strogo na kraju liste

	int dohvati(void);                                  // uzima se element sa
pocetka liste, vraca se njegova vrednost, i uklanja se iz liste

	int prazan(void);                             // ispitivanje da li je red
prazan (true ako je prazan i false ako nije)

	~Red(void);
};
```

A.7. Link.h

```
#pragma once
#pragma once
#include <iostream>

using namespace std;

class Link
{
	private:
	int c1, c2, opseg, *k;
	int static uID;
	int id;
public:
	Link(int a, int b, int op);

	int sused(int c);				// vraca id cvora koji je sused
cvoru c na pozvanom linku

	bool frekvencija(int f);			// vraca false ako je
frekfencija f zauzeta ili ako ne postoji, true ako je slobodna

	void zuzmiF(int f);					// zauzima frekfenciju f

	void oslobodiF(int f);			// oslobadja frekfenciju f

	~Link(void);
};
```

A.8. Cvor.h

```
#pragma once
#include "Link.h"
class Mreza;                                                    // samo
deklaracija klase mreza
class Cvor
{
      struct ElemL                                              // lista
linkova povezana na dati cvor
      {
            Link *l;
            ElemL *sled;
            ElemL(Link& link)
            {
                  l=&link;
                  sled = 0;
            }
      };

      int id;                                                         //
identifikacija cvora
      ElemL *prvi, *tek, *posl;
      Mreza* mreza;
public:

      Cvor(int ident, Mreza *m);

      int identitet();                    // vraca broj svog identieta

      int susedni(int i);                       // vraca id cvora na i-toj vezi,
-1 ako nema toliko veza

      bool prvoeraVeze(int i, int f);     // vraca true ako je konekcija na i-
toj vezi i frekvenciji f moguca

      void dodaj(Cvor *c, int k);         // ostvaruje vezu sa drugim cvorom,
prosledjenim u argumentu

      void dodaj (Link& link);            // ostvaruje vezu sa drugim cvorem,
preko prosledjenog linka u argumentu

      void skidaj(Link& link);            // brise vezu sa drugim cvorem, preko
prosledjenog linka u argumentu

      bool povezan(Cvor *c);              // vraca true vrednost ako cvor,
prosledjen kao argument ima ostvarenu vezu sa cvorem koji ispituje vezanost

      int povezanNa(Cvor *c);             // vraca interni broj linka preko
kojeg je cvor povezan sa cvorem c, -1 ako nisu povezani

      void zauzmiF(int f);    // zadatu frekvenciju na svim linkovima vezanim za
cvor

      void oslobodiF(int f);  // oslobadja frekvenciju na svim linkovima vezanim
za cvor
      ~Cvor();
};
```