SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3265

ANALIZA PERFORMANSI ALGORITAMA USMJERAVANJA

Adrijan Jakšić

Zagreb, lipanj 2013.

Sadržaj

Uvod

U ovom projektu ispitat ću performanse komunikacijskih mreža u mrežnom sloju komunikacijske mreže. Same performanse mreže ovise o mnogo detalja, kao što su širina pojasa, propusnosti i kašnjenje. Svaki sloj komunikacijske mreže pridonosi i utječe na neki način na sveukupnu performansu mreže. Ovaj završni rad će se koncentrirati na mrežni sloj gdje važnu ulogu ima algoritam po kojem se podaci usmjeravaju.

Najveći značaj dat ću algoritmima usmjeravanja paketa te usporedbi istih i davanje zaključka koji algoritam je najpovoljniji za što bolju performansu mreže. Testirat ću jedan statički i jedan dinamički algoritam usmjeravanja te ih potom analizirati. Kako bi se moglo pravilno ispitivati i analizirati performanse mreže prvo ću vas upoznati s osnovnim pojmovima komunikacijske mreže, kasnije sa samim mrežnim slojem u arhitekturi mreže i na kraju s algoritmima usmjeravanja.

1. Komunikacijska mreža

Komunikacijsku mrežu čine međusobno povezani komunikacijski sustavi na koje se spaja korisnička oprema, bilo računalna ili komunikacijska i druga oprema potrebna za pružanje informacijskih i komunikacijskih usluga te potporu aplikacija korisnicima kao što su poslužiteljska računala i drugi sustavi. Postoje dva osnovna načina komuniciranja u komunikacijskoj mreži. Prvi je definiran komunikacijskim kanalom. On predstavlja put kroz mrežu od izvorišta do odredišta koji se dodjeljuje na zahtjev (povezivanjem) ili trajno (najmom) i zauzet je za cijelo vrijeme komunikacije. Drugi način komuniciranja je paketom. Kod takvog načina komuniciranja informacija se dijeli na blokove kojima se dodaje zaglavlje s adresom na temelju koje ih se usmjerava od izvora do odredišta. Prednost ovog načina je ta što mreža nije cijelo vrijeme zauzeta već se zauzima kada izvor šalje pakete. Nedostatak je taj što je teško održavati vremenske uvjete komuniciranja (dolazi do kašnjenja i promjena kašnjenja). To jako utječe na performanse komunikacijske mreže. Također slanje paketa se može vršiti na dva načina: datagramski – svaki se paket šalje zasebno i moguće da različiti paketi prolaze drugim putevima kroz mrežu i virtualnim kanalom – prvo se odredi put kroz mrežu te se svi paketi njime šalju.

1.1. Arhitektura mreže

Arhitektura mreže je slojevita. Svaki sloj ima svoje funkcije u općem djelovanju mreže te utječu na performanse mreže općenito. Svakom sloju se specificiraju sučelja sa susjednim slojem kako bi viši sloj mogao koristiti usluge nižeg sloja. Svaki sloj ima svoje određene protokole po kojima se komunikacijska mreža ponaša. Svaka mreža se sastoji od aplikacijskog (aplikacijski, prezentacijski i sjednički), transportnog, mrežnog i fizičkog sloja (sloj podatkovne poveznice i sam fizički sloj). Aplikacijski sloj predstavljaju različiti protokoli koji omogućuju korisniku korištenje i pristupanje na mrežu. Transportni sloj omogućava transparentan prijenos podataka s kraja na kraj komunikacijske mreže uz prijenos bez pogreške (semantička transparentnost – TCP) ili prijenos uz najmanje kašnjenje (vremenska transparentnost – UDP). Mrežni sloj omogućuje komunikaciju između dva čvora u komunikacijskoj mreži, izravno ili preko drugih čvorova. Na fizičkoj razini se omogućuje slanje podataka između dva međusobno povezana čvora.

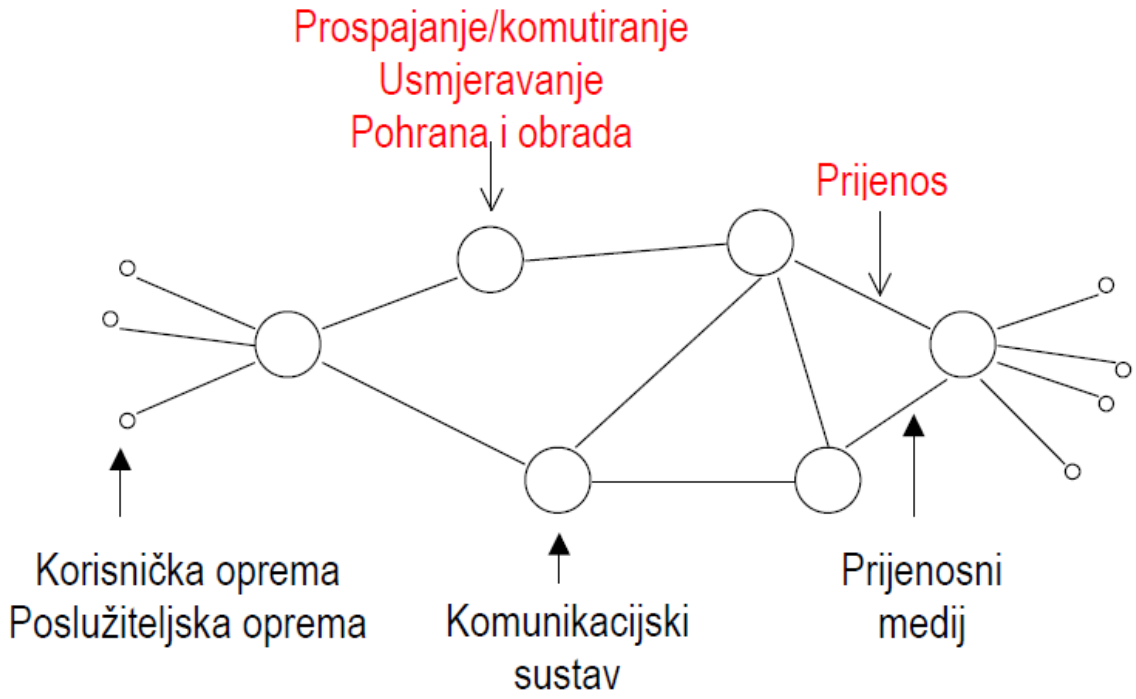
1.2. Performanse komunikacijske mreže

Performanse komunikacijske mreže su sposobnost mreže ili nekog njenog dijela da ostvari funkcije potrebne za komunikaciju dvaju korisnika ili korisnika i poslužiteljskog sustava. Njih čine više parametara kao što su: širina pojasa, propusnost, kašnjenje i slično. Svaki sloj utječe i pridonosi na performanse komunikacijske mreže u cjelosti. Širina pojasa je određena fizičkim osobinama kao npr. maksimalni broj bita koji se može prenijeti u sekundi. Propusnost je broj korisnih prenesenih bitova u sekundi. Ona je određena načinom prijenosa (podatkovna poveznica). Kašnjenje je vrijeme potrebno da bit s izvora stigne na odredište. Time se brine mrežni i transportni sloj. Sada se možemo upoznati s mrežnim slojem komunikacijske mreže.

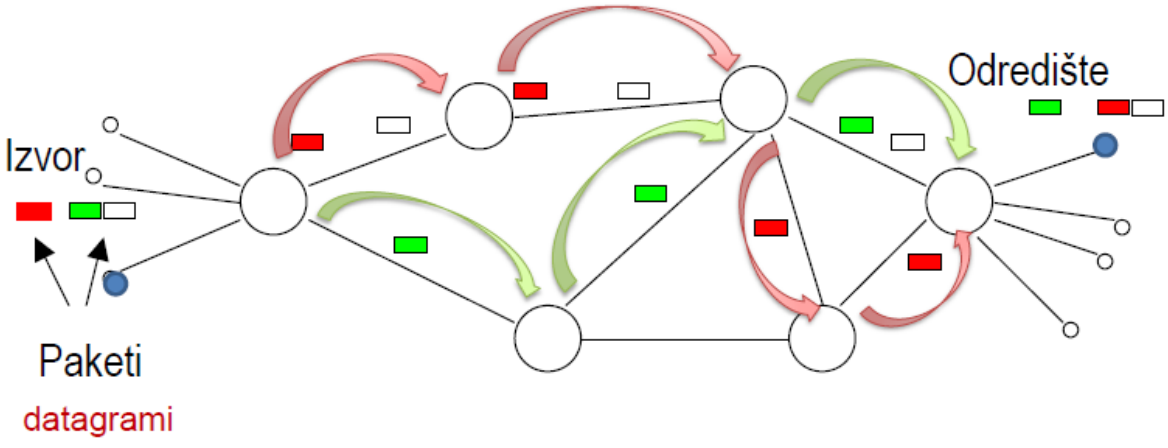
1.3. Mrežni sloj i načini komuniciranja

Osnovna zadaća mrežnog sloja je da omogućuje komunikaciju između dva krajnja, korisnička čvora u mreži, bilo izravno ili preko međučvorova. Zadaće su joj adresiranje (IP adrese), usmjeravanje jedinica podataka, kontrola zagušenja, pogrešaka i toka (očituje se u performansama mreže) te kao glavna zadaća međusobno povezivanje mreža i podmreža. Osnovna jedinica podataka je paket. Put paketa od izvorišta do odredišta paketa se upravlja adresiranjem te usmjeravanjem istih.

Sam prijenos od izvora do odredišta (usluga koju mrežni sloj pruža transportnom) može se riješiti spojnom ili nespojnom uslugom (internetski mrežni sloj), odnosno virtualnim kanalom ili datagramski. Spojna veza (usmjeravanje virtualnim kanalom) se sastoji od faze uspostave veze, transfera podataka i raskida veze. Nespojnu uslugu (datagrami) čini samo faza prijenosa podataka. S obzirom na ove dvije usluge mrežni sloj može biti i pouzdan i nepouzdan prilikom prijenosa podataka.



Slika 1: Komutacijska mreža na razini mrežnog sloja

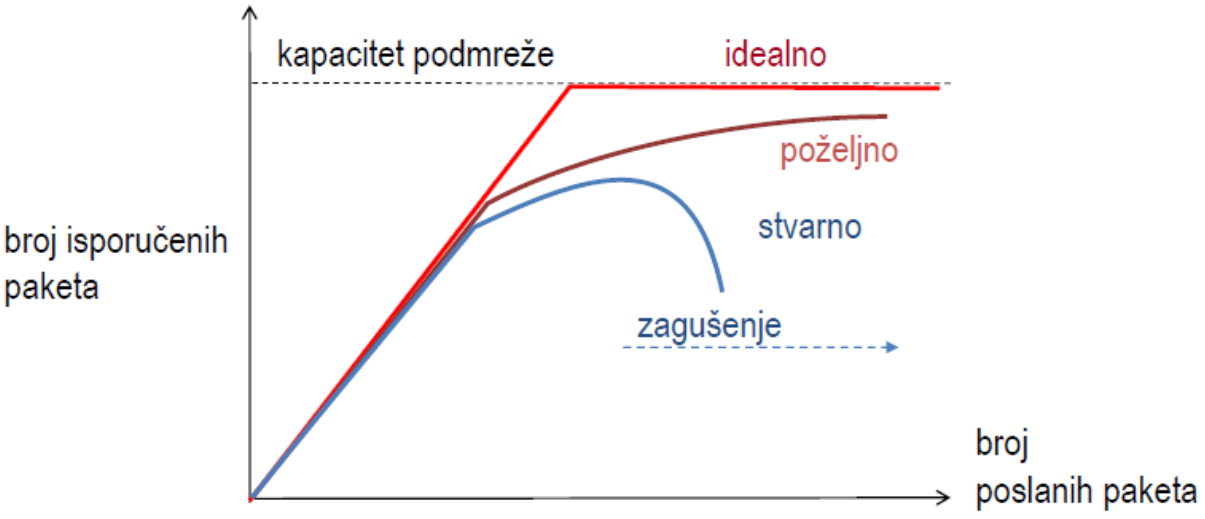


Slika 2: Datagramski način komuniciranja

Svaki čvor u komunikacijskoj mreži mora imati svoju jedinstvenu adresu kako bi se moglo pomoću nje odrediti gdje se on točno u mreži nalazi. Značenje jedinstvenosti mrežne adrese odnosi se na lokalnu konotaciju, ne i globalnu. Za razliku od MAC adrese na sloju podatkovne poveznice, mrežna adresa je logička, a ne fizička. IP adrese se dodjeljuju dinamički računalima u mreži pomoću aplikacijskog protokola DHCP. On daje računalu adresu u najam što znači da pri svakom spajanju na mrežu računala dobivaju nove IP adrese na određeno vrijeme. IP adresa je normirani 32-bitni identifikator koji globalno i jednoznačno određuje mrežno sučelje čvora u internetskom modelu mreže.

1.4. Upravljanje zagušenjem i tokom podataka

Po pitanju ove teme postoje rezlike u mrežama temeljenim na komutaciji paketa i onima temeljenim na komutaciji kanalom. Kod komutacije kanala isti se rezervira prije slanja te u pravilu ne dolazi do zagušenja (congestion). Jedini problem može nastati ako izvor prebrzo šalje no što ga odredišni čvor može primiti. Taj problem riješava kontrola toka (flow control). Mreža zasnovana na komutaciji paketa prihvaća stalno nove pakete i može doći do zagušenja mreže. Ono predstavlja degradaciju performansi mreže radi prevelikog broja paketa u prometu mreže koju ona ne može podnijeti.



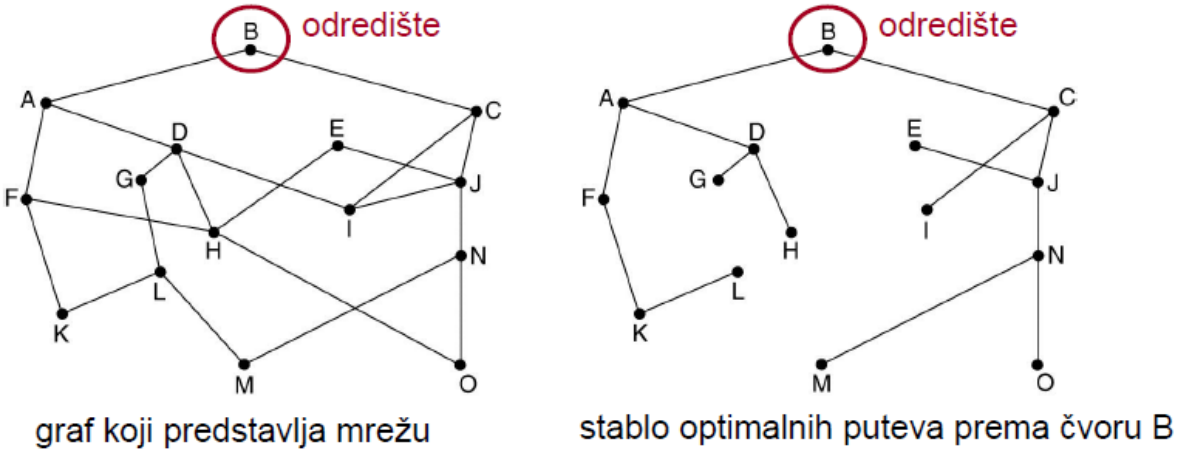
Slika 3: Propusnost u slučaju zagušenja

Za razliku od kontrole toka zagušenje je globalni problem. Kako opterećenje raste dolazi do gubitaka, pada propusnosti i povećanog kašnjenja. Razni uzroci su odgovorini za zagušenje mreže pa stoga postoje i razna rješenja. Npr. moguće je da razni izvori šalju paket na određeni usmjeritelj koji mora prosljediti sve pakete na isto sučelje. U takvom scenariju događa se čekanje, a zbog nedovoljne memorije moguć je i gubitak paketa. Ako povećamo memoriju tada će doći do većeg kašnjenja zbog duže faze obrade paketa, a većim kašnjenjem moguća je pojava isteka vremenske kontrole i retransmisije (ponovnog slanja paketa).

Povećanje brzine prijenosa također nije uvijek najbolje rješenje. Postoje samo dva načina iz teorije upravljanja zagušenjem, a to su rješenje otvorenom petljom i rješenje zatvorenom petljom. Kod rješenja s otvorenom petljom mrežni čvorovi moraju ograničiti prihvat novih zahtjeva, odbaciti pakete po potrebi, oblikovati promet i raspoređivati pakete unutar mreže. Rješenje zatvorenom petljom kaže da čvorovi moraju motriti sustav te otkriti zagušenje i mjesto gdje se nalazi, proslijediti tu informaciju drugim čvorovima te prilagoditi način rada sustava radi ispravljanja problema. Prilagodbe se često svode na slanje zahtjeva pošiljatelju da smanji brzinu slanja paketa.

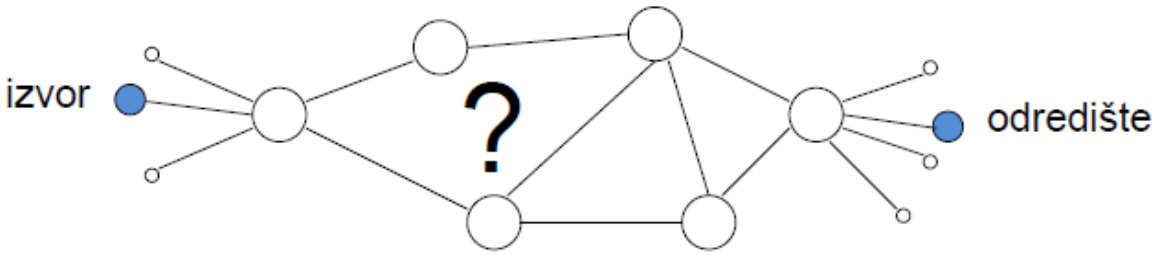
1.5. Komutacija paketa i usmjeravanje

Pitanje je kako se paketi šalju kroz komunikacijsku mrežu? Paket sadrži upravljačku informaciju kojom se oni komutiraju i usmjeravaju od izvora do cilja. U njoj su pohranjene IP adrese izvora i odredišta (kod datagrama) ili oznaka puta (kod virtualnog kanala). Komutacija paketa se sastoji od usmjeravanja – određivanje puta, tj. čvorova kojima se prolazi do odredišta i od prosljeđivanja – donošenje odluke u čvoru na koje odlazno sučelje poslati paket. Način na koji se paketi šalju i put kojim prolaze do odredišta ne mora biti jedinstven, ali cilj je pronaći najbolji mogući put kako bi performanse mreže bile bolje. Algoritmi kojima se određuje kojim putem putuju paketi zovu se algoritmi usmjeravanja. Taj problem usmjeravanja se zgodno da prikazati grafom u kojem čvorovi predstavljaju komutatore, a grane grafa poveznice među njima, pri tome je cilj pronaći najkraći put između zadanog para čvorova.



Slika 4: Problem pronalaska puta (usmjerenja)

Vrijednosti koje bi nosile grane mogle bi se odnositi na više različitih parametara, kao npr. stvarne udaljenost, propusnosti, kašnjenje, cijene komunikacije i slično. Rješenje problema usmjeravanja postiže se načelom optimalnosti. U primjeni, ako odredimo neki čvor kao odredište možemo napraviti stablo optimalnih puteva do odredišta. To stablo ne mora bit jedinstveno rješenje, već može biti nekoliko rješenja.



Slika 5: Pronalazak optimalnog puta

2. Algoritmi usmjeravanja

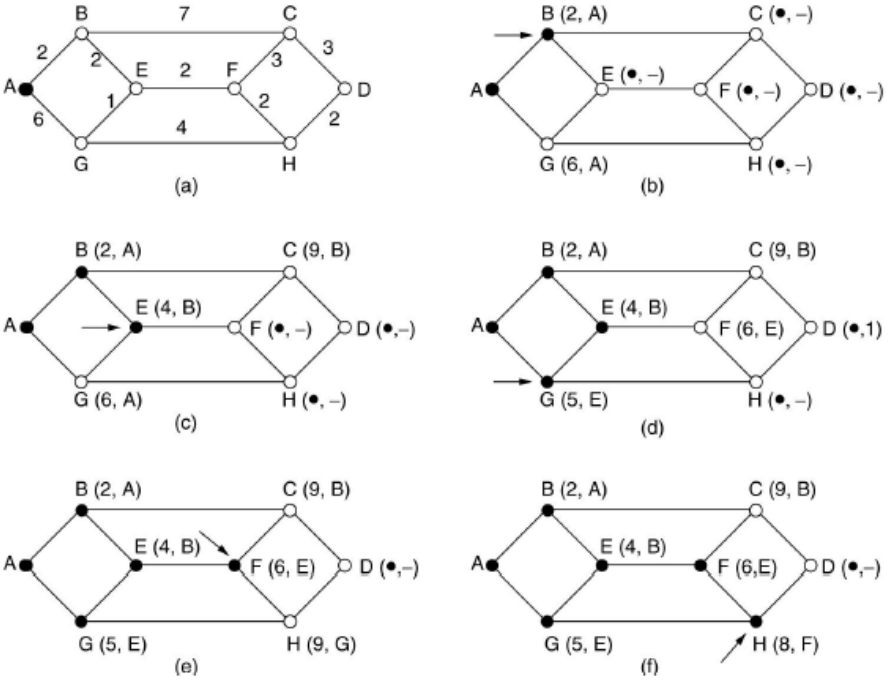
Postoje dvije osnovne kategorije algoritama usmjeravanja, a to su neadaptivni ili statički i adaptivni ili dinamički algoritmi. Odlike neadaptivnih su te da oni računaju putove na temelju jednog ili više zadanih kriterija za zadanu topologiju mreže. Ovaj tip algoritma je ograničen jer uzima samo u obzir početno stanje mreže i ne mijenja se. Ne uzima trenutno stanje mreže (opterećenje, prekide, kvarove) u mreži kao ni promjene nastale kroz vrijeme kao npr. dodavanje novih čvorova ili grana. Adaptivni algoritmi, naprotiv neadaptivnim, mjere trenutno stanje mreže i odluku o usmjeravanju donose na temelju istih. Kod ovakvih algoritam se postavlja pitanje što točno mjeriti (broj skokova, udaljenost, cijenu, opterećenje...), da li pratiti samo susjedne čvorove ili sve, i kada reagirati (periodički, pri promjeni topologije, opterećenja...).

U nastavku ću prokomentirati 4 glavna algoritma usmjerenja. Od statičkih algoritam usmjeravanja najkraćim putem i preplavljivanje, dok od dinamičkih usmjeravanje prema vektoru udaljenosti i usmjeravanju prema stanju poveznice. U primjeni se koriste još mnogo drugih algoritama kao npr. hijerarhijsko usmjeravanje, opće razašiljanje (broadcast), višeodredišno razašiljanje (multicast), usmjeravanje kada nema infrastrukture (ad-hoc mreže) itd.

2.1. Statički algoritmi usmjeravanja

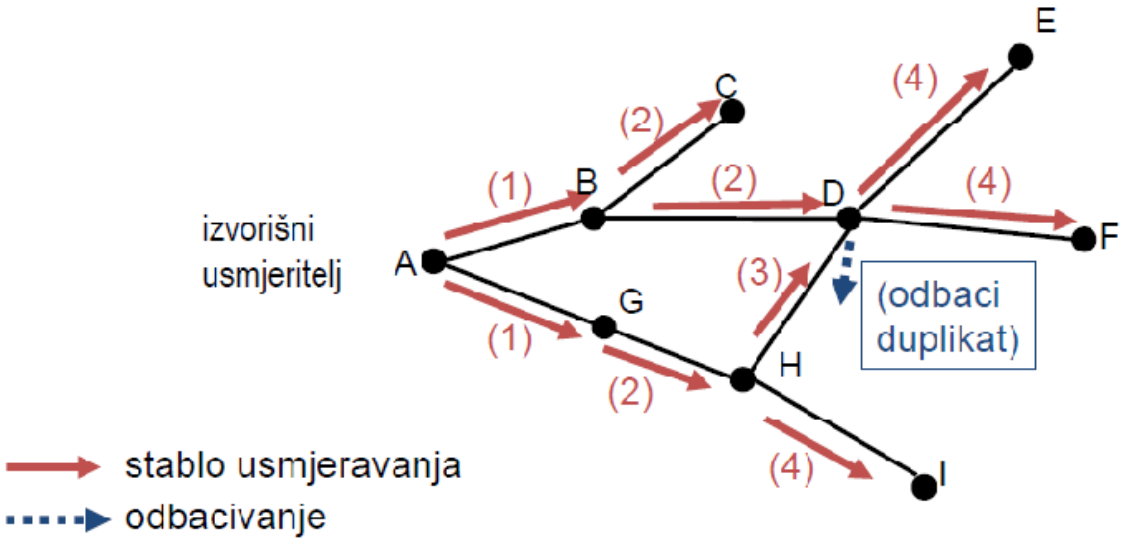
Usmjeravanje najkraćim putem je statički algoritam usmjeravanja koji traži put od izvora do odredišta tako da suma težina grana na tom putu bude minimalna. Kod primjene u mreži zanimljivo nam je za određeno odredište ili izvor pronaći odgovarajuće stablo najkraćeg puta. Postoje razni algoritmi kojima se računa najkraći put no najpoznatiji je Dijsktrin algoritam. Izmislio ga je nizozemski računalni znanstvenik Edsger Wybe Dijkstra, objavio 1959. godine. Taj algoritam rješava problem najkraćeg puta od nekog izvora do svih odredišta rezultirajući sa stablom najkraćeg puta. Algoritam funkcionira ovako: prvo formiramo dva skupa, u svakom će biti parovi koji se sastoje od udaljenosti od izvora do čvora i od oznake čvora. Prvo u prvi skup stavimo ishodište i udaljenost nula a drugom skupu sve

ostale čvore i onima koji si direktno spojeni s ishodištem pridjelimo tu udaljenost, a ostalima #. Nakon toga gledamo koji direktni čvor iz drugog skupa ima najmanju udaljenost i njega stavljamo u prvi skup sada za taj čvor gledamo direktne čvorove i osvježimo podatke (ako je neki čvor dobio novu vrijednost, ali veću nego prije, ne mijenjamo staru vrijednost). I tako ponavljamo sve do kraja, tj. do odredišnog čvora. Evo jednog skraćenog primjera dijkstre na slici 6.



Slika 6: Dijkstrin algoritam

Preplavljivanje (flooding) je isto statički algoritam i temelji se na jednom vrlo jednostavnom načelu. Ideja je nastala na temelju širenja vode, tj. poplave. Radi tako da svaki čvor kada primi paket stvara kopije i šalje ih na sva svoja sučelja osim onog s kojeg je primio paket. Kada primi isti paket koji je već ranije primio na nekom drugom sučenju on ga odbacuje tako da se duplikati ne šalju, a algoritam uvijek daje najkraći put. U praksi se javljaju neki problemi. Moguća je pojava petlji u mreži te beskonačno uvišestručavanje paketa. Postoji više mehanizama za rješavanje ovog problema. U praksi se primjenjuju dva: ograničenje maksimalnog broja prosljeđivanja nekog paketa i bilježenje već pristiglih paketa na čvor te se duplikati odbacuju. Bilježenje paketa se ostvaruje tako da izvorišni čvor označava svoje pakete brojevima u nizu, a ostali čvorovi to pamte. Slika 6 će nam pokazati kako algoritam funkcionira.

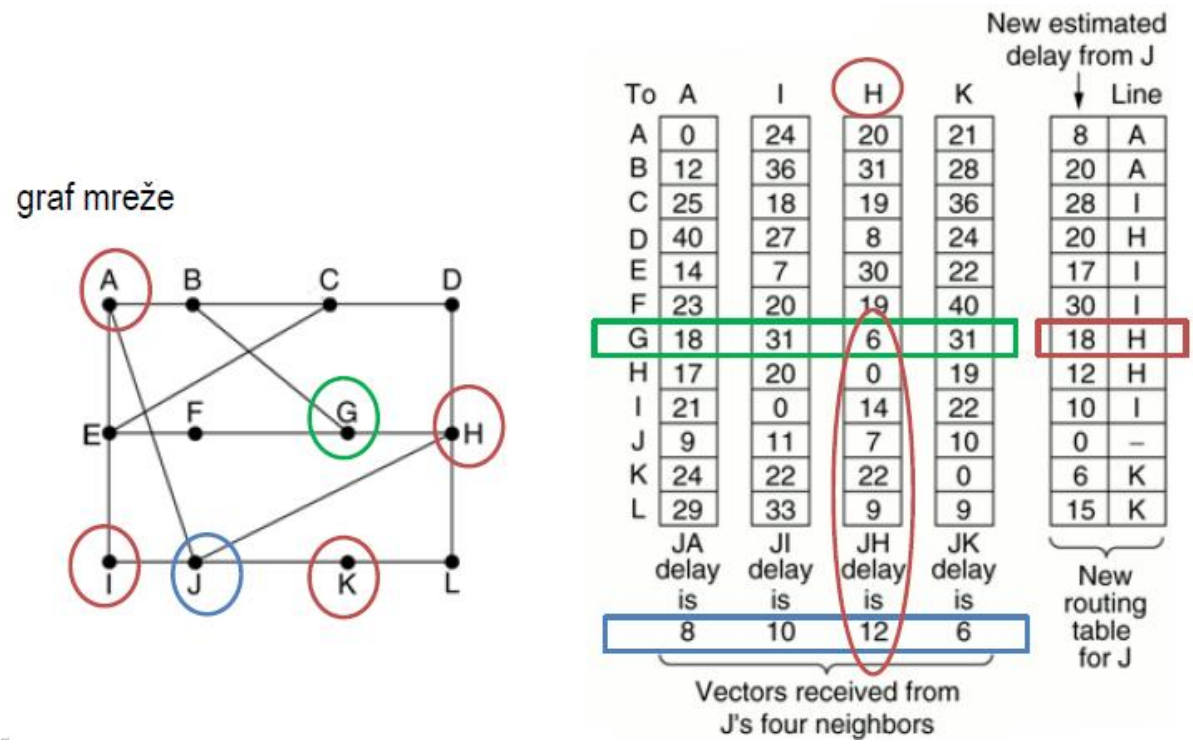


Slika 7: Preplavljivanje

Uočavamo iz ovog algoritma da se on može poboljšati tako da H-D poveznica se ne opterećuje zbog toga što će ionako čvor D odbacivati pakete. U praksi tu funkciju obavljaju protokoli usmjeravanja kao što su RIP (routing information protocol) i OSPF (open shortest path first).

2.2. Dinamički algoritmi usmjeravanja

Usmjeravanje prema vektoru udaljenosti (distance vector routing) dinamički je način usmjeravanja. Svaki usmjeritelj ima tablicu (vektor) u kojem se nalazi „najbolja poznata udaljenost“ za svako odredište prvi čvor prema njemu. Budući da je ovo dinamički algoritam tablica se osvježava na temelju razmjene podataka s drugim čvorovima u mreži. Jedan od prvih algoritama temeljen na ovoj ideji je algoritam Forda i Fulkersa, a on računa maksimalni tok u mreži tokova. Bellman i Ford su pak napravili algoritam koji računa stablo najkraćih puteva za težine koje mogu biti i negativne. Slika 7 prikazuje jedan algoritam usmjeravanja prema vektoru udaljenosti po kriteriju kašnjenja.



Slika 8: Usmjeravanje prema vektoru udaljenosti

Da bi algoritam mogao funkcionirati svaki čvor bi morao moći otkriti svoje susjede i saznati njihove mrežne adrese, izmjeriti kašnjenja do njih, osvežavati svoju tablicu usmjerenja s novim podacima, stvoriti paket koji sadržava vlastiti vektor udaljenosti te poslati paket susjednim čvorovima. Ovaj algoritam ima i dobrih i loših strana u pogledu performansi mreže.

Za ovu vrstu algoritama usmjerenja (dinamičku), zanimljivo je koliko je vremena potrebno da se promjena u mreži proširi do svih čvorova. U ovom algoritmu je očigledan nedostatak koji jako utječe na perfomanse mreže taj što sporo konvergira ka željenom stanju s time da bolje reagira na „dobre vijesti“ u mreži kao što je dodavanje čvora u mreži od „loših vjesti kao ispadanje čvora iz mreže. Algoritam na dodavanje čvora u mrežu reagira tako da neki susjed javi sada da ima kraći put do odredišta i onda se odmah svi paketi usmjeruju na taj čvor. Svaka sljedeća razmjena vektora prenosi dobru vijest dalje. Kod ispadanja čvora iz mreže prvi susjed to zna, ali ne i sljedeći, kod prve razmjene vektora zna 2. susjed, ali ne i 3. itd. Uvijek se u prometu mreže događaju neke „loše vijesti“ tako da ovaj algoritam ima velik nedostatak u performansama.

Usmjeravanje prema stanju poveznice (link state routing) temelji se na razmjeni podataka o topologiji i stvarno izmjerenih podataka o stanju poveznice konkretno kašnjenj među čvorovima i zatim primjeni Dijkstrinog algoritma za izračun najkraćeg puta prema svim ostalim čvorovima. Algoritam uzima u obzir stvarno stanje pa omogućuje bolju prilagodljivost na promjene, ali uz veću složenost algoritma. Po algoritmu, periodično se trebaju slati poruke susjedima te osvježavati o stanju poveznice te kada isto kada se dogodi neka promjena u topologiji mreže. Algoritam je složen jer se mora ispuniti zahtjev da se podaci puzdano distribuiraju pa to zahtjeva uvođenje numeriranje paketa, potvrde primitka, oznake starosti poruke i preplavljivanje uz neka proširenja. Svaki čvor u mreži radi na sljedeći način: otkriva susjede i njihove mrežne adrese, zatim mjeri kašnjenje prema njima, pa stvara i šalje svim čvorovima paket s podacima koje je doznao, a za to treba izračunati najkraći put do njih te osvježiti svoju tablicu usmjerenja.

Svaki od ovih algoritama je pokazao i svoje prednosti i nedostatke, ali usmjeravanje prema stanju poveznice se pokazao najbolji jer je najsloženiji i pazi na svaki detalj. U daljnjem dijelu rada prikazat ću i objasniti kako sam programski ostvario određene algoritme usmjeravanja. Kasnije ću prikazati dobivene rezultate te ih prokomentirati. Odlučio sam se za dijkstin algoritam kao predstavnika statičkih algoritama usmjeravanja te usmjeravanje prema vektoru udaljenosti za dinamičkog predstavnika.

3. Programsko ostvarenje algoritama usmjeravanja

Programske kodove sam pisao u programskom jeziku c++. Nisam koristio objektno programiranje, već samo strukture, main funkciju, pokoju funkciju koju sam napisao da ibavlja neki specifičan posao i neke gotove funkcije koje su mi bile jako korisne iz bogatih c++ biblioteka. Dijkstrin algoritam sam ostvario preko main i još dvije funkcije koje pozivam za prolazak kroz sam algoritam i bilježenje potrebnih podataka. Također, algoritam vektora udaljenosti sam ostvario na sličan način s tim da ima par načina rada budući je to dinamičan algoritam.

3.1. Dijkstrin algoritam usmjeravanja

Dijkstrin algoritam ostvaren je kroz s main funkcijom, funkcijom dijkstra i getPath funkcijom. U glavnoj funkciji sam iz datoteke primjer.txt učitavao broj čvora i broj grana kojima su povezani. Zatim sam u matricu učitavao vrijednost (kašnjenje) svake grane.

```
ifstream ulaz ("primjer.txt");
memset(matrica, 0, sizeof(matrica));
ulaz >> n >> br;
for (int i=0; i<br; i++) {
    ulaz >> a >> b >> c;
    matrica[a][b] = c;
}
for(int i=0; i<n; i++) {
    for(int j=0; j<n; j++)
        printf("%d ", matrica[i][j]);
    printf("\n");
}
```

Nakon toga printa se matrica na ekran da se vidi kako izgleda.

```
cout << "unesite izvor i odrediste: ";
cin >> i >> o;
GetSystemTime(&vrijeme1);
dijkstra(i);
getPath(o);
GetSystemTime(&vrijeme2);
trajanjems = 1000 * (vrijeme2.wSecond - vrijeme1.wSecond) +
            vrijeme2.wMilliseconds - vrijeme1.wMilliseconds;
cout << "trajanje algoritma: " << trajanjems << "ms“ << endl;
```

Onda program traži od korisnika da unese izvorišni i odredišni čvor u mreži, nakon toga pokreću se dvije funkcije gdje prva radi za zadani izvor najkraću udaljenost do svakog odredišta te put kojim će ići. Druga funkcija, getpath samo prolazi po tim čvorovima, ispisuje ih te ispisuje ukupnu udaljenost (kašnjenje) koje se obavilo. Pritom sam mjerio te ispisivao na ekran, unutar main funkcije, trajanje samog algoritma.

U funkciji dijkstra prvo sam inicijalizirao sve udaljenosti i veze među čvorovima kako bi algoritam mogao početi.

```
for (int i=0; i<n; i++) {
    udaljenost[i] = MAXINT;
    otac[i] = -1;
    obilazak[i] = false;
}
udaljenost[izvor] = 0;
```

```
  queue.push(pair <int,int> (udaljenost[izvor], izvor));
  while(!queue.empty()) {
    while(!queue.empty()) {
       cvor = queue.top();
       queue.pop();
    }
    i = cvor.second;
    if (!obilazak[i]) {
      obilazak[i] = true;
      for (j=0;j<n;j++)
      if (!obilazak[j] && matrica[i][j] > 0 && udaljenost[i] + matrica[i][j] < udaljenost[j]) {
         udaljenost[j] = udaljenost[i] + matrica[i][j];
         otac[j] = i;
         queue.push(pair <int,int>(udaljenost[j], j));
     }
    }
}
```

Koristio sam red parova kako bi mogao ostvariti teorijsko prolaženje dijkstinog algoritma kako je ranije opisano u teoriji. Jednostavno, kod za svaki čvor provjerava postoji li neki drugi čvor preko kojeg je kraća udaljenost do željenog čvora. Polje otac mi koristi kao očinski čvor. Dakle otac[j]=i; znači da je i otac od j, tj. da se do j-čvora najkraće dođe preko i-čvora.

U funkciju getpath predaje se kao argument odredište. Budući je funkcija dijsktra za odabrani izvor već pokrenuta, izračunate su sve udaljenosti od izvora pa tako i do našeg odredišta. Također izračunat je i put kojim moramo ići da bi stigli do odredišta. U funkciji getpath prvo se ispisuje, u datoteku rezultati.txt ukupna udaljenost od izvora do odredišta.

```
ofstream izlaz ("rezultati.txt");
izlaz << "Udaljenost do odredista je " << udaljenost[odrediste] << endl;
list<int> lista;
lista.push_front(odrediste);
while (otac[odrediste]!= -1) {
    lista.push_front(otac[odrediste]);
    odrediste = otac[odrediste];
}
izlaz << "Put kroz mrezu od izvora do odredista je:" << endl;
list<int>::iterator it;
for (it = lista.begin(); it != lista.end(); it++) {
   izlaz << *it << " ";
}
izlaz << endl;
```

Nakon toga program prolazi od odredišta unazad preko polja otac jer u njemu stoje svi prethodni čvorovi do izvora. Kako unazad ide od odredišta do izvora koristio sam listu s kojom sam po principu LIFO (last in first out) preokrenuo taj niz i na koncu iteratotom ispisao, u datoteku, sve čvorove kojim se prolazi najkraći put od zadanog izvora do odredišta.

3.2. Algoritam vektora udaljenosti

Algoritam vektora udaljenosti ostvario sam kroz main funkciju u kojoj unosimo određene podatke koje nas program traži i dvije funcije algoritam i izracunaj sljedeci cvor. Prvo sam napravio jedno globalno dvodimenzionalno polje matrica i jednu globalnu strukturu cvor nazvanu usmjeritelj. Definirao sam dvije vrijednosti MAX i MAXINT.

```
#define MAXINT (int)((pow(2,31)-1)/2-1)
#define MAX 100


unsigned int matrica[MAX][MAX];


struct cvor {
   unsigned udaljenost[MAX];
   unsigned preko[MAX];
} usmjeritelj[MAX];
```

U glavnoj funkciji sam inicijalizirao matricu i vrijednosti usmjeritelja za početni rad.

```
for (i=0;i<MAX;i++)
   for (j=0;j<MAX;j++) {
      usmjeritelj[i].udaljenost[j]=MAXINT;
      usmjeritelj[i].preko[j]=j;
   }
memset(matrica, MAXINT, sizeof(matrica));
```

Iz tekstualne datoteke (u floderu su primjer.txt i primjer2.txt, dvije različite topologije mreže) učitavam isto kao u dijkstri broj čvorova i broj grana. Nakon toga sve vrijednosti grana spremam u matricu, ali i u strukturu usmjeritelj. Nakon toga program traži da se unese izvor i odredište, a kasnije i način rada mreže (statički, dinamički, dodavanje novog čvora ili ispadanje čvora iz mreže). Nakon toga se pokreće funkcija algoritam koja će ispisati sve potrebne podatke, a u glavnom programu program mjeri i ispisuje na ekran brzinu izvršavanja kao što je radio i u dijkstinom kodu.

```
ifstream ulaz ("primjer2.txt");
ulaz >> n >> br;
for (i=0;i<n;i++) {
   matrica[i][i]=0;
   usmjeritelj[i].udaljenost[i]=0;
   usmjeritelj[i].preko[i]=i;
}
for (i=0;i<br;i++) {
   ulaz >> a >> b >> c;
   matrica[a][b] = c;
   usmjeritelj[a].udaljenost[b]=c;
   usmjeritelj[a].preko[b]=b;
}
cout << "Unesite izvor i odrediste: ";
cin >> izvor >> odrediste;
cout << "Unesite STAT za staticku mrezu, DIN za dinamicku, \n
        NEW za novi cvor u mrezi ili DEL za ispadanje jednog cvora iz mreze: ";
cin >> str;
algoritam(izvor,odrediste,n,str);
```

Funkcija Algoritam poziva funkciju izracunaj_sljedeci_cvor koja prolazi po svim kombinaciama čvorova i provjerava postoji li neki novi put koji je kraći i izvršava ga sve dok promjena postoji.

```
do {
   br=0;
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         for(k=0;k<n;k++)
            if(usmjeritelj[i].udaljenost[j]>matrica[i][k]+usmjeritelj[k].udaljenost[j]) {
               usmjeritelj[i].udaljenost[j]=matrica[i][k]+usmjeritelj[k].udaljenost[j];
               usmjeritelj[i].preko[j]=k;
               br++;
            }
} while (br!=0);
```

Funkcija algoritam je dosta duga jer u njoj se mijenjaju vrijednosti matrice u ovisnosti o predanom argumentu s tipkovnice, dakle STAT (statički način rada mreže), DIN (dinamički), NEW (dodavanje novog čvora) ili DEL (brisanje nekog čvora iz topologije mreže). Objasnit svaki dio koda posebno, ali prvo da komentiram usputni kod.

```
ofstream izlaz ("rezultati.txt");
izlaz << "Put kroz mrezu od izvora do odredista za " << str << " nacin rada je:" << endl;
izlaz << i << " ";
while (i!=o) {
   br++;
   if (str=="DIN" && br==1) {...
   }
```

```
    if (str=="NEW" && br==1) {...
    }
    if (str=="DEL" &&br==1) {...
    }
    izracunaj_sljedeci_cvor (n);
    j=usmjeritelj[i].preko[o];
    suma+=usmjeritelj[i].udaljenost[j];
    i=j;
    izlaz << i << " ";
}
izlaz << endl << "Udaljenost do odredista je " << suma << endl;
```

Funkcija prvo ispisuje, u datoteku rezultati.txt, prvi čvor prije ulaska u while petlju. Nakon toga provjerava koji je argument primljen i shodno tome mijenja matricu. Nakon toga se poziva funkcija koja izračuna sljedeći čvor, udaljenost do njega se pribroja sumi i sljedeći cvor se ispisuje u datoteku. Kada se izađe iz while petlje ispisuje se u datoteku i ukupan put od izvora do odredišta

```
if (str=="DIN" && br==1) {
    for (j=0;j<n;j++)
        for (k=0;k<n;k++)
            if (j!=k) {
                usmjeritelj[i].udaljenost[j]=matrica[i][j];
                usmjeritelj[i].preko[j]=j;
                if ((j==6 || k==6) && matrica[j][k]<20) {
                    matrica[j][k]+=3;
```

```
                usmjeritelj[i].udaljenost[j]=matrica[i][j];
            }
            if ((j==3 || k==3) && matrica[j][k]>2 && matrica[j][k]<20) {
                matrica[j][k]-=2;
                usmjeritelj[i].udaljenost[j]=matrica[i][j];
            }
            else if ((j==3 || k==3) && matrica[j][k]==2) {
                matrica[j][k]-=1;
                usmjeritelj[i].udaljenost[j]=matrica[i][j];
            }
        }
    }
```

Dinamički način usporava rad 6 čvora u mreži tako da sve njegove veze povećavanju kašnjenje za 3 (dotadašnje vrijednosti su bile 1-9), a 3 čvoru smanjuju za 2 kašnjenje osim u slučaju da je kašnjenje 2, tada smanji za 1, a ako je kašnjenje bilo 1 ostavlja ga. Odabrao sam ova dva čvora jer često preko njih idu podaci, ali moglo bi se implementirati za bilo koji čvor.

```
if (str=="NEW" && br==1) {
    for (j=0;j<n;j++)
        if (j==3) {
            usmjeritelj[j].udaljenost[10]=matrica[j][10]=1;
            usmjeritelj[j].preko[10]=10;
            usmjeritelj[10].udaljenost[j]=matrica[10][j]=1;
            usmjeritelj[10].preko[j]=j;
        }
```

```
        else if (j==6) {
            usmjeritelj[j].udaljenost[10]=matrica[j][10]=2;
            usmjeritelj[j].preko[10]=10;
            usmjeritelj[10].udaljenost[j]=matrica[10][j]=2;
            usmjeritelj[10].preko[j]=j;
        }
        else if (j==7) {
            usmjeritelj[j].udaljenost[10]=matrica[j][10]=2;
            usmjeritelj[j].preko[10]=10;
            usmjeritelj[10].udaljenost[j]=matrica[10][j]=2;
            usmjeritelj[10].preko[j]=j;
        }
        else if (j==10) {
            usmjeritelj[j].udaljenost[10]=matrica[j][10]=0;
            usmjeritelj[10].preko[j]=j;
        }
        else {
            usmjeritelj[j].udaljenost[10]=matrica[j][10]=MAXINT;
            usmjeritelj[j].preko[10]=10;
            usmjeritelj[10].udaljenost[j]=matrica[10][j]=MAXINT;
            usmjeritelj[10].preko[j]=j;
         }
    n++;
}
```

Dodavanje čvora se ostvaruje da ga nazovemo kao 10. čvor (do tada smo imali 0-9 redni broj čvorova) i ostvaruje se veza s 3., 6. i 7. čvorom te se veza sa samim sobom inicijalizira.

Ispadanje čvora iz mreže se realizira tako da se sve njegove grane postave na najveći broj integera, figurativno beskonačnost. Napravio sam da se 6 čvor izbacuje iz mreže jer ranije u dinamičkom radu napravio sam da se 6. Čvoru otežaju grane. Može se lako implementirati za bilo koji čvor. Statički način rada jednostavno preskače sve ove if uvjete te ne mijenja matricu.

```
if (str=="DEL" && br==1)
    for (j=0;j<n;j++) {
        usmjeritelj[j].udaljenost[6]=matrica[j][6]= MAXINT;
        usmjeritelj[6].udaljenost[j]=matrica[6][j]=MAXINT;
        usmjeritelj[6].preko[j]=usmjeritelj[j].preko[6]=6;
    }
```

4. Analiza performansi

U ovom radu, programski sam provjeravao dvije stvari koje utječuna performansu mreže. Mjerio sam vrijeme potrebno za izvršavanje određenog algoritma usmjerevanja te računao dužinu puta (zbroj težinskih vrijednosti grana, može se smatrati kašnjenjem) za različite načine rada mreže. Budući da moje topologije mreže su se sastojale od svega 10 čvorova i broj skokova na testiranim putovima nisu prelazili 5 nije bila velika razlika u vremenu izvršavanja algoritama. Također zbog relativno malog broja iteracija algoritmu su se više manje izvršavali u dužini 1ms, a u programskom jeziku c++, našao sam funkciju samo s najvećom finoćom timera upravo 1ms. Međutim iz svih mjerenja koja sam dobio ispisane na ekran zaključujem da se algoritam vektora udaljenosti duže izvršava od dijkstrinog algoritma kao što teorija govori. Većina mjerenja dijkstrinog vremena izvršavanja je bila 0ms, dok je pokoje izvršavanje bilo 1ms. Većina mjerenja vektora udaljenosti je bila 1ms. Algoritam vektora udaljenosti je svakako složeniji jer za svaki novi čvor u koji pređe ponovo računa optimalni put i usmjeravanje do sljedećeg čvora.

Izračune duljina puteva ću prikazati za redom sljedeće parove izvor-odredište: 1. 0-8, 2. 1-5, 3. 7-9, 4. 8-3, 5. 9-4 i to za dva primjera topologije mreže. Birao sam ih tako da bi u svakom načinu rada se na ponekom putu dogodila neka zanimljiva promjena. Redosljed prolazaka kroz čvorove mreže za ove parove izvor-odredišta na prvom primjeru topologije mreže u statičkoj mreži su:

0-8 -> 0, 3, 2, 5, 9, 8

1-5 -> 1, 2, 5

7-9 -> 7, 3, 2, 5, 9

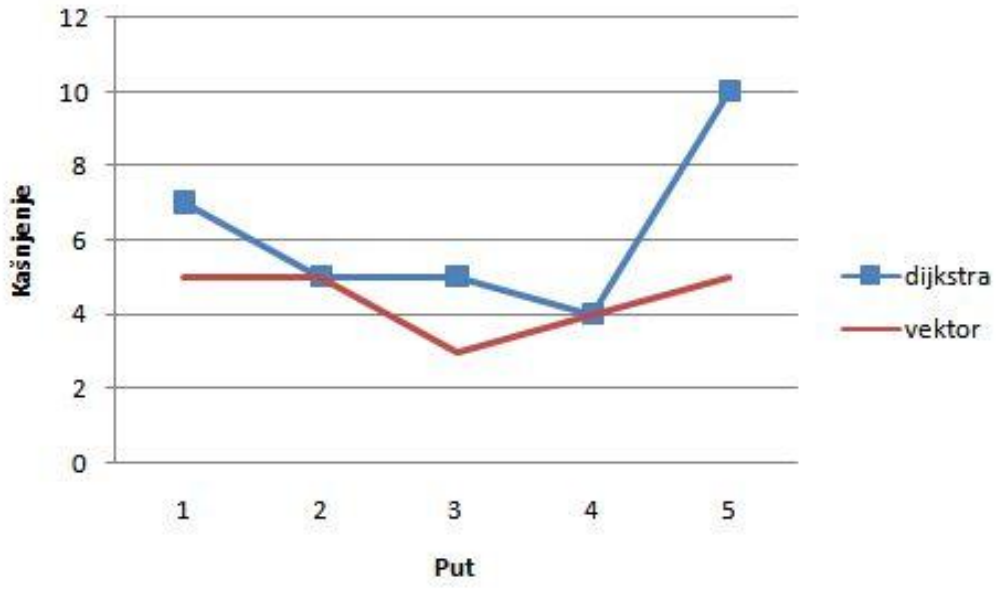
8-3 -> 8, 9, 5, 3

9-4 -> 9, 6, 4

Vrijednosti kašnjenja prikazana su na slici 9. Ona su ista za oba algoritma jer se mreža nije mijenjala, ali dijkstrin je algoritam se brže izvršio.

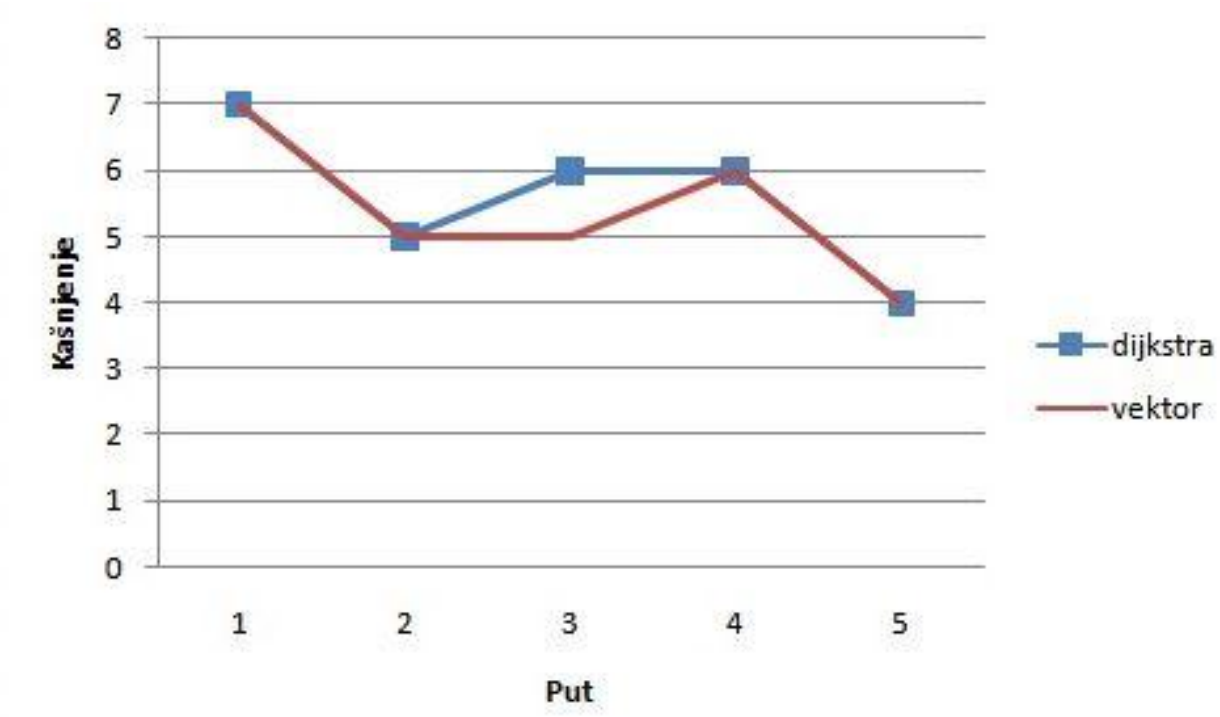
Slika 9: Mjerenje kašnjenja za statičku mrežu na primjeru 1

Na dinamičkom radu mreže (slika 10) dolazimo do zanimljivih usporedbi dvaju algoritama, ali i usporedbi s prijašnjom slikom da se vidi koja je rezlika jednog algoritma, a koja drugog u odnosu statičke mreže. Vidljivo je kako dijkstrin algoritam koji, mada je izračunao određeni najkraći put na početku, zbog promjena u mreži ponekad paketu treba duže, a ponekad kraće do cilja.



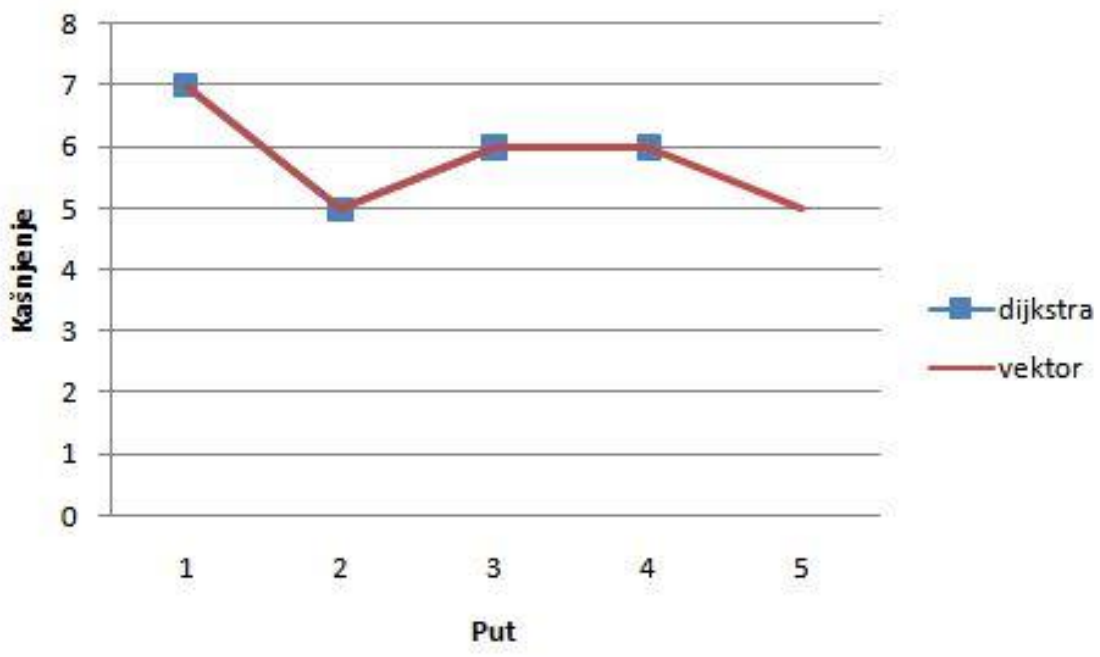
Slika 10: Mjerenje kašnjenja za dinamičku mrežu na primjeru 1

Dinamičan rad mreže implementiran je tako da se čvoru 6 otežaju sve grane za 3, a čvoru broj 3 olakšaju za 2 ukoliko im je vrijednost veća od 2 ili za 1 ukoliko im je vrijdnost jednaka 2. Samim time svaki par izvor-odredište koji u statičkoj mreži ima najkraći put takav da prolazi kroz čvorove 3 ili 6 će imati određene promjene kašnjenja do cilja. Vidljivo je da algoritam vektora udaljenosti mada mu treba nešto više vremena za izvršavanje je daleko korisniji u dinamičkoj mreži, a realne mreže su dinamičke. Valja još napomenuti kako se prvi i treći put promijenio na način da ide direktno iz čvora 3 u 5, a ne preko 2 kao ranije. Zadnji put promijenio je svoje usmjeravanje tako da više ne ide preko čvora 6 koji je „otežatni“ već ide put ovako 9, 8, 4.



Slika 11: Mjerenje kašnjenja prilikom dodavanja čvora u mrežu za primjer 1

Dodavanje čvora u mrežu je ostvareno spajanjem tog čvora s čvorovima 3, 6 i 7. Postavio sam niske vrijednosti tih grana mreže kako bi se češće našao kraći preko novog čvora međutim nisam u ovoj topologiji baš pogodio spoj s čvorovima pa se samo na jednom putu dogodio kraći put preko novog čvora 10. Međutim opet je vidljivo kako se u takvoj situaciji algoritam vektora udaljenosti bolje snalazi od statičkih algoritama. Također valja primjetiti kako dijkstrin algoritam se ponaša jednako kao u statičkoj mreži dok algoritam vektora udaljenosti ipak za jedan par izvor-odredište napao kraći put.



Slika 12: Mjerenje kašnjenja prilikom brisanja čvora iz mreže za primjer 1

Prilikom brisanja određenog čvora u mreži svi putevi podataka koji su prethodno izračunati preko njega se odbacuju. Iz tog razloga dinamički algoritmi pa tako i vektora udaljenosti su mnogo upotrebljiviji od statičkih. Dijkstrin algoritam bi u ovakvoj situaciji na 5. putu (9-4) zapeo na 6. čvoru jer kod je implementiran tako da se čvor broj 6 izbacuje iz mreže. Dužina puta za ostale putove koji ne idu na čvor 6 ostaju isti kao u statičkoj mreži. Posljednji put je promijenio svoje usmjeravanje kao i na dinamičkom načinu tako da ide preko čvora 8, a ne 6 koji je sada izbačen iz mreže.

Redosljed prolazaka kroz čvorove mreže za ove parove izvor-odredišta na prvom primjeru topologije mreže u statičkoj mreži su:

0-8 -> 0, 4, 6, 8

1-5 -> 1, 6, 4, 5

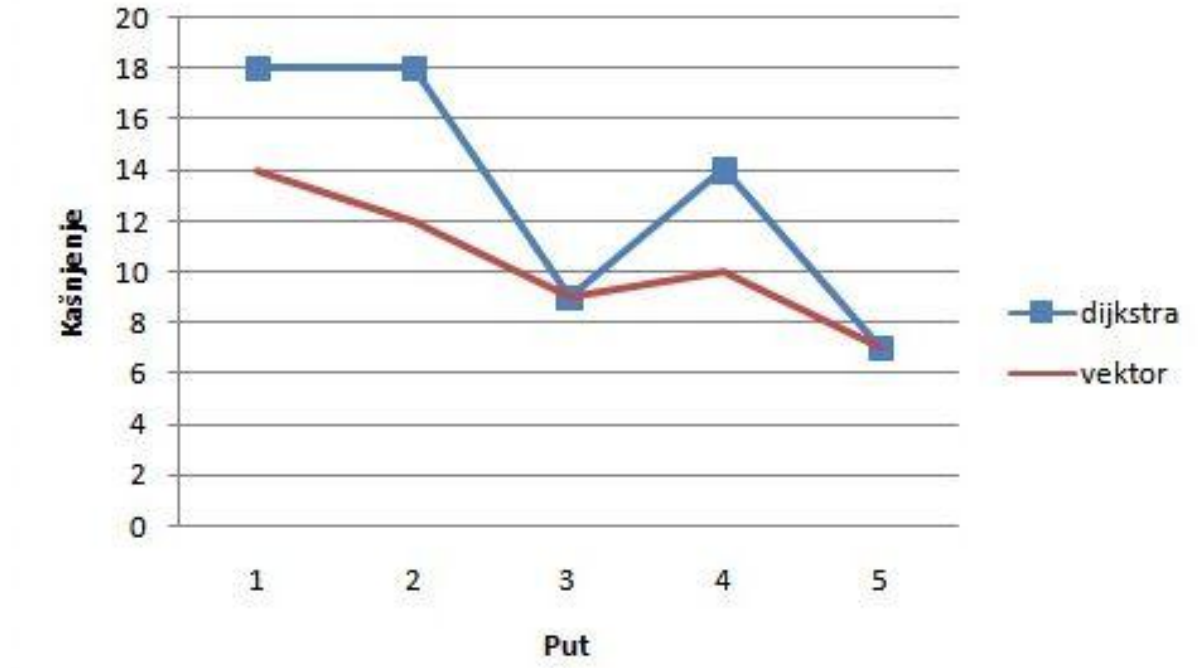
7-9 -> 7, 8, 9

8-3 -> 8, 6, 4, 3

9-4 -> 9, 4

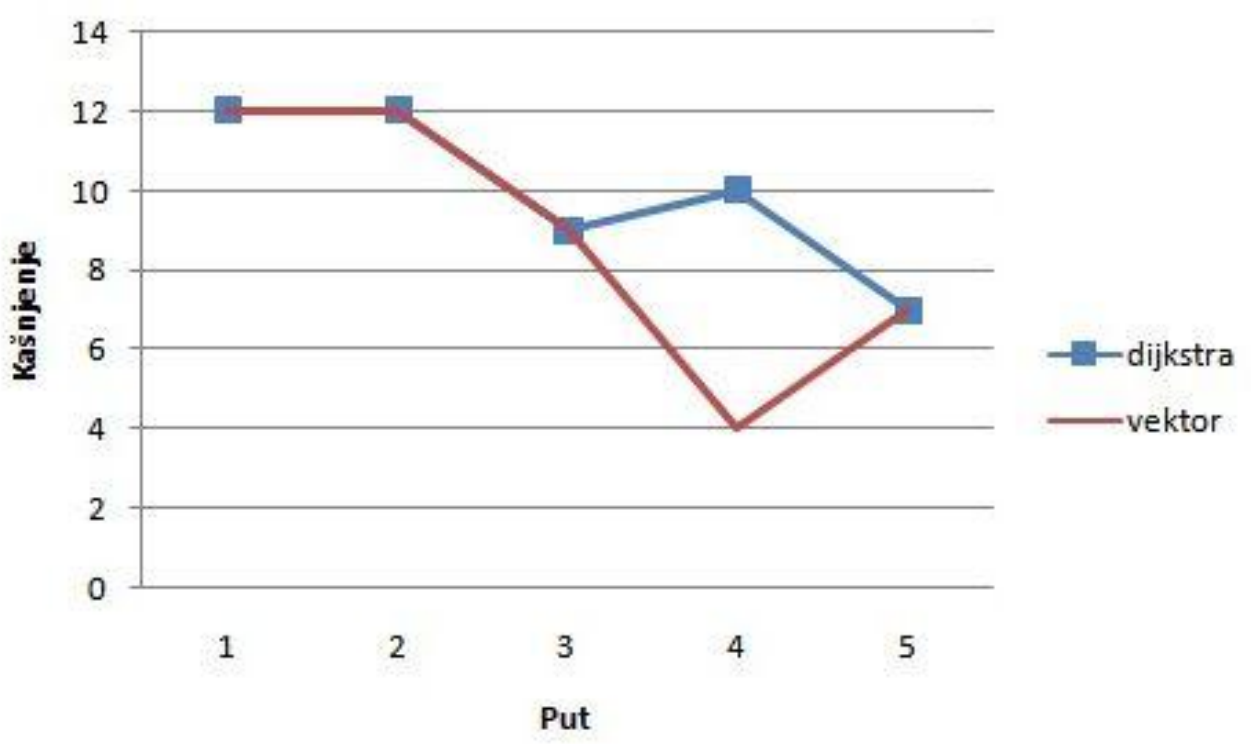
Slika 13: Mjerenje kašnjenja za statičku mrežu na primjeru 2

Također, kao ranije spomenuto, oba algoritma računaju jednako kašnjenje paketa od izvora do odredišta. Dijkstrin algoritam je ponovo bio brži u izvršavanju.



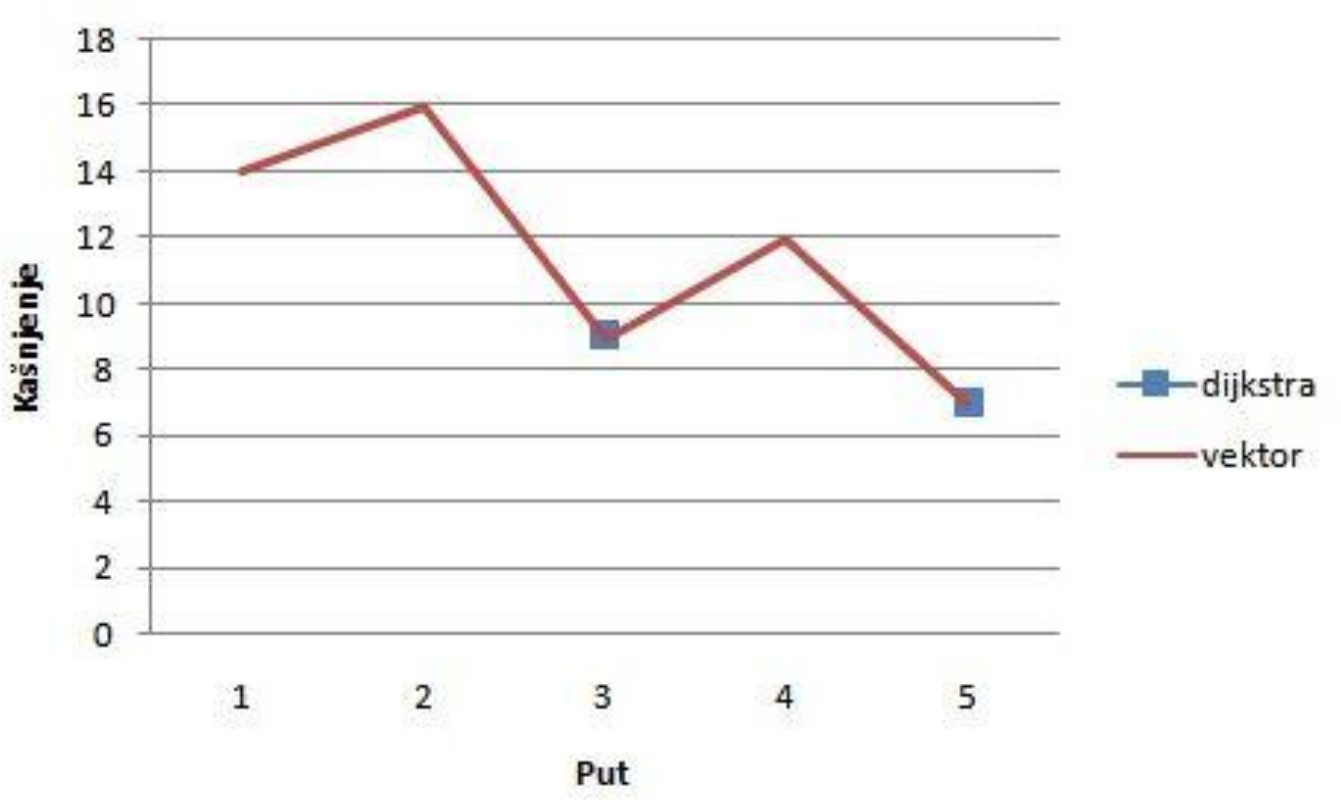
Slika 14: Mjerenje kašnjenja za dinamičku mrežu na primjeru 2

U ovom primjeru je čak očitija razlika između dva testirana algoritma. Razlog je taj što u statičkoj mreži tri puta su prolazila kroz čvor 6 koji je „otežano“ implementiran. Naravno putevi podataka kod algoritma vektora udaljenosti su svoje usmjeravanje promijenili i ne idu preko čvora 6.



Slika 15: Mjerenje kašnjenja prilikom dodavanja čvora u mrežu za primjer 2

Ovaj put također samo jedan par izvor-odredište je preopoznalo novi čvor kao brži način kako stići do odredišta, ali razlika je velika. Ovom prilikom samo put broj 4 je promijenio usmjeravanje na 10 čvor. Dinamički algoritam je jako koristan u ovim situacijama.



Slika 16: Mjerenje kašnjenja prilikom brisanja čvora iz mreže za primjer 2

Na ovom se primjeru jasno vidi kako u realnoj mreži, gdje se može dogoditi da neki čvor zaustavi rad jer se prezatrpao pristiglim paketima pa se kašnjenja i propusnost jako povećala ili pak se veza jednostavno prekinula s njim, statički algoritmi ne funkcioniraju dobro jer zapnu kod tog čvora i ne stignu do cilja. Dinamični algoritmi mada složeniji i dužeg izvršavanja su puno korisniji u realnom sustavu.

Zaključak

Postoje mnogi algoritmi usmjeravanja, kako statički, tako i dinamički. Smatram da su dijkstrin odnosno algoritam vektora usmjeravanja reprezentativni za svoju skupinu. Prvo što zaključujem iz dobivenih mjerenja je da su statički algoritmi brži, manje složenosti od dinamičkih te samim time bi za statičku mrežu bili idealniji od dinamičkih algoritama.

Problem kod vjerodostojnosti izračuna statičkog algoritma događa se onda kada je mreža dinamička, što znači da se mijenja. Može se mijenjati tako da se propusnost ili kašnjenje mijenja na određenim granama ili da se neki novi čvor dodaje u mrežu. Budući statički algoritmi, pa tako dijkstrin, uzimaju u obzir samo stanje mreže na početku slanja paketa oni imaju krive izračune puta, a javlja se i problem što ukoliko se neki čvor izbacuje iz mreže zbog nekog razloga, paket kod njega zastaje. Postoje određene implementacije koje riješavaju taj problem, ali to nije ni približno brzo kao što to rade dinamički algoritmi koji svakim prelazom u novi čvor provjeravaju trenutno stanje mreže i računaju najkraći put do cilja.

Vektor udaljenosti je na nekim primjerima značajno smanjio kašnjenje paketa to cilja. To mu omogućuje stalni uvid u trenutno stanje mreže pa može donijeti pravovremenu reakciju i odluku kamo da dalje šalje paket. Također zaključujem da reagira bolje na „dobre vijesti“ kao dodavanje čvora u mrežu, nego na „loše“ kao izbacivanje čvora iz mreže. Sveukupno gledajući smatram da su dinamički algoritmi puno upotrebljiviji od statičkih, osim u slučaju sigurno nepromjenjive mreže.

Literatura

Svaki autor piše popis literature na kraju svog poglavlja. Popis literature se piše stilom literatura.

[1] IGNAC LOVREK, MAJA MATIJAŠEVIĆ, GORDAN JEŽIĆ, DRAGAN JEVTIĆ. *KOMUNIKACIJSKE MREŽE.* RADNA INAČICA UDŽBENIKA V1.1, 2011.

[2] FETTERER, A. *A performance analysis of hierarchical shortest path algorithms*. IEEE Computer Society, November 1997.

[3] YINFEI PAN, "DESIGN ROUTING PROTOCOL PERFORMANCE COMPARISION IN NS2: AODV COMPARING TO DSR AS EXAMPLE, DEPTT OF CS, SUNY BINGHAMTON, VESTAL NY 13850

[4] TYAGI, S.S., Chauhan, R.K. Performance analysis of proactive and reactive routing protocols for ad hoc networks, *International Journal of Computer Applications*, Volume 1 - no. 14 (2010).

[5] Cormen, Thomas H., Leiserson, Charles E., Riverst, Ronald L., Stein, Clifford. *Introduction to algorithms*. MIT press 2001.

[6] Garcia-Luna-Aceves J.J., Murthy S. *A Path-Finding Algorithm for Loop-Free Routing.* IEEE/ACM Transactions on Networking, February 1997.

Sažetak

Naslov: Analiza performansi algoritama usmjeravanja

Sažetak: U ovom radu simulirali su se dijsktrin statički algoritam usmjeravanja i dinamički algoritam vektora udaljenosti. Za oba algoritma koristio se programski jezik c++. Testirani su bili kroz dvije različite topografije koje su radile i kao statičke i dinamičke ovisno o želji te se kasnije uspoređivali rezultati. Također mjerilo se vrijeme izvršavanja samog algoritma kao mjeru složenosti.

Ključne riječi: c++, algoritam usmjeravanja, performanse, kašnjenje, propusnost, duljina puta, statički i dinamički algorimi, dijkstrin algoritam, algoritam preplavljivanja, algoritam vektora udaljenosti, algoritam stanja poveznice.

Summary

Title: Peformance analysis od routing algorithms

Summary: In this work following algorithms were analyzed: dikstra and distance vector algorithm. Programming language used for simulation was c++. They were tested through two diffrent network topologies. Also there were 4 ways of network working that could be chosen from the program. As a measurment of complexity, the time of implemention was measured.

Keywords: c++, routing algorithm, performance, delay, permeability, path lenght, static and dynamic algorithms, dikstra algorithm, flooding algorithm, distance vector algorithm, link state algorithm

Skraćenice

TCP	*Transmission Control Protocol*	transportni protokol
UDP	*User Data Protocol*	transportni protokol
IP	*Internet Protocol*	mrežni internetski protokol
MAC	*Media Acces Control*	fizička adresa mrežnih uređaja
DHCP	*Dynamic Host Configuration Protocol*	mrežni protokol
RIP	*Routing Information Protocol*	protokol usmjeravanja
OSPF	*Open Shortest Path First*	protokol stanja poveznice
LIFO	*Last In First Out*	način upravljanja stogom, redom...
FIFO	*First In First Out*	način upravljanja stogom, redom...

Privitak

Instalacija programske podrške

C++ programski jezik je korišten u izradi ovog završnog rada. Funkcije korištene u programskom ostvarenju algoritama su funkcije koje se mogu koristiti na većini verzija microsoft visual studia. Ovaj rad je napravljen preko programa code blocks.

Upute za korištenje programske podrške

Pokretanje main programa uz po želji promjenu raznih varijabli. Kao ispis programa javit će jasno što se od vas traži da unese kako bi se program mogao izvšiti. Većinom su to unos izvora i odredišta uz opciju odabira statičkog ili nekog od dinamičkog ponašanja mreže.