SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Andro Milanović

UPRAVLJAČKI PROTOKOL RASPODIJELJENE PRIRUČNE MEMORIJE

MAGISTARSKI RAD

Zagreb, 2002.

Magistarski rad je izrađen na Zavodu za elektroniku,
mikroelektroniku, računalne i inteligentne sustave

Mentor: prof. dr. sc. Siniša Srbljić

Magistarski rad ima 99 stranica.

Rad br. _____

Povjerenstvo za ocjenu u sastavu:

1. prof. dr. sc. Uroš Peruško
2. prof. dr. sc. Siniša Srbljić
3. doc. dr. sc. Nikola Hadjina

Povjerenstvo za obranu u sastavu:

1. prof. dr. sc. Uroš Peruško
2. prof. dr. sc. Siniša Srbljić
3. doc. dr. sc. Nikola Hadjina

Datum obrane: 16. siječnja 2002.

Zahvaljujem prof. dr. sc. Siniši Srbljiću na vođenju i mogućnosti da praktični rad ostvarim u AT&T laboratorijima. Zahvaljujem se supruzi Nadži na razumijevanju i praktičnim savjetima. Zahvalan sam i kolegama koji su svojim prijedlozima pomogli pisanje rada.

Sadržaj

1. Uvod

Povećanjem broja Internet korisnika i količine multimedijskih sadržaja Internet promet eksponencijalno raste. Budući da razvoj mrežnih i računalnih tehnologija ne prati navedeni rast, javlja se znatno kašnjenje dohvata WWW (World Wide Web) stranica i ostalih Internet objekata. Kašnjenje uzrokuje nezadovoljstvo krajnjih korisnika i novčane troškove uzrokovane gubitkom vremena. Brojne organizacije i pružatelji Internet usluga stoga pokušavaju skratiti vrijeme dohvata Internet stranica. Jedan od načina ubrzanja dohvata objekata je uporaba priručnih memorija.

Priručne memorije na Internetu u početku su korištene samo u Internet pretraživačima. Navedena uporaba priručnih memorija ograničena je na ubrzanje rada jednog korisnika. Osim toga, ubrzanje se postiže samo ako korisnik višestruko pristupa istim stranicama. Učinkovitija primjena priručnih memorija na Internetu je zastupničko računalo. Zastupničko računalo prima zahtjeve od skupine korisnika čime se, u odnosu na priručne memorije Internet pretraživača, povećava vjerojatnost da je isti objekt višestruko tražen. Zadaća zastupnika je posredovanje između korisnika i poslužitelja tijekom procesa dohvata objekta. Korisničko računalo svoje zahtjeve upućuje zastupniku, a zastupnik poslužuje tražene objekte. Ako objekt prethodno nije bio zatražen od zastupnika, onda zastupnik traženi objekt dohvaća s poslužitelja i šalje ga korisniku. Ako je objekt prethodno bio tražen od zastupnika, onda ga zastupnik dohvaća iz lokalne priručne memorije i šalje korisniku bez komunikacije s poslužiteljem. Budući da je kašnjenje između zastupnika i korisnika manje od kašnjenja između poslužitelja i korisnika, posluživanjem objekata iz zastupnikove priručne memorije smanjuje se kašnjenje.

Nedostatak je sustava s jednim zastupnikom mala vjerojatnost da zastupnik ima traženi objekt. Zbog veće vjerojatnosti da više korisnika traži isti objekt, vjerojatnost pronalaženja objekta moguće je povećati ako se poveća broj korisnika po zastupniku. Povećanjem broja korisnika mogla bi se povećati učinkovitost zastupnika, ali se na taj način preopterećuje zastupničko računalo. Na temelju svojstva da vjerojatnost pronalaženja objekta na zastupniku raste s brojem korisnika, stvoreni su raspodijeljeni zastupnički sustavi. Raspodijeljeni zastupnički sustav sastoji se od skupa zastupnika koji međusobno komuniciraju. Korisnik šalje zahtjev jednom od zastupnika u sustavu. Ako zastupnik koji je primio zahtjev nema traženi objekt, onda on provjerava da li ga ostali zastupnici u sustavu imaju. Zastupnik potom dohvaća objekt od susjednog zastupnika. Ako nijedan susjedni zastupnik nema traženi objekt, onda se objekt dohvaća od poslužitelja. Raspodijeljeni zastupnički sustav smanjuje kašnjenje dohvata objekata ako bar jedan od zastupnika u sustavu ima traženi objekt. Budući da raspodijeljeni zastupnički sustav ima veći broj korisnika od izdvojenog zastupnika, veća je i vjerojatnost pronalaženja objekta u raspodijeljenom sustavu.

Raspodijeljeni zastupnički sustav učinkovit je samo ako je protokol za komunikaciju između zastupnika učinkovit i brz. Uspješnost zastupničkog sustava stoga u velikoj mjeri ovisi o pravilnom izboru upravljačkog protokola priručne memorije. Najrašireniji protokol za upravljanje zastupničkim sustavima je protokol ugrađen u sustav Squid [8-10]. Značajan nedostatak protokola Squid je

prekomjerna količina komunikacije koja brzo raste s brojem zastupnika u zastupničkom sustavu. Zbog nedostataka protokola Squid i ostalih upravljačkih protokola, u okviru tvrtke AT&T razvijen je upravljački protokol zasnovan na direktoriju [23,24]. Direktorijskim protokolom značajno se smanjuje mrežno opterećenje i opterećenje zastupničkih računala uzrokovano komunikacijom između zastupnika.

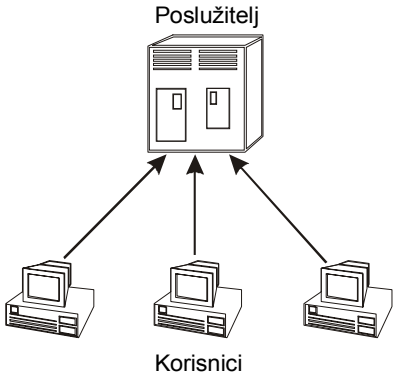
Magistarski rad opisuje programsko ostvarenje direktorijskog protokola i uspoređuje direktorijski i Squid protokol temeljem rezultata mjerenja. U svrhu programskog ostvarenja u sustavu Squid proučeni su komunikacijski standardi koji se koriste u suvremenim zastupničkim sustavima. Analiziran je protokol koji sustav Squid koristi za upravljanje raspodijeljenim zastupničkim sustavima. Na temelju komunikacijskih standarda i postojećeg upravljačkog protokola definirane su nove komunikacijske poruke za upravljanje raspodijeljenom priručnom memorijom. Nakon programskog ostvarenja direktorijskog protokola, uspoređena su njegova svojstva sa svojstvima protokola Squid. U svrhu usporedbe protokola ostvaren je mjerni sustav za mjerenje kašnjenja direktorijskog i Squid protokola. Programski su ostvareni mjerni programi i definirani uvjeti izvođenja mjerenja. Prikupljeni rezultati grafički su obrađeni i analizirani. Na temelju analize uspoređena su svojstva proširivosti direktorijskog i Squid protokola.

U drugom je poglavlju prikazana uporaba raspodijeljenih priručnih memorija u Internet sustavima. Poglavlje počinje pregledom razvoja priručnih memorija, te predstavlja model raspodijeljene priručne memorije. Potom je opisan komunikacijski protokol ICP namijenjen povezivanju zastupničkih sustava. Detaljno su opisani upravljački protokoli Squid, Berkeley i direktorijski protokol. Prikazana je jednostavna analitička usporedba opterećenja navedenih upravljačkih protokola. Na kraju poglavlja dan je opis ostalih znanstvenih istraživanja s područja protokola za upravljanje raspodijeljenim priručnim memorijama u Internet sustavima. Četvrto poglavlje opisuje programsko ostvarenje upravljačkog protokola zasnovanog na direktoriju. U prvom odjeljku kratko je opisan zastupnički sustav Squid. Drugi odjeljak opisuje strukturu sustava Squid i način programskog ostvarenja direktorijskog protokola. Opis usporednih mjerenja svojstava protokola Squid i direktorijskog protokola nalazi se u petom poglavlju. Prvi dio poglavlja opisuje uvjete mjerenja i mjerni sustav. U drugom dijelu petog poglavlja prikazani su rezultati mjerenja. Na kraju rada dani su zaključak i pregled korištene literature.

2. Raspodijeljene priručne memorije u Internet sustavima

2.1. Razvoj priručnih memorija

Osnovno načelo rada Interneta je dohvat stranica i ostalih objekata od poslužiteljskih računala. Korisnik šalje zahtjev za objektom poslužitelju, a poslužitelj odgovara slanjem objekta korisniku. Jednostavan model korisnik-poslužitelj prikazan je na slici 1. Zbog velikog broja korisnika Interneta i velikog prometa koji korisnici stvaraju, model korisnik-poslužitelj značajno opterećuje komunikacijsku mrežu i poslužiteljska računala. Opterećenje komunikacijske mreže izazvano je prenošenjem objekata, a poslužiteljska su računala opterećena primanjem zahtjeva i odgovaranjem na zahtjeve korisnika. Brzinu dohvata objekata na Internetu moguće je povećati uporabom bržih mreža i bržih poslužitelja, ali rast broja korisnika Internet sustava i rast njihovih potreba veći su od brzine razvoja računala i mrežnih komunikacijskih tehnologija.

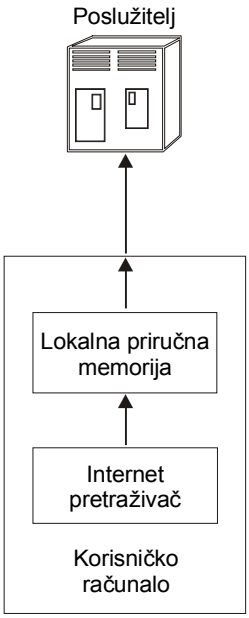


Slika 1: Model korisnik-poslužitelj

Obzirom da je poslužiteljsko računalo složeni sustav, njegova brzina ovisi o mnogo čimbenika. Iako snaga procesora ima stalan eksponencijalan porast, ostali dijelovi računala poput sabirnice, memorije i diskovnog podsustava unapređuju se znatno sporije. Stoga je porast brzine računala znatno sporiji od porasta brzine procesora pa brzina računala kao složenog sustava ne raste dovoljno brzo za ispunjavanje potreba Internet korisnika. Nedostatan porast brzine problem je i na području računalnih mreža gdje se nove tehnologije pojavljuju sporije nego na području računala. Dodatni su problem računalnih mreža visoki troškovi uvođenja novih mrežnih arhitektura. Postavljanje računalnih mreža donosi velike materijalne troškove koji nisu vezani samo uz cijenu nove tehnologije. Primjer navedenih troškova je postavljanje novih vodova u zgradi ili između udaljenih mjesta. Sporo pojavljivanje novih tehnologija i znatni troškovi postavljanja mreža usporavaju rast brzine računalnih mreža.

Budući da sklopovska poboljšanja nisu dovoljna, ubrzanje rada Interneta postiže se i unapređivanjem programske potpore poput poslužiteljskih programa ili TCP/IP podsustava. Na osnovi detaljnog proučavanja arhitekture operativnih sustava, računalnog sklopovlja i korisničkih zahtjeva unapređuju se svojstva programske potpore. Cilj promjena programske potpore je postizanje minimalnog

kašnjenja računalnog sustava. Budući da dobici postignuti navedenim promjenama nisu dovoljni za ostvarivanje očekivanja korisnika, izmijenjeno je načelo dohvata objekata od poslužitelja. Model korisnik-poslužitelj proširuje se uvođenjem priručnih memorija. Priručne memorije izvorno su korištene u procesorskim sustavima gdje se njihovom uporabom postižu velika ubrzanja. Ubrzanje dobiveno uporabom priručnih memorija u procesorskim je sustavima značajno, jer se njihovom uporabom smanjuje učestalost sporog čitanja i pisanja u radnu memoriju. Najjednostavniji način ubrzanja odziva Internet sustava uporabom priručne memorije predstavlja lokalna priručna memorija Internet pretraživača (engl. *browser*). Priručna memorija pretraživača ubrzava rad korisnika koji koristi Internet pretraživač. Ubrzanje se postiže samo tijekom višestrukog pristupanja istog korisnika istim stranicama unutar ograničenog vremena. Ako korisnik ne pristupa istim stranicama ili ako se stranice dinamički mijenjaju, onda priručna memorija pretraživača ne smanjuje kašnjenje dohvata stranica.

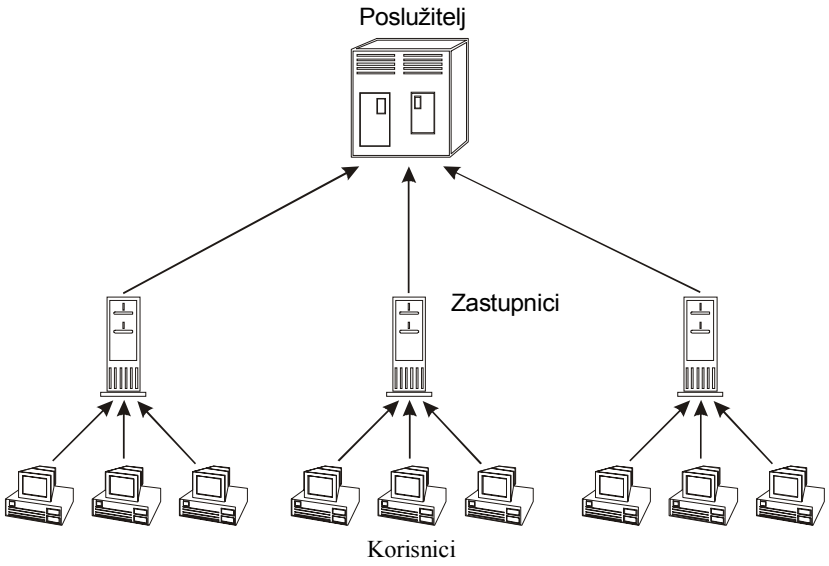


Slika 2: Model Internet pretraživača s lokalnom priručnom memorijom

Sustav Internet pretraživača s lokalnom priručnom memorijom prikazan je na slici 2. Nakon što korisnik zatraži otvaranje nove Internet stranice, pretraživač redom dohvaća objekte koji čine stranicu. Za svaki objekt pretraživač provjerava da li je spremljen u lokalnoj priručnoj memoriji. Ako u lokalnoj memoriji postoji valjana preslika traženog objekta, onda se objekt dohvaća iz priručne memorije pretraživača umjesto od poslužitelja. Ako u priručnoj memoriji pretraživača nije pronađena valjana preslika traženog objekta, onda pretraživač dohvaća objekt od poslužitelja. Nakon što je prenesen na korisničko računalo, objekt se sprema u lokalnu priručnu memoriju i potom se obrađuje i prikazuje u Internet pretraživaču.

Nedostatak lokalne priručne memorije Internet pretraživača je ograničenost na jednog korisnika. Ubrzanje se postiže samo ako isti korisnik pristupa istim stranicama. Problem je da korisnik tijekom rada na Internetu pristupa velikom skupu različitih stranica. Zbog toga se Internet stranice rijetko

dohvaćaju iz lokalne priručne memorije. Korisnik pristupa istim stranicama u rijetkim slučajevima kao što su vršne stranice hijerarhijski organiziranih Web stranica ili ponovno proučavanje određene stranice. Osim toga, ponovni pristup dinamičkim stranicama također ne donosi ubrzanja zbog potebe provjere da li su promijenjene stranice na poslužitelju. Posljedica opisanih nedostataka lokalne priručne memorije Internet pretraživača je malo smanjenje kašnjenja dohvata objekata. Smanjenje kašnjenja javlja se u malom skupu slučajeva kad korisnik višestruko pristupa stranicama koje se ne mijenjaju dinamički.



Slika 3: Model korisnik-zastupnik-poslužitelj

Slika 3 prikazuje učinkovitiji način poboljšanja brzine odziva Internet sustava uporabom zastupnika. Na slici je prikazan Internet sustav u kojem zastupnici djeluju kao priručne memorije. Zastupnici (engl. *proxy*) su posebna računala koja čine zasebni razred računala na Internetu. Zastupnik posreduje između korisnika i poslužitelja tijekom dohvata objekta i sprema dohvaćene objekte u svoju lokalnu priručnu memoriju. Priručna memorija zastupnika sastoji se od radne memorije i prostora na čvrstom disku. Na čvrsti disk spremaju se svi dohvaćeni objekti, a radna memorija ubrzava pristup disku spremanjem često traženih objekata.

Jedan zastupnik poslužuje više korisnika čime je, u odnosu na priručne memorije Internet pretraživača, povećana vjerojatnost višestrukog dohvata istog objekta. Rad sa zastupnikom započinje tako da korisnik u postavke svog pretraživačkog programa unese adresu zastupnika. Internet pretraživač potom sve korisničke zahtjeve umjesto poslužiteljima šalje zastupniku. Nakon što primi zahtjev od korisnika, zastupnik provjerava da li u svojoj priručnoj memoriji ima valjanu presliku traženog objekta. Ako u priručnoj memoriji nije pronađena valjana preslika traženog objekta, onda zastupnik otvara vezu s poslužiteljem i prenosi mu zahtjev. Traženi objekt potom se prenosi od poslužitelja do zastupnika. Nakon što je objekt spremio u lokalnu memoriju, zastupnik objekt šalje korisniku koji ga je zatražio.

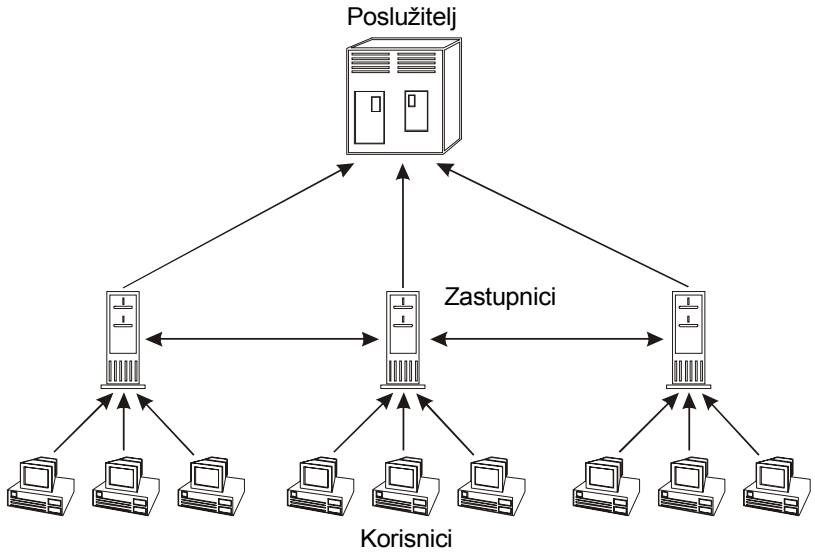
Ako korisnik zatraži objekt koji je prije toga već bio tražen, onda zastupnik provjerom sadržaja priručne memorije utvrđuje da ona sadrži presliku traženog objekta. Zastupnik potom provjerava da li su ispunjeni uvjeti jednoznačnosti podataka, odnosno da li je preslika objekta valjana. Ako je preslika valjana, onda zastupnik na korisnikov zahtjev odgovara slanjem objekta koji se već nalazi u njegovoj priručnoj memoriji. Budući da je kašnjenje između zastupnika i korisnika manje od kašnjenja između poslužitelja i korisnika, posluživanjem objekata iz lokalne priručne memorije zastupnika skraćuje se vrijeme odziva.

U analizi rada zastupnika koriste se termini pogotka i promašaja. Ako je nakon primanja korisnikovog zahtjeva u priručnoj memoriji zastupnika pronađena valjana preslika traženog objekta, onda se govori o pogotku. Ako lokalno nije pronađena valjana preslika objekta, onda se kaže da je došlo do promašaja. Pogotci u lokalnu memoriju zastupnika smanjuju opterećenje Internet sustava na dva načina. Pogotcima se smanjuje opterećenje poslužiteljskog računala i rasterećuje se dio Internet veza od zastupnika do poslužitelja. Opterećenje poslužitelja je manje jer poslužitelj prima manje zahtjeva, a Internet veze između zastupnika i poslužitelja su rasterećene jer prenose manju količinu podataka. Smanjenje opterećenja Internet sustava nije izravno vidljivo korisnicima, ali korisnici primjećuju znatno manje kašnjenje dohvata Internet stranica. Razna istraživanja [12,13,18] pokazuju da zastupnici i zastupnički sustavi znatno smanjuju kašnjenje dohvata WWW objekata. Osim kašnjenja, zastupnički sustavi smanjuju i mrežni promet između lokalne mreže i Interneta. U početku razvoja zastupničkih sustava, smanjenje Internet prometa bilo je temeljni motiv postavljana zastupnika u raznim organizacijama.

Učinkovitost zastupnika i priručnih memorija u Internet sustavima ograničena je s tri uvjeta. Prvi uvjet traži da je veza korisničkog i zastupničkog računala brža od veze s poslužiteljskim računalom. Drugi uvjet traži da zastupničko računalo ima brži odziv od poslužitelja, a treći uvjet zahtijeva lokalnost podataka. Lokalnost podataka u Internet sustavima se izražava kao udio pogodaka u priručnu memoriju. Udio pogodaka (engl. *hit rate*) je omjer broja zahtjeva koji se poslužuju izravno iz priručne memorije i ukupnog broja zahtjeva koje priručna memorija primi. Veći udio pogodaka u priručnu memoriju povećava učinkovitost priručne memorije i skraćuje prosječno kašnjenje dohvata objekata. Prva dva uvjeta učinkovitosti zastupnika moguće je zadovoljiti odabirom brzih zastupničkih računala i postavljanjem brzih veza između zastupnika i korisnika. Uvjet lokalnosti podataka, međutim, predstavlja značajan problem. Lokalnost podataka izravno ovisi o svojstvima Internet prometa organizacije koja koristi zastupnički sustav. Ako organizacija ima mali broj korisnika, ako korisnici dohvaćaju različite stranice ili ako se objekti dinamički mijenjaju, onda zastupnik često dohvaća objekte od poslužitelja. U navedenim uvjetima stoga ne dolazi do ubrzanja rada s Internetom i ne smanjuje se mrežni promet. Budući da svi korisnički zahtjevi prolaze kroz zastupnički sustav, kašnjenje dohvata objekata za krajnje korisnike u tom slučaju čak je veće nego u sustavu bez zastupnika.

Mala lokalnost podataka značajan je problem Internet sustava. Dok je udio pogodaka u priručnu memoriju višeprocesorskih sustava u prosjeku veći od 99.99%, u Internet sustavima je udio pogodaka

manji od 40%. Stoga je učinkovitost priručnih memorija u Internet sustavima znatno manja nego u višeprocesorskim sustavima. Budući da je kašnjenje dohvata objekta od poslužitelja znatno veće od kašnjenja dohvata objekta od zastupnika, zastupnici smanjuju prosječno vrijeme dohvata objekata unatoč niskom udjelu pogodaka. Učinkovitost zastupnika moguće je poboljšati povećanjem broja korisnika koji pristupaju istom zastupniku. Povećanjem broja korisnika po zastupniku raste vjerojatnost višestrukog dohvata objekta od strane dva ili više različitih korisnika. Navedenim pristupom nije moguće riješiti problem neučinkovitosti Internet zastupnika, jer se povećanjem broja korisnika po zastupniku zagušuju zastupničko računalo i veze prema njemu.



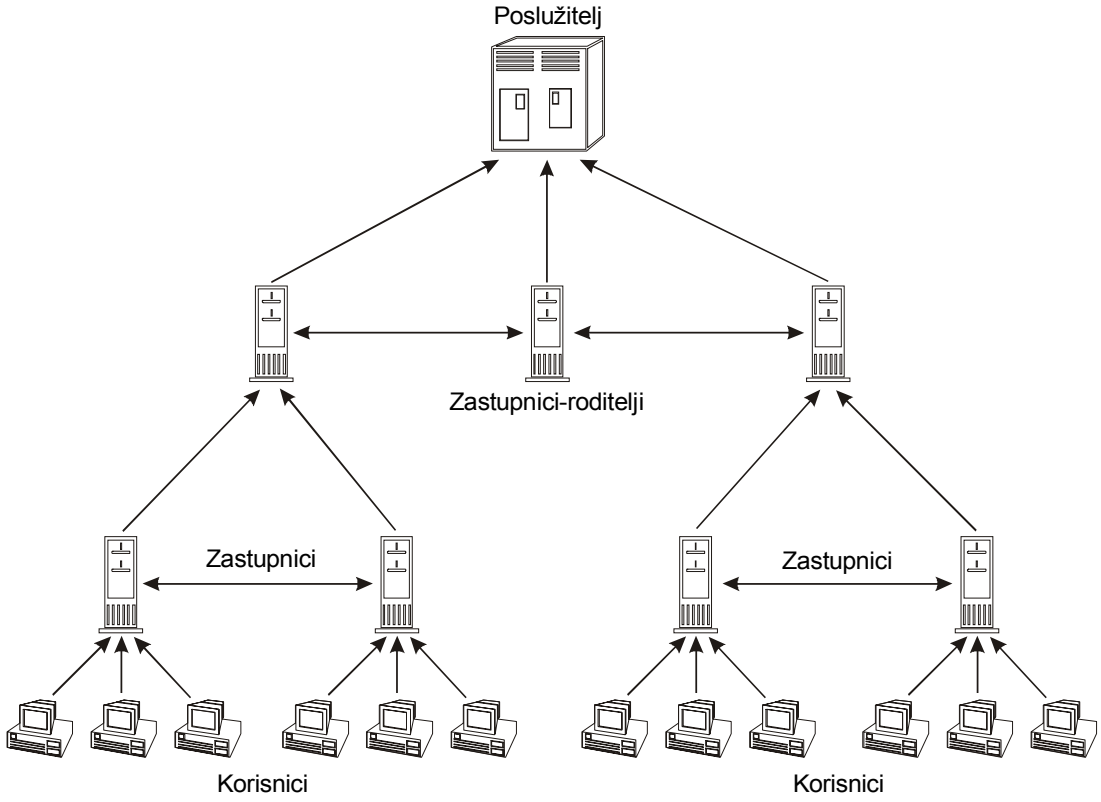
Slika 4: Model korisnik-povezani zastupnici-poslužitelj

Budući da povećanje broja korisnika zagušuje zastupnika, osmišljen je složeni sustav koji uklanja zagušenje. Zastupnici se međusobno povezuju čime se dobiva raspodijeljeni zastupnički sustav prikazan na slici 4. Zastupnici međusobno razmjenjuju informacije o sadržaju svojih lokalnih priručnih memorija i dohvaćaju objekte od ostalih zastupnika u sustavu. U raspodijeljenim zastupničkim sustavima javlja se termin glavnog zastupnika. Glavni zastupnik (engl. *master cache*) je zastupnik od kojeg korisnik početno traži objekt. Budući da je svakom zastupniku u sustavu pridijeljen skup korisnika koji mu šalju zahtjeve, svaki zastupnik može biti glavni zastupnik.

Raspodijeljeni zastupnički sustav prikazan na slici 4 djeluje kao cjelina. Nakon što primi zahtjev od korisnika, glavni zastupnik provjerava sadržaj svoje lokalne priručne memorije. Ako u lokalnoj priručnoj memoriji ne postoji valjana preslika traženog objekta, onda glavni zastupnik pokušava objekt dohvatiti od ostalih zastupnika u skupini. Ako nijedan susjedni zastupnik nema traženi objekt, onda glavni zastupnik dohvaća objekt od poslužitelja. Opisanim postupkom stvorena je raspodijeljena priručna memorija. Prednost raspodijeljene priručne memorije je da uz isti broj korisnika po zastupniku postiže veći udio pogodaka od izoliranih zastupnika, a ne povećava opterećenje pojedinih zastupničkih računala. Povećanje udjela pogodaka posljedica je svojstva da je ukupni broj korisnika u raspodijeljenoj priručnoj memoriji jednak zbroju korisnika po svakom zastupniku. Zbog većeg broja

korisnika koji koriste istu, raspodijeljenu priručnu memoriju raste vjerojatnost višestrukog traženja istog objekta.

Harvest [5-7] je prvi programski sustav koji je u sustave raspodijeljene priručne memorije na Internetu uveo međusobnu komunikaciju zastupnika. Sustav Harvest nastao je tijekom istraživačkog rada čiji cilj je bio stvaranje sustava indeksiranja i pohrane Internet dokumenata. Korisnicima sustava Harvest omogućeno je brzo pronalaženje i brzi dohvat dokumenata. U cilju ubrzanja dohvata dokumenata, unutar sustava Harvest razvijen je i zastupnički sustav. Iako je zastupnički sustav dodatna mogućnost Harvesta, tijekom vremena se pokazalo da je zastupnička sastavnica najznačajnije postignuće istraživanja vezanog uz Harvest.



Slika 5: Hijerarhijska struktura Harvest sustava

Osim međusobne komunikacije zastupnika, zastupnički sustav Harvesta prvi je uveo i hijerarhijsku strukturu zastupničkog sustava. U hijerarhijskom sustavu, svaki zastupnik razlikuje dvije vrste zastupnika s kojima surađuje: susjede (engl. *neighbor*) i roditelje (engl. *parent*). Susjedi su zastupnici koji se nalaze na istoj hijerarhijskoj razini s promatranim zastupnikom, a roditelji su zastupnici na višoj razini. Roditelji mogu imati svoje roditelje pa je dobivena hijerarhijska struktura čiji primjer je prikazan na slici 11. Prikazana hijerarhijska struktura ima dvije razine, ali sustav Harvest ne ograničava broj razina koje se mogu ostvariti u stvarnim sustavima. Broj razina ograničavaju praktični problemi izgradnje hijerarhijske strukture s velikim brojem razina.

Osnovne pretpostavke učinkovitosti Harvesta i ostalih raspodijeljenih zastupničkih sustava su znatno brži odziv na mreži kojom su povezana zastupnička računala od odziva na mreži prema

poslužiteljskom računalu i znatno brži odziv zastupničkih računala od odziva poslužiteljskog računala. Osim toga, učinkovitost raspodijeljenog zastupničkog sustava značajno ovisi i o svojstvima upravljačkog protokola. Upravljački protokol raspodijeljene priručne memorije definira komunikacijske postupke kojima zastupnici u sustavu razmjenjuju podatke o stanju svojih lokalnih priručnih memorija. Upravljačkim protokolom definirani su i uvjeti dohvata objekata od susjednih zastupnika. Budući da raspodijeljeni zastupnički sustav u proces dohvata objekata unosi dodatno kašnjenje potrebno za određivanje da li susjedni zastupnici imaju traženi objekt, svojstva upravljačkog protokola znatno utječu na kašnjenje dohvata Internet objekata. Upravljački protokoli raspodijeljenih priručnih memorija stoga su predmet brojnih znanstvenih istraživanja.

Osnovni nedostaci sustava Harvest su njegova osnovna namjena i zastarjelost. Autori nisu predvidjeli uporabu Harvesta isključivo za stvaranje raspodijeljene priručne memorije pa je njegova uporaba ograničena. Osim toga, sustav Harvest zastario je s obzirom na svoju arhitekturu, upravljanje datotečnim sustavom i podršku novim protokolima kao što je inačica broj 1.1 protokola HTTP (HyperText Transfer Protocol). Zbog navedenih nedostataka Harvest se više ne koristi u tržišnim proivodima i znanstvenim istraživanjima. Na osnovi Harvesta razvijeni su mnogi zastupnički sustavi od kojih je najpoznatiji sustav Squid. Squid je uporabljen kao osnova ostvarenja direktorijskog upravljačkog protokola u praktičnom dijelu ovog rada. Upravljački protokol Squid stoga se podrobno opisuje u odjeljku 3.4, a zastupnički sustav Squid je opisan u odjeljku 4.1.

Hijerarhijska struktura uvedena sustavom Harvest ne doprinosi značajno smanjenju kašnjenja dohvata objekata. Osnovna zamisao hijerarhijske strukture je da zastupnici na nižim razinama u svojim priručnim memorijama drže često dohvaćane objekte, a da njihovi roditelji spremaju rjeđe tražene objekte. Harvestov upravljački protokol pretpostavlja da više razine zastupničkog sustava povećavaju udio pogodaka što povećava učinkovitost raspodijeljene priručne memorije. U tu svrhu potrebno je da roditeljski zastupnici imaju višestruko veće memorijske kapacitete od zastupnika na nižim razinama. Veći memorijski kapaciteti potrebni su, jer roditeljski zastupnici trebaju spremiti više objekata od zastupnika na nižim razinama. Navedeni zahtjev je nedostatak hijerarhijske strukture, jer ga nije isplativo ostvariti u svim primjenama. U sustavu koji u cijelosti ostvaruje hijerarhijsku strukturu, zastupnici na najvišoj razini imaju memorijske kapacitete dovoljne za pohranu svih objekata na Internetu.

Značajan nedostatak hijerarhijske strukture je i dodatno kašnjenje koje ona unosi. U slučaju promašaja u lokalnim priručnim memorijama svih zastupnika jedne razine, glavni zastupnik objekt ne dohvaća izravno od poslužitelja, nego zahtjev šalje roditeljskom zastupniku. Ako roditeljski zastupnik i njemu susjedni zastupnici nemaju traženi objekt, onda ga roditeljski zastupnik dohvaća od poslužitelja ili od svog roditeljskog zastupnika. Uz pretpostavku sustava Harvest da roditeljski zastupnici imaju rjeđe tražene objekte, udio pogodaka u njihove lokalne memorije je niži nego za podređene zastupnike. Prema tome, hijerarhijskom strukturom unosi se dodatno kašnjenje koje nije nadoknađeno većim udjelom pogodaka. Zbog upitne koristi hijerarhijske strukture, u ovom radu se promatra ponašanje

upravljačkih protokola samo unutar jedne razine. Promatrana razina može biti samostalna ili može činiti dio hijerarhijske strukture.

Osim problema lokalnosti podataka, priručne memorije na Internetu pokušavaju riješiti i problem jednoznačnosti podataka. Ako objekt na poslužitelju i preslika u priručnoj memoriji zastupnika ili Internet pretraživača nisu jednoznačni, onda korisnici dobijaju zastarjele objekte. Problem jednoznačnosti nije riješen u HTTP i FTP (File Transfer Protocol) protokolima [3,4] pa je potrebno riješiti ga u protokolima više razine. U svrhu rješavanja problema jednoznačnosti, inačica broj 1.1 HTTP protokola [4] definira vremenska ograničenja valjanosti objekata. Pritom se odabir pravilnih vrijednosti većim dijelom zasniva na poopćenim pretpostavkama koje ne mogu zadovoljiti sve potrebe održavanja jednoznačnosti. Definirane su četiri inačice vremenskog određivanja valjanosti objekata.

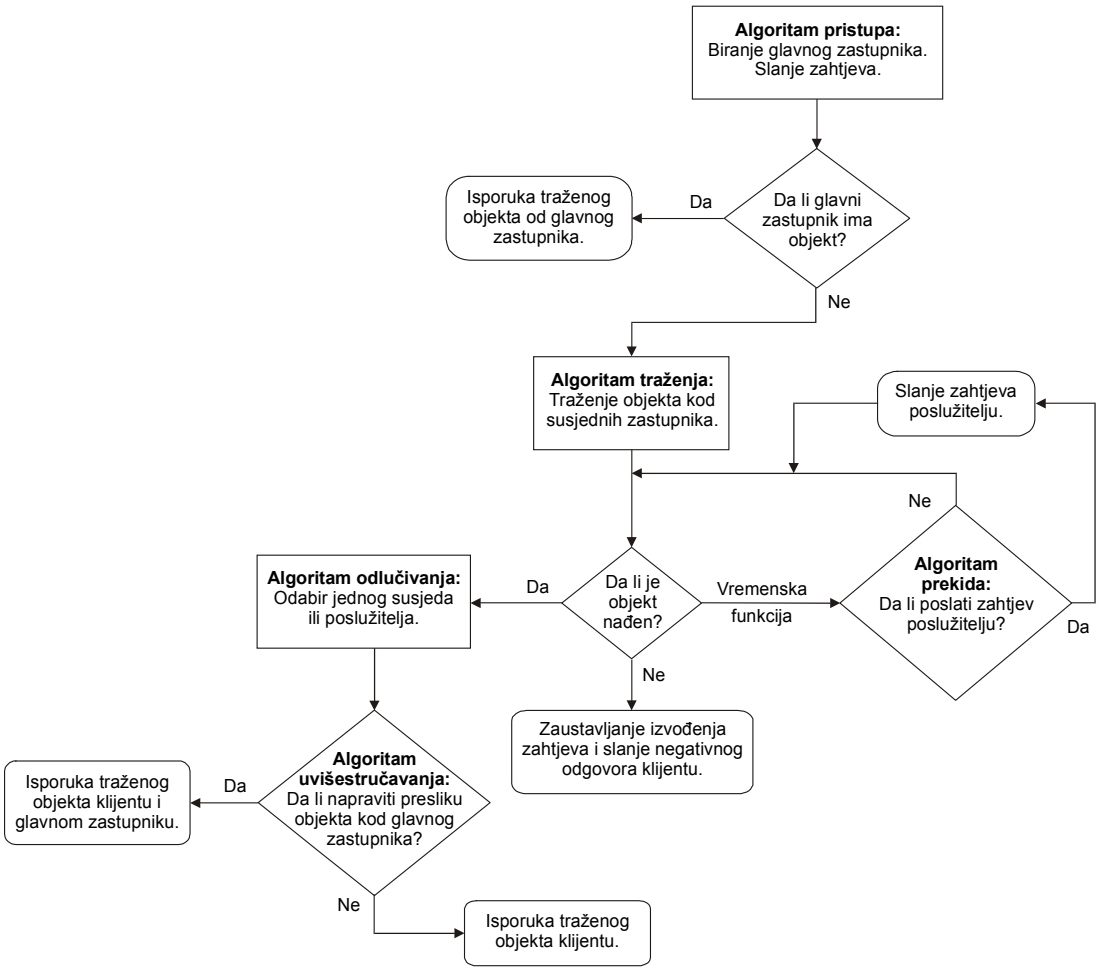
U prvoj inačici, svim Internet objektima pridjeljuje se podrazumijevano vrijeme valjanosti. Budući da jedinstveno vrijeme valjanosti ne odgovara dinamici promjena svih objekata, njegovom uporabom nije dobiveno zadovoljavajuće rješenje problema jednoznačnosti. Druga inačica vremenskog određivanja valjanosti objekata je da poslužitelj svakom objektu preko odgovarajućih HTTP polja pridijeli zasebno vrijeme isteka valjanosti (engl. *expires*). Na temelju zadanog vremena isteka valjanosti, Internet pretraživač i zastupnik mogu odrediti da li je objekt koji imaju u priručnoj memoriji valjan u određenom trenutku. Budući da je vrijeme isteka valjanosti teško procijeniti i da od korisnika poslužitelja zahtijeva postavljanje pravilnih vrijednosti, navedeni se pristup rijetko koristi. Treća inačica vremenskog održavanja jednoznačnosti podrazumijeva da poslužitelj svakom objektu pridjeljuje polje s vremenom zadnje promjene (engl. *last-modified*). Iako je informacija o zadnjoj promjeni pouzdana i lako se stvara, navedenim pristupom se procjena točnog vremena valjanosti samo prebacuje s poslužitelja na korisnika. Zbog toga treća inačica ne predstavlja poboljšanje u odnosu na prvu inačicu vremenskog određivanja valjanosti objekata. Četvrta inačica vremenskog održavanja jednoznačnosti je uvjetni upit korisnika ili zastupnika. Uvjetnim upitom se od poslužitelja traži objekt samo ako se objekt promijenio od trenutka prethodnog dohvata (engl. *if-modified-since*). Opisana inačica predstavlja najpouzdaniju metodu održavanja jednoznačnosti, ali je njezin značajan nedostatak potreba za slanjem upita poslužitelju. Navedenim upitom opterećuju se mreža i poslužitelj, te se stvara dodatno kašnjenje ako u priručnoj memoriji Internet pretraživala ili zastupnika postoji valjana preslika objekta.

Održavanje jednoznačnosti značajno utječe na priručne memorije na Internetu, jer se održavanjem jednoznačnosti povećava kašnjenje dohvata Internet objekata. Osim toga, potreba za dohvatom valjanih preslika smanjuje udio pogodaka u priručne memorije što smanjuje njihovu učinkovitost. Jednoznačnost objekata na Internetu složen je problem za koji još uvijek ne postoji zadovoljavajuće rješenje.

2.2. Model raspodijeljene priručne memorije

U svrhu preglednije usporedbe pojedinih upravljačkih protokola raspodijeljene priručne memorije, u radu se koristi model raščlambe i usporedbe svojstava raspodijeljenih sustava. Model prikazan na slici 6 dijeli raspodijeljenu priručnu memoriju, odnosno njezin upravljački protokol, na slijedeće nezavisne algoritme:

1. algoritam pristupa
2. algoritam traženja
3. algoritam prekida
4. algoritam odlučivanja
5. algoritam uvišestručavanja



Slika 6: Algoritmi upravljačkog protokola raspodijeljene priručne memorije

Slika 6 prikazuje kako se nabrojani algoritmi međusobno povezuju u veću cjelinu, odnosno u složeni algoritam upravljačkog protokola. Osim algoritma pristupa, koji izvodi korisnik ili zastupnik, sve ostale algoritme izvode isključivo zastupnici. Raščlambom upravljačkog protokola na pet osnovnih algoritama dobija se mehanizam za analizu i lakšu usporedbu pojedinih protokola. Za razumijevanje odabranog modela potrebno je razumjeti svrhu i djelovanje pojedinih algoritama koji čine model.

Algoritam pristupa određuje koji zastupnik se bira kao glavni zastupnik. Algoritam se počinje izvoditi nakon što korisničko računalo pokrene dohvat novog objekta. Jedan od načina ostvarenja algoritma pristupa je jednostavan i sastoji se od statičkog određivanja glavnog zastupnika u postavkama Internet pretraživača. Prednosti navedenog pristupa su male promjene Internet pretraživača i jednostavnost, a glavni nedostatak je neprilagodljivost opterećenju zastupničkog sustava. Drugi pristup ostvarenju algoritma je kružni (engl. *round robin*) algoritam odabira glavnog zastupnika na korisniku. Prednost je kružnog algoritma raspodjela opterećenja zastupnika. Osnovni nedostatak je potreba da svi korisnici znaju za sve zastupnike u sustavu što ograničava dinamičko dodavanje zastupnika u sustav raspodijeljene priručne memorije. Daljnji nedostatak je ograničena korisnost kružne raspodjele opterećenja. Treća mogućnost izvedbe algoritma pristupa je da algoritam izvode zastupnici i da zastupnički sustav brine o opterećenju pojedinih zastupnika. Nakon što korisnik pristupi zastupničkom sustavu, navedeni algoritam preusmjerava korisnika na najbližeg i najmanje opterećenog zastupnika. Prednost navedenog pristupa je visoko učinkovita raspodjela opterećenja, no značajan nedostatak je složena izvedba algoritma na zastupnicima. Osim toga, potrebne su i značajne promjene Internet pretraživača radi ostvarivanja mehanizma preusmjeravanja glavnog zastupnika.

Drugi je algoritam u slijedu izvođenja algoritam traženja kojeg izvode zastupnici. Algoritam traženja u raspodijeljenom zastupničkom sustavu pronalazi zastupnike koji imaju traženi objekt. Najveće razlike između pojedinih zastupničkih sustava očituju se u algoritmu traženja. Jedan od načina ostvarenja algoritma traženja je slanje upita svim susjednim zastupnicima. Nedostatak navedenog pristupa je veliko opterećenje koje se stvara brojnim upitima. Navedeni nedostatak moguće je ukloniti sužavanjem skupa zastupnika za koje se sumnja da imaju traženi objekt. Algoritam traženja moguće je izvoditi slanjem *multicast* upita koji je dovoljno poslati jednom, a primaju ga svi susjedni zastupnici. Iako slanje *multicast* upita smanjuje mrežni promet, njihovim slanjem se ne smanjuje opterećenje zastupničkih računala. Svi susjedni zastupnici primaju *multicast* upit, obrađuju ga i odgovaraju na njega. Treći način ostvarenja algoritma traženja nije zasnovan na slanju upita. Skup URL-ova (Uniform Resource Locator) dijeli se na podskupove koji se pridijeljuju pojedinim zastupnicima. Neovisno o tom koje korisničko računalo šalje zahtjev i neovisno o opterećenju sustava, određeni zastupnik uvijek je odgovoran za sve URL-ove iz podskupa koji mu je pridijeljen. Navedeni algoritam jednostavan je i brz, ali donosi značajne nedostatke u pogledu neučinkovite raspodjele opterećenja i povećanog vremena kašnjenja između glavnog zastupnika i korisnika.

Algoritam prekida koristi se u sustavima koji koriste algoritme traženja zasnovane na temelju slanja upita susjednim zastupnicima. Tijekom slanja upita, postoji mogućnost da se odgovori susjednih zastupnika zagube ili da značajno kasne. U tom slučaju, potreban je algoritam prekida koji prekida započeto pretraživanje i glavnog zastupnika upućuje da dohvati traženi objekt izravno s poslužitelja. Budući da tijekom slanja *multicast* upita nije poznat očekivani broj odgovora, algoritam prekida čini značajnu sastavnicu zastupničkih sustava koji koriste *multicast* mehanizam. Za mehanizme slanja upita koji nisu zasnovani na *multicast* mehanizmu, moguće je algoritam prekida ostvariti pomoću unaprijed određenog vremena čekanja. Međutim, za ostvarenje kvalitetnog algoritma zasnovanog na

multicast mehanizmu potrebno je da algoritam prekida uzima u obzir razne parametre i da je dinamičan. Problemi pravilnog odabira vremena prekida čine razvoj optimalnog algoritma prekida složenim.

Algoritam odlučivanja izvodi se nakon što je utvrđeno da više susjednih zastupnika ima traženi objekt. Algoritam odlučivanja odlučuje od kojeg susjednog zastupnika glavni zastupnik dohvaća traženi objekt. Navedeni algoritam značajno utječe na raspoređivanje opterećenja između zastupnika. Osim algoritma raspodjele opterećenja, u algoritam odlučivanja moguće je ugraditi i preusmjeravanje korisnika. Glavni zastupnik šalje korisničkom računalu zahtjev za preusmjeravanjem nakon što utvrdi koji susjedni zastupnik sadrži traženi objekt. Korisničko računalo potom šalje novi zahtjev odabranom zastupniku. Prednost opisanog pristupa je uklanjanje dodatnog prenošenja objekata između dva zastupnika. Nedostatak preusmjeravanja su značajne promjene koje je potrebno ostvariti na korisničkom pretraživaču. Osim toga, mehanizmom preusmjeravanja ne osigurava se učinkovita raspodjela opterećenja.

Konačno, algoritam uvišestručavanja određuje da li glavni zastupnik, nakon dohvata objekta za korisnika, lokalno sprema presliku traženog objekta. Ako se objekt ne spremi u lokalnu memoriju glavnog zastupnika, onda se smanjuje zalihost lokalnih memorija zastupnika. Zalihost podrazumijeva pojavu više preslika istog objekta u raspodijeljenom zastupničkom sustavu. Smanjivanjem zalihosti povećava se iskorištenost memorijskog prostora sustava raspodijeljene priručne memorije. Veća iskorištenost memorije povećava udio pogodaka u raspodijeljenu priručnu memoriju i učinkovitost raspodijeljenog zastupničkog sustava. Značajan nedostatak navedenog pristupa je da se dohvaćeni objekti ne spremaju u lokalnu memoriju glavnog zastupnika. U slučaju višestrukih zahtjeva za istim objektom zahtjevi se ne poslužuju iz lokalne memorije glavnog zastupnika, nego se korisnici svaki put preusmjeravaju na susjednog zastupnika. Budući da je kašnjenje između susjednog zastupnika i korisnika veće od kašnjenja između glavnog zastupnika i korisnika, preusmjeravanje povećava kašnjenje dohvata objekta.

2.3. Protokol ICP

Za točno opisivanje pojedinih upravljačkih protokola raspodijeljene priručne memorije, prikladno je koristiti standardni, zajednički komunikacijski protokol. U tu svrhu najprikladniji je protokol ICP (Internet Cache Protocol) koji je standardiziran i definiran RFC (Request For Comments) dokumentima [9,10]. Iako ne definira sve poruke potrebne za ostvarivanje svih upravljačkih protokola opisanih u ovom radu, ICP protokol pruža osnovni komunikacijski mehanizam. Osnovni mehanizam jednostavno se proširuje dodatnim porukama potrebnim za opis pojedinog upravljačkog protokola.

Protokol ICP definira međusobnu komunikaciju zastupnika u Internet sustavima. ICP je razvijen na temelju protokola upotrijebljenih u sustavima Harvest [5-7] i Squid [8]. Početna istraživanja na protokolu ostvario je Peter Danzig sa University of Southern California. Protokol je potpuno definiran, predložen i prihvaćen od Internet zajednice tek nakon stvaranja sustava Squid u laboratoriju

NLANR (National Laboratory for Applied Network Research). NLANR je državna organizacija SAD-a koju financira NSF (National Science Foundation). ICP je relativno jednostavan protokol s malim opterećenjem mrežnih kapaciteta i računala po poruci. Iako je protokol jednostavan i brz, njegov značajan nedostatak leži u velikoj količini poruka koje se razmjenjuju pri njegovoj uporabi. Iako velika količina poruka nije vezana uz protokol za međusobnu komunikaciju Internet zastupnika, autori ICP protokola pogriješili su vežući komunikacijski protokol uz upravljački protokol raspodijeljenih priručnih memorija. U RFC dokumentu [9] autori daju definiciju ICP protokola i opisuju komunikacijske mehanizme, a u dokumentu [10] autori podrobnije opisuju primjenu ICP protokola na način koji oni smatraju prikladnim i koji odgovara upravljačkom protokolu ugrađenom u Squid. Posljedica navedene pogreške autora je miješanje ICP protokola sa Squidovim protokolom za upravljanje raspodijeljenim zastupničkim sustavom. Navedeni protokoli u znatnom dijelu nisu međusobno ovisni. Ovisnost između njih postoji samo zbog toga što ICP protokol osigurava komunikacijske mehanizme upravljačkom protokolu raspodijeljene priručne memorije. U ovom radu stoga se strogo odvojeno promatraju komunikacijski dio ICP standarda, koji se naziva ICP protokolom, i upravljački protokol raspodijeljene priručne memorije, koji se u radu naziva Squid protokolom.

2.3.1. Namjena protokola

ICP protokol namijenjen je razmjeni poruka o sadržaju lokalnih memorija zastupničkih sustava. Poruke ICP protokola su jednostavne i kratke. ICP stoga postiže jednostavnost programskog ostvarenja i veliku brzinu komunikacije. ICP poruke prenose se UDP (User Datagram Protocol) protokolom [1,2] koji je nepouzdani prijenosni mehanizam pa Internet sustav ne jamči prijenos poruke do primatelja. UDP protokol prikladniji je za prijenos ICP poruka od pouzdanog TCP (Transmission Control Protocol) protokola [1,2]. Prednosti prijenosa UDP protokolom su znatno jednostavnija i brža komunikacija, manje poruke, te jednostavnije i brže programsko ostvarenje. TCP protokol u odnosu na UDP stvara dodatne nadzorne poruke potrebne za potvrdu prijema podataka, te za otvaranje i zatvaranje TCP veze. Osim toga, poruke TCP protokola su veće duljine jer sadrže dodatne informacije potrebne za ostvarivanje pouzdanog mehanizma prijenosa podataka. Primjerice, UDP zaglavlje ima duljinu 8 okteta, a minimalno TCP zaglavlje ima 20 okteta. Zbog nadzornih poruka i veće duljine zaglavlja, kašnjenje u prijenosu TCP poruka veće je od kašnjenja UDP poruka.

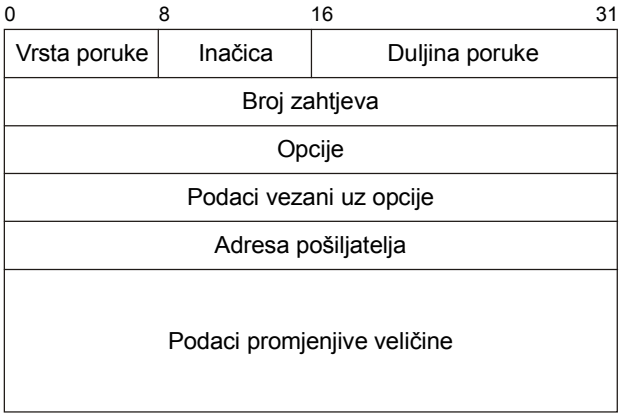
Manje kašnjenje prijenosa poruka značajna je prednost UDP protokola u zastupničkim sustavima. Radi postizanja manjeg kašnjenja, zastupnički sustav tijekom izvođenja algoritma traženja ograničava čekanje na odgovor susjednih zastupnika. Stoga dugotrajan postupak otvaranja i zatvaranja veze, te slanje potvrde primitka podataka koje provodi TCP protokol, smanjuju učinkovitost zastupničkog sustava.

Dodatni razlog za uporabu UDP protokola je činjenica da gubitak ICP poruke ne uzrokuje niti grešku u prijenosu traženog objekta korisniku niti neposluživanje korisničkog zahtjeva. Jedine negativne posljedice gubitka ICP poruke su manji udio pogodaka u raspodijeljenu priručnu memoriju i neznatno

povećanje kašnjenja dohvata objekata. Budući da je pouzdanost današnjih mreža dobra, odnosno poruke se rijetko gube, uporaba UDP protokola ne uzrokuje značajno smanjenje udjela pogodaka ni povećanje kašnjenja dohvata objekata.

2.3.2. ICP poruke

Svaka ICP poruka sastoji se od zaglavlja konstantne veličine i dijela s podacima promjenjive veličine. Slika 7 prikazuje strukturu ICP poruke. Na slici 7 svaki redak predstavlja 32 bita, a iznad prvog retka su označeni položaji značajnijih bitova. Veličina ICP zaglavlja je 20 okteta, što je istovremeno i najmanja veličina ICP poruke. Na poruku prikazanu na slici 7 prije slanja preko Interneta nadodaju se UDP i IP (Internet Protocol) zaglavlja pa je veličina IP poruke kojom se prenosi ICP poruka veća od 20 okteta.



Slika 7: Struktura ICP poruke

Pojedina polja u ICP poruci imaju slijedeća značenja:

Vrsta poruke – polje širine 8 bita koje određuje osnovno značenje poruke.

Inačica – polje širine 8 bita u kojem je zapisana inačica ICP protokola. U uporabi su dvije inačice protokola, broj 2 i broj 3. Pritom se u postojećim sustavima koristi samo inačica broj 2.

Duljina poruke – određuje ukupnu duljinu poruke u oktetima. Iako je širina polja 16 bita, maksimalna dopuštena duljina poruke iznosi 16384 okteta.

Broj zahtjeva – pomoću navedenog 32-bitnog broja razlikuju se pojedini zahtjevi. Broj zahtjeva stvara se tijekom slanja upita tako da ne dolazi do ponavljanja brojeva. U odgovoru na primljeni upit preuzima se broj zahtjeva naveden u upitu. Opisanim postupkom ostvaruje se jednostavno sparivanje upita i pripadnih odgovora.

Opcije – 32-bitno polje namijenjeno za opcijske zastavice kojima se ostvaruje proširivanje protokola.

Podaci vezani uz opcije – polje širine 32 bita koje služi za upis podataka koji nose dodatne informacije o odabranim opcijama.

Adresa pošiljatelja – ne koristi se u praksi, ali je zadržano radi sukladnosti. Polje je izvorno trebalo sadržavati IPv4 adresu pošiljatelja.

Podaci promjenjive veličine – u navedenom dijelu ICP poruke prenose se podaci koji nemaju stalnu duljinu i koji se ne pojavljuju u svim vrstama poruka. Jedna od najčešćih uporaba navedenog polja je prijenos URL-a u obliku C znakovnog niza.

U inačici broj 2 ICP protokola [9] definirano je ukupno 10 vrsta poruka. Vrsta poruke označava se brojčanom vrijednosti. Definicija protokola ostavlja veliki neiskorišteni raspon u slijedu brojeva za buduća proširenja. Prema definiciji protokola zauzet je brojčani raspon od 0 do 23. Budući da je za označavanje vrste poruke predviđeno osam bita, za proširenja skupa poruka na raspolaganju je velik raspon vrijednosti od 24 do 255. U tablici 1 dan je pregled i kratko objašnjenje postojećih vrsta poruka.

Vrsta poruke	Namjena
ICP_INVALID	Poruka *ICP_INVALID* nikad se ne šalje, već je pokazatelj greške u prijenosu poruke.
ICP_QUERY	Šalje se susjednim zastupnicima radi ispitivanja da li imaju traženi objekt. U dijelu poruke namijenjenom podacima promjenjive veličine upisana je IPv4 adresa izvornog tražitelja objekta tj. korisnika. Navedena adresa omogućava preusmjeravanje korisnika na zastupnika koji ima traženi objekt. Nakon adrese slijedi URL traženog objekta u obliku C znakovnog niza.
ICP_HIT	Poruka *ICP_HIT* je odgovor na upit glavnog zastupnika i potvrđuje postojanje traženog objekta u lokalnoj memoriji susjednog zastupnika.
ICP_MISS	*ICP_MISS* poruka šalje se kao odgovor na upit glavnog zastupnika. Navedenom porukom javlja se da traženi objekt ne postoji u lokalnoj memoriji susjednog zastupnika.
ICP_ERR	Poruka *ICP_ERR* šalje se kao odgovor na upit glavnog zastupnika. Namjena poruke je obavijest glavnom zastupniku da u poruci s upitom postoji greška.
ICP_SECHO	Poruka služi za simuliranje upita na poslužitelju traženog objekta. Šalje se na poslužiteljev *echo* priključak (engl. *port*) odakle se nepromijenjena vraća pošiljatelju. Ako se *ICP_SECHO* poruka vrati prije bilo kojeg drugog pozitivnog odgovora, onda zastupnik objekt dohvaća izravno od poslužitelja. U dio predviđen za dodatne podatke upisuje se URL objekta.
ICP_DECHO	Poruka *ICP_DECHO* slična je poruci *ICP_SECHO*, ali se ne šalje poslužitelju, već se šalje susjednom zastupniku koji ne podržava ICP protokol.
ICP_MISS_NOFETCH	Navedena poruka slična je poruci *ICP_MISS*. Temeljna razlika je da susjedni zastupnik slanjem *ICP_MISS_NOFETCH* poruke od glavnog zastupnika zahtijeva da ne pokušava dohvatiti traženi objekt od njega.
ICP_DENIED	*ICP_DENIED* poruku susjedni zastupnik šalje glavnom zastupniku ako glavni zastupnik u tom trenutku nema pravo pristupiti traženom objektu
ICP_HIT_OBJ	Navedena poruka istog je značenja kao i *ICP_HIT* poruka, ali ona prenosi traženi objekt. U dijelu poruke predviđenom za podatke promjenjive veličine upisani su URL objekta, veličina objekta i tijelo objekta. Poruka *ICP_HIT_OBJ* koristi se za ubrzanje dohvata malih objekata. Budući da se opisanim slanjem objekta izbjegava HTTP obrada objekta i da se koristi UDP protokol, standard [9] ne preporuča uporabu navedene poruke.

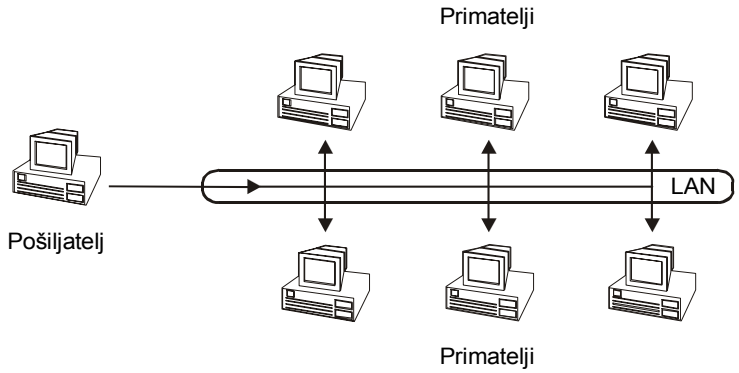
Tablica 1: Pregled standardnih vrsta ICP poruka

2.3.3. Načini slanja poruka

ICP poruke prenose se UDP protokolom. Osim protokola za prijenos podataka, tijekom analize pojedinih upravljačkih protokola promatra se i način slanja jedne poruke skupini primatelja. U raznim mrežnim primjenama poput emitiranja slikovnih i zvučnih zapisa, javlja se potreba za slanjem identičnog sadržaja skupini primatelja. Slanje identičnog sadržaja skupini primatelja javlja se i u raspodijeljenim zastupničkim sustavima tijekom slanja upita susjednim zastupnicima. U svrhu smanjenja mrežnog prometa, moguće je poslati samo jednu poruku koju će dobiti svi primatelji iz određenog skupa. Navedena potreba uzrok je postojanja nekoliko načina slanja poruka s obzirom na broj primatelja. Tri osnovna načina slanja poruka u Internet sustavima su *unicast*, *broadcast* i *multicast*.

Slika 8: *Unicast* slanje poruka

Unicast način slanja je uobičajeni način slanja poruke jednom primatelju prikazan na slici 8. Navedeni način slanja najčešće se koristi, jer u većini primjena Interneta u komunikaciji sudjeluju samo dva računala. Prednost *unicast* načina slanja poruka je mogućnost uporabe u sprezi s protokolima za pouzdani i nepouzdani prijenos podataka. Nedostatak navedenog načina slanja poruka je slanje zasebne poruke svakom primatelju iako sve poruke imaju identični sadržaj. Zbog toga dolazi do nepotrebne zalihosti informacija koje se šalju mrežom.

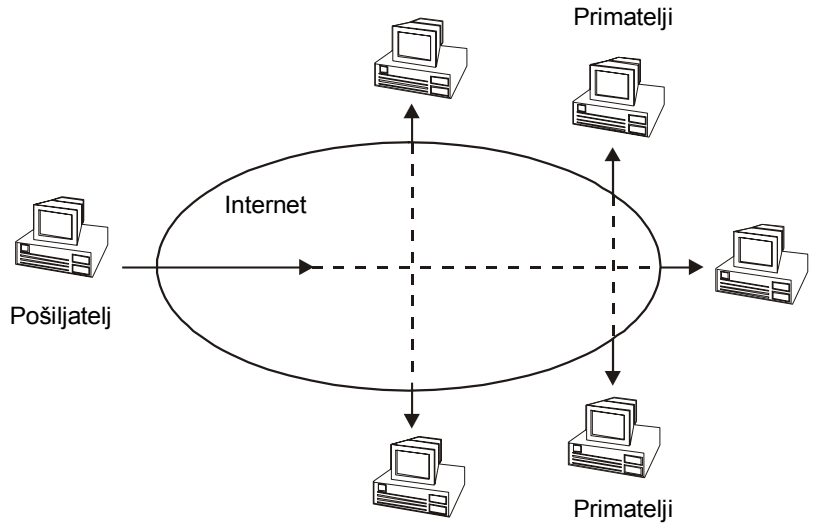


Slika 9: *Broadcast* slanje poruka

Nepotrebno opterećivanje mreže višestrukim slanjem istih podataka moguće je ukloniti nekim od načina slanja jedne poruke prema više primatelja. Temeljni način slanja iste poruke skupini primatelja naziva se *broadcast*. Navedeni način slanja poruka podržan je u jezgrenom dijelu TCP/IP paketa protokola i podrazumijeva slanje jedne poruke svim računalima koja se nalaze na jednoj podmreži. Slika 9 prikazuje opisani način slanja poruka.

TCP/IP standard određuje da *broadcast* poruku primaju sva računala koja se nalaze na adresiranoj podmreži. Pritom podmreža u većini slučajeva odgovara lokalnoj mreži (LAN). Ne postoji preduvjet

na položaj pošiljatelja tako da se pošiljatelj može nalaziti ili na istoj podmreži ili na ostatku Interneta. Uporaba *broadcast* načina slanja prikladna je u primjenama kod kojih se identične informacije šalju svim računalima na jednoj lokalnoj mreži. Primjer primjene *broadcast* načina slanja je prozivka računala na lokalnoj mreži. Nedostaci opisanog načina slanja poruka su grubo određivanje skupa primatelja poruka i nemogućnost uporabe u sprezi s protokolima za pouzdani prijenos, poput TCP protokola. Manje značajan problem *broadcast* načina slanja poruka nastaje iz potrebe da *broadcast* mehanizam podržava i fizički sloj mreže, poput *Etherneta*. Ako fizički sloj mreže ne podržava *broadcast* način slanja poruka, poput ATM-a (Asynchrnous Transfer Mode), onda TCP/IP protokol, odnosno usmjernik simulira slanje *broadcast* poruke. U stvarnim se sustavima simulacija ostvaruje tako da usmjernik svakom primatelju pošalje zasebnu *unicast* poruku. Opisanom simulacijom gube se prednosti i smisao uporabe *broadcast* načina slanja poruka. Navedene prednosti i nedostaci ograničavaju *broadcast* način slanja poruka na lokalne mreže i uporabu u sprezi s UDP protokolom.



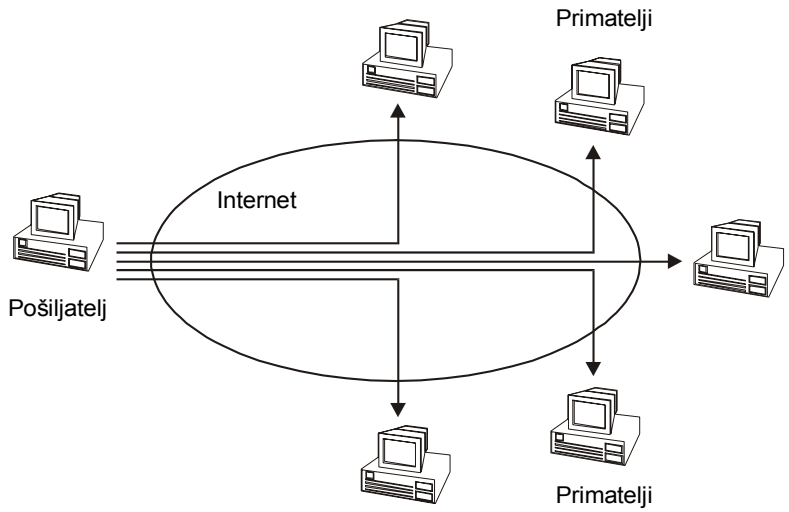
Slika 10: *Multicast* slanje poruka

Multicast način slanja poruka, prikazan na slici 10, ispravlja jedan od temeljnih nedostataka *broadcast* mehanizma. Umjesto da je skup primatelja ograničen na računala spojena na jednu podmrežu, skup primatelja *multicast* poruke čine računala koja se nalaze bilo gdje na Internetu. Za rad *multicast* mehanizma potrebno je svako primateljsko računalo prijaviti kao člana određene *multicast* skupine. Pošiljatelj šalje jednu poruku koju *multicast* poslužitelji potom prenose do ciljnih računala. Ako se na istoj podmreži nalazi samo jedan primatelj, *multicast* poslužitelj primatelju dostavlja poruku pomoću *unicast* mehanizma. Nalazi li se na jednoj podmreži više primatelja, *multicast* poslužitelj pokušava upotrijebiti *multicast* poruku ako je lokalna mreža podržava.

Iako opisani način slanja poruka nudi mehanizam slanja identične poruke skupini primatelja, jedan nedostatak značajno ograničava uporabljivost *multicast* mehanizma. Za ostvarenje *multicast* načinja slanja poruka nužni su *multicast* poslužitelji. *Multicast* poslužitelji vode popise članova pojedinih *multicast* grupa, te usmjeravaju i uvišestručavaju *multicast* pakete na temelju IP adresa članova grupe. U svrhu potpunog djelovanja *multicast* sustava, potrebno je na svakom Internet usmjerniku pokrenuti

multicast poslužitelj. Budući da ne spadaju u temeljni dio TCP/IP paketa protokola, *multicast* poslužitelji su vrlo rijetki na današnjem Internetu. Prema tome, *multicast* način slanja nije moguće uporabiti u potrebnom opsegu u današnjim sustavima. Osim opisanog nedostatka, *multicast* način slanja nije moguće koristiti u sprezi s TCP protokolom. Daljnji manji nedostatak *multicast* protokola odnosi se na zadnji stupanj dostave *multicast* poruka krajnjim računalima koja se nalaze na istoj podmreži. Budući da većina lokalnih mreža ne nudi jednostavnu podršku za *multicast*, u zadnjem stupnju slanja koristi se *unicast* mehanizam dostave poruka primateljima. *Unicast* mehanizam koristi se neovisno o broju primatelja koji se nalaze na istoj lokalnoj mreži.

Slanje *ICP_QUERY* upita pomoću *multicast* mehanizma značajno smanjuje mrežnu komunikaciju u raspodijeljenim sustavima priručne memorije. Mnogi istraživači područja raspodijeljenih zastupničkih sustava stoga u svojim radovima [11,15,16,18,20,21] preporučaju ili podrazumijevaju uporabu *multicast* načina slanja poruka. Pritom istraživači navode da su svjesni nepostojanja podrške za *multicast* mehanizam. Kao zamjenu za *multicast* mehanizam autori navode mehanizam *logičkog multicasta*.



Slika 11: Logički *multicast*

Logički *multicast* u osnovi je simulacija *multicast* načina slanja poruka koju izvodi pošiljatelj. *Multicast* mehanizam simulira se slanjem *unicast* poruka svakom primatelju poruke. Navedeni način slanja poruka u osnovi odgovara *unicast* slanju pa su broj poruka i mrežno opterećenje jednaki kao i kod *unicast* načina slanja poruka. Svakom primatelju pošiljatelj šalje zasebnu poruku s identičnim sadržajem, što je prikazano na slici 11. Navedena slika prikazuje primjer slanja identične poruke logičkim *multicast* mehanizmom. U prikazanom primjeru pošiljatelj šalje pet identičnih poruka na pet različitih adresa. Usporedbom navedenog broja poruka sa jednom *multicast* porukom prikazanom na slici 10, zaključuje se da logički *multicast* stvara višestruko veći promet u odnosu na *multicast* način slanja poruka. Navedeni promet istovjetan je prometu koji stvara *unicast* mehanizam.

Određeni problemi u analizi svojstava i razumijevanju pojedinih upravljačkih protokola raspodijeljene priručne memorije nastaju ako autori članaka izričito ne navedu da razmatraju logički *multicast* kao

način slanja upita. U krajnjim slučajevima, autori ponekad zanemaruju posljedice neraspoloživosti pravog *multicast* mehanizma i analizu svojstava upravljačkog protokola izvode na temelju broja poruka koje stvara *multicast* način slanja poruka. Budući da logički *multicast* način slanja poruka simuliran *unicast* mehanizmom stvara višestruko veći broj poruka tijekom slanja upita, analitički rezultati dobiveni na opisani način ne odgovaraju stvarnim sustavima.

2.4. Upravljački protokol Squid

Zastupnički sustav Squid [8] nasljednik je sustava Harvest i prvi predstavnik nove generacije zastupničkih sustava. Harvest je složeni sustav kojemu je uloga zastupnika samo jedna od namjena, a naglasak sustava je na indeksiranju i pohrani dokumenata. Za razliku od Harvesta, Squid je specijalizirani zastupnički sustav. Sustav Squid je od Harvesta gotovo u cijelosti preuzeo protokol za upravljanje raspodijeljenom priručnom memorijom, a preuzeo je i mogućnost stvaranja zastupničke hijerarhije. Sa stajališta upravljačkog protokola raspodijeljene priručne memorije, razlike između navedenih sustava svode se na manja poboljšanja uvedena u Squidu i standardizaciju komunikacijskog i upravljačkog protokola u obliku ICP standarda [9,10]. Temeljna su poboljšanja sustava Squid u odnosu na zastupničku sastavnicu sustava Harvest bolje rukovanje memorijom i datotečnim sustavom, te podrška za inačicu 1.1 HTTP protokola. Stoga se u daljnjem objašnjenju rada upravljačkog protokola ugrađenog u Squid i Harvest koristi opis algoritma ugrađenog u Squid. Osim toga, umjesto naziva Harvest/Squid protokol korišten je kraći naziv, Squid protokol.

2.4.1. Način rada protokola

Algoritam pristupa Squid protokola jednostavan je kao i u većini raspodijeljenih zastupničkih sustava. Internet pretraživač bira glavnog zastupnika na temelju postavki koje zadaje korisnik programa. Navedene postavke nije moguće dinamički mijenjati tijekom rada sa zastupničkim sustavom. Zbog navedenog ograničenja ne postoji mogućnost dinamičkog rasporeda opterećenja između zastupnika. Nakon što od korisnika primi zahtjev za objektom, glavni zastupnik provjerava da li se traženi objekt nalazi u lokalnoj priručnoj memoriji i da li je valjan. Ako u priručnoj memoriji ne postoji valjana preslika objekta, onda se izvodi prozivanje drugih zastupnika, odnosno, pokreće se algoritam traženja.

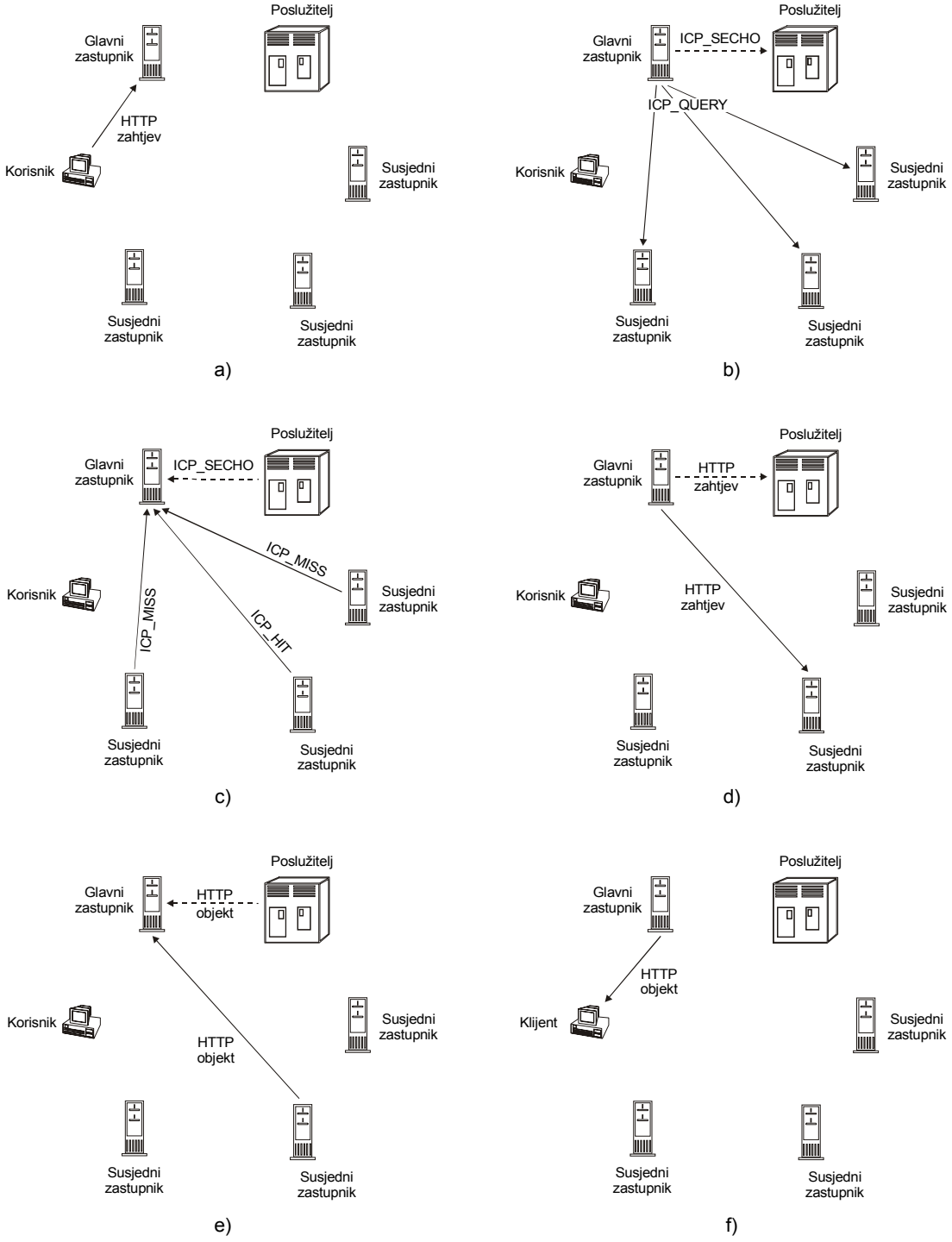
Algoritam traženja izvodi se slanjem upita (engl. *query*) svim susjednim zastupnicima iste hijerarhijske razine i roditeljskom zastupniku. Poseban oblik upita moguće je poslati i poslužitelju na kojem se izvorno nalazi traženi objekt. Upit poslužitelju šalje se na *echo* priključak i služi za provjeru da li je odziv poslužitelja brži od odziva zastupnika. Upiti zastupnicima i poslužitelju šalju se *unicast* paketima pomoću ICP protokola. Svi susjedni zastupnici koji su primili upit odgovaraju jednostavnom porukom koja govori da li pojedini susjedni zastupnik ima ili nema traženi objekt. Za razliku od susjednih zastupnika, poslužitelj preko svoje *echo* priključnice vraća poruku identičnu poslanoj. Nakon primanja svih odgovora, izvodi se algoritam odlučivanja.

Algoritam odlučivanja odabire onog susjednog zastupnika čiji je potvrdni odgovor prvi primljen na glavnom zastupniku. Ako se upit poslan poslužitelju vrati prije potvrdnog odgovora nekog od susjednih zastupnika, onda algoritam odlučivanja odabire poslužitelja. Opisanim algoritmom odabire se računalo koje je prvo odgovorilo na upit. Prema tome, očekuje se da odabrano računalo najbrže posluži traženi objekt. Navedeno očekivanje ne ostvaruje se uvijek, jer je moguće da računalo odgovor na upit pošalje znatno brže nego što je u mogućnosti poslati traženi objekt.

Algoritam prekida Squid protokola odlučuje pristupiti dohvatu objekta s poslužitelja ako su svi odgovori bili negativni ili ako je istekao unaprijed određeni vremenski interval. Konačno, algoritam uvišestručavanja izveden je tako da Squid uvijek sprema presliku objekta u lokalnu memoriju glavnog zastupnika. Nova preslika sprema se u lokalnu priručnu memoriju glavnog zastupnika bez obzira da li neki od susjednih zastupnika već ima presliku dohvaćenog objekta.

2.4.2. Komunikacijski slijed

Slika 12 prikazuje slijed poruka koje Squid protokol razmjenjuje tijekom dohvata jednog objekta. Navedena slika omogućava određivanje broja poruka koje se šalju tijekom postupka dohvata. Postoji više slijedova događaja tijekom dohvata objekta. Izvođenje određenog slijeda ovisi o okolnostima u kojima se zastupnički sustav i Internet nalaze. Pod okolnostima se podrazumijeva: da li glavni zastupnik ima traženi objekt, da li neki susjedni zastupnik ima traženi objekt i relativni odnos vremena odziva susjednih zastupnika u odnosu na vrijeme odziva poslužitelja. Na slici 12, punim crtama je označen slijed poruka u okolnostima u kojima glavni zastupnik nema traženi objekt, jedan od susjednih zastupnika ima traženi objekt, a kašnjenje komunikacije sa susjednim zastupnikom manje je od kašnjenja komunikacije s poslužiteljem. Isprekidanim crtama označena je mogućnost slanja upita poslužitelju i pripadni poslužiteljev odgovor. Osim toga, isprekidanim crtama označen je i dohvat objekta od poslužitelja ako je njegov odgovor najbrži ili ako nijedan susjedni zastupnik nema traženi objekt.



Slika 12: Komunikacijski slijed protokola Squid

Svi mogući komunikacijski slijedovi započinju stvaranjem zahtjeva na korisničkom računalu. U trenutku utvrđivanja potrebe dohvata određenog objekta korisnik iz vlastitih postavki čita adresu pridijeljenog mu glavnog zastupnika i šalje zahtjev. Slanje korisničkog zahtjeva prikazano je na slici 12a. Najkraći mogući komunikacijski slijed izvodi se ako glavni zastupnik ima valjanu presliku traženog objekta. U tom slučaju, glavni zastupnik vraća traženi objekt korisniku kao što je prikazano na slici 12f. U navedenom komunikacijskom slijedu nema međusobne komunikacije zastupnika.

Opisani komunikacijski slijed identičan je u svim promatranim upravljačkim protokolima pa nije uzet u obzir tijekom usporedbe opterećenja.

Drugi komunikacijski slijed odvija se ako glavni zastupnik nema traženi objekt. Glavni zastupnik u tom slučaju susjednim zastupnicima šalje *ICP_QUERY* poruke koje sadrže URL traženog objekta. Izvorni Squid protokol predviđa slanje *ICP_QUERY* poruka logičkim *multicast* mehanizmom kao što je prikazano na slici 12b. Ako je administrator sustava to omogućio, onda se usporedno s *ICP_QUERY* porukama poslužitelju šalje *ICP_SECHO* poruka. *ICP_SECHO* poruka služi provjeri brzine odziva poslužitelja i šalje se na poslužiteljev *echo* priključak. Nakon primitka *ICP_QUERY* poruke, susjedni zastupnici provjeravaju da li u lokalnoj memoriji imaju presliku traženog objekta. Na temelju navedene provjere susjedni zastupnici odgovaraju *unicast* slanjem *ICP_HIT* i *ICP_MISS* poruka glavnom zastupniku. *ICP_HIT* poruka označava da susjedni zastupnik ima valjanu presliku traženog objekta u svojoj priručnoj memoriji, a *ICP_MISS* poruka označava suprotni slučaj. Poslužitelj preko svog *echo* priključka vraća primljenu *ICP_SECHO* poruku glavnom zastupniku. Slanje svih navedenih odgovora prikazano je na slici 12c.

Na temelju primljenih odgovora, glavni zastupnik odabire onog susjednog zastupnika čiji je *ICP_HIT* odgovor primio prvi. Ako glavni zastupnik primi *ICP_SECHO* poruku prije pozitivnih odgovora susjednih zastupnika, onda se traženi objekt dohvaća od poslužitelja. Ako glavni zastupnik od svih susjednih zastupnika primi negativne odgovore ili ako u zadanom vremenu ne dobije sve očekivane odgovore, onda se zahtjev za objektom također šalje poslužitelju. Opisani događaji prikazani su na slici 12d. Nakon dohvata traženog objekta od susjednog zastupnika ili od poslužitelja, što je prikazano na slici 12e, glavni zastupnik sprema presliku objekta u vlastitu priručnu memoriju. Traženi objekt potom se dostavlja korisniku kako je prikazano na slici 12f.

2.4.3. Opterećenje protokola

U cilju analitičke usporedbe različitih upravljačkih protokola raspodijeljene priručne memorije potrebno je procijeniti opterećenje koje pojedini protokol izaziva. Prijenos objekta korisniku i dohvat objekta od poslužitelja ne ubrajaju se u opterećenje jer su zajednički svim upravljačkim protokolima. Osim toga, s obzirom da većina protokola nakon promašaja na glavnom zastupniku dohvaća objekt od susjednog zastupnika ili od poslužitelja, opterećenje koje nastaje kad glavni zastupnik dohvaća objekt također ne ulazi u opterećenje upravljačkog protokola. U opterećenje se stoga ubraja samo opterećenje zastupničkog sustava, koje je uzrokovano komunikacijom upravljačkog protokola. Navedeno opterećenje nastaje tijekom izvođenja algoritma traženja.

Opterećenje se dijeli na dva osnovna dijela: opterećenje komunikacijske mreže i opterećenje zastupničkih računala. Opterećenje komunikacijske mreže izražava se kao broj IP paketa koji se šalju tijekom izvođenja upravljačkog protokola. Opterećenje zastupničkih računala procjenjuje se pomoću broja obrada poruka upravljačkog protokola koje zastupnička računala šalju i primaju. Slanje poruka i obrada primljenih poruka troše određenu količinu procesorskog vremena pa se opterećenje zastupničkih računala izražava kao broj vremenskih jedinica obrade na procesoru.

Budući da slanje *ICP_SECHO* upita opterećuje poslužiteljsko računalo, u većini Squid sustava slanje navedenog upita je onemogućeno. Prema tome, u procjeni opterećenja koje stvara Squid protokol navedeni upiti su zanemareni. Prebrojavanjem poruka i određivanjem opterećenja zastupničkih računala na temelju slike 12 dobiveni su izrazi za mrežno opterećenje N_{paketa} i opterećenje zastupničkih računala T_{obrade}. Obje formule izražavaju opterećenje po jednom zahtjevu.

$$N_{\text{paketa}}=(1-p)\cdot(n+n) \tag{3.4.1}$$

$$T_{\text{obrade}}=(1-p)\cdot(2\cdot n\cdot t+2\cdot n\cdot t) \tag{3.4.2}$$

N_{paketa} ... broj mrežnih paketa koje stvara upravljački protokol
T_{obrade} ... broj vremenskih jedinica obrade u procesoru
t ... prosječno vrijeme potrebno za obradu jedne poruke u procesoru
p ... vjerojatnost da jedan zastupnik ima traženi objekt
n+1 ... broj zastupnika u raspodijeljenom zastupničkom sustavu
n ... broj susjednih zastupnika

Formula (3.4.1) sastoji se od dva podizraza. Podizraz (1-p) vjerojatnost je da glavni zastupnik nema traženi objekt. Podizraz (n+n) predstavlja n *ICP_QUERY* poruka koje glavni zastupnik logičkim *multicastom* šalje susjedima i n *unicast* odgovora susjednih zastupnika. Susjedni zastupnici odgovaraju *ICP_HIT* ili *ICP_MISS* porukama. Množenjem prvog i drugog podizraza dobiven je očekivani broj IP paketa po korisničkom zahtjevu.

Formula (3.4.2) dobivena je na analogni način pa je prvi podizraz isti kao u formuli (3.4.1). Drugi podizraz sastoji se od dva dijela. Prvi dio, izraz $2\cdot n\cdot t$, procesorsko je vrijeme potrebno za pripremanje i obradu n *ICP_QUERY* upita. Izraz se množi s 2, jer se svaki upit priprema na glavnom zastupniku, a potom se i obrađuje na zastupniku koji je primio upit. Drugi izraz $2\cdot n\cdot t$ procesorsko je vrijeme potrebno za pripremanje i obradu n odgovora *ICP_HIT* ili *ICP_MISS*. I drugi dio podizraza množi se s množiteljem 2, jer se odgovori pripremaju na susjednim zastupnicima, a potom se obrađuju na glavnom zastupniku. Dobivene formule koriste se u analitičkoj usporedbi protokola opisanoj u odjeljku 2.7.

2.5. Upravljački protokol Berkeley

U cilju rješavanja problema mrežnog opterećenja koje stvara Squid protokol, autori protokola nazvanog Berkeley [11] predlažu da se za slanje upita od glavnog zastupnika prema susjednim zastupnicima koristi *multicast* mehanizam. Navedenim mehanizmom šalje se samo jedna, identična poruka grupi primatelja. Uporabom *multicast* slanja poruka smanjuje se broj poruka potrebnih za slanje upita susjednim zastupnicima. Daljnje smanjenje mrežnog opterećenja Berkeley protokol ostvaruje zahtjevom da susjedni zastupnici ne šalju negativni odgovor ako nemaju traženi objekt. Berkeley protokol, prema tome, nema *ICP_MISS* poruku, ali je zato potrebno da glavni zastupnik izvodi složen i učinkovit algoritam prekida.

2.5.1. Način rada protokola

Algoritam pristupa Berkeley protokola izvodi se tako da korisnik slučajno bira kojem zastupniku u raspodijeljenom sustavu priručne memorije šalje upit. Autori Berkeley protokola nastoje opisanim algoritmom ravnomjerno rasporediti opterećenje. Berkleyev algoritam pristupa ima dva značajna nedostatka. Potrebno je da korisnici imaju točan popis svih zastupnika u sustavu što je teško ostvariti. Osim toga, mrežna udaljenost od korisnika do zastupnika nije ista za sve zastupnike u sustavu pa kašnjenje ovisi o izboru zastupnika kojem korisnik šalje zahtjev. Zbog navedenih nedostataka i pojednostavljenog raspoređivanja opterećenja, opisani algoritam pristupa ne koristi se u postojećim zastupničkim sustavima.

Odabrani glavni zastupnik izvodi algoritam traženja. Algoritam traženja šalje *multicast* upit svim susjednim zastupnicima. Susjedni zastupnici nakon primanja upita provjeravaju sadržaj svojih lokalnih priručnih memorija. Ako u lokalnoj priručnoj memoriji postoji valjana preslika traženog objekta, onda susjedni zastupnik odgovara potvrdnom porukom. Ako susjedni zastupnik nema valjanu presliku traženog objekta, onda ne šalje odgovor glavnom zastupniku. Algoritam odlučivanja jednak je algoritmu odlučivanja Squid protokola, odnosno, glavni zastupnik dohvaća traženi objekt od susjednog zastupnika čiji je potvrdni odgovor prvi primio.

Budući da susjedni zastupnici ne šalju negativne odgovore, algoritam prekida ima znatan utjecaj na svojstva Berkeley protokola. Algoritam prekida ostvaren je na osnovi isteka unaprijed određenog vremena. Ako u navedenom vremenu ne dobije nijedan pozitivni odgovor, onda glavni zastupnik dohvaća traženi objekt od poslužitelja. Odabir vremena čekanja određuje svojstva Berkeley protokola. Ako je vrijeme čekanja prekratko, onda glavni zastupnik dohvaća objekte od poslužitelja umjesto od susjednih zastupnika. U suprotnom slučaju, ako je vrijeme čekanja predugo, onda se nepotrebno unosi dodatno kašnjenje tijekom dohvata objekta.

Algoritam uvišestručavanja Berkeley protokola predložen u dokumentu [11], znatno se razlikuje od algoritama ostalih protokola za upravljanje raspodijeljenim priručnim memorijama. Posebnost Berkeley protokola javlja se u slučaju da traženi objekt postoji u lokalnoj priručnoj memoriji nekog susjednog zastupnika. Ako nijedan zastupnik u sustavu nema traženi objekt u svojoj lokalnoj priručnoj memoriji, onda se objekt nakon dohvata od poslužitelja sprema u priručnu memoriju glavnog zastupnika. Ako neki susjedni zastupnik ima traženi objekt, onda glavni zastupnik ne dohvaća objekt nego preusmjerava korisnika na navedenog susjednog zastupnika. Preusmjeravanjem korisnika na susjednog zastupnika, Berkeley protokol smanjuje pojavu zalihosti objekata. Preusmjeravanje se izvodi tako da glavni zastupnik korisniku pošalje poruku o preusmjeravanju na susjednog zastupnika koji ima traženi objekt.

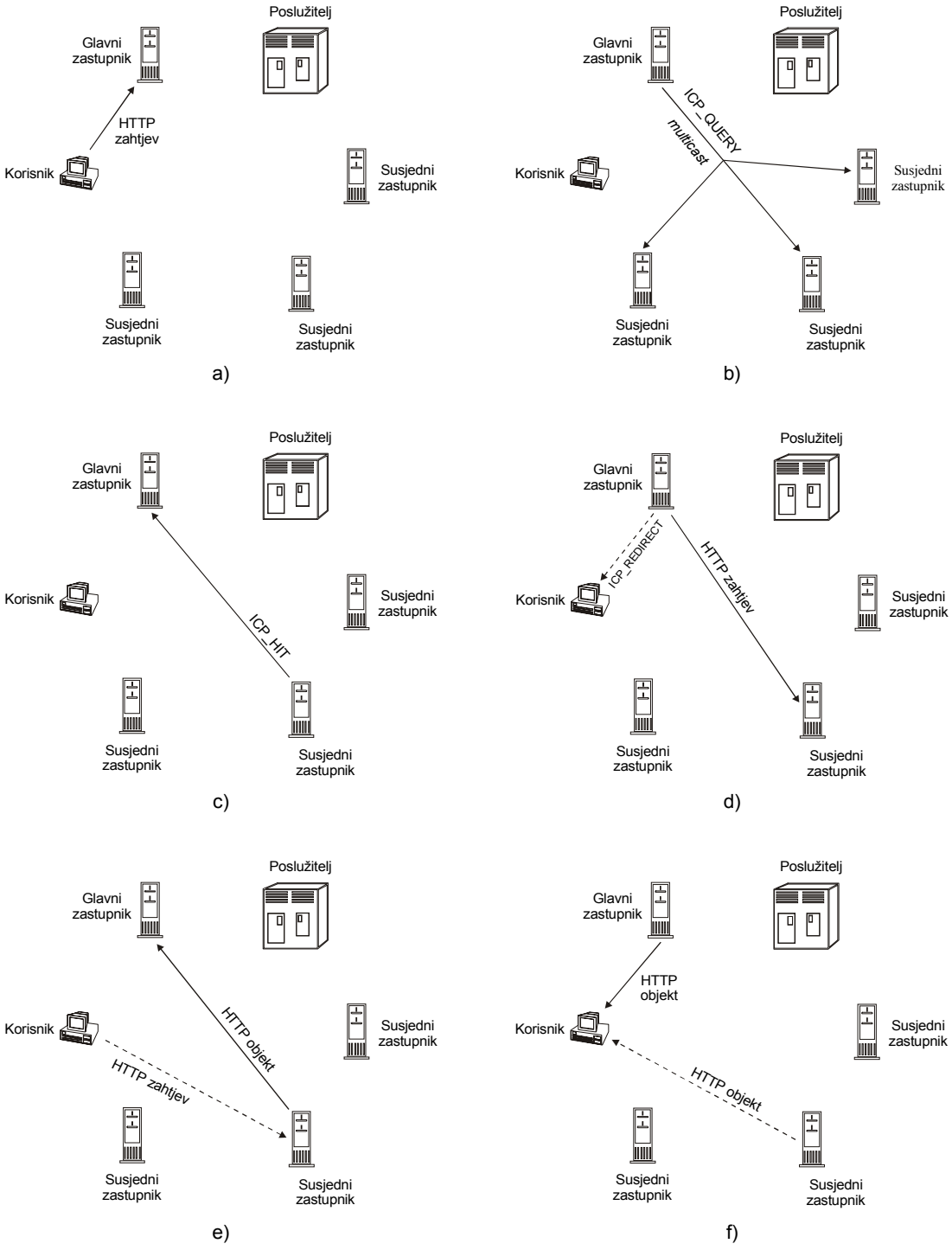
2.5.2. Komunikacijski slijed

Na slici 13 prikazan je komunikacijski slijed Berkeley protokola tijekom dohvata jednog objekta. Iako i Berkeley protokol ima više slijedova događaja, zbog istovjetnosti sa Squid protokolom, na slici 13

nisu prikazana dva slijeda. Prvi neprikazani slijed odnosi se na okolnosti u kojima glavni zastupnik ima traženi objekt i izravno ga dostavlja korisniku. Drugi neprikazani slijed događaja je dohvat objekta od poslužitelja nakon što je utvrđeno da nijedan zastupnik u sustavu nema traženi objekt.

Iako se preusmjeravanje ne koristi u većini programskih ostvarenja Berkeley protokola, slika 13 prikazuje komunikacijski slijed preusmjeravanja isprekidanim crtama. Mehanizam presumjeravanja nije podržan u postojećim sustavima, jer zahtijeva značajne promjene u korisničkom programu.

Iako ICP protokol [9,10] nema poruku preusmjeravanja, u svrhu opisa Berkeley protokola definirana je poruka oznake *ICP_REDIRECT*. Navedenu poruku glavni zastupnik šalje korisniku. Poruka sadrži URL objekta koji je nađen u lokalnoj priručnoj memoriji susjednog zastupnika i IP adresu susjednog zastupnika. Korisnik nakon primanja *ICP_REDIRECT* poruke ponavlja cijeli postupak dohvata traženog objekta od susjednog zastupnika navedenog u poruci. Navedeni susjedni zastupnik na taj način postaje novi glavni zastupnik za dohvat traženog objekta.



Slika 13: Komunikacijski slijed protokola Berkeley

Temeljni komunikacijski slijed započinje korisnikovim zahtjevom Internet pretraživaču. Korisničko računalo potom određuje adresu glavnog zastupnika i šalje zahtjev kao što je prikazano na slici 13a. Nakon što utvrdi da nema traženi objekt, glavni zastupnik *multicast* mehanizmom šalje *ICP_QUERY* upit svim susjednim zastupnicima. Navedeni mehanizam omogućuje pošiljatelju slanje jedne poruke koju prima više primatelja kako prikazuje slika 13b. Susjedni zastupnici provjeravaju da li imaju traženi objekt i ako ga pronađu u vlastitoj priručnoj memoriji, onda glavnom zastupniku šalju

ICP_HIT poruku. Slika 13c prikazuje slanje *ICP_HIT* poruke. Budući da mehanizam preusmjeravanja nije podržan u većini sustava, na slici 13d prikazano je da nakon primanja prve *ICP_HIT* poruke glavni zastupnik pokreće dohvat objekta od susjednog zastupnika. Na slici 13e prikazan je prijenos objekta od susjednog zastupnika prema glavnom zastupniku, a na slici 13f traženi se objekt dostavlja korisniku.

Osim opisanog slijeda događaja, slike 13d-13f isprekidanim crtama prikazuju i komunikacijski slijed koji se izvodi ako sustav podržava preusmjeravanje. Glavni zastupnik u tom slučaju šalje *ICP_REDIRECT* poruku korisničkom računalu kao na slici 13d. Korisničko računalo potom šalje novi HTTP zahtjev susjednom zastupniku navedenom u *ICP_REDIRECT* poruci što je prikazano na slici 13e. Slika 13f prikazuje slanje objekta korisniku od navedenog susjednog zastupnika.

2.5.3. Opterećenje protokola

U svrhu procjene opterećenja koje stvara Berkeley protokol, potrebno je na temelju slike 13 izračunati broj mrežnih paketa i vremenskih jedinica obrade na zastupničkim računalima. Prilikom izračuna mrežnog opterećenja i opterećenja zastupničkih računala zanemaren je mehanizam preusmjeravanja. Izrazi za opterećenje računaju se kao prosjek po jednom korisničkom zahtjevu.

$$N_{\text{paketa}}=(1-p)\cdot(1+p\cdot n) \tag{3.5.1}$$

$$T_{\text{obrade}}=(1-p)\cdot(t+n\cdot t+2\cdot p\cdot n\cdot t) \tag{3.5.2}$$

N_{paketa} ... broj mrežnih paketa koje stvara upravljački protokol
T_{obrade} ... broj vremenskih jedinica obrade u procesoru
t ... prosječno vrijeme potrebno za obradu jedne poruke u procesoru
p ... vjerojatnost da jedan zastupnik ima traženi objekt
n+1 ... broj zastupnika u raspodijeljenom zastupničkom sustavu
n ... broj susjednih zastupnika

Formule (3.5.1) i (3.5.2) sastoje se od dva podizraza. Prvi podizraz (1-p) zajednički je za obje formule i predstavlja vjerojatnost da glavni zastupnik nema traženi objekt. Drugi podizraz formule (3.5.1) sastoji se od zbrajanja dva dijela. Prvi dio drugog podizraza predstavlja jedan *multicast* paket potreban za slanje *ICP_QUERY* poruke. Drugi dio, $p*n$, očekivani je broj *ICP_HIT* poruka koje su odgovori susjednih zastupnika. Navedeni broj dobiva se množenjem vjerojatnosti da jedan zastupnik ima traženi objekt s brojem susjednih zastupnika.

Drugi podizraz u formuli (3.5.2) sastoji se od zbrajanja tri dijela. Prvi je dio drugog podizraza procesorsko vrijeme potrebno za stvaranje *ICP_QUERY* poruke na glavnom zastupniku. Drugi dio je procesorsko vrijeme obrade primljene *ICP_QUERY* poruke na n susjednih zastupnika. Treći dio obuhvaća procesorsko vrijeme pripreme $p*n$ *ICP_HIT* poruka na susjednim zastupnicima i obradu navedenih poruka na glavnom zastupniku. Navedeni dio množi se s dva, jer se *ICP_HIT* poruke pripremaju na susjednim zastupnicima, a potom se obrađuju na glavnom zastupniku. Množitelj $p*n$ u trećem podizrazu predstavlja očekivani broj zastupnika koji imaju traženi objekt.

2.6. Upravljački protokol zasnovan na direktoriju

Upravljački protokol raspodijeljene priručne memorije zasnovan na direktoriju stvoren je s namjerom potpunog smanjenja opterećenja koje nastaje izvođenjem upravljačkog protokola. Berkeley protokol jednostavnom zamisli uporabe *multicast* mehanizma za slanje upita značajno smanjuje mrežni promet, ali ne rješava problem opterećenja zastupničkih računala. Za razliku od njega, direktorijski protokol osim smanjenja broja IP paketa, smanjuje i procesorsko opterećenje zastupničkih računala. Smanjenje opterećenja raspodijeljenog zastupničkog sustava ostvareno je uporabom direktorija koji opisuje sadržaje priručnih memorija zastupnika. Upravljački protokol zasnovan na direktoriju razvio je prof. dr. sc. Siniša Srbljić kao dio većeg projekta tijekom rada u IPTO (Internet Platform Technology Organization) organizaciji AT&T Labs istraživačkih laboratorija u San Joseu, California, SAD. Protokol je razvijen 1995. godine i patentiran u SAD-u [23,24]. Kraći opis i analiza svojstava direktorijskog protokola prikazani su u članku [28].

2.6.1. Poruke direktorijskog protokola

Programsko ostvarenje direktorijskog protokola uključuje proširenje skupa ICP poruka i definiciju standardnih komunikacijskih postupaka. Definiranje novih poruka i određivanje načina prijenosa informacija vezanih uz direktorij značajan su dio ovog magistarskog rada. Pregled direktorijskih poruka kojima je proširen ICP protokol dan je u tablici 2. Sve navedene poruke koriste standardni format ICP poruka opisan u odjeljku 2.3.2.

Vrsta poruke	Namjena poruke
ICP_DIR_GET	Šalje se kad glavni zastupnik traži informacije iz direktorija, a listu za traženi objekt održava susjedni zastupnik. Poruka u dijelu namijenjenom za podatke promjenjive veličine ima URL traženog objekta.
ICP_DIR_ADD	Navedenom porukom susjedni zastupnik traži od direktorijskog zastupnika da ga unese u direktorijsku listu za navedeni objekt. U dijelu namijenjenom dodatnim podacima nalazi se URL objekta.
ICP_DIR_DEL	Poruku šalje susjedni zastupnik direktorijskom zastupniku. Porukom se traži izbacivanje susjednog zastupnika iz direktorijske liste vezane za navedeni objekt. URL objekta upisan je u prostoru za dodatne podatke.
ICP_DIR_DATA	Navedenu poruku direktorijski zastupnik šalje glavnom zastupniku kao odgovor na poruku *ICP_DIR_GET*. Poruka prenosi podatke iz direktorijske liste za traženi objekt. Na početku dodatnih podataka je URL objekta čija lista je zatražena. Nakon URL-a naveden je popis imena zastupnika koji u svojoj lokalnoj memoriji imaju traženi objekt. Imena zastupnika odvojena su dvotočkama, a popis završava s dvije dvotočke. Ako je lista prazna, onda se popis sastoji samo od dvije dvotočke.

Tablica 2: Pregled poruka direktorijskog protokola

2.6.2. Način rada protokola

U direktorij se za svaki pojedini objekt upisuje lista zastupnika koji objekt imaju u svojoj priručnoj memoriji. Na početku rada sustava direktorij je prazan, a proširuje se svakim korisničkim zahtjevom. Tijekom slanja upita susjednim zastupnicima, glavni zastupnik dohvaća iz direktorija listu zastupnika koji imaju traženi objekt. Glavni zastupnik potom *unicast* mehanizmom šalje upite susjednim zastupnicima. Nakon dohvata objekta, glavni zastupnik se dodaje u listu zastupnika koji imaju navedeni objekt.

S obzirom da održavanje direktorija stvara dodatno opterećenje, direktorij je raspodijeljen između zastupničkih računala. Za raspodjelu direktorija koristi se funkcija raspršenog adresiranja (engl. *hash*) koja ravnomjerno raspodjeljuje direktorij na sve zastupnike u sustavu raspodijeljene memorije. Svaki zastupnik u sustavu održava direktorijske liste za skup objekata, pri čemu se informacija o jednom objektu nalazi samo na jednom zastupniku. Opisanim pristupom osigurana je proširivost sustava s obzirom na broj zastupnika.

Uvođenjem direktorijskog protokola u raspodijeljeni zastupnički sustav, osim pojma glavnog zastupnika, javlja se i pojam direktorijskog zastupnika. Direktorijski je zastupnik jedan od zastupnika u sustavu koji održava direktorijsku listu za objekt koji se trenutno dohvaća. Zadaće su direktorijskog zastupnika bilježenje dodavanja objekata u lokalnu priručnu memoriju susjednih zastupnika,

bilježenje izbacivanja objekata iz lokalne priručne memorije susjednih zastupnika i dostavljanje podataka iz direktorija susjednim zastupnicima. Budući da je svaki zastupnik u sustavu direktorijski zastupnik za određeni skup objekata, direktorijski zastupnik istodobno izvodi sve funkcije običnog zastupnika. Ako korisnik pošalje zahtjev za objektom zastupniku koji održava direktorijsku listu za traženi objekt, onda je navedeni zastupnik istodobno direktorijski zastupnik i glavni zastupnik.

Algoritam pristupa direktorijskog protokola isti je kao u Squidu, odnosno, korisnikov Internet pretraživač na temelju vlastitih postavki bira glavnog zastupnika. Nakon što glavni zastupnik primi zahtjev od korisnika i ne pronađe traženi objekt u svojoj priručnoj memoriji, pokreće se algoritam traženja.

Algoritam traženja temelji se na dohvatu informacija iz direktorija. Pritom postoje dva moguća slijeda događaja zbog mogućnosti da se direktorij za traženi objekt nalazi na glavnom ili na susjednom zastupniku. Ako je direktorij za traženi objekt lokalno spremljen, onda zastupnik izravno dohvaća podatke iz direktorija. U suprotnom slučaju, glavni zastupnik šalje direktorijskom zastupniku upit za čitanje podataka iz direktorija. Direktorijski zastupnik odgovara porukom koja sadrži listu zastupnika koji imaju traženi objekt u svojoj lokalnoj priručnoj memoriji. Nakon što je pribavio podatke iz direktorija, glavni zastupnik provjerava da li je dobivena lista zastupnika prazna. Ako u direktorijskoj listi nema zapisa, onda je ostvaren jedan od uvjeta izvođenja algoritma prekida. Algoritam prekida u tom slučaju nalaže glavnom zastupniku da izravno zatraži objekt od poslužitelja. Budući da glavni zastupnik ne čeka odgovore susjednih zastupnika, opisani način izvođenja algoritma prekida smanjuje kašnjenje zahtjeva u odnosu na protokole Squid i Berkeley.

Ako u direktorijskoj listi postoji najmanje jedan zapis, onda algoritam traženja šalje upite susjednim zastupnicima navedenima u listi. Slanje upita susjedima potrebno je zato što se komunikacija vezana uz direktorij zasniva na ICP protokolu. ICP koristi nepouzdani komunikacijski protokol UDP pa je moguća pojava netočnih podataka ili nejednoznačnosti u direktoriju. Slanjem upita susjedima otklanja se utjecaj nejednoznačnosti. Osim toga, izgradnjom sustava koji pretpostavlja postojanje netočnih informacija povećava se otpornost sustava na druge uzroke nejednoznačnosti. Primjeri ostalih uzroka su ispadi zastupničkih računala ili pogreške u programskom ostvarenju. U svrhu dodatnog ograničenja opterećenja zastupničkog sustava uzrokovanog upravljačkim protokolom, glavni zastupnik ne šalje upite neograničenom broju susjednih zastupnika navedenih u direktorijskoj listi.

Algoritam odlučivanja sudjeluje u slanju upita ograničavanjem broja susjednih zastupnika kojima se šalju upiti. Algoritam odlučivanja zbog ravnomjernog raspoređivanja opterećenja slučajno bira α zastupnika kojima se šalje upit. U praktičnoj izvedbi, za vrijednost parametra α obično se bira mali broj, između 3 i 10. U sustavima čija je pouzdanost mreže i računala visoka, parametar α ima manju vrijednost. U sustavima sa slabom pouzdanosti, α ima veću vrijednost što povećava vjerojatnost da neki od susjednih zastupnika potvrdno odgovori na upit.

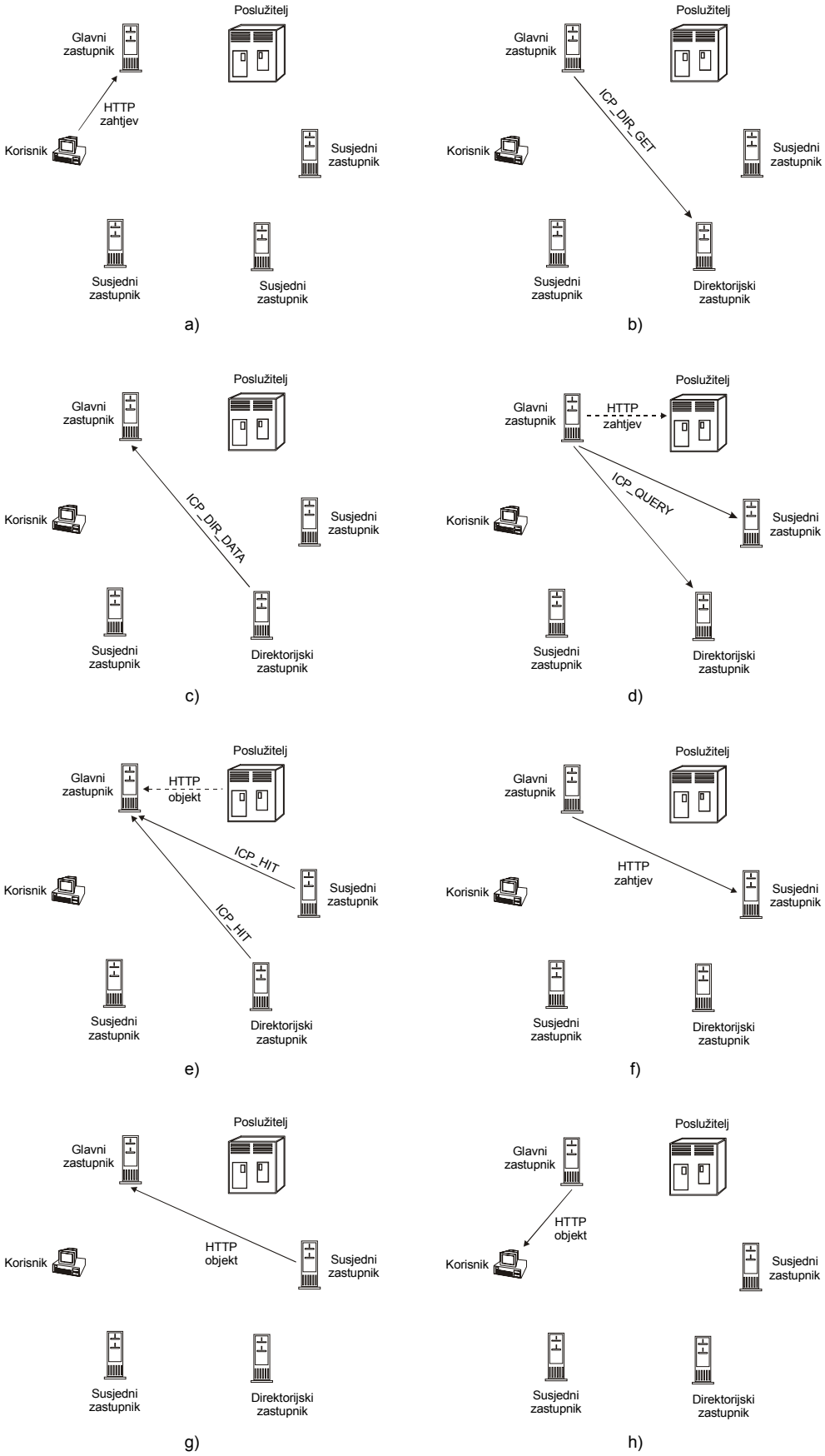
Drugi dio algoritma odlučivanja vezan je uz odgovore susjednih zastupnika. Susjedni zastupnici šalju potvrdne i negativne odgovore glavnom zastupniku. Ako je sustav pouzdan, onda su zbog uporabe

direktorija svi odgovori pozitivni. Nakon primanja prvog potvrdnog odgovora, algoritam odlučivanja bira susjednog zastupnika koji je navedeni odgovor poslao. Glavni zastupnik potom šalje HTTP zahtjev odabranom susjednom zastupniku.

Slanje negativnih odgovora omogućuje glavnom zastupniku da na temelju brojanja primljenih odgovora zaključi da su svi upitani susjedni zastupnici odgovorili negativno. U tom slučaju ostvaren je drugi uvjet za izvođenje algoritma prekida pa glavni zastupnik pristupa dohvatu traženog objekta od poslužitelja. Treći je uvjet za izvođenje algoritma prekida istek očekivanog vremena primanja odgovora od susjednih zastupnika. Ako u predviđenom vremenu glavni zastupnik ne dobije odgovor na upit za direktorijskom listom koji je poslan direktorijskom zastupniku ili ako ne dobije sve odgovore na upite za objektom koji su poslani susjednim zastupnicima, onda se objekt dohvaća izravno od poslužitelja. Algoritam uvišestručavanja direktorijskog protokola sprema sve dohvaćene objekte u lokalnu memoriju glavnog zastupnika.

2.6.3. Komunikacijski slijed

Slika 14 prikazuje slijed poruka koje direktorijski protokol razmjenjuje pri dohvatu objekta koji je korisnik zatražio. Na slici 14 nije prikazan komunikacijski slijed koji nastaje ako glavni zastupnik ima traženi objekt u svojoj priručnoj memoriji. Na slici nije prikazan ni komunikacijski slijed kojim glavni zastupnik dohvaća traženi objekt od poslužitelja nakon što od susjednih zastupnika nije dobiven pozitivni odgovor. Punim crtama je na slici označen slijed koji se izvodi u okolnostima u kojima glavni zastupnik nema traženi objekt, direktorijski zastupnik je jedan od susjednih zastupnika i preslika traženog objekta postoji na jednom od susjednih zastupnika. Isprekidanim je crtama na slici 14 označen poseban dio komunikacijskog slijeda koji se ostvaruje u slučaju da je direktorijska lista za traženi objekt prazna.



Slika 14: Komunikacijski slijed direktorijskog protokola

Postupak dohvata objekta počinje slanjem zahtjeva glavnom zastupniku prikazanom na slici 14a. Ako glavni zastupnik nema traženi objekt, onda se pristupa čitanju informacija iz direktorija. Direktorij se može nalaziti na glavnom zastupniku ili na nekom od susjednih zastupnika. Ako je glavni zastupnik istodobno i direktorijski zastupnik, onda se informacije u direktoriju dohvaćaju bez slanja ICP poruka. Komunikacijski se slijed u tom slučaju nastavlja slanjem *ICP_QUERY* upita prikazanim na slici 14d. Ako glavni zastupnik nije direktorijski zastupnik, onda se direktorijskom zastupniku šalje *ICP_DIR_GET* poruka kao što je prikazano na slici 14b. Slika 14c pokazuje da direktorijski zastupnik odgovara porukom *ICP_DIR_DATA* koja sadrži podatke iz direktorija. Poruka *ICP_DIR_DATA* šalje se i ako u direktoriju ne postoje podaci vezani uz traženi objekt. Nakon pripremanja poruke *ICP_DIR_DATA* direktorijski zastupnik dodaje glavnog zastupnika u listu za traženi objekt.

Ako je dohvaćena direktorijska lista za traženi objekt prazna, onda glavni zastupnik šalje zahtjev za objektom poslužitelju što je prikazano isprekidanom crtom na slici 14d. Poslužitelj potom dostavlja objekt glavnom zastupniku što prikazuje isprekidana crta na slici 14e, a glavni zastupnik dostavlja objekt korisniku kao što je prikazano na slici 14h.

Ako direktorijska lista za traženi objekt nije prazna, onda glavni zastupnik odabire α susjednih zastupnika kojima šalje *ICP_QUERY* upite. Pomoću navedenih upita, prikazanih na slici 14d provjerava se da li zastupnici navedeni u direktoriju doista imaju traženi objekt. Ako se direktorijski zastupnik nalazi u listi za traženi objekt, onda glavni zastupnik može i njega odabrati za slanje *ICP_QUERY* upita. Slika 14e prikazuje postupak odgovaranja susjednih zastupnika. Na primljeni *ICP_QUERY* upit svaki zastupnik odgovara *ICP_HIT* ili *ICP_MISS* porukom. Glavni zastupnik šalje HTTP zahtjev onom susjednom zastupniku čiju je *ICP_HIT* poruku primio prvu kao što je prikazano na slici 14f. Nastavak postupka dohvata objekta jednostavan je i sastoji se od dohvata objekta od susjednog zastupnika prikazanog na slici 14g i slanja objekta korisniku što je prikazano na slici 14h.

2.6.4. Opterećenje protokola

Na temelju slijeda poruka prikazanog na slici 14 računa se opterećenje zastupničkog sustava uzrokovano izvođenjem direktorijskog upravljačkog protokola. Opterećenje se izražava kao broj IP paketa i broj vremenskih jedinica obrade na zastupničkim računalima po korisničkom zahtjevu. Opterećenje direktorijskog protokola izraženo je slijedećim formulama.

$$N_{\text{paketa}}=(1-p)\cdot(2\cdot r+2\cdot y)+q\cdot r \tag{3.6.1}$$

$$T_{\text{obrade}}=(1-p)\cdot[(2\cdot t+2\cdot t)\cdot r+4\cdot y\cdot t]+2\cdot q\cdot r\cdot t \tag{3.6.2}$$

N_{paketa} ... broj mrežnih paketa koje stvara upravljački protokol

T_{obrade} ... broj vremenskih jedinica obrade u procesoru

t ... prosječno vrijeme potrebno za obradu jedne poruke u procesoru

p ... vjerojatnost da jedan zastupnik ima traženi objekt

$n+1$ ... broj zastupnika u raspodijeljenom zastupničkom sustavu

n ... broj susjednih zastupnika

$r=\frac{n}{n+1}$ ... vjerojatnost da direktorijsku listu održava susjedni zastupnik

α ... maksimalni broj *ICP_QUERY* poruka koje šalje glavni zastupnik

$y=\min(p\cdot n, \alpha)$ ... broj *ICP_QUERY* poruka koje se šalju susjednim zastupnicima

$q \approx (1-p)^{n+1}$ ... vjerojatnost izbacivanja objekta iz lokalne memorije

Prvi podizraz formule (3.6.1) na početku ima množitelj (1-p) koji označava vjerojatnost da glavni zastupnik nema traženi objekt. Drugi dio prvog podizraza sastoji se od dva dijela. Podizraz $2\cdot r$ odnosi se na dvije poruke, poruku *ICP_DIR_GET* koju šalje glavni zastupnik i poruku *ICP_DIR_DATA* kojom direktorijski zastupnik vraća podatke iz direktorija. Navedene dvije poruke pomnožene su s vjerojatnosti da se direktorijska lista za traženi objekt održava na jednom od susjednih zastupnika. Podizraz $2\cdot y$ predstavlja y *ICP_QUERY* poruka koje se šalju zastupnicima iz liste i pripadnih y odgovora koji mogu biti *ICP_HIT* ili *ICP_MISS*. y je složena funkcija kojom se ograničava broj upita koje šalje glavni zastupnik. Ograničenje se ostvaruje odabirom manje vrijednosti između izraza $p\cdot n$ i α. Izraz $p\cdot n$ očekivani je broj susjednih zastupnika koji imaju traženi objekt. Navedeni izraz odgovara broju zastupnika u direktorijskoj listi za traženi objekt. Ako je broj zastupnika u listi manji od α, onda se upiti šalju svim zastupnicima s liste. Ako je broj zastupnika u listi veći od α, onda se šalje α upita.

Drugi podizraz formule (3.6.1) prikazuje očekivani broj poruka *ICP_DIR_DEL*. Navedeni broj dobiven je množenjem vjerojatnosti q s vjerojatnosti r. Parametar q predstavlja vjerojatnost da je za dodavanje objekta u lokalnu priručnu memoriju glavnog zastupnika potrebno prethodno izbaciti neki objekt iz nje. Vjerojatnost q izražena je približno kao vjerojatnost da nijedan zastupnik u raspodijeljenoj priručnoj memoriji nema traženi objekt. Navedena se aproksimacija temelji na pretpostavci da je lokalna memorija glavnog zastupnika u trenutku dohvata objekta puna. Vrijednost parametra q u stvarnim je sustavima manja od navedene.

Prvi član formule (3.6.2) sadrži množitelj (1-p) kojim se izražava vjerojatnost da glavni zastupnik nema traženi objekt. Drugi dio člana sastoji se od dva podizraza. Podizraz $(2\cdot t+2\cdot t)\cdot r$ je količina obrade potrebna za pripremanje i obradu *ICP_DIR_GET* poruke, te potom za pripremanje i obradu *ICP_DIR_DATA* poruke. Podizraz na kraju sadrži množitelj r koji određuje vjerojatnost da je direktorijska lista za traženi objekt na susjednom zastupniku. Drugi podizraz, $4\cdot y\cdot t$ predstavlja pripremanje i obradu y *ICP_QUERY* upita koje šalje glavni zastupnik, a primaju ih susjedni

zastupnici, te pripremanje i obradu odgovora *ICP_HIT* ili *ICP_MISS* koje šalju susjedni zastupnici, a prima ih glavni zastupnik. Drugi član formule (3.6.2) izražava vrijeme pripreme i obrade poruke *ICP_DIR_DEL*. Navedeni član množi se s vjerojatnosti izbacivanja objekta i s vjerojatnosti da je direktorijska lista za traženi objekt na susjednom zastupniku.

2.6.5. Optimistični i pesimistični pristup održavanju direktorija

Definicija direktorijskog protokola [23,24] predviđa dva pristupa održavanju informacija u direktoriju. Prema vjerojatnosti dohvata objekata, nazvani su pesimističnim i optimističnim pristupom. Oba pristupa vezana su uz odluku u kojem se trenutku glavni zastupnik dodaje u direktorijsku listu objekta. U prethodnom opisu direktorijskog protokola opisan je optimistični pristup. U optimističnom pristupu direktorijski zastupnik dodaje glavnog zastupnika u direktorijsku listu odmah nakon primanja *ICP_DIR_GET* poruke. Navedeni pristup nazvan je optimističnim, jer direktorijski zastupnik pretpostavlja da će glavni zastupnik u budućem, kratkom vremenu dobaviti traženi objekt.

Ako glavni zastupnik nije uspio dohvatiti traženi objekt, onda direktorij ne sadrži točne podatke. Opterećenje koje nastaje zbog netočnih podataka očituje se u obliku suvišnih upita zastupnicima koji nemaju traženi objekt. Navedeno opterećenje nije znatno u pouzdanim sustavima pa je optimistični pristup namijenjen uporabi u takvim sustavima.

Točnost direktorija u optimističnom je pristupu moguće povećati ugradnjom posebnog mehanizma. Nakon neuspjelog dohvata objekta od poslužitelja glavni zastupnik šalje direktorijskom zastupniku poruku *ICP_DIR_DEL*. Navedeni mehanizam održava točnost direktorija, ali značajno povećava složenost programskog ostvarenja zbog potrebe otkrivanja neuspjelih dohvata objekata.

U sustavima male pouzdanosti primjereniji je pesimistični pristup. U pesimističnom pristupu direktorijski zastupnik ne dodaje glavnog zastupnika u direktorijsku listu odmah nakon primanja *ICP_DIR_GET* upita. Održavanje točnosti direktorija prepušteno je glavnom zastupniku. Glavni zastupnik, nakon dohvata traženog objekta od poslužitelja, izravno zahtijeva dodavanje u direktorijsku listu. Dodavanje glavnog zastupnika u direktorijsku listu zahtijeva se slanjem *ICP_DIR_ADD* poruke direktorijskom zastupniku. Prednost navedenog pristupa je veća pouzdanost direktorija, a osnovni je nedostatak dodatna ICP poruka potrebna za dodavanje glavnog zastupnika u direktorijsku listu. Slanje poruke *ICP_DIR_ADD* povećava mrežno opterećenje raspodijeljenog zastupničkog sustava i prosječno kašnjenje zahtjeva.

2.7. Analitička usporedba upravljačkih protokola

Složena mjera za analitičku usporedbu učinkovitosti pojedinih upravljačkih protokola je prosječno kašnjenje tijekom ispunjavanja korisničkih zahtjeva. Precizno izračunavanje prosječnog kašnjenja složen je i zahtjevan proces. Za pojednostavljenu analitičku usporedbu upravljačkih protokola stoga se u ovom radu koristi proračun opterećenja koje stvaraju upravljački protokoli. Navedena usporedba

moguća je samo ako se radi o protokolima sličnih temeljnih algoritama. Squid, Berkeley i direktorijski protokol zadovoljavaju navedeni uvjet.

Formule za mrežno opterećenje i opterećenje zastupničkih računala pojedinih upravljačkih protokola prikupljene su zbog lakše usporedbe na jednom mjestu, u tablici 3. Formule za opterećenja prikazane u tablici 3 dobivene su grupiranjem članova formula (3.4.1), (3.4.2), (3.5.1), (3.5.2), (3.6.1) i (3.6.2). U navedene formule potom su uvrštene vrijednosti za parametre q i r, te je pretpostavljeno da je $y=\alpha$. Izrazi u tablici predstavljaju opterećenje po jednom korisničkom zahtjevu. Mrežno opterećenje izraženo je u broju IP paketa, a opterećenje zastupničkih računala izraženo je u vremenskim jedinicama obrade na procesoru.

Protokol	Mrežno opterećenje (N_{paketa}, IP paketa po zahtjevu)
Squid	$2\cdot(1-p)\cdot n$
Berkeley	$(1-p)\cdot(1+p\cdot n)$
Direktorij	$2\cdot(1-p)\cdot(1+\alpha)+(1-p)^{n+1}-\frac{2\cdot(1-p)+(1-p)^{n+1}}{n+1}$
Protokol	**Opterećenje računala (T_z, jedinica obrade po zahtjevu)**
Squid	$4\cdot(1-p)\cdot n\cdot t$
Berkeley	$(1-p)\cdot(1+(1+2\cdot p)\cdot n)\cdot t$
Direktorij	$2\cdot\left(2\cdot(1-p)\cdot(1+\alpha)+(1-p)^{n+1}-\frac{2\cdot(1-p)+(1-p)^{n+1}}{n+1}\right)\cdot t$

t ... prosječno vrijeme potrebno za obradu jedne poruku u procesoru

p ... vjerojatnost da jedan zastupnik ima traženi objekt

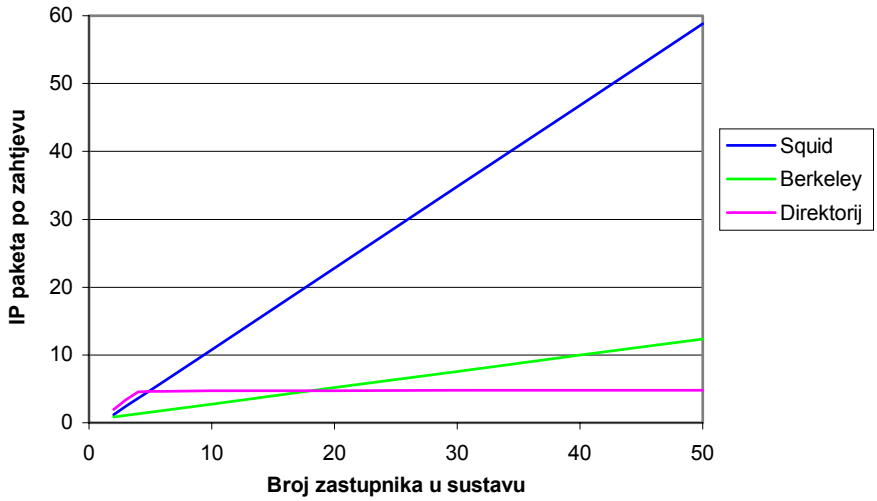
n+1 ... broj zastupnika u raspodijeljenom zastupničkom sustavu

n ... broj susjednih zastupnika

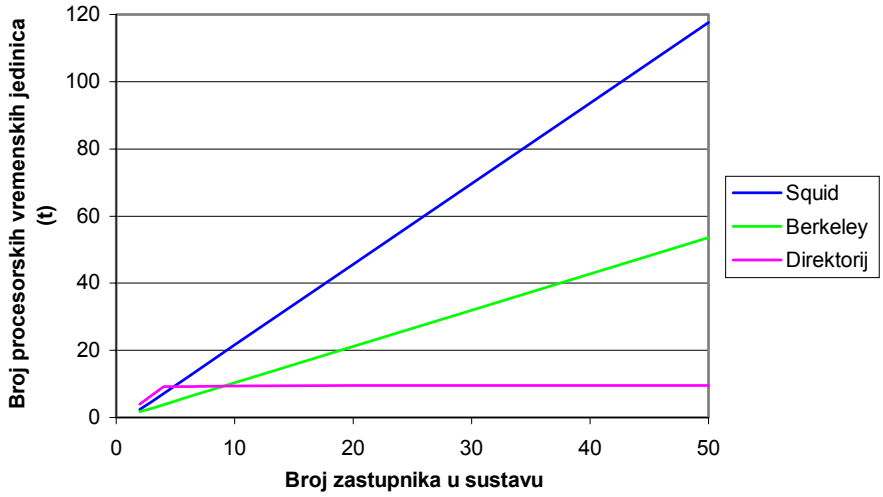
α ... maksimalni broj *ICP_QUERY* poruka koje šalje glavni zastupnik

Tablica 3: Opterećenje upravljačkih protokola

Izrazi navedeni u tablici 3 omogućuju jednostavnu analitičku usporedbu Squid, Berkeley i direktorijskog protokola. Na temelju izraza za opterećenje zaključuje se da opterećenja Squid i Berkeley protokola linearno rastu s brojem zastupnika u sustavu. Za razliku od navedenih protokola, opterećenje direktorijskog protokola neznatno raste s brojem zastupnika u sustavu, odnosno teži u konačnu vrijednost. S obzirom na složenost izraza prikazanih u tablici 3, odnosi između opterećenja znatno su jasniji na grafičkom prikazu porasta opterećenja datom na slikama 15 i 16. Za izradu navedenih grafova korišten je interval od 2-50 zastupnika u sustavu. Vrijednost parametra p, koji određuje vjerojatnost da određeni zastupnik ima traženi objekt, iznosi 0.4. Pretpostavljena je ista vjerojatnost pronalaženja objekta u raspodijeljenom zastupničkom sustavu za sve upravljačke protokole. Parametar α, koji određuje broj zastupnika kojima direktorijski protokol šalje *ICP_QUERY* poruku, iznosi 3.



Slika 15: Opterećenje zastupničkog sustava IP paketima



Slika 16: Opterećenje zastupničkog sustava obradom ICP poruka

Slika 15 pokazuje da broj IP paketa, koje stvaraju upravljački protokoli Squid i Berkeley, linearno ovisi o broju zastupnika u sustavu. Krivulja Berkeley protokola ima blaži rast od krivulje Squid protokola. Formula navedena u tablici 3 pokazuje da je nagib krivulje Berkeley protokola proporcionalan s vjerojatnosti da određeni zastupnik ima traženi objekt (p). Na slici 15 vidljivo je da krivulja direktorijskog protokola ima neznatan porast težeći u konstantnu vrijednost. Navedenu vrijednost moguće je izvesti iz formule (3.6.1). Ako $n \to \infty$, onda broj IP paketa teži k $(1-p)\cdot 2\cdot(\alpha+1)$. Za korištene vrijednosti parametara p i α broj navedeni izraz teži k vrijednosti 4.8.

Usporedba procesorskih opterećenja prikazanih na slici 16 pokazuje slične odnose. Opterećenje Squid i Berkeley protokola linearno ovisi o broju zastupničkih računala. U odnosu na krivulju mrežnog opterećenja, krivulja procesorskog opterećenja za Berkeley protokol ima znatno veći nagib koji je bliži nagibu krivulje Squid protokola. Kao i mrežno opterećenje, procesorsko opterećenje direktorijskog protokola teži konstantnoj vrijednosti uz neznatni porast na početku krivulje. Na

temelju formule (3.6.2) izvodi se da uz $n \to \infty$ vrijednost procesorskog opterećenja direktorijskog protokola teži k $2 \cdot (1-p) \cdot 2 \cdot (\alpha+1) \cdot t$. Navedena vrijednost uz upotrijebljene vrijednosti parametara p i α iznosi $9.6 \cdot t$.

Analitička usporedba protokola pokazuje da su Berkeley i direktorijski protokol bolji od Squid protokola, jer stvaraju manje opterećenje na mreži i na zastupničkim računalima. Promatrajući mrežno opterećenje, Berkeley protokol pokazuje značajno manji porast broja IP paketa od Squid protokola. Analizom procesorskog opterećenja uočava se da je porast opterećenja Berkeley protokola znatno bliže porastu opterećenja Squid protokola. Daljnji je nedostatak Berkeley protokola zadržavanje linearne ovisnosti opterećenja o broju zastupnika u zastupničkom sustavu što stvara zagušenje u slučaju velikog povećanja broja zastupnika u sustavu. Navedena svojstva Berkeley protokola nastaju zato što svi zastupnici u sustavu primaju i obrađuju *ICP_QUERY* poruke unatoč slanju *multicast* upita. Osim toga, zastupnici koji imaju traženi objekt šalju *ICP_HIT* odgovore koje potom obrađuje glavni zastupnik.

Za razliku od Squid i Berkeley protokola, opterećenje protokola zasnovanog na direktoriju ne ovisi o broju zastupnika u sustavu. Budući da direktorijski protokol zbog održavanja direktorija stvara veće optećenje, pri malom broju zastupnika njegovo je opterećenje veće od opterećenja ostalih protokola. Osim toga, uz mali broj zastupnika u sustavu direktorijski protokol ne ograničava slanje upita susjednim zastupnicima i njihovu obradu. Porastom broja zastupnika, do izražaja dolazi smanjenje skupa susjednih zastupnika kojima se šalje *ICP_QUERY* upit. Opterećenje dodatno smanjuje uporaba parametra α koji ograničava skup susjednih zastupnika kojima se šalju upiti.

Na temelju prikazane analitičke usporedbe tri protokola, moguće je donijeti zaključak o svojstvima proširivosti pojedinih protokola s obzirom na broj zastupnika u sustavu. Protokol Squid nije pogodan za proširivanje, odnosno, za sustave s velikim brojem zastupnika. Temeljni nedostatak navedenog protokola je jaki linearni porast opterećenja pri povećanju broja zastupnika u raspodijeljenom zastupničkom sustavu. Berkeley protokol smanjuje mrežno opterećenje, ali ne ublažava značajno procesorsko opterećenje zastupničkih računala. Upravljački protokol zasnovan na direktoriju smanjuje oba opterećenja i održava ih konstantnim. Neovisnost opterećenja o broju zastupnika značajna je prednost direktorijskog protokola iako je njegovo opterećenje pri malom broju zastupnika veće od opterećenja Squid i Berkeley protokola. Opisane prednosti i nedostaci tri uspoređena protokola, navode na zaključak da je protokol zasnovan na direktoriju prikladan za uporabu u sustavima s velikim brojem zastupnika. Sustavi s malim brojem zastupnika trebali bi koristiti protokol Squid. Razlog za uporabu Squid protokola u malim sustavima njegova je raširenost i jednostavnost, te činjenica da Berkeley protokol pri malom broju zastupnika nema znatno manje opterećenje od Squid protokola, a znatno je složeniji za uporabu.

Opisana analitička usporedba Squid, Berkeley i direktorijskog protokola izvedena je na štetu direktorijskog protokola. Zbog jednostavnosti, zanemareno je smanjenje opterećenja direktorijskog protokola ako nijedan zastupnik u sustavu nema traženi objekt. Zanemareno je i ubrzanje u odnosu na

protokole Squid i Berkeley koje pri tome nastaje. Osim toga, pretpostavljeno je da vrijednost parametra y iznosi α iako je y pri malom broju zastupnika u sustavu manji od α. Konačno, vrijednost je parametra q u stvarnim sustavima manja. Navedene vrijednosti parametara y i α povećavaju procjenu opterećenja koje stvara direktorijski protokol. Berkeley protokolu je zbog jednostavnosti dana prednost, jer su zanemareni problemi ostvarenja *multicast* mehanizma slanja poruka opisani u odjeljku 3.3.3. Izvedeno opterećenje Berkeley protokola bilo bi veće uz pretpostavku da jedna *multicast* poruka stvara više IP paketa, kao što je slučaj u većini stvarnih sustava.

Najznačajniji nedostatak prikazane analitičke usporedbe je da rezultati nisu preneseni u vremensku domenu. Osnovna je mjera usporedbe svojstava upravljačkih protokola raspodijeljene priručne memorije prosječno kašnjenje po korisničkom zahtjevu. Veliki skup parametara i nepredvidivih kombinacija čine izračun prosječnog kašnjenja složenim i nepreciznim. Točan analitički izraz za kašnjenje stoga je teško izvesti. Dio članka [28] čini prikaz analitičkih rezultata usporedbe Squid, Berkeley i direktorijskog protokola. Analitički rezultati u navedenom članku izraženi su u obliku prosječnog kašnjenja po korisničkom zahtjevu. Prosječno kašnjenje dobiveno je izračunom opterećenja koji se temelji na opterećenju upravljačkih protokola i opterećenju prijenosa HTTP objekata. Uporabom M/G/1 modela, dobiveno opterećenje preneseno je u vremensku domenu. Rezultati dobiveni na opisani način dovode do zaključaka koji su istovjetni zaključcima opisanim u ovom odjeljku.

Analitička usporedba protokola ne uzima u obzir i ne predviđa sve parametre koji utječu na svojstva protokola. Primjer parametra koji je teško procijeniti je parametar p, odnosno vjerojatnost da određeni zastupnik ima traženi objekt. Navedena vjerojatnost mijenja se u ovisnosti o upotrijebljenom protokolu i o okolnostima u kojima se nalazi zastupnički sustav. Jedini je način, koji obuhvaća sve parametre koji utiču na svojstva upravljačkog protokola, mjerenje kašnjenja u stvarnim sustavima. Ako se pravilno izvede, onda navedeni pristup daje znatno točnije rezultate od analitičkog pristupa. Točne rezultate daje samo izravno mjerenje u sustavu koji je u stvarnoj, svakodnevnoj uporabi. Budući da za većinu istraživanja mjerenje u stvarnom sustavu nije ostvarivo, rezultati dobiveni mjerenjem vjerodostojniji su od analitičkih rezultata, ali ne odgovaraju u potpunosti stvarnim sustavima. Rezultati mjerenja svojstava simuliranog sustava izvedenog s protokolom Squid i direktorijskim protokolom opisani su u poglavlju 4.

2.8. Ostali upravljački protokoli i istraživanja

Na području upravljačkih protokola za raspodijeljene sustave priručne memorije postoji velik broj istraživanja. Uz tri prethodno opisana, postoje i druga rješenja za povezivanje zastupnika u raspodijeljeni sustav. Samo neka rješenja primjenjuju se u stvarnim sustavima. U primjeni su najčešći Squid protokol koji je raširen zbog popularnosti Squida, te CARP protokol koji je podržan i u Squidu, ali je raširen zbog ostvarenja u Microsoft Proxy Serveru.

2.8.1. RPD-SD protokol

Autori članka [15] istražuju problem proširivosti raspodijeljenih zastupničkih sustava s obzirom na broj zastupnika i utjecaj hijerarhijske strukture na opća svojstva sustava. Na početku istraživanja, proučavana su svojstva zastupničke sastavnice sustava Harvest i donesen je zaključak da raspodijeljeni sustav građen oko Harvesta ima loša svojstva sa stajališta proširivosti s obzirom na broj zastupnika u sustavu. Autori članka [15] uočavaju da proširivost ograničava neučinkovit protokol za pronalaženje susjednog zastupnika koji ima presliku traženog objekta. Stoga je u navedenom radu izgrađen i ispitan alternativni sustav koji se temelji na protokolu nazvanom CSD.

Skraćenica CSD dolazi od izraza *Central Synchronous Directory* što ukazuje na činjenicu da se u sustavu definira centralizirani i usklađeni direktorij. CSD protokol radi tako da se u sustav postavlja posebno računalo na kojem se izvodi središnja aplikacija za održavanje direktorija. Na središnjem računalu istodobno može raditi i zastupnik, ali u tom se slučaju usporavaju veći sustavi. U direktorij se za svaki objekt sprema adresa zastupnika koji navedeni objekt ima u svojoj lokalnoj priručnoj memoriji. Nakon što od korisnika primi zahtjev za nekim objektom, glavni zastupnik provjerava da li traženi objekt postoji u lokalnoj priručnoj memoriji. Ako objekt nije pronađen lokalno, onda glavni zastupnik šalje upit središnjem direktoriju. Ako u središnjem direktoriju postoji zapis da se traženi objekt nalazi na nekom od susjednih zastupnika, onda glavni zastupnik dobija *REDIRECT* poruku s adresom navedenog susjednog zastupnika. Glavni zastupnik potom dohvaća traženi objekt od navedenog susjednog zastupnika. Ako traženi objekt nije pronađen u direktoriju, onda se glavnom zastupniku šalje *MISS* poruka nakon čega se objekt izravno dohvaća od poslužitelja.

U svrhu održavanja direktorija, zastupnici tijekom svakog dohvata objekta izvan zastupničkog sustava ili izbacivanja objekta iz lokalne memorije, razmjenjuju određeni broj poruka sa središnjim računalom. Uočljiv je najveći nedostatak CSD protokola. Iako je protokol dizajniran s ciljem postizanja proširivosti sustava s obzirom na broj zastupnika, on u osnovi nije proširiv. Postojanje središnjeg održavatelja direktorija onemogućava proširivost opisanog sustava. Tijekom većeg porasta broja zastupnika u sustavu središnji direktorij postaje preopterećen. Uporabom bržeg računala u svrhu održavanja središnjeg direktorija samo se pomiče prag preopterećenja, ali se ne postiže neograničena proširivost. Dodatni nedostatak CSD protokola koji autori nisu naveli je da središnji direktorij glavnom zastupniku šalje adresu samo jednog susjednog zastupnika koji sadrži traženi objekt. Ako je navedeni objekt u međuvremenu izbačen iz lokalne memorije susjednog zastupnika i ako neki drugi susjedni zastupnik ima traženi objekt, onda dolazi do lažnog promašaja. Lažni promašaj je pojava da glavni zastupnik zaključi da u sustavu ne postoji preslika traženog objekta pa objekt dohvaća od poslužitelja. Na navedni način, lažni promašaj uzrokuje nepotrebno dodatno kašnjenje. Sami autori protokola CSD uočili su da protokol nije proširiv u onoj mjeri koju su očekivali pa predlažu novi protokol.

U cilju poboljšanja CSD protokola razvijen je RPD-SD protokol također opisan u članku [15]. Skraćenica RPD-SD dolazi od naziva *Replicated Partial Directory of Shared Documents*.

Proučavanjem prometa u raspodijeljenom zastupničkom sustavu, autori protokola su zaključili da je potrebno uvesti dva nova pojma, pojam dijeljenih zahtjeva i pojam dijeljenih objekata. Dijeljeni zahtjev je zahtjev za objektom koji je prethodno tražen. Dijeljeni zahtjev u pouzdanom raspodijeljenom zastupničkom sustavu rezultira pogotkom. Dijeljeni objekt je objekt koji je bio tražen u više korisničkih zahtjeva. Autori su promatranjem zastupničkih sustava utvrdili da je 70% zahtjeva dijeljeno, ali je istovremeno samo 30% objekata dijeljeno. Iako većina korisničkih zahtjeva rezultira pogotkom, navedeni pogotci ostvaruju se na znatno manjem skupu objekata.

Autori smatraju da je na temelju navedenih saznanja moguće smanjiti promet uzrokovan upravljačkim protokolom raspodijeljene priručne memorije. RPD-SD protokol ne sprema direktorij samo na središnjem računalu, već svaki zastupnik sadrži presliku dijela informacija iz direktorija. Informacije zapisane u preslikama mogu se ponavljati na različitim zastupnicima. Uporabom preslikanog direktorija uklanja se problem opterećenja središnjeg direktorija. Radi smanjenja veličine i troškova održavanja preslikanih informacija RPD-SD protokol koristi djelomične preslike. Djelomične preslike nastaju izbacivanjem informacija iz direktorija uz nastojanje da se ne gube korisni sadržaji. Na temelju prije opisanoga proučavanja dijeljenja objekata, autori predlažu da replicirani djelomični direktorij treba sadržavati samo informacije o dijeljenim objektima.

U svrhu određivanja objekata koji su dijeljeni, RPD-SD protokol koristi središnji poslužitelj. Središnji poslužitelj, za razliku od CSD protokola, ne daje informaciju o sadržaju direktorija, već održava listu dijeljenih objekata. RPD-SD protokol radi tako da glavni zastupnik tijekom izvođenja algoritma traženja provjerava samo vlastiti lokalni direktorij i na temelju postojećih informacija odlučuje da li objekt dohvatiti od susjednog zastupnika ili izravno od poslužitelja. Budući da je potrebno održavati jednoznačnost informacija u lokalnoj djelomičnoj preslici direktorija, zastupnik se periodički povezuje sa središnjim poslužiteljem. Tijekom veze sa središnjim poslužiteljem, zastupnik poslužitelju šalje podatke o objektima koje je dohvatio i izbacio, te preuzima dopune informacija za djelomično preslikani direktorij. Središnji poslužitelj održava središnju presliku direktorija, prima informacije od zastupnika i šalje im nadopune. Na temelju podataka o dijeljenosti objekata središnji poslužitelj dodaje i izbacuje pojedine objekte iz preslikanog djelomičnog direktorija.

Iako autori smatraju da RPD-SD protokol rješava nedostatak proširivosti CSD protokola, navedeni problem na opisani način nije značajno smanjen. RPD-SD protokol, kao i CSD protokol, nije proširiv zbog prisutnosti središnjeg poslužitelja kojeg je moguće preopteretiti ako se prijeđe određeni broj zastupnika u sustavu. Direktorij s djelomičnim preslikama sadrži manje informacija od punog direktorija, ali njegova veličina nije znatno manja. Zbog toga su i troškovi održavanja direktorija s djelomičnim preslikama na svakom zastupniku značajni. Opterećenje zastupnika povećava potreba za slanjem podataka o svim traženim objektima središnjem poslužitelju. Središnji poslužitelj značajno je opterećen primanjem i slanjem podataka o svim promjenama u direktoriju. Osim toga, središnji poslužitelj odlučuje da li je pojedini objekt postao ili prestao biti dijeljen. Navedena obrada u zahtijeva više procesorskog vremena, nego obrada zahtjeva koja se izvodi u središnjem direktoriju. Daljnji je nedostatak RPD-SD protokola složeno programsko ostvarenje. Konačno, RPD-SD protokol osjetljiv

je na gubitak informacija o objektima u zastupničkom sustavu što smanjuje udio pogodaka u sustav raspodijeljene priručne memorije. Do gubitka informacija dolazi zbog periodičke razmjene informacija u direktoriju i zbog uvođenja pojma dijeljena objekata. Zbog kašnjenja prijenosa informacija od nekog susjednog zastupnika, moguće je da glavni zastupnik u svojoj replici direktorija nema podatak o dijeljenom objektu kojeg traži. Osim toga, svi zahtjevi obrađeni prije nego određeni objekt postaje dijeljen u RPD-SD protokolu bit će promašaji, jer do trenutka postavljanja navedenih zahtjeva objekt nije bio u skupu dijeljenih objekata. Zbog opisanih nedostataka RPD-SD protokol ne smatra se proširivim s obzirom na broj zastupnika u sustavu.

2.8.2. Summary Cache

Sustav raspodijeljene priručne memorije opisan u članku [16] dobio je ime po sažetcima (engl. *summary*) koje drži svaki zastupnik u sustavu. Sažetak je u *Summary Cache* sustavu lokalno spremljen direktorij. U sažetcima su zapisane informacije o sadržaju lokalnih priručnih memorija svih susjednih zastupnika. Navedeni se pristup u članku [16] naziva direktorijskim, jer koristi popis smještaja pojedinih objekata u raspodijeljenom zastupničkom sustavu. *Summary Cache* sličan je Squid protokolu, a značajne razlike pristune su samo u algoritmu traženja. Nakon što dobije zahtjev od korisnika i utvrdi da se traženi objekt ne nalazi u lokalnoj memoriji, glavni zastupnik provjerava sadržaj sažetka. Glavni zastupnik na temelju sažetka određuje koji susjedni zastupnici imaju traženi objekt i potom šalje *ICP_QUERY* upite odgovarajućim zastupnicima. Slanjem *ICP_QUERY* upita, *Summary Cache* povećava vjerojatnost pronalaženja objekata u lokalnim priručnim memorijama susjednih zastupnika s obzirom na mogućnost pojave grešaka u sažetcima.

Autori *Summary Cache* protokola smatraju da je proširivost dobivene arhitekture osigurana činjenicom da nije nužno da sažeci precizno opisuju trenutno stanje lokalne memorije susjednih zastupnika. Jedini uvjet koji sažetci trebaju ispunjavati je da sadrže nadskup skupa objekata koje pojedini zastupnici imaju u svojim lokalnim memorijama. U tom slučaju, prema autorima protokola, mogu se pojaviti samo dvije vrste "grešaka": lažni promašaji i lažni pogotci. Lažni promašaj nastaje ako neki susjedni zastupnik ima traženi objekt, a glavni zastupnik na temelju svog sažetka, koji nije osvježen, zaključi da nijedan od susjednih zastupnika nema traženi objekt. Lažni pogodak nastaje ako susjedni zastupnik izbaci traženi objekt iz lokalne memorije, a glavni zastupnik zbog kašnjenja pri održavanju sažetaka zaključi da navedeni zastupnik još uvijek ima traženi objekt.

U svrhu smanjivanja memorijskih kapaciteta potrebnih za pohranu sažetaka, autori *Summary Cache* protokola predlažu uporabu *Bloom* filtera. Radi se o matematičkoj metodi za određivanje elemenata nekog skupa opisanoj u članku [17]. Bloom filteri djeluju tako da definiraju skup funkcija raspršenog adresiranja i vektor bitova. Za svaki element iz skupa računaju se vrijednosti *hash* funkcija i postavljaju odgovarajući bitovi u vektoru. Osnovne prednosti Bloom filtera su malo zauzeće memorijskog prostora i prilagodljivost memorijskog zauzeća potrebama. Ako se odabere manji broj funkcija raspršenog adresiranja i vektor s manje bitova, onda češće dolazi do preklapanja vrijednosti za različite objekte. S obzirom na složenost funkcija raspršenog adresiranja, njihov broj je stalan, a

mijenja se samo veličina vektora bitova. Uporabom navedenog matematičkog aparata, autori protokola smanjuju memorijske potrebe za pohranu sažetaka. Memorijske potrebe po preporukama autora protokola iznose 1-10% ukupne veličine radne memorije zastupnika.

Kao dva čimbenika koji ograničavaju proširivost sustava s obzirom na broj zastupnika, autori *Summary Cachea* navode dodatni promet uzrokovan upravljačkim protokolom raspodijeljene priručne memorije i memorijske kapacitete potrebne za spremanje sažetaka. Pri tome zanemaruju dodatno vrijeme obrade na procesoru uzrokovano izvođenjem upravljačkog protokola. Radi smanjivanja dodatnog prometa, autori odlučuju upotrijebiti periodičko obnavljanje sažetaka. Svaki zastupnik periodički šalje sadržaj svoje lokalne memorije svim susjednim zastupnicima. U svom radu [16], autori proučavaju odnos između vremena osvježavanja i udjela pogodaka u raspodijeljeni zastupnički sustav. Navedene vrijednosti su obrnuto proporcionalne, odnosno, čim je veći razmak između pojedinih osvježavanja sažetaka tim je manji udio pogodaka u sustavu.

Primjetna je nelogičnost u analizi svojstava *Summary Cachea*. Autori navode uvjet uspješnosti *Summary Cache* protokola kojim se traži da je sažetci sadržavaju nadskup skupa svih objekata koje pojedini zastupnici imaju u svojim lokalnim priručnim memorijama. Budući da je održavanje sažetaka podložno određenom kašnjenju, sažetci ne ispunjavaju navedeni uvjet što smanjuje učinkovitost protokola.

Osim neispunjavanja osnovnog uvjeta, opisana arhitektura ima više nedostataka. Jedan je nedostatak zauzeće memorijskog prostora i zahtjevno održavanje sažetaka. Računanje vrijednosti četiri *hash* funkcije za svaki objekt, kao što autori predlažu, zahtijeva značajne procesorske kapacitete. Osim toga, potrebno je na svim računalima održavati sažetke za sve susjedne zastupnike. Konačno, ako se žele smanjiti memorijski kapaciteti, onda je potrebno da je vektor bitova Bloom filtera mali što uzrokuje često preklapanje pojedinih objekata. Posljedica preklapanja je povećana količina lažnih pogodaka u raspodijeljenom zastupničkom sustavu.

Daljnji je nedostatak smanjenje točnosti sažetaka i udjela pogodaka. Do navedenih pojava dolazi zbog smanjivanja memorijskih zahtjeva i prorjeđivanja poruka koje prenose informacije potrebne za osvježavanje direktorija. Navedeni mehanizmi ne osiguravaju proširivost koja je ugrožena činjenicom da s porastom broja zastupnika raste opterećenje pojedinih zastupnika, jer oni osvježavaju sažetke svih susjeda. Tijekom osvježavanja sažetaka koristi se logički *multicast* mehanizam. Unatoč uporabi logičkog *multicasta*, zastupnici su opterećeni neprestanim slanjem i obradama poruka osvježavanja koje šalju i primaju od susjednih zastupnika. Svojstva *Summary Cache* protokola dodatno kvari prijedlog autora da se TCP protokol koristi za osvježavanje sažetaka. TCP protokol nije pogodan za primjene u kojima je važno postići malo kašnjenje.

S obzirom na složenost odabira kvalitetnih funkcija raspršenog adresiranja, te složenost i sporost osvježavanja sažetaka, *Summary Cache* protokol nije proširiv. Protokol za upravljanje raspodijeljenim zastupničkim sustavom sličan *Summary Cacheu* programski je ostvaren, osim u probnim sustavima, i u Squid sustavu. Naziv navedenog protokola je *CacheDigest* i opisan je u dokumentu [29].

2.8.3. Hint Cache

Sustav sličan prethodno opisanom sustavu *Summary Cache* predlažu autori rada [18]. Pod nazivom *Hint Cache* autori stvaraju sustav u kojem svaki zastupnik sadrži tablicu u kojoj su zapisani svi objekti koji se nalaze u sustavu. Za svaki objekt zapisana je informacija koji najbliži zastupnik sadrži presliku tog objekta. Navedena tablica u osnovi je direktorij s podacima o sadržaju lokalnih priručnih memorija zastupnika. Osnovne su promjene i u *Hint Cache* sustavu ograničene na algoritam traženja. Glavni zastupnik nakon primanja zahtjeva od korisnika provjerava da li se objekt nalazi u lokalnoj memoriji. Ako objekt nije pronađen u lokalnoj memoriji, onda glavni zastupnik provjerava tablicu savjeta (engl. *hint*). Ako za traženi objekt u tablici postoji zapis, onda glavni zastupnik čita adresu pripadnog zastupnika i šalje mu *ICP_QUERY* upit.

Dodatno unapređenje koje autori predlažu odnosi se na hijerarhiju zastupnika. Predložena je *push* tehnologija kojom se objekti prenose od zastupnika više razine na zastupnike niže razine. Navedena tehnologija djeluje tako da zastupnik više razine za svaki objekt vodi popis podređenih zastupnika koji su ga tražili. *Push* tehnologija se aktivira ako neki od podređenih zastupnika zatraži objekt, a objekt više nije valjan. Nadređeni zastupnik potom dohvaća valjanu presliku traženog objekta i objekt šalje tražitelju i svim podređenim zastupnicima koji su navedeni objekt prethodno tražili.

Za programsko ostvarenje direktorija autori koriste tehnike raspršenog adresiranja i 64-bitne MD5 ključeve. MD5 ključevi su standardizirani način pretvaranja poruka promjenjive duljine u vrijednost fiksne duljine i opisani su u dokumentu [19]. *Hint Cache* i Squid sustav koriste MD5 ključeve za pretvaranje URL-a u 64-bitnu informaciju. Vjerojatnost da se dva različita URL-a preslikaju u isti MD5 ključ iznimno je mala. Svaki zapis u direktoriju sastoji se od MD5 ključa i identifikacijskog broja susjednog zastupnika. Sadržaj dobivenog direktorija periodički se osvježava. Poruke osvježavanja šalju se svim susjednim zastupnicima i nose podatke o promjenama u lokalnoj memoriji jednog zastupnika. Slično radu [16], autori *Hint Cachea* analiziraju optimalni interval za slanje novih informacija susjedima, te dolaze do vrijednosti od nekoliko minuta.

Nedostaci opisanog sustava raspodijeljene priručne memorije slični su nedostacima *Summary Cachea*. Značajni je nedostatak *Hint Cache* protokola održavanje direktorija. Slanje novih informacija svakih nekoliko minuta u sustavima sa više desetaka zastupnika praktički znači da svaki zastupnik prima navedene informacije od susjeda svakih nekoliko sekundi. Osim toga, svaki zastupnik šalje određene informacije svim susjednim zastupnicima što stvara značajno opterećenje i mrežnih i računalnih resursa, pogotovo ako se koristi *logički multicast*. *Hint Cache* protokol ima određene prednosti u odnosu na *Summary Cache* poput izbacivanja složenih funkcija raspršenog adresiranja i promjene strukture direktorija tako da pokazuje jednu lokaciju za svaki objekt umjesto da pokazuje sadržaj lokalnih memorija za sve susjedne zastupnike. Prednost spremanja samo jedne lokacije je smanjivanje memorijskih potreba upravljačkog protokola. Međutim, budući da se koristi UDP prijenos poruka osvježavanja, moguć je gubitak informacija o sadržaju lokalnih memorija. Kao posljedica, raste vjerojatnost da zastupnik spremljen u direktoriju nema traženi objekt. S obzirom da se objekt

pokušava dohvatiti od samo jednog susjednog zastupnika, smanjen je udio pogodaka u raspodijeljeni zastupnički sustav u odnosu na sustav koji ima popis svih zastupnika koji sadrže određeni objekt.

Push tehnologija, koju autori *Hint Cache* sustava predlažu, ima nekoliko nedostataka. Navedena tehnologija troši dodatne resurse za održavanje popisa podređenih zastupnika i za slanje objekata istima. Iako se ovaj rad ne bavi problemima hijerarhijskih pričuvnih memorija, njihova je korisnost upitna. Time je korisnost *push* tehnologije značajno smanjenja. Ako se u obzir uzmu istraživanja koja pokazuju da *push* tehnologija stvara previše nepotrebnih slanja objekata i činjenica da je za njeno programsko ostvarenje potrebno izvršiti značajne izmjene na zastupničkim sustavima, onda je očito da su nedostaci *push* tehnologije veći od njenih prednosti.

Iako *Hint Cache* sustav ima manje memorijske potrebe i stvara nešto manje opterećenje u odnosu na *Summary Cache* sustav, zbog opisanih problema *Hint Cachea* također ne predstavlja kvalitetno rješenje za osiguranje proširivosti sustava s obzirom na broj zastupnika.

2.8.4. CARP

Cache Array Routing Protocol (CARP) opisan u dokumentu [14] razlikuje se od svih dosad opisanih rješenja. Značajne razlike između Squida i CARP-a pojavljuju se u algoritmu traženja, algoritmu odlučivanja i algoritmu uvišestručavanja. Određene razlike moguće su i u algoritmu pristupa. CARP sustavi pomoću raspršenog adresiranja dijele skup svih URL-ova na podskupove. Dobiveni podskupovi pridjeljeni su pojedinim zastupnicima. Kad glavni zastupnik dobije zahtjev, on izračunava koji susjedni zastupnik je zadužen za traženi objekt i potom mu šalje zahtjev. Nakon što dohvati traženi objekt od susjednog zastupnika, glavni zastupnik objekt prosljeđuje korisniku i ne sprema ga lokalno. Ako je glavni zastupnik zadužen za traženi objekt i u lokalnoj memoriji ne postoji valjana preslika, onda glavni zastupnik izravno dohvaća objekt od poslužitelja i sprema ga lokalno. Ako kontaktirani susjedni zastupnik nije imao traženi objekt, onda glavni zastupnik čeka da ga navedeni susjedni zastupnik dohvati od poslužitelja i potom ga proslijeđuje korisniku.

CARP sustav djeluje tako da s unaprijed određenog URL-a dohvati tablicu koja navodi i opisuje sve zastupnike u sustavu raspodijeljene priručne memorije. U tablici se navodi i vrijeme isteka valjanosti tablice čime se postiže određena mogućnost dinamičke promjene sustava. Tablica osim popisa zastupnika sadrži dodatne informacije poput opterećenosti zastupnika i veličine lokalne memorije. Funkcija raspršenog adresiranja je jednostavna. Računa se istovremeno za URL traženog objekta i za svakog zastupnika u sustavu. *Hash* vrijednost URL-a računa se samo pomoću zbrajanja i rotacije bitova u 32-bitnom cijelom broju za svaki znak iz URL-a. Odgovarajuća vrijednost za zastupnike računa se na isti način, s tim da se na kraj još dodaje jedno množenje s konstantom i rotacija bitova. Navedeni dodatak uveden je zbog sličnosti imena zastupnika. U opisani postupak računanja *hash* vrijednosti zastupnika moguće je dodati i korekciju zbog opterećenja. Opterećenje se zadaje kao konstanta i daje procjenu mogućnosti pojedinog zastupničkog računala. Nakon što su *hash* vrijednosti za URL i zastupnika izračunate, one se kombiniraju kroz *XOR* funkciju, a rezultat se množi s

konstantom i rotira. Dobiveni rezultat daje značaj pojedinog zastupnika za dani URL. Glavni zastupnik odabire onog susjednog zastupnika koji ima najveći značaj.

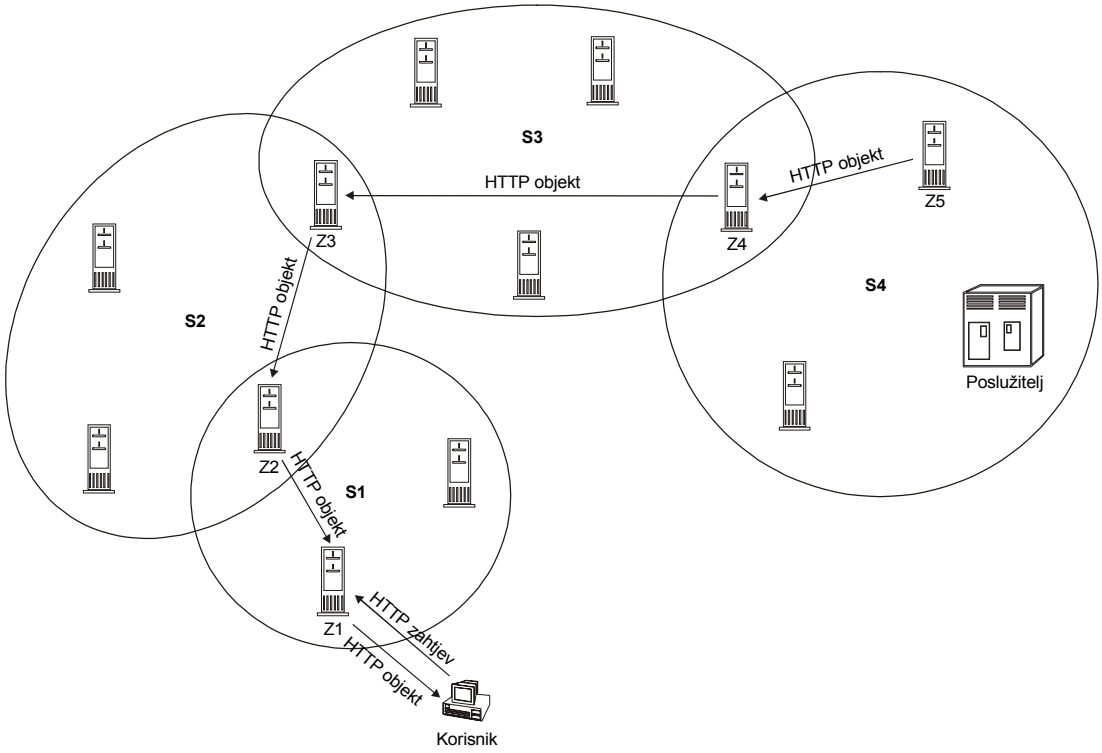
Opisani sustav ima više prednosti od kojih su najznačajnije brz algoritam traženja i povećanje iskoristivosti raspodijeljenog zastupničkog sustava. Traženje susjeda koji ima presliku traženog objekta brzo je zbog jednostavne funkcije raspršenog adresiranja. Iskoristivost sustava povećana je time što se u sustavu ne pojavljuju višestruke preslike istog objekta, odnosno, jedan objekt će biti u lokalnoj memoriji najviše jednog zastupnika u cijelom sustavu. Manje značajna prednost je jednostavnost algoritma odlučivanja. Dodatno ubrzanje CARP sustava postiže se uporabom klijenata koji su svjesni CARP sustava. Svjesni korisnik izvodi algoritam pristupa koji kao glavnog zastupnika odabire zastupnika koji je zadužen za traženi URL. Opisanim algoritmom smanjuje se broj skokova koje prolazi objekt na putu prema korisniku i skraćuje prosječno kašnjenje po zahtjevu. U prednosti sustava CARP, ubraja se i činjenica da je dodatni promet kojeg stvara upravljački protokol mali i sastoji se samo od povremenih čitanja tablice zastupnika. Učestalost čitanja tablice zastupnika sasvim je prilagodljiva i ovisi o stupnju dinamičnosti sustava koji se želi postići. Pod opisanim uvjetima CARP protokol je proširiv i pogodan za uporabu u sustavima s velikim brojem zastupnika.

Nedostaci CARP sustava ovise o specifičnostima pojedinih organizacija u kojima se on koristi. Temeljni nedostatak je činjenica da se jednom dohvaćeni objekt ne sprema na glavnom zastupniku tako da će budući zahtjevi za tim objektom uvijek biti dohvaćani posredno preko susjednih zastupnika. Ako je veza sa susjednim zastupnikom nešto sporija, onda opisani algoritam uvišestručavanja značajno povećava kašnjenje. Ostali problemi proizlaze iz funkcije raspršenog adresiranja. Svi zastupnici u sustavu koriste istu funkciju što ograničava dinamičnost sustava. Za promjenu *hash* funkcije potrebno je izmijeniti izvorni kôd zastupničkog programa. Potrebno je pažljivo osmisliti funkciju raspršenog adresiranja zbog postizanja ravnomjernog rasporeda objekata i zahtjeva bez obzira na broj zastupnika u sustavu. Odabirom pogrešne funkcije raspršivanja pojedini zastupnici mogu biti preopterećeni brojnim zahtjevima. Moguć je i slučaj da su neki zastupnici prisiljeni izbacivati objekte zbog nedostatka prostora u lokalnoj memoriji, dok su istovremeno ostali zastupnici neopterećeni ili njihova lokalna memorija nije potpuno iskorištena. Problem funkcije raspršivanja povećava standard [14] koji određuje koja se funkcija raspršenog adresiranja koristi. Pritom je odabrana jednostavna funkcija koja se brzo izračunava. Navedena funkcija ne ispunjava istovremeno zahtjev za ravnomjernim rasporedom objekata između zastupnika i zahtjev za ravnomjernim rasporedom opterećenja. CARP protokol temeljni je protokol Microsoft Proxy Server zastupnika, a podržan je kao dodatna mogućnost i u Squid sustavu.

2.8.5. AWC

Projekt Adaptive Web Caching (AWC) započet je na Computer Science odjelu sveučilišta UCLA u SAD-u. AWC je sustav dinamičkog samouređivanja zastupnika koji je opisan u dva rada [20,21]. Temeljna zamisao AWC sustava je da zastupnici tvore virtualne zastupničke skupine. Svaki zastupnik član je jedne ili više grupa. Ovisno o trenutnom stanju sustava, opterećenju, raspoloživosti zastupnika

i poslužitelja i korisničkim potrebama, zastupnici mogu dinamički i samostalno mijenjati svoje članstvo u pojedinim grupama. Zastupnici unutar jedne skupine međusobno komuniciraju preko protokola za upravljanje raspodijeljenim zastupničkim sustavima. Navedeni protokol nije strogo definiran u opisu AWC sustava, nego autori preporučaju uporabu protokola Squid ili Berkeley. Upotrijebiti se mogu i neki od novijih protokola opisanih u odjeljku 2.8 ili direktorijski protokol čije je programsko ostvarenje i mjerenje svojstava tema ovog magistarskog rada. Autori AWC sustava predviđaju uporabu *multicast* mehanizma za slanje poruka smatrajući da on osigurava malo opterećenje sustava i malo kašnjenje zahtjeva. Budući da su svjesni problema ostvarenja *multicast* mehanizma koji su opisani u odjeljku 2.3.3, autori AWC-a preporučaju uporabu Squid protokola dok se ne razvije učinkovitiji protokol.



Slika 17: Dohvat objekta u AWC sustavu

Slika 17 prikazuje primjer dohvata objekta u AWC sustavu. Dohvat započinje povezivanjem korisnika na virtualnu zastupničku skupinu S1. Korisnik postavlja zahtjev svom glavnom zastupniku Z1 koji prosljeđuje upit svim susjednim zastupnicima unutar virtualne skupine. Zastupnici Z2, Z3 i Z4 članovi su više skupina i služe kao veza između skupina prenoseći dobiveni upit u sve skupine kojih su članovi. Virtualna zastupnička skupina S4 povezana je s poslužiteljem koji izvorno ima traženi objekt. U skupini S4 velika je vjerojatnost da jedan od zastupnika već ima traženi objekt, a na slici 17 radi se o zastupniku Z5. Zastupnik Z4 dohvaća objekt od zastupnika Z5, te ga time prenosi u skupinu S3. Objekt potom dohvaćaju zastupnici Z3 i Z2 prenoseći objekt u skupine S2 i S1. Objekt, konačno, dohvaća zastupnik Z1 i šalje ga korisniku.

Dva su temeljna nedostatka opisanog sustava. Prvi nedostatak je potencijalno velik broj skokova između različitih virtualnih zastupničkih skupina. Termin skok označava prijelaz objekta iz jedne virtualne skupine u drugu, odnosno prijenos objekta između zastupnika koji služe kao veze između virtualnih skupina. Svaki skok donosi dodatno kašnjenje tijekom dohvata objekta. Drugi nedostatak čini predloženi upravljački protokol koji se zasniva na *multicast* mehanizmu za kojeg su i sami autori AWC-a svjesni da još nije u općoj uporabi na Internetu. Budući da je moguće navedeni upravljački protokol jednostavno zamijeniti nekim drugim protokolom, navedeni je nedostatak relativno lako ispraviti.

AWC nudi zanimljivu i potencijalno korisnu mogućnost dinamičkog uređivanja skupina zastupnika. Razvoj opisanog sustava iznimno je složen zbog potrebe razvoja niza protokola koji bi AWC sustavu osiguravali dinamičnost. Osnovni je problem AWC sustava razvoj, jer navedeni protokoli do danas nisu razvijeni. Problem razvoja je značajan, jer je AWC projekt, prema informacijama s Web stranica projekta privremeno napušten.

Nijedan upravljački protokol raspodijeljene priručne memorije opisan u ovom odjeljku ne osigurava proširivost sustava s obzirom na broj zastupnika. Zbog toga direktorijski protokol u ovom trenutku pruža najbolje rješenje za velike raspodijeljene zastupničke sustave koji se sastoje od više desetaka ili stotina zastupnika. Proučavanje znanstvenih istraživanja ipak omogućava uočavanje problema na koje su naišli drugi istraživači. Na temelju njihovih iskustava moguće je poboljšati vlastita rješenja.

3. Programsko ostvarenje protokola zasnovanog na direktoriju

3.1. Zastupnički sustav Squid

Kao osnova programskog ostvarenja upravljačkog protokola raspodijeljene priručne memorije zasnovanog na direktoriju odabran je zastupnički sustav Squid [8]. Postoji niz razloga za odabir sustava Squid. Zastupnički sustav Squid ima vlastiti upravljački protokol raspodijeljene memorije koji je učinkovito programski ostvaren. Squid je potpuno besplatan sustav koji spada u grupu proizvoda otvorenog kôda. Izvorni kôd Squid sustava svima je dostupan i uz određena minimalna ograničenja moguće ga je slobodno mijenjati. Navedena ograničenja proizlaze iz licence GNU General Public License pod kojom se Squid dostavlja korisnicima. Ostali raspoloživi zastupnički programi čiji je izvorni kôd dostupan javnosti i čija je promjena dozvoljena, uglavnom su zastarjeli ili su lošije kvalitete. Tržišna rješenja, kao druga mogućnost, ne otkrivaju izvorni kôd široj javnosti osim uz posebna odobrenja koja se dobijaju u posebnim uvjetima. Daljnji problem su vlasnička prava nakon promjene kôda tržišnih proizvoda. Stoga je za ostvarenje direktorijskog protokola jedina utemeljena alternativa uporabi Squid sustava razvoj cjelovitog zastupničkog sustava. Razvoj vlastitog sustava sa stajališta istraživanja upravljačkih protokola nije opravdan zbog dugog vremena razvoja i velikih sredstava koja su potrebna za razvoj. Osim navedenih prednosti Squida, za njegov odabir važan je i podatak da se radi o jednom od najraširenijih zastupničkih sustava koji predstavlja jaku konkurenciju tržišnim proizvodima. Squid se, između ostalog, koristi i u mnogim znanstvenim istraživanjima i radovima s područja zastupničkih sustava na Internetu.

3.1.1. Namjena Squida

Squid je zastupnički sustav za Web korisnike koji posreduje tijekom dohvata HTTP (WWW), FTP i Gopher objekata. Za zaštićeni prijenos podataka Squid podržava sigurni SSL (Secure Socket Layer) protokol. Za ostvarenje nadzora i održavanja tijekom rada sustava, Squid izvodi SNMP (Simple Network Management Protocol) protokol i održava odgovarajuću bazu podataka o radu sustava. Squid je izvorno napisan i prilagođen radu na raznim inačicama operativnog sustava Unix. Postoje određeni napori da se Squid prenese i na Windows operativni sustav. Slično operativnom sustavu Linux, u stvaranju Squida sudjeluju mnogi pojedinci. Početak projekta i daljnji nadzor razvoja kôda vodi Duane Wessels iz NLANR-a. Nositelj autorskih prava na Squid je University of California, San Diego.

Squid je namijenjen uporabi u malim i srednje velikim organizacijama. Predviđen je za postavljanje na mjestu gdje se lokalna mreža spaja s Internetom. Unatoč postojanju određenih zaštitnih, vatrozidnih (engl. *firewall*) mogućnosti u Squid sustavu, njegove mogućnosti zaštite lokalne mreže su male. Zbog toga autori Squida preporučaju korištenje u sprezi s nekim od zaštitnih vatrozidnih sustava. Na temelju opažanja dobivenih iz praktične uporabe, sustav Squid ima relativno loša svojstva proširivosti s obzirom na broj zastupnika u sustavu. Stoga se ne preporuča uporaba Squid sustava u

sustavima s velikim brojem zastupnika. Time je Squid ograničen na uporabu u organizacijama koje ne prelaze broj od nekoliko stotina korisnika, a u krajnjim slučajevima moguća je njegova uporaba za zastupanje nekoliko tisuća korisnika.

3.1.2. Svojstva Squida

Zamisao o stvaranju Squida potječe od zastupničke sastavnice sustava Harvest [5-7]. Sustav Harvest stvoren je s ciljem sveobuhvatnog rukovanja dokumentima na Internetu. Harvest uključuje pohranu i posluživanje dokumenata, njihovo indeksiranje i pretraživanje, a zastupnički dio je samo sastavnica kojom se nastojalo ubrzati pristup dokumentima. Autori Squid sustava zaključili su da je zastupnički dio Harvesta koristan u široj primjeni na Internetu pa su ga odlučili izdvojiti i unaprijediti. Jedno je od najvažnijih unapređenja zastupničkog sustava Squid upravljanje memorijom i datotečnim sustavom. U novijim inačicama Squida, sve operacije čitanja i pisanja podataka preko TCP/IP protokola, odnosno preko priključnica (engl. *socket*) izvode se pomoću neblokirajućih funkcija. Na navedeni se način postiže bolji odziv sustava i manji gubitak procesorskog vremena na razne sistemske zadaće poput prebacivanja kontrole između dretvi. Poboljšanje ostvaruje i arhitektura programskog sustava koja je zasnovana na neblokirajućim, kratkim zadaćama. Njima je omogućeno izvođenje cijelog sustava unutar samo jedne dretve. Uporaba jedne dretve za glavne zadaće daje niz prednosti. Najistaknutija prednost je manje vrijeme odziva sustava. Sustav se na opisani način ubrzava, jer nema potrebe za promjenama konteksta tijekom prelaska s jedne na drugu dretvu. Istovremeno se ne gube ni sustavski resursi potrebni za stvaranje i uklanjanje dretvi.

Daljnje je unapređenje Squida u odnosu na Harvest standardizacija ICP protokola koji definira komunikaciju unutar zastupničkog sustava. Nakon programskog ostvarenja inačice 2 ICP protokola u Squidu, on je postao i Internet standardom [9,10]. Standardizacija navedenog protokola predstavlja znatan doprinos raspodijeljenim zastupničkim sustavima, jer definira standardne načine komunikacije između zastupnika, te standardizira poruke koje zastupnici razmjenjuju. Osim poruka za komunikaciju unutar iste hijerarhijske razine, ICP protokol definira i poruke za razmjenu informacija o objektima između podređenih i nadređenih zastupnika. U definiciju protokola uključena je i komunikacija s poslužiteljem objekta i komunikacija sa zastupnicima koji ne podržavaju ICP protokol. Osim ICP protokola, Squid podržava još tri protokola za upravljanje sustavom raspodijeljene priručne memorije. Dva od navedenih protokola su *CacheDigest* i CARP opisani u odjeljcima 2.8.3 i 2.8.4. Treći podržani protokol je HTCP (HyperText Caching Protocol) [30]. HTCP protokol ima sličnu namjenu kao i ICP protokol. Osnovna je razlika u odnosu na ICP da HTCP omogućuje prenošenje dodatnih informacija o zahtjevima. Temeljem dodatnih informacija, glavni zastupnik točnije opisuje korisnički zahtjev i time dobija točniji odgovor da li susjedni zastupnik ima valjanu presliku traženog objekta. Zanimljivost HTCP protokola je da podržava *push* metodu održavanja jednoznačnosti objekata.

Za pokretanje Squid sustav potrebno je brzo zastupničko računalo koje raspolaže znatnim količinama memorije. Squid sustav za pohranu Internet objekata istodobno koristi radnu memoriju (RAM) i prostor na čvrstim diskovima. Zahtjevi na zastupničko računalo slični su zahtjevima na WWW

poslužiteljska računala. Istraživanje pristupnih svojstava WWW sustava opisano u članku [12] predlaže veličinu diskovne priručne memorije od jednog gigaokteta na svakih 100,000 zahtjeva dnevno. Diskovni prostor navedene veličine u novije doba više ne predstavlja značajan problem. Problem velikih diskovnih sustava priručne memorije nije cijena memorije, već indeksiranje podataka i brz pristup podacima. U svrhu indeksiranja podataka Squid stoga koristi radnu memoriju i razne tehnike za brzi pristup poput raspršenog adresiranja. Stoga je potrebno da radna memorija ima veliki kapacitet. Preporučena količina radne memorije iznosi od 1% do 10% veličine prostora na tvrdom disku rezerviranog za zastupnički sustav. Ovisno o veličini organizacije u kojoj se Squid sustav postavlja, potrebni su diskovi veličine više desetaka gigaokteta i radna memorija reda veličine gigaokteta.

Squid sustav sastoji se od nekoliko izvršnih programa, skripti i središnje datoteke s postavkama. Zastupnički sustav pokreće se pozivom glavnog programa nazvanog `squid`. Sve prilagodbe Squida izvode se preko središnje datoteke s postavkama pod imenom `squid.conf` koja se nalazi u `etc` podkazalu osnovnog kazala u koje je sustav postavljen. U navedenoj datoteci nalazi se veliki broj postavki koje omogućuju prilagodbu sustava raznim uvjetima. Squid na izlazu daje nadzorne poruke koje se zapisuju u dnevničke datoteke u podkazalu `logs`.

Prije pokretanja Squid sustava potrebno je pravilno prilagoditi sve postavke potrebama sustava na kojem se sustav postavlja. Navedene postavke uključuju priključak na kojem zastupnik prima HTTP zahtjeve, priključak na koji se primaju ICP poruke i adrese *multicast* grupa. Potrebno je nabrojati i zastupnike u sustavu, njihovu vrstu (susjed ili roditelj), te priključke na kojima očekuju HTTP zahtjeve i ICP poruke. U datoteci s postavkama moguće je namjestiti i vremena čekanja na određene događaje i postavke vezane uz radnu memoriju i prostor na disku koji se koristi kao lokalna memorija. Jedna od važnijih postavki tijekom raznih ispitivanja zastupničkih sustava je postavljanje razine ispisa nadzornih informacija u dnevničke datoteke. Za navedenu postavku vezane su postavke s imenima dnevničkih datoteka. U datoteci s postavkama Squida moguće je prilagoditi i vremena osvježavanja dokumenata prema njihovoj vrsti. Navedena postavka korisna je tijekom održavanja jednoznačnosti podataka u priručnoj memoriji. Jedna od većih grupa postavki odnosi se na postavljanje popisa pristupa (ACL – engl. *Access Control List*). Na temelju popisa pristupa, uz određene grupe vežu se prava dohvaćanja HTTP objekata ili dozvola odgovaranja na ICP zahtjeve. Nabrojane postavke manji su dio velikog skupa postavki, a navedene su radi prikaza mogućnosti prilagodbe Squid sustava.

Vođenje dnevničkih datoteka znatna je prednost Squida. U Squid sustav ugrađen je mehanizam upravljanja dnevnicima i funkcija za ispis nadzornih poruka koja je jednostavna za korištenje. Moguće je postavljati različite razine ispisa nadzornih poruka za svaki pojedini modul, odnosno datoteku s izvršnim kôdom. Sve nadzorne poruke vezane uz rad zastupničkog dijela sustava upisuju se u datoteku `logs/cache.log`. Postavljanjem odgovarajuće razine ispisa poruka i uporabom uobičajenih alata za rad s tekstualnim dokumentima moguće je učinkovito izdvojiti poruke vezane uz dio sustava koji se ispituje. Zbog navedenih prednosti, mehanizam nadzornih poruka Squid sustava korišten je tijekom programskog ostvarenja direktorijskog protokola i mjerenja svojstava upravljačkih

protokola raspodijeljene priručne memorije. Osim opće dnevničke datoteke `cache.log`, Squid vodi i dnevnik pristupa u datoteci `logs/access.log` i dnevnik rada sa lokalnom memorijom u datoteci `logs/store.log`.

Osnovni problem koji onemogućava uporabu samostalnih zastupničkih računala u većim sustavima je preopterećenost zastupničkog računala. Preopterećenost nastaje zbog prevelikog broja korisnika koji pristupaju jednom zastupniku. U cilju uporabe Squid sustava u većim organizacijma, predviđeno je povezivanje izoliranih zastupnika u jedan sustav pomoću Squid protokola opisanog u odjeljku 2.4. U analizi navedenoj u odjeljku 2.7 pokazano je da upravljački protokol Squid uzrokuje veliko kašnjenje po zahtjevu ako se koristi u sustavima s velikim brojem zastupnika. Osim ravnopravne suradnje zastupnika, Squid podržava i uporabu zastupničke hijerarhije radi poboljšanja svojstava zastupničkog sustava. Squid sustav razlikuje dvije vrste zastupnika, nadređene i ravnopravne zastupnike. Ravnopravni zastupnici su svi zastupnici unutar jedne zastupničke grupe. Nadređeni zastupnik najčešće je samo jedan i zajednički za cijelu grupu zastupnika iste razine.

Squid sustav omogućava djelomično poboljšanje loših svojstava upravljačkog protokola slanjem *multicast* ICP upita slično protokolu Berkeley opisanom u odjeljku 2.5. U sustavu Squid mogućuje je definirati *multicast* grupu u kojoj se zastupnik nalazi. Svi ICP upiti susjednim zastupnicima u tom se slučaju šalju *multicast* mehanizmom. Slanje *multicast* upita ne smanjuje mrežni promet i opterećenje računala u znatnoj mjeri što pokazuje analitički model iz odjeljka 2.7. Osim toga, nedostatak *multicast* mehanizma je i potreba za postavljanjem *multicast* poslužitelja. U nedostatku pravog *multicast* mehanizma Squid koristi *logički multicast* koji je samo drugo ime za *unicast* mehanizam. Problemi *multicast* mehanizma detaljnije su objašnjeni u odjeljku 2.3.3.

S programerskog stajališta Squid sustav je pogodan za proširenja i programsko ostvarenje raznih upravljačkih algoritama raspodijeljene priručne memorije. Navedena pogodnost proizlazi iz otvorenosti izvornog kôda Squida. Pogodnosti pridonosi i svojstvo da je Squid besplatan sustav i da je izmijenjeni kôd dozvoljeno koristiti u razne svrhe. Daljnja prednost Squida dolazi iz njegove djelomično modularne arhitekture. Izvorni kôd podijeljen je u više od stotinu datoteka i pisan je u programskom jeziku C. Modularnost je ostvarena semantičkim grupiranjem pojedinih dijelova Squid sustava u datoteke s izvornim kôdom. Na primjer, sve procedure za ostvarenje ICP protokola nalaze se u datoteci `icp_v2.c`, a sav kôd vezan uz pohranu objekata u memoriju i na disk nalazi se zbog svoje opsežnosti u više datoteka koje počinju prefiksom `store`.

Nedostatak je sustava Squid da ne osigurava jednostavno proširivanje gotovim modulima. Squid nema sustav dodavanja modularnih proširenja kroz pozive C funkcija iz osnovnog izvornog kôda. Zbog toga izmjene na Squidu i programsko ostvarenje upravljačkih protokola zahtijevaju detaljno proučavanje izvornog kôda i mehanizma rada Squida. Nakon proučavanja arhitekture Squid sustava potrebno je odabrati najpovoljnije mjesto umetanja novog kôda. Zbog djelomične modularne strukture Squida, u nekim slučajevima potrebno je ograničiti neke mogućnosti Squida ili promijeniti dizajn upravljačkog protokola radi omogućavanja programskog ostvarenja.

3.2. Ugradnja direktorijskog protokola u sustav Squid

Iako je definicija protokola ICP navedena u dokumentima [9,10] u znatnoj mjeri vezana za upravljački protokol raspodijeljene priručne memorije nazvan Squid, ICP je u osnovi općeniti protokol za komunikaciju između zastupnika. ICP je stoga moguće proširiti i primijeniti u kombinaciji s drugim upravljačkim protokolima. Proširenje je moguće zato što su poruke koje protokol ICP koristi dovoljno općenite. Istovremeno, definicija ICP protokola korisnicima na raspolaganje stavlja dovoljno velik raspon nezauzetih vrijednosti pa je moguće na jednostavan način dodavati nove poruke.

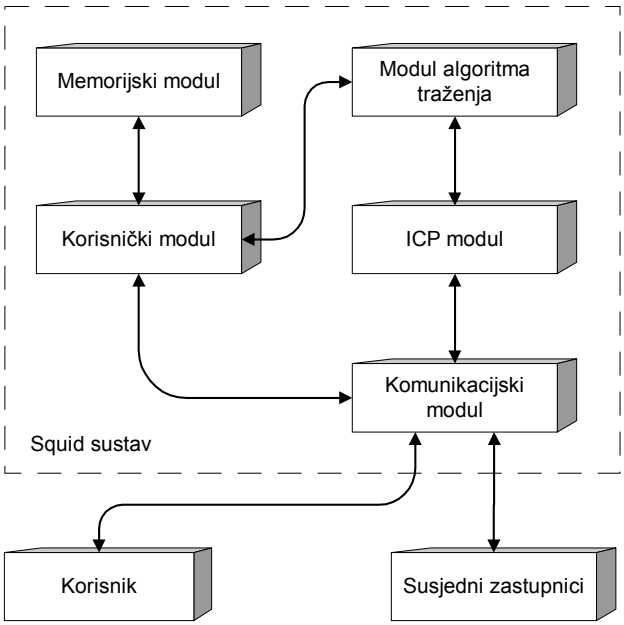
U svrhu programskog ostvarenja direktorijskog protokola potrebno je proširiti skup poruka ICP protokola i izvršiti odgovarajuće promjene na zastupničkom sustavu Squid. Prije izvođenja promjena potrebno je temeljito proučiti programsko ostvarenje upravljačkog protokola u sustavu Squid i proučiti odnos upravljačkog protokola prema cjelini zastupničkog sustava. Direktorijski protokol i poruke potrebne za njegovo izvođenje opisane su u odjeljku 2.6, a slijedeći odjeljak opisuje programsko ostvarenje upravljačkog protokola.

3.2.1. Način programskog ostvarenja upravljačkog protokola u Squid sustavu

Programsko ostvarenje upravljačkog protokola u Squid sustavu vezano je uz proces dohvata objekta koji je korisnik tražio. Stoga je za razumijevanje ostvarenja upravljačkog protokola potrebno proučiti cijeli tijek rada sustava, uključujući i postavljanje sustava nakon pokretanja glavnog programa. Početno postavljanje sustava uključuje niz radnji. Squid na početku određuje osnovne postavke vezane uz baratanje s memorijom i rad s datotekama. Dohvaća se IP adresa i ime računala na kojem se sustav izvodi, bilježi se vrijeme pokretanja programa i parsiraju se opcije navedene u naredbenom retku (engl *command line options*). Nakon toga, parsira se datoteka s postavkama potrebnim za pokretanje i prilagođavanje Squida. Sustav nastavlja rad postavljanjem datoteka, komunikacijskih potprograma i ostalih dijelova programa. U drugoj fazi određivanja postavki programa, određuju se HTTP i ICP priključci na kojima Squid sustav prima dolazni promet i otvara se opća dnevnička datoteka. Nakon toga, pripremaju se zasebni moduli Squida kao što su priručne memorije za IP, DNS (Domain Name System), preusmjeravanje, autentikacija, dnevnik pristupa, podsustav za baratanje greškama i drugo.

Po završetku postavljanja svih podsustava, Squid ulazi u temeljnu radnu petlju. Prvi je korak temeljne petlje izvođenje događaja (engl. *event*). U Squid sustavu događaj je poziv određenog potprograma u unaprijed određenom trenutku. Postoje dvije vrste događaja, vremenski događaji i komunikacijski događaji. Vremenskim je događajima moguće odrediti u kojem trenutku počinje izvršavanje pridružene zadaće. Navedeni događaji postavljaju se u rep za izvođenje. Komunikacijski događaji u Squid sustavu uzrokovani su TCP/IP komunikacijom. TCP/IP komunikaciju koja stvara događaje čine primanje i slanje UDP paketa, uspostavljanje i zatvaranje TCP veze, pojava podataka na priključnici, uspješno slanje podataka s priključnice i razne greške.

Opisani mehanizam pokretan događajima temelj je arhitekture Squid sustava. Navedeni dizajn omogućuje Squidu izbjegavanje uporabe više dretvi. Autori sustava Squid smatraju da višedretvena arhitektura nije prikladna u vremenski zahtjevnim primjenama, jer se njom troše memorijski i procesorski kapaciteti računala. Memorijski prostor troši se za pohranu stanja pojedinih dretvi i stogove, a procesorsko se vrijeme troši za prebacivanje konteksta između pojedinih dretvi.



Slika 18: Pojednostavljena shema sustava Squid

Slika 18 prikazuje pojednostavljenu shemu sustava Squid. Prikazana shema dostatna je za opis rada upravljačkog protokola Squida. Proces dohvata objekta počinje tako da korisnik pošalje zahtjev za objektom zastupniku. Otvaranje TCP veze na HTTP priključku pokreće odgovarajući događaj u komunikacijskom modulu. Komunikacijski modul prenosi primljeni HTTP zahtjev korisničkom modulu. Nakon primanja HTTP zahtjeva, korisnički modul pristupa memorijskom modulu zbog provjere da li u lokalnoj priručnoj memoriji postoji traženi objekt. Za opis programskog ostvarenja upravljačkog protokola u sustavu Squid značajan je samo slučaj kad u lokalnoj memoriji ne postoji valjana preslika traženog objekta. Nakon što je utvrdio potrebu za dohvatom objekta izvana, korisnički modul pokreće postupak prozivanja susjeda prenoseći nadzor na algoritam traženja. Modul algoritma traženja dohvaća listu susjednih i nadređenih zastupnika kojima je potrebno poslati upite i priprema odgovarajuće *ICP_QUERY* poruke. Navedene poruke šalju se pomoću ICP modula. ICP modul stvara UDP poruke i postavlja ih u rep UDP poruka za slanje koji se nalazi u komunikacijskom modulu.

Nakon što pošalje *ICP_QUERY* poruke susjednim zastupnicima, komunikacijski modul priprema mehanizam za primanje odgovora. ICP odgovori koje šalju susjedni zastupnici primaju se u komunikacijskom modulu i potom se proslijeđuju ICP modulu. ICP modul obrađuje dobivene poruke i šalje ih modulu algoritma traženja. Modul algoritma traženja čeka dok ne dobije bar jedan pozitivan odgovor, dok ne stignu svi negativni odgovori ili dok ne istekne unaprijed određeno vrijeme. Nakon

prvog primljenog pozitivnog odgovora, modul algoritma traženja nastavlja primati odgovore susjednih zastupnika, ali ih zanemaruje. Odluku o dohvatu objekta od poslužitelja ili od susjednog zastupnika, modul algoritma traženja prenosi korisničkom modulu. Korisnički modul potom pokreće dohvat HTTP objekta pomoću komunikacijskog modula. Komunikacijski modul proslijeđuje HTTP zahtjev, a sve informacije koje dolaze kao odgovor na zahtjev prenose se korisničkom modulu. Tijekom dohvata objekta od susjednog zastupnika ili poslužitelja, korisnički modul pristupa memorijskom modulu radi pohrane objekta u lokalnu pričuvnu memoriju. Nakon što je primljen cijeli traženi objekt, korisnički modul ponovo pristupa memorijskom i komunikacijskom modulu, te objekt dostavlja korisniku.

Datim opisom pokrivena su opća načela rada Squida i postupak dohvaćanja objekata. Za programsko ostvarenje upravljačkog protokola zasnovanog na direktoriju potrebno je proučiti još dva dijela Squida. Prvi je dio jednostavan i odnosi se na popis vrijednosti pomoću kojih su predstavljene pojedine poruke ICP protokola. Većina konstanti koje se u Squidu koriste, uključujujući sve vrijednosti vezane uz ostvarene protokole, nabrojane su u datoteci `enums.h`. Vrijednosti u ovoj datoteci pridružuju se konstantama pomoću pobrojanih lista (engl. *enumeration list*).

Zadnji dio potreban za programsko ostvarenje direktorijskog protokola složeniji je i odnosi se na rukovanje objektima u lokalnoj memoriji zastupnika. U svrhu održavanja točnosti direktorija potrebno je zabilježiti brisanje objekata iz lokalne priručne memorije zastupnika. Izbacivanje objekta iz lokalne memorije moguće je uočiti samo na temelju odgovarajućih poziva funkcija iz memorijskog modula. Analizom ponašanja Squida, uočeno je da se izbacivanje objekata iz lokalne memorije izvodi u samo jednoj funkciji. Namjena navedene funkcije je oslobađanje prostora u lokalnoj memoriji. Funkcija za oslobađanje memorije poziva se periodički ili u slučaju manjka memorije.

3.2.2. Način ugradnje direktorijskog protokola u Squid

Ugradnja direktorijskog protokola u sustav Squid počinje definiranjem vrijednosti kojima se označavaju direktorijske ICP poruke. U odjeljku 2.3.2 opisane su standardne ICP poruke navedene u dokumentu [9] i način na koji su poruke predstavljene. Svakoj ICP poruci pridružena je određena brojčana vrijednost. Tablica 4 prikazuje vrijednosti pridružene standardnim ICP porukama.

Vrsta poruke	Brojčana vrijednost
ICP_INVALID	0
ICP_QUERY	1
ICP_HIT	2
ICP_MISS	3
ICP_ERR	4
ICP_SECHO	10
ICP_DECHO	11
ICP_MISS_NOFETCH	21
ICP_DENIED	22
ICP_HIT_OBJ	23

Tablica 4: Vrijednosti standardnih ICP poruka

Osim standardnih vrijednosti prikazanih u tablici 4, sustav Squid koristi dijelove raspona od 5 do 9 i od 12 do 20 za posebne poruke koje nisu definirane u standardu ICP protokola [9]. Zbog toga je za proširenje skupa ICP poruka direktorijskim porukama odabran raspon vrijednosti od 24 do 27. Tablica 5 prikazuje brojčane vrijednosti pridružene pojedinim direktorijskim porukama. Vrijednosti navedene u tablici 5 umeću se u odgovarajuću pobrojanu listu u datoteci `enums.h`.

Vrsta poruke	Brojčana vrijednost
ICP_DIR_GET	24
ICP_DIR_ADD	25
ICP_DIR_DEL	26
ICP_DIR_DATA	27

Tablica 5: Vrijednosti direktorijskih ICP poruka

Osim umetanja navedenih brojčanih vrijednosti, za ugradnju novog upravljačkog protokola raspodijeljene priručne memorije potrebno je izvršiti određene izmjene u izvornom kôdu zastupničkog sustava Squid. Iako je pisan modularno, sustav Squid ne raspolaže mehanizmom za umetanje korisničkih proširenja. Mehanizam proširenja trebao bi podržavati pozive vanjskih korisničkih potprograma koji prikupljaju podatke i izvode određene izmjene informacija koje se obrađuju u Squid sustavu. Budući da navedeni mehanizam nije osiguran, potrebno je proučiti druge načine ugradnje direktorijskog protokola. S obzirom da je izvorni kôd Squida modularan i dan na slobodnu uporabu, moguće je izvesti izmjene na način koji nije složen. Osnovni je preduvjet ostvarivanja izmjena dobro razumijevanje izvornog kôda Squida. Ako se potrebe novog protokola poklapaju s izvedbom Squid sustava, onda je dovoljno u izvorni kôd dodati nove potprograme i izvesti manje izmjene u postupcima obrade zahtjeva, slanja upita susjedima i sl. Direktorijski protokol i neke od protokola opisanih u odjeljku 2.8, moguće je ostvariti uz određena manja ograničenja mogućnosti navedenih protokola.

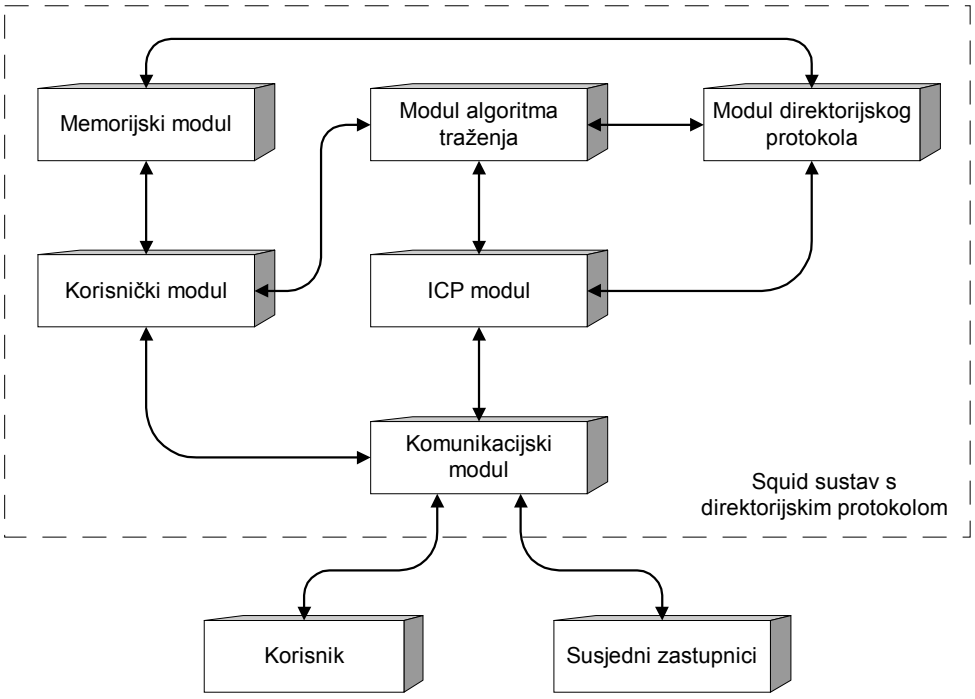
Pristup zasnovan na izmjenama izvornog kôda sustava Squid ima određene nedostatke. Osnovni nedostatak je potreba da se tijekom ispitivanja i promjena nadodanog kôda prekompilira čitav Squid sustav. S obzirom na složenost Squida, prevođenje cijelog sustava tijekom svake izmjene značajno usporava razvoj novog sustava. Uz navedeni nedostatak vezan je i problem promjene inačica Squida. Nakon što se pojavi nova inačica Squida poželjno je, zbog ispravki grešaka ili poboljšanih svojstava, upotrijebiti je kao osnovu vlastitog zastupničkog sustava. Pritom je potrebno prebaciti velike količine vlastitog kôda u novi izvorni kôd Squida. Značajni problem predstavljaju i moguće opsežne promjene izvornog kôda nove inačice Squida.

Iako je potonji problem moguće samo ublažiti, rješenje prva dva navedena problema daje uporaba vanjskih poziva potprograma, odnosno dinamičke biblioteke. Uporabom dinamičkih biblioteka dovoljno je samo jednom promijeniti izvorni kôd Squida, neovisno o mogućom promjenama kôda koji se dodaje. Potrebne promjene na sustavu Squid uključuju povezivanje s dinamičkom bibliotekom, stvaranje sučelja između funkcija u biblioteci i funkcija u izvornom Squidu i umetanje povratnih poziva potprograma iz biblioteke na odgovarajuća mjesta. Opisani pristup ublažava problem

usklađivanja proširenja s novim inačicama Squida. Budući da se uporabom dinamičkih biblioteka značajno smanjuje količina kôda koji se dodaje u izvorni kôd Squid sustava, umetanje proširenja u nove inačice Squida izvodi se znatno brže.

Ugradnja direktorijskog protokola u sustav Squid započinje umetanjem kôda koji otvara dinamičku biblioteku. Prikladno je mjesto za umetanje navedenog kôda početni dio programa koji izvodi pripreme postavki sustava. Nakon otvaranja dinamičke biblioteke, iz nje se poziva funkcija koja izvodi potrebne pripreme. Između ostalog, navedena funkcija preko pokazivača priprema dvije grupe funkcija radi povezivanja izvornog kôda Squida i dinamičke biblioteke. Prvu grupu čine funkcije iz dinamičke biblioteke koje se pozivaju s odgovarajućih mjesta u izvornom kôdu Squida. Druga skupina su izvorne funkcije Squida koje su potrebne dinamičkoj biblioteci za programsko ostvarenje direktorijskog protokola. Opisani postupak je nužan, jer niti izvorni kôd Squida niti izvorni kôd dinamičke biblioteke u trenutku povezivanja nisu svjesni postojanja vanjskih funkcija u drugom modulu. Osim navedenog povezivanja funkcija, tijekom postavljanja dinamičke biblioteke čita se datoteka s postavkama vezanim uz direktorijski protokol i otvara se zasebna dnevnička datoteka.

Slijedeći je korak programskog ostvarenja direktorijskog protokola stvaranje modula direktorijskog protokola i njegovo povezivanje s tri modula Squid sustava. Modul direktorijskog protokola povezuje se s memorijskim modulom, ICP modulom i modulom algoritma traženja. Načelna shema sustava Squid s ugrađenim direktorijskim protokolom prikazana je na slici 19.



Slika 19: Shema sustava Squid s ugrađenim direktorijskim protokolom

Sustav prikazan na slici 19 djeluje slično sustavu opisanom u prethodnom odjeljku. Razlika između navedenih sustava javlja se samo tijekom prijenosa izvođenja na modul algoritma traženja. U navedeni modul dodan je poziv funkcije iz dinamičke biblioteke modula direktorijskog protokola.

Umjesto da modul algoritma traženja izvede slanje upita susjedima, modul direktorijskog protokola pokreće dohvat podataka iz direktorija. Ako se direktorijska lista za traženi objekt nalazi na susjednom zastupniku, onda direktorijski modul pristupa ICP modulu i preko njega šalje *ICP_DIR_GET* poruku direktorijskom zastupniku. Nakon primanja poruke *ICP_DIR_DATA* od direktorijskog zastupnika, nadzor se prenosi s komunikacijskog modula na ICP modul pa na modul direktorijskog protokola. Modul direktorijskog protokola potom dohvaća podatke iz direktorijske liste, stvara listu zastupnika kojima se šalje *ICP_QUERY* upit. Slanje upita modul direktorijskog protokola prepušta modulu algoritma traženja. Ako je direktorijska lista za traženi objekt prazna, onda direkorijski modul dojavljuje modulu algoritma traženja da je objekt potrebno dohvatiti od poslužitelja ili nadređenog zastupnika. Daljnji dohvat objekta izvodi se kao u sustavu opisanom u prethodnom odjeljku.

Nakon primanja *ICP_DIR_GET*, *ICP_DIR_DEL* ili *ICP_DIR_ADD* poruke na direktorijskom zastupniku, podaci se preko komunikacijskog i ICP modula prenose direktorijskom modulu. Modul direktorijskog protokola potom izvodi tražene promjene na direktorijskoj listi, odnosno stvara *ICP_DIR_DATA* poruku i preko ICP modula šalje navedenu poruku glavnom zastupniku. Tijekom stvaranja liste zastupnika koji imaju traženi objekt, modul direktorijskog protokola pristupa memorijskom modulu radi provjere da li u lokalnoj priručnoj memoriji direktorijskog zastupnika postoji preslika traženog objekta. Navedena je provjera nužna, jer direktorijska lista u svrhu uštede memorijskog prostora ne sadržava zapis za objekte spremljene na direktorijskom zastupniku.

Zadnja promjena u sustavu Squid izvedena je u memorijskom modulu. Navedeni modul između ostalih zadaća, periodički ili u slučaju nedostatka memorijskog prostora u lokalnoj priručnoj memoriji izvodi izbacivanje objekata iz lokalne memorije. Izbacivanje se izvodi stvaranjem LRU (engl. *Least Recently Used*) liste. Budući da je potrebno održavati jednoznačnost direktorija, nakon izbacivanja određenog objekta memorijski modul šalje obavijest modulu direktorijskog protokola. Ako se direktorijska lista za izbačeni objekt nalazi na susjednom zastupniku, onda modul direktorijskog protokola stvara poruku *ICP_DIR_DEL* i preko ICP modula šalje navedenu poruku direktorijskom zastupniku. S ciljem ostvarivanja bolje jednoznačnosti sustava, ako zastupnik primi *ICP_QUERY* poruku, a u lokalnoj priručnoj memoriji nema valjanu presliku traženog objekta, onda se direktorijskom zastupniku također šalje *ICP_DIR_DEL* poruka. Slanje navedene poruke temelji se na pretpostavci da je *ICP_QUERY* upit dobiven na temelju netočnih informacija u direktoriju.

Opisano programsko ostvarenje ima dvije značajke vezane uz izvedbu direktorijskog protokola. Funkcije raspršenog adresiranja, kojima se direktorij raspodjeljuje između zastupnika, preuzete su iz sustava Squid. Jedna od navedenih funkcija upotrijebljena je i za ostvarenje raspršenog adresiranja koje se koristi u izvedbi direktorija unutar jednog zastupnika. Uporabom raspršenog adresiranja ostvaren je brži dohvat informacija iz lokalnog direktorija. Druga je značajka opisanog programskog ostvarenja ugradnja dodatne tablice koja povezuje URL-ove objekata s pripadnim MD5 [19] ključevima. Navedenim ključevima sustav Squid zamjenjuje URL-ove promjenjive duljine. Uvođenje tablice potrebno je zato što objekt u procesu dohvata dobija nekoliko različitih ključeva. Zbog toga,

prije potpune pohrane objekta u lokalnoj priručnoj memoriji nije moguće objekt dohvatiti preko njegovog URL-a.

Promjene opisane u ovom odjeljku čine potpuni skup promjena izvedenih na izvornom kôdu Squida. Opisane promjene su jednostavne, nije ih teško unijeti u kôd i nisu opsežne. Navedeni postupak programskog ostvarenja direktorijskog protokola osigurava da tijekom izdavanja nove inačice Squida nije potrebno izvoditi opsežne promjene na novom izvornom kôdu. Osim funkcija koje čine sučelje između Squida i dinamičke biblioteke, potpuno je odvojeno programsko ostvarenje direktorija od izvornog kôda Squida. Prevođenje, ispravljanje i provjera kôda direktorijskog protokola stoga se izvodi znatno brže nego u slučaju da je navedeni kôd u cijelosti uključen u izvorni kôd sustava Squid.

3.3. Moguća poboljšanja programskog ostvarenja

Jedan od osnovnih nedostataka opisanog programskog ostvarenja direktorijskog protokola je izbacivanje mnogih mehanizama ugrađenih u sustav Squid koji nisu nužni za usporedbu učinkovitosti protokola. Izbačene su provjere prava pristupa za direktorijski protokol, vođenje statistika i drugi mehanizmi. Bolje povezivanje sa Squidom ostvarilo bi se podržavanjem navedenih mehanizama. Osim toga, moguće su opsežnije promjene vezane uz precizno postavljanje vremenskog intervala predviđenog za dohvat direktorija. Potrebno je istražiti i na odgovarajući način promijeniti ponašanje sustava u slučaju kad se direktorij ne dohvati u predviđenom vremenu.

Značajna poboljšanja moguće je izvesti na direktoriju. Prvo poboljšanje vezano za direktorij sastoji se od zamjene URL-a s MD5 [19] ključevima. U sadašnjoj podatkovnoj strukturi direktorija svaki objekt ima svoj popis zastupnika koji navedeni objekt imaju u svojoj lokalnoj memoriji. Pojedini objekti u popisu razlikuju se po URL-ovima. Nedostatak navedenog pristupa je promjenjiva duljina URL-ova. Osim toga, URL-ovi mogu biti znatne duljine. Prva je posljedica uporabe tekstualnih URL-ova sporo pretraživanje direktorija, jer se koristi uspoređivanje znakovnih nizova. Druga je posljedica različita duljina zapisa zbog koje se na zapise u direktorijskim listama troši veći memorijski prostor. MD5 ključevi izvorno imaju duljinu od 128 bita i dobivaju se izračunavanjem složenih funkcija. Funkcije su dizajnirane tako da je osigurano nisko preklapanje ključeva. Pod preklapanjem ključeva smatra se preslikavanje dva različita URL-a u isti ključ. Uvođenjem MD5 ili sličnih ključeva smanjuje se vrijeme pretraživanja, smanjuje se prostor potreban za pohranu zapisa i ostvaruju se zapisi stalne duljine koji su pogodniji za uporabu u raznim metodama brzog adresiranja.

Daljnje poboljšanje mehanizma direktorija odnosi se na poboljšanje brzine pristupa zapisima iz direktorija. Poželjno bi bilo primijeniti neke od tehnika brzog pristupa zapisima poput naprednijeg raspršenog adresiranja. Osim toga, radi smanjenja memorijskog prostora koji direktorij zauzima i ubrzanja rada direktorija moguće je ograničiti direktorijske liste za određeni objekt. Za ostvarenje navedenog ograničenja potrebno je razviti složene mehanizme koji određuju pod kojim uvjetima se stari zapisi izbacuju iz direktorija u cilju oslobađanja mjesta za nove zapise.

U opisanom programskom ostvarenju koristi se jednostavan mehanizam raspoređivanja opterećenja. Nakon što dobije direktorijsku listu za traženi objekt u kojoj postoji više od α zapisa, glavni zastupnik odabire α zastupnika kojima šalje *ICP_QUERY* upiti. Raspoređivanje opterećenja izvedeno je slučajnim odabirom α zastupnika iz skupa zastupnika koji imaju traženi objekt. Učinkovitija provedba raspoređivanja opterećenja uključivala bi kružni mehanizam odabira zastupnika. Kružni je mehanizam znatno složenije ostvariti. Jedno moguće rješenje je izmjena redoslijeda zastupnika na direktorijskom računalu prije slanja *ICP_DIR_DATA* poruke. Nedostatak je navedenog pristupa, osim povećanja složenosti programskog ostvarenja, dodatno vrijeme potrebno za izvođenje rotacije zastupnika. Između ostalog, potrebno je provesti dodatno istraživanje s ciljem pronalaženja optimalne funkcije za raspršeno adresiranje. Zadaća poboljšanje funkcije je uravnoteživanje veličine podskupova URL-ova za čije su direktorije pojedini zastupnici odgovorni. Navedenim postupkom, direktorij se optimalno raspoređuje između zastupnika.

Konačno, jedno od značajnih poboljšanja upravljačkog protokola raspodijeljene priručne memorije zasnovanog na direktoriju je povećanje jednoznačnosti podataka u direktoriju. Budući da direktorij koristi nepouzdani mehanizam prijenosa informacija, informacije u direktoriju mogu biti netočne ili nepotpune. S obzirom da je nepouzdani prijenosni mehanizam UDP odabran zbog učinkovitosti, pouzdanost direktorija nije moguće povećati prelaskom na pouzdani protokol TCP. Uporaba TCP protokola, osim toga, ne rješava problem jednoznačnosti, jer postoji mogućnost da objekt koji se nalazi u lokalnoj memoriji nekog zastupnika nije valjan u trenutku u kojem ga glavni zastupnik pokuša dohvatiti. Osim toga, moguće je izbacivanje objekta iz lokalne priručne memorije zastupnika nakon što je glavni zastupnik dobio informaciju iz direktorija, a prije nego je glavni zastupnik poslao ICP upit ili HTTP zahtjev zastupniku na kojem je objekt izbačen. Zbog navedenih razloga problem održavanja točnosti direktorija promatra se kao problem jednoznačnosti podataka. S obzirom na ograničenost postojećih istraživanja u području jednoznačnosti podataka, u smislu održavanja točnosti direktorija moguće je uvesti značajna poboljšanja.

Neka od navedenih poboljšanja nije moguće izvesti bez opsežnih promjena na Squidu. Zbog navedenog problema, jedna od razmatranih mogućnosti tijekom izrade rada bila je izgradnja vlastitog zastupničkog sustava koji u osnovnom dizajnu podržava direktorijski protokol. Osim toga, postojala je i zamisao stvaranja zastupničkog sustava napisanog tako da podržava jednostavne izmjene upravljačkog protokola raspodijeljene priručne memorije. Navedene zamisli napuštene su zbog složenosti sustava Squid. Stvaranje zastupničkog sustava koji je po svojstvima, nevezanim za upravljački protokol, istovjetan Squidu zahtijeva znatno veći i dugotrajniji projekt od opisanoga. Pokazalo se da programsko ostvarenje direktorijskog protokola na osnovi Squida, zadovoljava osnovne potrebe usporedbe upravljačkih protokola. Za ozbiljnije primjene direktorijskog protokola potreban je znatno opsežniji rad na prilagodbi Squida. U krajnjem slučaju, navedeni rad je po složenosti jednak stvaranju novog zastupničkog sustava.

4. Mjerenje svojstava upravljačkih protokola

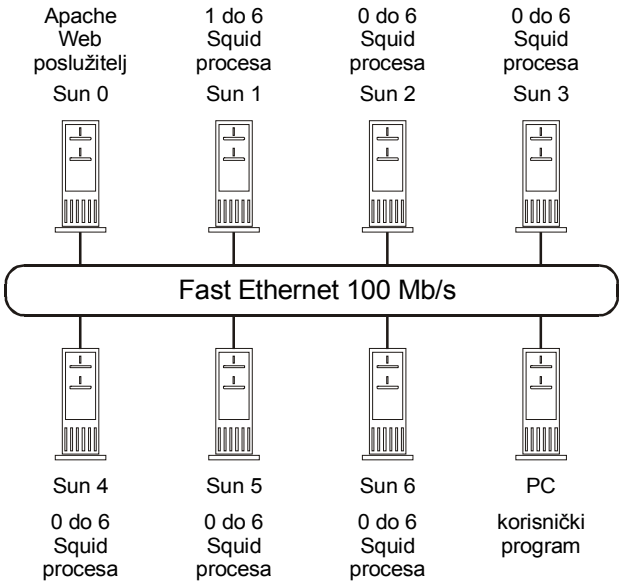
4.1. Uvjeti mjerenja

U cilju usporedbe sposobnosti proširivanja, mjerena su svojstva dva protokola, Squid protokola i direktorijskog protokola. Mjerenja svojstava izvedena su pomoću modificiranog zastupničkog programa Squid. Modificirani zastupnički program dobiven je programskim ostvarenjem direktorijskog protokola, opisanim u poglavlju 4. U datoteku s postavkama zastupničkog programa dodan je parametar koji služi kao prekidač za uključivanje i isključivanje upravljačkog protokola zasnovanog na direktoriju.. Uporaba parametra-prekidača omogućava jednostavnu promjenu upravljačkog protokola. Promjena upravljačkog protokola ostvaruje se zaustavljanjem zastupničkog programa, promjenom navedenog parametra i ponovnim pokretanjem programa. Pritom ostale postavke zastupničkog sustava ostaju iste i nakon promjene upravljačkog protokola. Dobiveni rezultati usporedbe stoga su neovisni o postavkama sustava.

Mjerenja su izvedena unutar odjela IPTO AT&T Labs u San Joseu u Californiji, SAD. Za mjerenja je raspoloživo bilo sedam Sun računala sa Solaris operativnim sustavom i jedno osobno računalo (PC). Četiri Sun računala bila su slabiji modeli SparcStation 20, dok su preostala tri Sun računala bili Ultra 10 sustavi. PC računalo bilo je zasnovano na Pentium III procesoru radnog takta od 500 MHz i operativnom sustavu Windows 98. Jedno Sun Ultra 10 računalo iskorišteno je kao Web poslužitelj. Na poslužiteljsko računalo instaliran je i pokrenut Apache poslužiteljski sustav. Ostalih šest Sun računala služila su kao zastupnička računala na kojima su pokretani Squid zastupnički programi.

Tijekom mjerenja su na raspolaganju bila navedena, ograničena sredstva koja nisu dostatna za izvođenje testova punog opsega. Za točnije testove potreban je znatno veći broj zastupničkih i korisničkih računala. Potrebna je i izdvojena mreža s poslužiteljem koji simulira vrijeme odziva koje se javlja na Internetu. Raspoloživ broj zastupničkih računala bio je malen. Pri šest zastupnika u sustavu razlike svojstava upravljačkih protokola nisu značajne pa je posebnim postupkom ostvaren veći broj zastupnika. Povećani broj zastupnika ostvaren je pokretanjem više zastupničkih programa, odnosno, procesa na istom računalu. Najveći broj Squid procesa pokrenutih na jednom zastupničkom računalu iznosio je šest. Budući da je u sustavu bilo šest računala, najveći broj zastupnika u sustavu iznosio je 36. Pri navedenom broju zastupnika u raspodijeljenom zastupničkom sustavu primjetne su značajne razlike između protokola čija svojstva su mjerena.

Osobno računalo služilo je za stvaranje i slanje korisničkih zahtjeva. Budući da jedan korisnik nije dovoljan za ostvarivanje mjerenja, korisničke zahtjeve slao je korisnički program koji je pokretao više korisničkih dretvi. Pomoću više dretvi simulirano je opterećenje zastupničkog sustava s više paralelnih zahtjeva. Sva mjerna računala bila su spojena na isti segment FastEthernet lokalne mreže kapaciteta 100 Mb/s. Slika 20 prikazuje strukturu mjernog sustava.

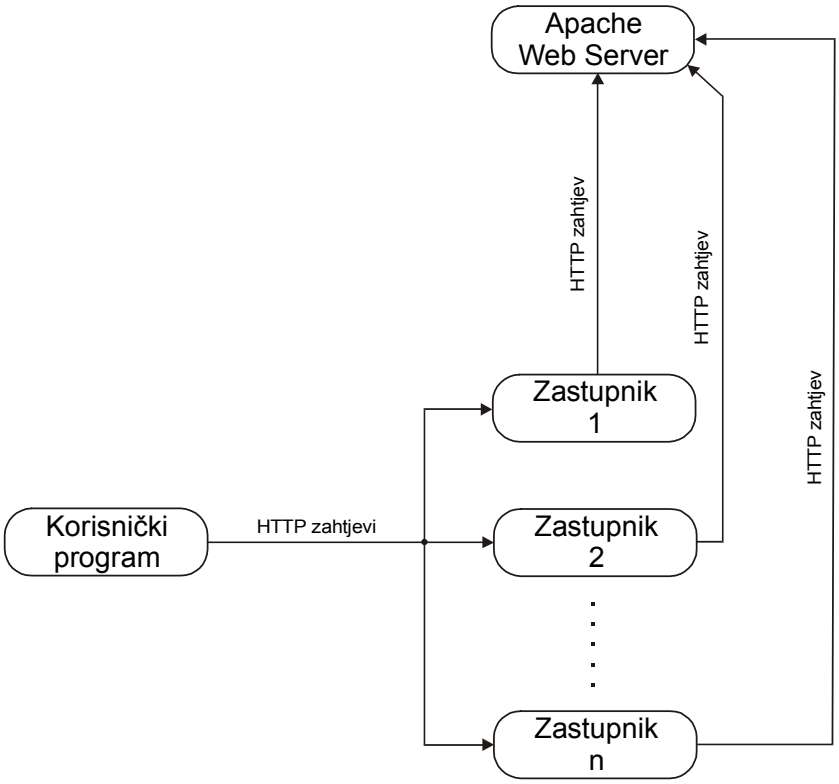


Slika 20: Struktura mjernog sustava

Mjerenja nisu izvršena na izdvojenom segmentu mreže zbog organizacijskih razloga. Za mjerenja je korištena mreža koju u svakodnevnom radu koriste djelatnici AT&T Labs IPTO. Budući da je mreža velikog kapaciteta i da su mjerenja izvođena izvan radnog vremena organizacije, utjecaj mrežnog prometa koji ne stvara mjerni sustav nije znatan. Rezultati mjerenja ipak mogu sadržavati greške koje su posljedica prometa na mahove. U svrhu smanjivanja utjecaja prometa na mahove, sva mjerenja su izvođena u većim vremenskim intervalima i ponavljana 10 puta.

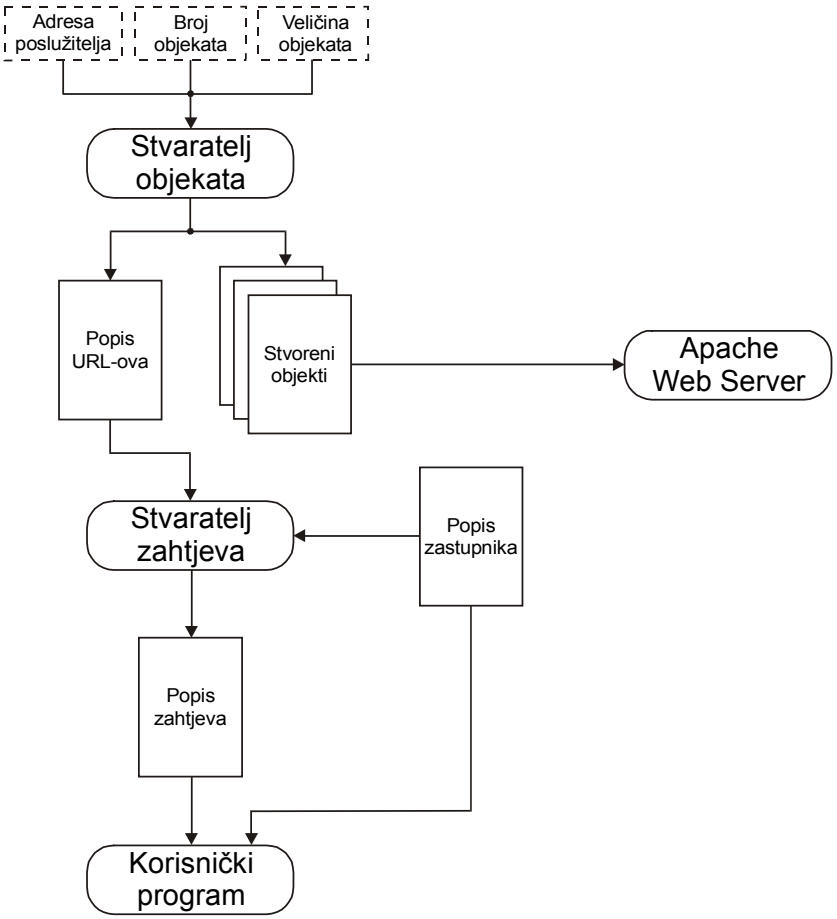
4.2. Mjerni sustav

Osim pravilnog postavljanja i pokretanja Web poslužitelja i zastupničkih programa, za izvođenje mjerenja svojstava upravljačkih protokola potrebno je umjetno stvarati korisničke zahtjeve. Umjetno stvoreni zahtjevi upotrijebljeni su zbog nemogućnosti izvođenja mjerenja sa stvarnim korisnicima. Za potrebe mjerenja napisan je poseban korisnički program. Korisnički program šalje HTTP zahtjeve zastupnicima u raspodijeljenom zastupničkom sustavu. Svi traženi HTTP objekti nalaze se na mjernom Apache Web poslužitelju pa zastupnici dohvaćaju objekte od navedenog poslužitelja. Slika 21 prikazuje shemu mjernog sustava.



Slika 21: Shema mjernog sustava

Mjerni sustav umjetno stvara objekte koje korisnički program traži od zastupnika u sustavu raspodijeljene priručne memorije. Osim toga, umjetno se stvara razdioba zahtjeva u odnosu na objekte i razdioba zahtjeva u odnosu na zastupnike. Objekte stvara program nazvan *stvaratelj objekata*, a razdiobu zahtjeva stvara program *stvaratelj zahtjeva*. Navedeni programi nisu dio korisničkog programa čija je jedina namjena slanje zahtjeva i dohvat objekata od zastupnika. Međusobne ovisnosti programa koji čine mjerni sustav prikazane su na slici 22. Na slici 22 pravokutnicima su prikazane datoteke s postavkama, a isprekidanim su crtama uokvireni ulazni parametri programa.



Slika 22: Međusobne ovisnosti programa u mjernom sustavu

Pri pozivu programa *stvaratelj objekata* zadaje se adresa poslužitelja na kojem će objekti biti smješteni, broj objekata i njihova veličina. Program se poziva na slijedeći način:

```
CreateObjects mahern.ipo.att.com:2010 100 20000
```

Sadržaji objekata različiti su i ograničeni na čitljive (engl. *readable*) ASCII znakove. Stvoreni objekti prenose se pomoću FTP alata na Apache Web poslužitelj. Osim datoteka sa sadržajem objekata, program na izlazu stvara datoteku s popisom URL-ova objekata. Navedena datoteka nazvana je `DirTester.url`, a isječak iz nje prikazan je na slici 23.

```
mahern.ipo.att.com:2010/Object00000000.dat
mahern.ipo.att.com:2010/Object00000001.dat
mahern.ipo.att.com:2010/Object00000002.dat
```

Slika 23: Isječak iz datoteke `DirTester.url`

Program *stvaratelj zahtjeva* na ulazu čita datoteku `DirTester.url` i datoteku s popisom zastupnika u zastupničkom sustavu. Datoteka s popisom zastupnika nazvana je `DirTester.conf` i stvara se ručno. U navedenoj datoteci, osim popisa zastupnika navedene su sve postavke korisničkog programa, odnosno broj korisničkih dretvi, broj zahtjeva i popis zastupnika s njihovim DNS imenima i

brojevima priključaka. Primjer datoteke `DirTester.conf` prikazan je na slici 24. Datoteku `DirTester.url` moguće je ručno napisati ili upotrijebiti izlaznu datoteku programa *stvaratelj objekata*. Poziv programa *stvaratelj zahtjeva* nema parametara, jer se sve postavke čitaju iz navedenih datoteka. Temeljem zadanog broja zahtjeva, program stvara popis zahtjeva koji su po uniformnoj razdiobi raspoređeni na zadane zastupnike i objekte. Popis zahtjeva ispisuje se u izlaznu datoteku `DirTester.in`. Svaki zapis u navedenoj datoteci sastoji se od URL-a objekta i rednog broja zastupnika kojem se šalje zahtjev za navedenim objektom. Isječak iz datoteke `DirTester.in` prikazan je na slici 25.

```
n_threads 10
n_requests 1000
proxy lancelot 1983
proxy lancelot 1984
proxy lancelot 1985
proxy lancelot 1986
```

Slika 24: Primjer datoteke s postavkama `DirTester.conf`

```
6 mahern.ipo.att.com:2010/Object00000027.dat
2 mahern.ipo.att.com:2010/Object00000055.dat
6 mahern.ipo.att.com:2010/Object0000001e.dat
10 mahern.ipo.att.com:2010/Object0000003c.dat
```

Slika 25: Isječak iz datoteke `DirTester.in`

Korisnički program pokreće se bez parametara, jer se svi podaci koje koristi nalaze u datotekama `DirTester.conf` i `DirTester.in`. Korisnički program iz datoteke `DirTester.conf` čita podatke o zastupnicima u sustavu raspodijeljene priručne memorije. Osim toga, iz navedene se datoteke čita broj objekata i broj paralelnih zahtjeva koji se šalju. Iz datoteke `DirTester.in` korisnički program čita popis korisničkih zahtjeva.

Korisnički program izvodi se u više dretvi. Uz temeljnu dretvu, program pokreće određeni broj korisničkih dretvi. Broj korisničkih dretvi odgovara broju paralelnih zahtjeva navedenom u datoteci `DirTester.conf`. U mjerenjima su korištene vrijednosti od 5, 10, 15 i 20 paralelnih zahtjeva pa korisnički program pokreće toliko korisničkih dretvi. Na slikama 26 i 27 prikazani su algoritmi rada temeljne i korisničkih dretvi.

```
pročitaj datoteku s postavkama;
stvori sinkronizacijske objekte;
stvori dretve;
otvori datoteku s popisom zahtjeva;
dok nije kraj datoteke s popisom zahtjeva
{
   pročitaj indeks zastupnika i URL objekta;
   čekaj slobodnu dretvu ili signal za prekid;
   ako je signaliziran prekid
      izađi iz petlje;
   inače
   {
      prenesi podatke o zahtjevu slobodnoj dretvi;
      signaliziraj početak rada slobodnoj dretvi;
   }
}
pričekaj završetak rada svih dretvi;
ispiši rezultate mjerenja;
oslobodi sve korištene resurse;
```

Slika 26: Algoritam rada temeljne dretve

```
za uvijek
{
   čekaj signal za početak rada ili prekid programa;
   ako je signaliziran prekid programa
      izađi iz petlje;
   otvori vezu sa zastupnikom;
   pošalji zahtjev za objektom;
   primi odgovor;
   izračunaj i pohrani vrijeme dohvata objekta;
   signaliziraj spremnost za idući zahtjev;
}
signaliziraj spremnost za izlazak;
```

Slika 27: Algoritam rada korisničke dretve

Nakon pokretanja korisničkog programa, temeljna dretva pročita datoteku s postavkama, pripremi sve podatkovne strukture i korištene resurse uključujući i signalizacijske objekte. Osim toga, temeljna dretva u pripremnom dijelu podijeli zahtjeve svim pokrenutim korisničkim dretvama i ulazi u glavnu petlju. U glavnoj petlji temeljna dretva čeka da neka od korisničkih dretvi završi dohvat objekta i potom joj predaje novi zahtjev koji treba izvršiti. Opisanim načinom postiže se konstantno opterećenje zastupničkog sustava stalnim brojem paralelnih korisničkih zahtjeva.

Svaka korisnička dretva istodobno postavlja jedan zahtjev zastupničkom sustavu. Nakon što je dohvatila traženi objekt, korisnička dretva izračuna vrijeme potrošeno za dohvat objekta i prijavljuje se kao raspoloživa temeljnoj dretvi. Nakon što su svi zahtjevi iz ulazne datoteke obrađeni, temeljna dretva signalizira završetak rada svim korisničkim dretvama i čeka da one završe s radom. Korisnički program potom ispisuje broj dohvaćenih objekata, ukupnu veličinu objekata i prosječno trajanje dohvata jednog objekta. Trajanje dohvata predstavlja kašnjenje po korisničkom zahtjevu koje je u mjerenjima upotrijebljeno kao mjera ocjene svojstva upravljačkog protokola raspodijeljene priručne memorije.

Opisana podjela mjernog sustava na tri programa omogućuje znatnu prilagodljivost mjernog sustava. Mjerni sustav može koristiti umjetno stvorene objekte ili objekte preuzete s Internet sustava. Osim umjetne uniformne razdiobe zahtjeva po zastupnicima i objektima, moguće je upotrijebiti razdiobu koja točnije opisuje stvarne sustave ili upotrijebiti zapis slijeda korisničkih zahtjeva (engl. *trace* ili *access log*) iz stvarnih Internet sustava. Za ostvarenje navedenih prilagodbi nisu potrebne izmjene izvornog kôda mjernih programa, već je dovoljno pravilno stvoriti sve datoteke s postavkama u mjernom sustavu.

Nakon usporedbe opisanog načina testiranja s načinima koji su korišteni u radovima [15,16,18] primjetne su određene razlike. Navedeni radovi s područja raspodijeljenih zastupničkih sustava, koriste dvije metode za dobivanje ocjene svojstava zastupničkih sustava. Prva je metoda simulacija rada zastupničkog sustava. Navedena metoda koristi zaspise slijedova korisničkih zahtjeva. Simulatori se izvode podjelom skupa korisnika navedenih u zapisima na nekoliko podskupova. Dobiveni podskupovi predstavljaju zamišljene organizacijske cjeline. Svaka organizacijska cjelina ima vlastitog zastupnika kojem se obraća. Organizacijske su cjeline povezane preko zastupnika. Nakon podjele korisnika u organizacijske cjeline i određivanja zastupnika, definira se ponašanje upravljačkog protokola za dobiveni sustav raspodijeljene priručne memorije. Simulator čita zapis slijeda korisničkih zahtjeva i simulira ponašanje sustava kao da u njemu postoje zamišljeni zastupnici koji izvode ispitivani upravljački protokol raspodijeljene priručne memorije. Tijekom simulacije, simulator vodi popis objekata koje pojedini zastupnik ima u svojoj priručnoj memoriji. Na temelju popisa objekata, simulator odlučuje da li se objekt dohvaća od glavnog zastupnika, susjednog zastupnika ili poslužitelja.

Simulator koristi jednostavan model zastupničkog sustava. Ponašanje sustava simulira se provjerom da li u lokalnoj priručnoj memoriji zastupnika postoji traženi objekt. Tijekom simulacije procjenjuje se količina međusobne komunikacije susjednih zastupnika. Prednost je uporabe simulatora izbjegavanje stvaranja složenog stvarnog sustava potrebnog za izvođenje mjerenja. Uporaba simulatora ima više nedostataka. Simulacijom se procjenjuje samo količina dodatnog prometa koji stvara međusobna komunikacija zastupnika. Osim toga, simulator daje nepreciznu procjenu udjela pogodaka u sustavu raspodijeljene priručne memorije. Stvarni zastupnici ne ponašaju se u skladu s predviđanjima simulatora, jer simulator nije u mogućnosti točno procijeniti u kojem trenutku neki objekt postaje nevaljan ili bude izbačen iz priručne memorije zastupnika. Postojeći simulatori

procjenjuju mrežni promet, a ne uzimaju u obzir stvarno kašnjenje korisničkih zahtjeva koje nastaje kao posljedica mrežnog prometa. Konačno, simulator ne uzima u obzir opterećenje računala na kojem se zastupnik izvodi. Zbog navedenih nedostataka, korist simulatora svodi se na procjenu dodatnog prometa koji stvaraju pojedini upravljački protokoli. Dodatni promet s približno jednakom točnosti moguće je procijeniti analitičkim modelom pa uporaba simulatora nije kvalitetno rješenje za određivanje svojstava upravljačkih protokola.

Druga je metoda procjene svojstava upravljačkih protokola raspodijeljene priručne memorije promatranje rada stvarnog sustava. Navedena metoda slična je metodi s umjetnim zahtjevima koja se koristi u ovom magistarskom radu. Promatraju se stvarna računala koja rade kao zastupnici, računala koja šalju korisničke zahtjeve i dodatna računala poput WWW ili DNS poslužitelja. Iako promatranje stvarnih sustava daje najpouzdanije ocjene svojstava pojedinih protokola, zbog praktičnih problema promatrani se sustavi pojednostavljuju. Pojednostavljenje kvari rezultate pa dobiveni rezultati ne odgovaraju stvarnim sustavima. Najčešće se zbog nedostatka računala izbacuju WWW poslužitelji iz mjernog sustava. WWW poslužitelji se u tom slučaju simuliraju umjetnim umetanjem traženih objekata u lokalne priručne memorije zastupnika. Navedeneo umetanje objekata značajno kvari točnost rezultata.

Smanjenje broja zastupničkih računala jedan je od kompromisa tijekom pripremanja metode promatranja stvarnih sustava. Navedenim kompromisom znatno se smanjuje smisao uporabe metode promatranja. Uz mali broj zastupničkih računala u sustavu raspodijeljene priručne memorije razlike između kašnjenja upravljačkih protokola nisu značajne. Razlike nisu značajne, jer dodatni promet i obrada pojedinih zahtjeva imaju mali udio u kašnjenju. Istraživači koji koriste metodu promatranja stvarnih sustava, svjesni su navedenih nedostataka. Oni stoga metodu promatranja koriste samo za provjeru ispravnosti programskog ostvarenja. Ograničenjem metode na provjeru ispravnosti, metoda gubi svoj značaj u smislu ocjene svojstava raspodijeljenog zastupničkog sustava.

Način mjerenja ostvaren u ovom radu ima nekoliko prednosti i nedostataka u odnosu na opisane metode drugih autora. Osnovni je nedostatak primijenjene metode mjerenja umjetno stvaranje zahtjeva. Metode koje drugi autori koriste najčešće su zasnovane na zapisu sa slijedom korisničkih zahtjeva u kojem se detaljno opisuju zahtjevi pojedinih korisnika. Zahtjevi koji se šalju na temelju navedenih zapisa slični su stvarnom sustavu po vremenskoj razdiobi zahtjeva i po razdiobi veličine objekata. Uporabom umjetno stvorenih objekata, gubi se dio kvalitete ocjene svojstava i ostvareni model se udaljava od stvarnog sustava.

Prednosti su korištenog modela jednostavno ostvarenje korisničkog programa, jednostavno slanje zahtjeva i kraće vrijeme izvođenja mjerenja. Mjerenje izvedeno u ovom radu traje kraće, jer tijekom praćenja zapisa korisničkih zahtjeva simulirano vrijeme odgovara vremenu u zapisima pa je moguće da jedno mjerenje potraje više dana. Najveća je prednost mjernog sustava korištenog u radu uporaba poslužiteljskog računala i velik broj zastupnika. Uporabom poslužiteljskog računala ostvaren je sustav

koji je sličan stvarnom Internet sustavu. Na osnovi velikog broja zastupnika, moguće je odrediti svojstva proširivosti pojedinih upravljačkih protokola.

Osnovne su razlike ostvarenog mjernog sustava prema stvarnim sustavima umjetno stvoreni zahtjevi i objekti, više zastupničkih procesa koji se izvode na istom računalu i lokalni poslužitelj WWW objekata. Uporaba umjetno stvorenih zahtjeva obrazložena je u odjeljku 4.2. Izvođenje više zastupničkih procesa na jednom računalu nužno je zbog ograničenih sredstava. Izvođenje više procesa na jednom računalu čini dobivene rezultate djelomično netočnima. Postavljanje više zastupničkih procesa na jedno računalo ekvivalentno je uporabi više sporijih računala. Stoga navedeni pristup nema znatan utjecaj na rezultate.

Posljednji navedeni nedostatak, postavljanje poslužitelja WWW objekata na lokalnu mrežu, ima više uzroka. Ostvarenje dohvata objekata od stvarnih poslužitelja, ograničeno je s tri praktična problema. Prvi je problem prikupljanje reprezentativnog uzorka objekata i poslužitelja od kojih se objekti dohvaćaju. Drugi je problem zaobilaženje zastupnika koji se nalazi između lokalne mreže i Interneta, a koji unosi velika odstupanja u mjerene rezultate. Treći je problem promjenjivost opterećenja Interneta. Prosječni promet na vezi između mjernog sustava i poslužitelja u određenom vremenskom razdoblju značajno se mijenja. Znatna odstupanja u vremenu dohvata objekta s poslužitelja značajno utječu na dobivene rezultate. Budući da je namjena mjernog sustava usporedba svojstava dva protokola, navedeni problemi onemogućili su ostvarivanje usporedbe s realnim poslužiteljima.

Postavljanje zastupničkih računala na istu lokalnu mrežu zajednički je nedostatak svih metoda opisanih u ovom odjeljku. U stvarnim sustavima, zastupnici se postavljaju na odvojene lokalne mreže. Zbog toga je kašnjenje komunikacije između zastupnika koje se dobiva na temelju opisanih metoda manje od kašnjenja u stvarnim sustavima. Navedeno kašnjenje ima znatni utjecaj na kašnjenje korisničkih zahtjeva pa navedeni nedostatak utječe na točnost rezultata. U istraživanjima se izbjegava postavljanje zastupnika na različite mrežne segmente, jer su za ostvarenje takvih sustava potrebna znatna sredstva.

4.3. Rezultati mjerenja

Mjerenja ostvarena u ovom radu koriste prosječno kašnjenje po korisničkom zahtjevu za mjeru usporedbe upravljačkih protokola raspodijeljene priručne memorije. Rezultati mjerenja prikazani su u obliku grafova. Na *y* osima grafova prikazano je prosječno kašnjenje, a na *x* osima vrijednost parametra čiji se utjecaj mjeri. Rezultati mjerenja podijeljeni su u tri grupe, ovisno o parametru čiji je utjecaj ispitivan. Grafovi prikazani u zasebnim odjeljcima stoga na *x* osi imaju broj zastupnika u sustavu, broj klijenata ili veličinu objekata. U svim izvedenim mjerenjima parametar α bio je konstantan. Parametar α ograničava broj zastupnika iz direktorija kojima se šalje upit i podrobnije je objašnjen u odjeljku 2.6.

4.3.1. Utjecaj broja zastupnika

Broj zastupnika u sustavu najvažniji je promatrani parametar, jer određuje proširivost sustava zastupnika s obzirom na broj zastupnika u sustavu. Ako je raspodijeljeni zastupnički sustav proširiv, onda tijekom povećanja broja zastupnika u sustavu ne raste kašnjenje po korisničkom zahtjevu. U ovom odjeljku prikazani su rezultati mjerenja prosječnog kašnjenja po zahtjevu za različiti broj zastupnika u sustavu. Broj zastupnika mijenjan je od 2 do 36. U sustavima sa šest i manje zastupnika, na svakom zastupničkom računalu pokrenut je jedan Squid proces. U sustavima s više od šest zastupnika, na svakom računalu je pokrenuto više Squid procesa. Budući da je korišteno ukupno šest mjernih računala, u sustavu od 36 zastupnika na svakom zastupničkom računalu pokrenuto je šest Squid procesa. Rezultati su podijeljeni na pet grafova, ovisno o broju Squid procesa istovremeno pokrenutih na jednom računalu.

Razlog za odvajanje pojedinih rezultata je usporenje rada zastupničkih procesa koji se izvode na istom računalu. Svaki Squid proces djeluje kao nezavisni zastupnik s vlastitim prostorom u radnoj memoriji (RAM) i na čvrstom disku. Usporenje rada Squid procesa javlja se zato što više procesa dijeli resurse jednog računala. Dijeljenje računala povećava kašnjenje po korisničkom zahtjevu. Rezultate dobivene uz različiti broj zastupničkih procesa stoga nije moguće međusobno uspoređivati. Ako se više zastupničkih procesa izvodi na istom računalu, onda se sva komunikacija između njih odvija unutar računala. Sve poruke koje navedeni zastupnici izmjenjuju ne izlaze na mrežu pa su opterećenje mreže i prosječno kašnjenje u tom slučaju manji nego u slučaju da se svaki zastupnik izvodi na zasebnom računalu. Zbog toga je osnovni nedostatak pokretanja više zastupničkih procesa na istom računalu promjena relativnog odnosa kašnjenja na zastupničkim računalima i kašnjenja na mreži. Povećanjem broja zastupničkih procesa povećava se kašnjenje uzrokovano obradom na računalu, a smanjuje se kašnjenje uzrokovano slanjem poruka i podataka preko mreže.

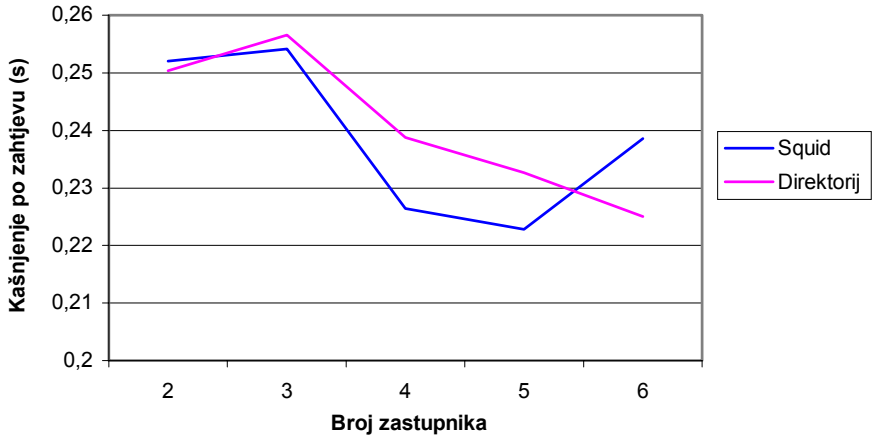
Predzadnji graf u ovom odjeljku prikazuje rezultate za kontinuirano povećavanje broja zastupnika od 2 do 36. Na navedenom grafu primjetni su znatni porasti kašnjenja pri prijelazima na veći broj Squid procesa po računalu. Zadnji graf u odjeljku prikazuje kašnjenje za sustave s 6 do 36 zastupnika. Kašnjenje je mjereno povećavanjem broja zastupnika za šest. Navedenim povećanjem zadržan je

konstantan broj Squid procesa po zastupničkom računalu, a povećanje sustava se ostvaruje dodavanjem novog računala koje izvodi šest Squid procesa. Dobiveni rezultati stoga ne sadrže stepenice.

Budući da je mjerenjima prikazanim u ovom odjeljku određivan utjecaj broja zastupnika u sustavu na kašnjenje po zahtjevu, ostali parametri sustava su konstantni. Mjerenja u ovom odjeljku izvođena su s 10 paralelnih zahtjeva i s objektima veličine 20,000 okteta.

Rezultati za dva do šest zastupnika

Slika 28 prikazuje rezultate mjerenja kašnjenja na sustavima koji imaju od dva do šest zastupnika. Pritom je na svakom zastupničkom računalu bio pokrenut po jedan Squid proces. Graf na slici 28 pokazuje da su za sustav od dva zastupnika razlike između dva protokola zanemarive. Povećanjem na tri zastupnika u sustavu, kašnjenja oba protokola rastu. Kašnjenje Squid protokola raste manje od kašnjenja direktorijskog protokola. Uzrok većeg porasta kašnjenja direktorijskog protokola je opterećenje uzrokovano održavanjem direktorija. Navedeno opterećenje prevladava do broja od pet zastupnika u sustavu. Tijekom povećanja na četiri i pet zastupnika u sustavu kašnjenje oba upravljačka protokola pada. Pad kašnjenja uzrokovan je raspodjelom opterećenja na više zastupničkih računala.



Ukupni broj zastupnika	2	3	4	5	6
Broj računala	2	3	4	5	6
Zastupnika po računalu	1				
Broj paralelnih zahtjeva	10				
Veličina objekta	20,000 okteta				

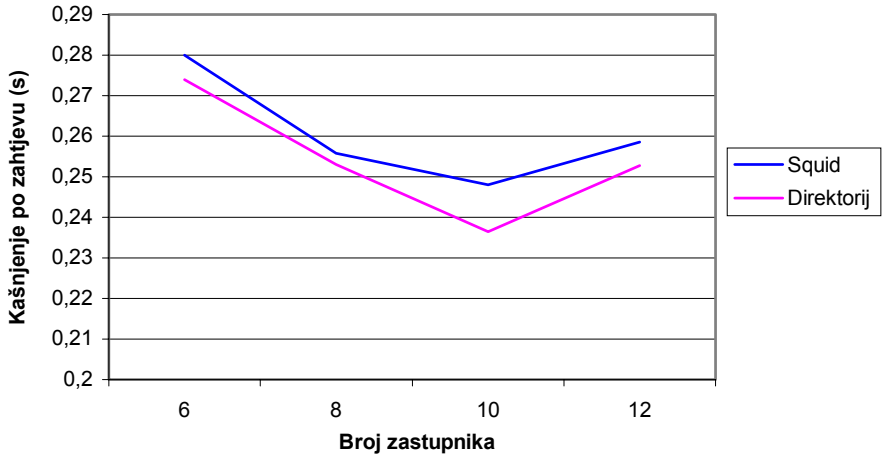
Slika 28: Utjecaj 2-6 zastupnika na kašnjenje

Dodavanjem šestog zastupnika u sustav kašnjenje Squid protokola počinje rasti, a kašnjenje protokola zasnovanog na direktoriju i dalje pada. Uzrok porasta kašnjenja Squid protokola je povećana količina komunikacije između zastupnika. Za šest zastupničkih računala količina komunikacije je približno pet puta veća nego za dva zastupnička računala. Kašnjenje direktorijskog protokola nastavlja padati, jer

zahvaljujući informacijama iz direktorija direktorijski protokol smanjuje količinu komunikacije između zastupnika. Osim toga, korisnički zahtjevi se raspoređuju na više zastupničkih računala.

Rezultati za šest do dvanaest zastupnika

Rezultati prikazani na slici 29 dobiveni su mjerenjem kašnjenja u sustavu s dva Squid procesa na svakom zastupničkom računalu. Graf na slici 29 pokazuje da upravljački protokol zasnovan na direktoriju ima manje kašnjenje od Squid protokola. Kašnjenje je manje na cijelom mjernom rasponu od šest do dvanaest zastupnika. U početnom dijelu krivulje kašnjenja po zahtjevu padaju. Nakon što sustav dosegne broj od deset zastupnika, kašnjenje oba protokola počinje rasti. Usporedbom s istovjetnim kašnjenjem za šest zastupnika na slici 28, vidljivo je da je kašnjenje za šest zastupnika na slici 29 veće. Veće kašnjenje uzrokovano je povećanim brojem zastupničkih procesa koji se izvode na istom računalu.

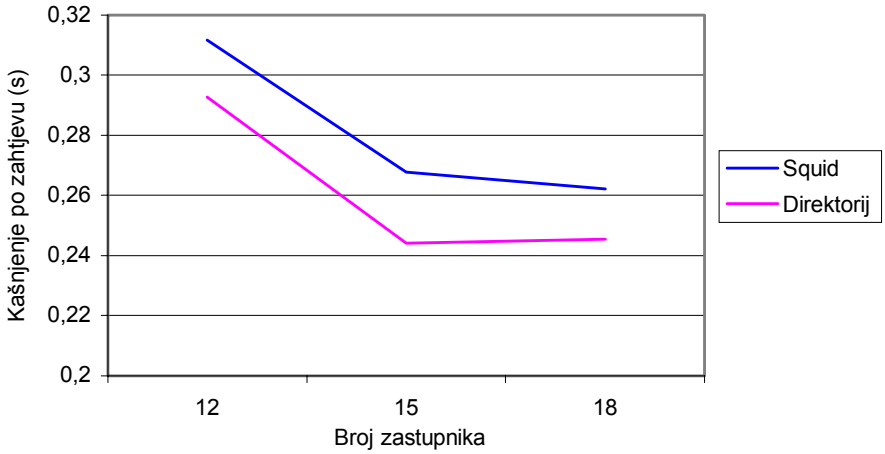


Ukupni broj zastupnika	6	8	10	12
Broj računala	3	4	5	6
Zastupnika po računalu	2			
Broj paralelnih zahtjeva	10			
Veličina objekta	20,000 okteta			

Slika 29: Utjecaj 6-12 zastupnika na kašnjenje

Rezultati za 12 do 18 zastupnika

Slika 30 prikazuje rezultate mjerenja izvedenog s tri zastupnička procesa po računalu. Na grafu se vidi da prosječno kašnjenje nakon povećanja broja procesa raste. Kašnjenje direktorijskog protokola je na cijelom mjernom rasponu manje od kašnjenja Squid protokola. Prosječno kašnjenje pada s dodavanjem novih računala u zastupnički sustav. Navedeni pad značajan je tijekom povećanja sa 12 na 15 zastupničkih računala, ali za 18 računala postaje zanemariv za oba protokola.

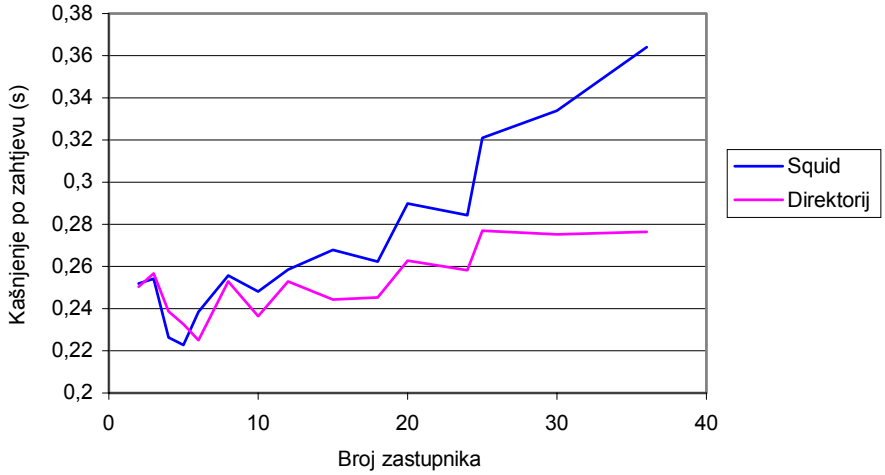


Ukupni broj zastupnika	12	15	18
Broj računala	4	5	6
Zastupnika po računalu	3		
Broj paralelnih zahtjeva	10		
Veličina objekta	20,000 okteta		

Slika 30: Utjecaj 12-18 zastupnika na kašnjenje

Rezultati za 2 do 36 zastupnika

Rezultati prikazani u prethodnom odjeljku sastoje se od samo tri točke, jer se tijekom dodavanja novih zastupničkih procesa u mjerenjima izbjegavalo preklapanje broja zastupnika u zastupničkom sustavu. Jedina iznimka od navedenog pravila su rubne točke. Na primjer, sustav od 10 zastupnika može se ostvariti na dva načina. Prvi način je pokretanje po 2 zastupnička procesa na 5 računala, a drugi način je pokretanje po 5 zastupničkih procesa na 2 računala. Mjerenja za sustav od 10 zastupnika nisu izvedena na oba načina nego samo jednom, odnosno pokretanjem po 2 zastupnička procesa na 5 računala. Zbog toga broj točaka za određeni broj zastupničkih procesa po računalu pada s porastom broja zastupnika. Za vrijednosti iznad tri zastupnička procesa po računalu, broj točaka više nije dovoljan za iscrtavanje grafova. Rezultati mjerenja pokazali su da je navedeni izbor točaka djelomično pogrešan što se vidi na grafu sa slike 31. Na navedenom grafu prikazani su svi izmjereni rezultati, neovisno o broju pokrenutih zastupničkih procesa.



Ukupni broj zastupnika	2	3	4	5	6	8	10	12	15	18	20	24	25	30	36
Broj računala	2	3	4	5	6	4	5	6	5	6	5	6	5	6	6
Zastupnika po računalu	1	1	1	1	1	2	2	2	3	3	4	4	5	5	6
Broj paralelnih zahtjeva	10														
Veličina objekta	20,000 okteta														

Slika 31: Utjecaj 2-36 zastupnika na kašnjenje

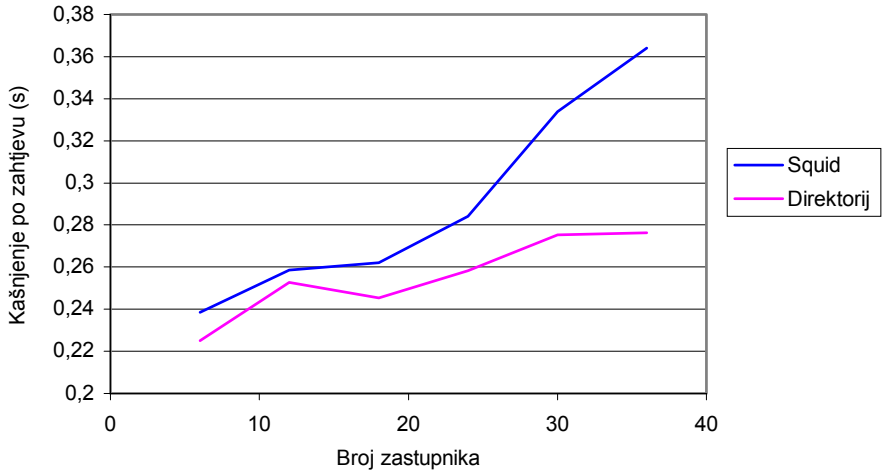
Slika 31 prikazuje rezultate mjerenja dobivene postupnim povećavanjem broja zastupnika u sustavu od 2 do 36. Pritom je prvo povećavan broj računala, a potom je povećavan broj zastupničkih procesa po računalu. Odnose između broja zastupnika, broja računala i broja procesa prikazuje tablica na slici 31. Uočljiva je znatna isprekidanost grafa. Povećanjem broja zastupnika sa 6 na 8 javlja se prva stepenica zbog naglog povećanja kašnjenja. Uzrok porasta kašnjenja je povećanje međuzastupničke komunikacije uz stalno opterećenje koje stvara korisnički program. Istodobno je broj računala, koja čine procesnu snagu sustava, smanjen sa šest na četiri zastupnička računala. Opisana pojava ponavlja se i tijekom prelazaka s 12 na 15 zastupnika, s 18 na 20 zastupnika, te s 24 na 25 zastupnika.

Nakon svakog skoka, kašnjenje se blago smanjuje što je posljedica dodavanja novih računala u sustav. Kašnjenje se smanjuje unatoč povećanju količine komunikacije između zastupnika. Pri manjem broju zastupnika, smanjenje kašnjenja je izrazitije nego uz veći broj zastupnika. Dva su uzroka navedenog ponašanja. Ako je broj zastupničkih procesa po računalu mali, onda je povećanje mrežnog prometa uzrokovano dodavanjem računala u sustav manje nego uz veći broj zastupničkih procesa po računalu. Na primjer, tablica na slici 31 prikazuje da se pri povećanju broja zastupnika s 5 na 6 broj računala poveća s 5 na 6. Pri prijelazu s 15 na 18 zastupnika, broj računala također se poveća s 5 na 6. Ako povećanje mrežnog prometa nakon prelaska s 5 na 6 zastupnika iznosi x, onda povećanje mrežnog prometa s 15 na 18 zastupnika iznosi $3 \cdot x$. U potonjem slučaju, mrežni promet je narastao tri puta, a broj računala se povećao za jedan kao i u prvom slučaju. Drugi je uzrok manjeg smanjenja kašnjenja mali utjecaj komunikacije između zastupnika u odnosu na ukupno kašnjenje uz mali broj zastupnika. Povećanjem broja zastupnika raste utjecaj njihove komunikacije na ukupno kašnjenje pa tijekom povećanja s 25 na 30 zastupnika ne dolazi do smanjenja kašnjenja unatoč povećanju računalne snage.

Iako se rezultati dobiveni uz različit broj zastupničkih procesa po računalu ne mogu izravno uspoređivati, graf na slici 31 služi određivanju trendova. Ako se zanemare skokovi na grafu, onda se vidi da su u početku, uz mali broj zastupnika u sustavu, rezultati različitih protokola slični. U početnom dijelu je količina komunikacije između zastupnika mala pa ne postoji razlog za veće razlike u kašnjenju. Protokol zasnovan na direktoriju u početnom dijelu grafa pokazuje lošije rezultate od protokola ugrađenog u Squid. Troškovi održavanja direktorija i kašnjenje koje unose upiti za dohvat direktorija usporavaju odziv zastupničkog sustava, a ne donose dobitke u pogledu ograničenja komunikacije između zastupnika. Uz više od šest zastupnika u zastupničkom sustavu, razlika kašnjenja Squid protokola i direktorijskog protokola raste. Razlika kašnjenja postaje sve veća pri čemu direktorijski protokol ima manje kašnjenje. U krajnjoj točki s 36 zastupnika, kašnjenje Squid protokola je više od 30% veće od kašnjenja protokola zasnovanog na direktoriju. Primjetno je da kašnjenje direktorijskog protokola teži konačnoj vrijednosti, a kašnjenje Squid protokola nelinearno raste. Navedena konačna vrijednost kašnjenja direktorijskog zastupnika bila bi znatno manja da tijekom mjerenja nisu stvorene stepenice uzrokovane povećanjem broja zastupničkih procesa po računalu.

Rezultati za stalni broj zastupničkih procesa po računalu

Slika 32 prikazuje rezultate dobivene mjerenjem kašnjenja zastupničkih sustava u kojima je broj zastupničkih procesa po računalu konstantan i iznosi šest. Zbog konstantnog broja zastupničkih procesa, ne mijenja se kapacitet obrade korisničkih zahtjeva u pojedinim zastupničkim procesima. Graf na slici 32 stoga je znatno pravilniji od prethodnog i točnije prikazuje trendove. Prosječno kašnjenje raste s povećanjem broja zastupnika. Na cijelom mjernom rasponu direktorijski protokol ima manje kašnjenje od Squid protokola. Razlika između protokola raste s povećanjem broja zastupnika pri čemu je porast kašnjenja direktorijskog protokola znatno blaži. Kašnjenje direktorijskog protokola prestaje rasti nakon što broj zastupnika u sustavu postane veći od 30.



Ukupni broj zastupnika	6	12	18	24	30	36
Broj računala	1	2	3	4	5	6
Zastupnika po računalu	6					
Broj paralelnih zahtjeva	10					
Veličina objekta	20,000 okteta					

Slika 32: Utjecaj 6-36 zastupnika na kašnjenje

4.3.2. Utjecaj broja paralelnih zahtjeva

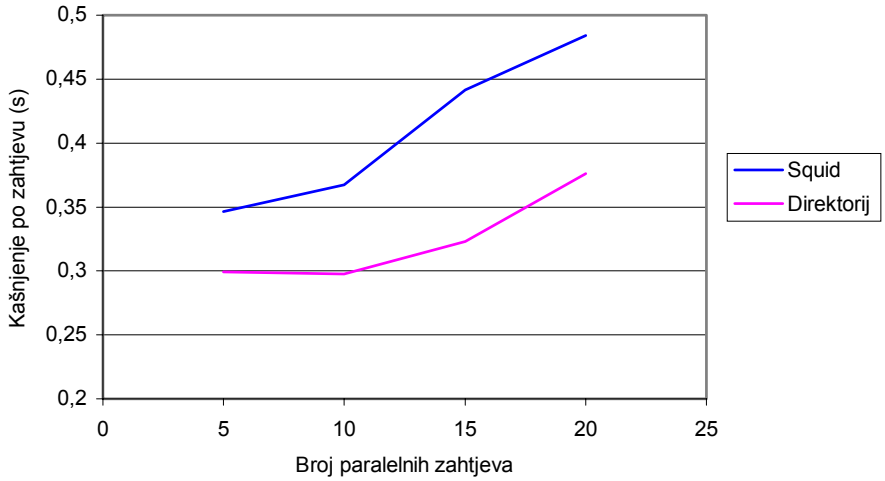
Broj korisnika koji istodobno pristupaju zastupniku značajno utječe na svojstva zastupničkog sustava. Utjecaj se očituje u povećanju mrežnog prometa, povećanju procesorskog vremena i potrošnji sustavskih resursa. Posluživanje što većeg broja korisnika uz što manji rast kašnjenja dohvata objekata temeljni je i najteži zahtjev koji se postavlja zastupničkim sustavima. Mjerenja prikazana u ovom odjeljku prikazuju ovisnost kašnjenja dohvata objekata o broju korisničkih zahtjeva.

Opterećenje raspodijeljenog zastupničkog sustava korisničkim zahtjevima, simulirano je u ovom radu pomoću paralelnih korisničkih zahtjeva. Paralelne zahtjeve stvara PC računalo istovremenim slanjem određenog broja HTTP zahtjeva zastupnicima. Detalji izvođenja korisničkog programa opisani su u odjeljku 4.2. Opisani pristup je prva aproksimacija ponašanja stvarnih sustava. Korisnik tijekom čitanja Web stranica ne dohvaća objekte konstantno, već dohvaća pojedine Web stranice koje se sastoje od više objekata. Nakon što Internet pretraživač dohvati sve objekte koji čine Web stranicu, korisnik određeno vrijeme proučava dohvaćenu stranicu. Stoga se tijekom čitanja Web stranice ne stvaraju novi zahtjevi. Zbog opisanog ponašanja korisnika, broj paralelnih zahtjeva ostvaren u mjerenjima odgovara ponašanju znatno većeg broja stvarnih korisnika. Točan broj predstavljenih korisnika teško je odrediti, jer on ovisi o ponašanju korisnika, o vremenu koje korisnik potroši na proučavanje Web stranice i o veličini i broju objekata na Web stranici.

Mjerenje utjecaja korisničkog opterećenja na upravljačke protokole raspodijeljene priručne memorije, ostvareno je mijenjanjem broja paralelnih zahtjeva tijekom izvođenja mjerenja. Dobiveni grafovi na svojoj *y* osi prikazuju kašnjenje izraženo u sekundama, a na *x* osi je broj paralelnih zahtjeva. Rezultati i pripadni grafovi podijeljeni su u četiri skupine, ovisno o veličini umjetno stvorenih objekata. Korištene su četiri veličine objekata, od 5,000, 10,000, 20,000 i 40,000 okteta. Ostali parametri su u svim mjerenjima konstantni. Korišteno je 36 zastupničkih procesa ostvarenih pokretanjem po šest Squid procesa na šest zastupničkih računala.

Rezultati za objekte veličine 5,000 okteta

Slika 33 prikazuje rezultate mjerenja dobivene mijenjanjem broja paralelnih zahtjeva uz konstantnu veličinu objekata od 5,000 okteta. Graf na slici 33 prikazuje da povećanjem opterećenja sustava dolazi do povećanja prosječnog kašnjenja. Opterećenje sustava iskazano je brojem paralelnih zahtjeva. Razlog porasta kašnjenja je povećana količina podataka koju zastupnički sustav obrađuje u istoj jedinici vremena. Dodatno vrijeme potrebno za prijenos i obradu upita i zahtjeva uzrokuje porast prosječnog vremena dohvata objekta. Na slici 33 vidljivo je znatno manje kašnjenje upravljačkog protokola zasnovanog na direktoriju u odnosu na Squid protokol. Već u početnom dijelu krivulje direktorijski protokol ima približno 15% manje kašnjenje. Porastom broja paralelnih zahtjeva, povećava se razlika u kašnjenju ispitivanih protokola. Squid protokol pritom ima veći porast kašnjenja nego direktorijski protokol.

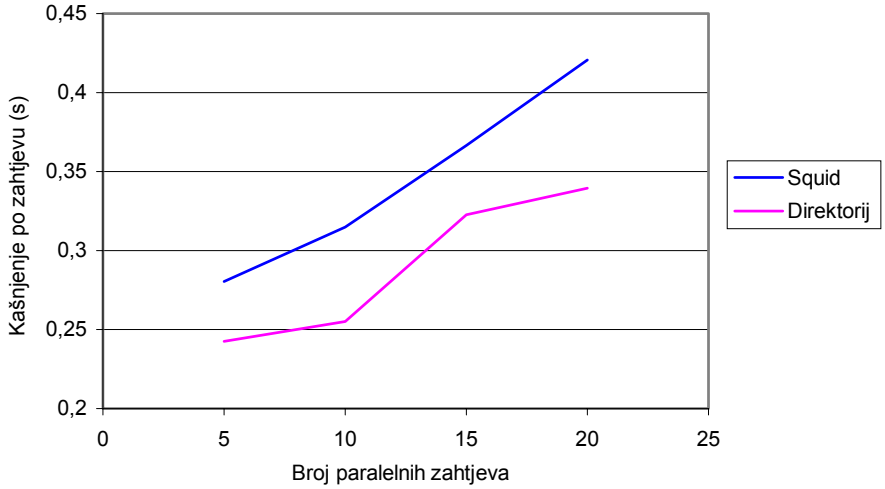


Ukupni broj zastupnika	36			
Broj računala	6			
Zastupnika po računalu	6			
Broj paralelnih zahtjeva	5	10	15	20
Veličina objekta	5,000 okteta			

Slika 33: Utjecaj broja zahtjeva uz objekte od 5,000 okteta

Rezultati za objekte veličine 10,000 okteta

Slika 34 prikazuje utjecaj broja paralelnih zahtjeva na prosječno kašnjenje uz uporabu objekata od 10,000 okteta. Na cijelom mjernom rasponu protokol zasnovan na direktoriju daje bolje rezultate od Squid protokola. Graf na slici 34 prikazuje brz rast prosječnog kašnjenja oba protokola. Porast kašnjenja proporcionalan je porastu broja paralelnih zahtjeva. Ukupni porast prosječnog kašnjenja direktorijskog protokola blaži je od porasta prosječnog kašnjenja Squid protokola. Relativna razlika u vremenima kašnjenja navedenih protokola manja je od razlike koja se javlja ako se koriste objekti od 5,000 okteta. Do smanjenja razlike kašnjenja dolazi zato što porastom veličine objekata raste udio vremena dohvata objekata u prosječnom kašnjenju. Udio kašnjenja uzrokovanog upravljačkim protokolom sve je manji pa su i razlike kašnjenja manje.

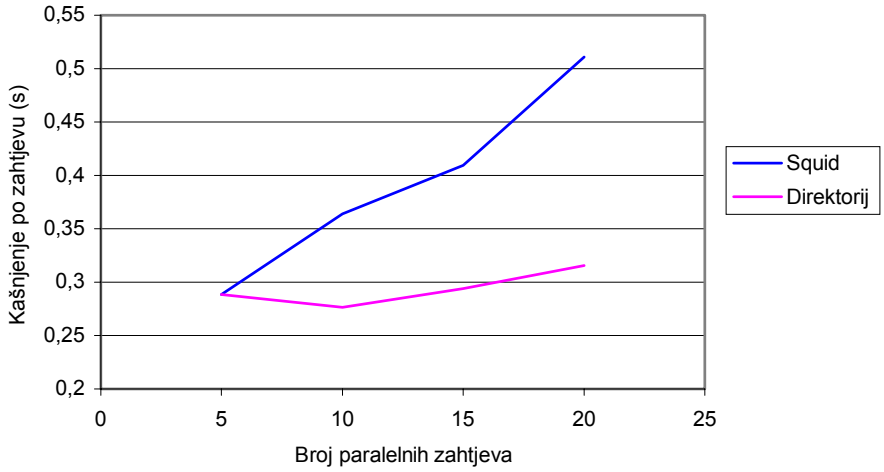


Ukupni broj zastupnika	36			
Broj računala	6			
Zastupnika po računalu	6			
Broj paralelnih zahtjeva	5	10	15	20
Veličina objekta	10,000 okteta			

Slika 34: Utjecaj broja zahtjeva uz objekte od 10,000 okteta

Rezultati za objekte veličine 20,000 okteta

Nakon povećanja veličine objekata na 20,000 okteta izveden je slijedeći skup mjerenja. Dobiveni rezultati prikazani su na slici 35. Pri malom broju paralelnih zahtjeva kašnjenje Squid protokola i direktorijskog protokola približno je jednako. Povećavanjem broja paralelnih zahtjeva prosječno kašnjenje Squid protokola strmo raste. Kašnjenje dohvata objekata direktorijskog protokola istodobno ostaje približno konstantno. Kašnjenje direktorijskog protokola, uz manja odstupanja, iznosi 0.3 sekundi. Budući da kašnjenje Squid protokola brzo raste, a kašnjenje direktorijskog protokola ostaje konstantno, razlika između ispitivanih protokola raste na cijelom mjernom rasponu.

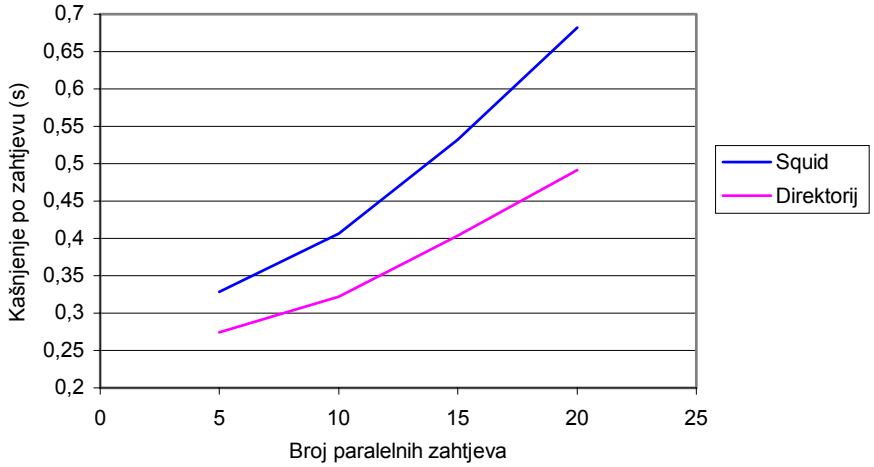


Ukupni broj zastupnika	36			
Broj računala	6			
Zastupnika po računalu	6			
Broj paralelnih zahtjeva	5	10	15	20
Veličina objekta	20,000 okteta			

Slika 35: Utjecaj broja zahtjeva uz objekte od 20,000 okteta

Rezultati za objekte veličine 40,000 okteta

Povećanjem veličine objekata na 40,000 okteta prosječno kašnjenje dohvata objekta povećalo se u odnosu na prethodnu skupinu mjerenja. Izmjerene rezultate prikazuje slika 36. Već uz mali broj paralelnih zahtjeva vidljiva je razlika u kašnjenju između Squid protokola i direktorijskog protokola. Porastom broja paralelnih zahtjeva od 5 prema 20, kašnjenje Squid protokola brzo raste. Kašnjenje protokola zasnovanog na direktoriju u ovom mjerenju nije konstantno, već raste. Porast kašnjenja direktorijskog protokola primjetno je sporiji od porasta kašnjenja Squid protokola. Na punom broju zahtjeva razlika u kašnjenju između ispitivanih protokola višestruko se povećala u odnosu na mali broj zahtjeva.



Ukupni broj zastupnika	36			
Broj računala	6			
Zastupnika po računalu	6			
Broj paralelnih zahtjeva	5	10	15	20
Veličina objekta	40,000 okteta			

Slika 36: Utjecaj broja zahtjeva uz objekte od 40,000 okteta

Rezultati mjerenja opisani u ovom odjeljku pokazuju da prosječno kašnjenje dohvata objekata u raspodijeljenom zastupničkom sustavu raste s porastom broja paralelnih zahtjeva. Navedeni rezultati su u skladu s očekivanjima. Usporedba Squid protokola i direktorijskog protokola pokazuje da direktorijski protokol ima manje kašnjenje na cijelom rasponu od 5 do 20 paralelnih zahtjeva za sve korištene veličine objekata. Direktorijski protokol osim toga ima i manji porast kašnjenja u odnosu na Squid protokol, što je posebno značajno za ostvarivanje velikih zastupničkih sustava.

4.3.3. Utjecaj veličine objekata

Veličina objekta utječe na kašnjenje dohvata objekta iz raspodijeljene priručne memorije. Kašnjenje pri dohvatu objekta sastoji se od vremena potrebnog upravljačkom protokolu da provjeri da li se objekt nalazi u raspodijeljenoj priručnoj memoriji i od vremena potrebnog da se objekt dohvati. Tijekom dohvata moguće je da se objekt prenosi jednom ili dvaput, ovisno o tome da li je pronađen u lokalnoj priručnoj memoriji glavnog zastupnika ili se dohvaća od susjednog zastupnika ili poslužitelja. Jedan je prijenos potreban za slanje objekta od glavnog zastupnika do korisnika i on je uvijek prisutan. Drugi je prijenos, koji se ne pojavljuje pri svakom zahtjevu, potreban za dohvat objekta od susjednog zastupnika ili poslužitelja.

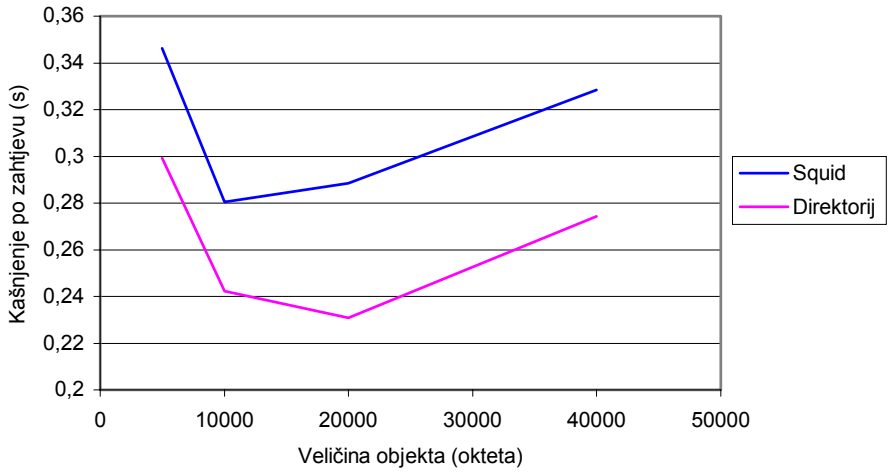
Utjecaj veličine objekta na trajanje prijenosa objekta očit je, jer prijenos većih objekata traje dulje. Vrijeme prijenosa objekta nije linearno ovisno o veličini objekata i raste sporije od porasta veličine objekata. Navedena pojava opisana je u člancima [25, 26], a njen uzrok su dodatne informacije i obrada koja se javlja tijekom prijenosa svakog objekta (engl. *overhead*). Posljedica je da se s velikim objektima postiže veća propusnost, nego s malim objektima.

Daljnje povećanje kašnjenja odnosi se na prosječno kašnjenje uzrokovano upravljačkim protokolom raspodijeljene priručne memorije. Budući da upravljački protokol za svaki traženi objekt stvara kontrolne poruke čiji broj i veličina ne ovise o veličini objekta, promet uzrokovan upravljačkim protokolom ovisit će o broju zahtjeva u jedinici vremena. Dohvaćanje malih objekata odvija se brže od dohvaćanja velikih objekata pa je u jedinici vremena moguće dohvatiti veći broj malih objekata. Veći broj zahtjeva u jedinici vremena stvara veće opterećenje uzrokovano upravljačkim protokolom. Zbog toga ukupno kašnjenje po zahtjevu ne pada ispod minimuma određenog opterećenjem sustava.

Mjerenje utjecaja veličine objekata na svojstva upravljačkih protokola raspodijeljene priručne memorije, izvedeno je mijenjanjem veličine objekata. Ponašanje sustava mjereno je u četiri grupe mjerenja s 5, 10, 15 i 20 paralelnih zahtjeva. Mjerni sustavi imali su 36 zastupnika ostvarenih pokretanjem po šest Squid procesa na šest zastupničkih računala. Grafovi s odgovarajućim rezultatima na svojoj *y* osi prikazuju kašnjenje izraženo u sekundama, a na *x* osi veličinu objekata.

Rezultati za 5 paralelnih zahtjeva

Slika 37 prikazuje rezultate dobivene mjerenjem prosječnog kašnjenja uz pet paralelnih zahtjeva. Direktorijski protokol ima manje kašnjenje na cijelom području na kojem su mjerenja izvođena. U početnom dijelu krivulje za objekte veličine 5,000 okteta Squid protokol ima približno 15% veće kašnjenje. Povećanjem veličine objekata s 5,000 na 10,000 okteta kašnjenje znatno pada. Nakon navedenog pada, prosječno kašnjenje po zahtjevu počinje rasti. Pritom je brzina rasta kašnjenja oba protokola približno jednaka. Primjetno je da se minimum kašnjenja direktorijskog protokola postiže s većim objektima nego kod Squid protokola. Squid protokol minimalno kašnjenje postiže s objektima veličine 10,000 okteta, a direktorijski protokol s objektima veličine 20,000 okteta.



Ukupni broj zastupnika	36			
Broj računala	6			
Zastupnika po računalu	6			
Broj paralelnih zahtjeva	5			
Veličina objekta (okteta)	5,000	10,000	20,000	40,000

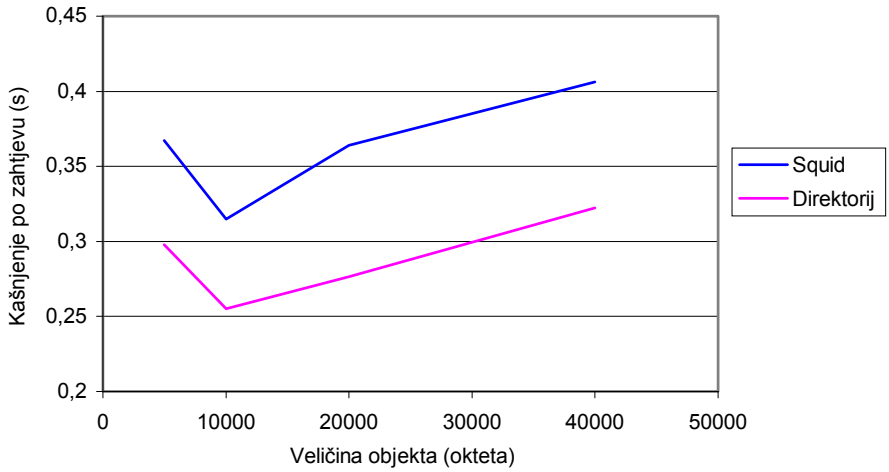
Slika 37: Utjecaj veličine objekta uz 5 paralelnih zahtjeva

U rezultatima mjerenja prikazanim na slici 37 u početnom dijelu krivulje javlja se pad kašnjenja. Ukupno kašnjenje sastoji se od dva dijela. Prvi dio je kašnjenje zbog prijenosa objekta koje ovisi o veličini objekta. Drugi dio ukupnog kašnjenja je kašnjenje koje unosi upravljački protokol raspodijeljene priručne memorije. Navedeni dio kašnjenja ovisi o broju objekata koji se prenose. Smanjenje udjela kašnjenja uzrokovanog upravljačkim protokolom u ukupnom kašnjenju uzrok je pada ukupnog kašnjenja na početku grafa prikazanog na slici 37. Mali objekti, kao što su u prikazanim mjerenjima objekti veličine 5,000 okteta, brže se prenose preko mreže. Zbog toga je udio kašnjenja prijenosa kod malih objekata manji nego kod većih objekata. Značaj kašnjenja koje uzrokuje upravljački protokol u slučaju prijenosa malih objekata je veći, jer zastupnički sustav dohvaća više malih objekata. Povećanjem veličine objekata, udio kašnjenja vezanog uz veličinu objekata raste. Istodobno opada broj objekata zbog čega pada i udio kašnjenja vezanog za upravljački protokol. Prema tome, porastom veličine objekata do određenog praga, ukupno kašnjenje prijenosa objekta pada. Nakon što se prijeđe navedeni prag, udio kašnjenja vezanog uz prijenos objekata počinje

preuzimati značajniji udio u ukupnom kašnjenju. Posljedica je rast ukupnog kašnjenja zbog porasta vremena prijenosa objekata.

Rezultati za 10 paralelnih zahtjeva

Rezultate dobivene s 10 paralelnih zahtjeva prikazuje slika 38. Protokol zasnovan na direktoriju ima bolje rezultate na cijelom mjernom rasponu veličina objekata. Kašnjenje direktorijskog protokola u prosjeku je približno 20% manje od kašnjenja Squid protokola. Nakon početnog znatnog pada i postizanja minimuma, kašnjenje oba protokola raste približno jednakom brzinom. Primjetno je da se pri opterećenju s 10 paralelnih zahtjeva minimumi kašnjenja ispitivanih protokola poklapaju na 10,000 okteta.

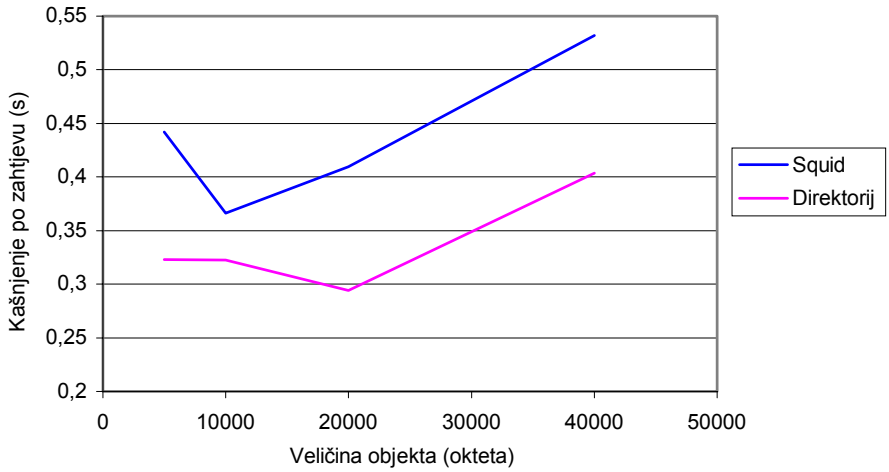


Ukupni broj zastupnika	36			
Broj računala	6			
Zastupnika po računalu	6			
Broj paralelnih zahtjeva	5			
Veličina objekta (okteta)	5,000	10,000	20,000	40,000

Slika 38: Utjecaj veličine objekta uz 10 paralelnih zahtjeva

Rezultati za 15 paralelnih zahtjeva

Povećanjem broja paralelnih zahtjeva na 15 dobiveni su rezultati prikazani na slici 39. Na cijelom je mjernom području kašnjenje direktorijskog protokola manje od kašnjenja Squid protokola. Kašnjenje direktorijskog protokola u prosjeku je približno 20% manje od Squid protokola. Minimum kašnjenja za Squid protokol nalazi se na veličini objekata od 10,000 okteta. Početni pad kašnjenja direktorijskog protokola dovodi do minimuma pri veličini objekata od 20,000 okteta. Nakon postizanja minimuma, kašnjenja oba protokola rastu s približno istim nagibom.

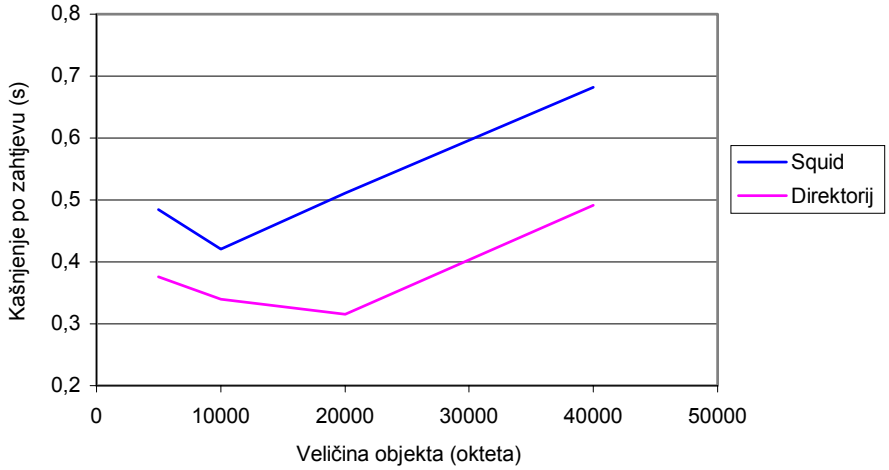


Ukupni broj zastupnika	36			
Broj računala	6			
Zastupnika po računalu	6			
Broj paralelnih zahtjeva	5			
Veličina objekta (okteta)	5,000	10,000	20,000	40,000

Slika 39: Utjecaj veličine objekta uz 15 paralelnih zahtjeva

Rezultati za 20 paralelnih zahtjeva

Konačnim povećanjem opterećenja zastupničkog sustava na 20 paralelnih zahtjeva dobiveni su rezultati predstavljeni na slici 40. Kao i u prethodnim mjerenjima, direktorijski protokol ima manje kašnjenje od Squid protokola na cijelom mjernom rasponu. U prosjeku je kašnjenje direktorijskog protokola 30% manje od kašnjenja Squid protokola. Oba protokola imaju početni pad kašnjenja po zahtjevu, nakon čega postižu minimum. Minimum kašnjenja Squid protokola postiže se za objekte veličine 10,000 okteta, a minimum kašnjenja direktorijskog protokola za objekte veličine 20,000 okteta. Daljnjim povećanjem veličine objekata kašnjenje oba protokola raste. Porast kašnjenja Squid protokola i direktorijskog protokola približno je jednak.



Ukupni broj zastupnika	36			
Broj računala	6			
Zastupnika po računalu	6			
Broj paralelnih zahtjeva	5			
Veličina objekta (okteta)	5,000	10,000	20,000	40,000

Slika 40: Utjecaj veličine objekta uz 20 paralelnih zahtjeva

U rezultatima mjerenja prikazanim u ovom odjeljku krivulje kašnjenja Squid i direktorijskog protokola imaju točku minimuma za objekte veličine 10,000, odnosno 20,000 okteta. Prema desnoj strani grafa na kojoj veličina objekata teži u beskonačnost, ukupno kašnjenje po zahtjevu oba protokola linearno raste prema beskonačnosti. Na lijevoj strani grafa na kojoj veličina objekata teži prema nuli, ukupno kašnjenje po zahtjevu raste prema konačnoj vrijednosti. Navedena konačna vrijednost ovisi o količini mrežnog prometa i procesorskog opterećenja koje stvara upravljački protokol. Prema tome, konačna vrijednost ovisna je o upotrijebljenom upravljačkom protokolu raspodijeljene priručne memorije.

Osim opisanog svojstva, rezultati mjerenja prikazani u ovom odjeljku pokazuju da je prosječno kašnjenje po zahtjevu direktorijskog protokola manje od kašnjenja Squid protokola. Kašnjenje je manje za sve korištene veličine objekata i za sva korištena opterećenja sustava. Minimum kašnjenja

direktorijskog protokola postiže se za veće objekte nego kod Squid protokola. Brzina pada i porasta kašnjenja Squid protokola i direktorijskog protokola približno je ista u svim mjerenjima.

5. Zaključak

Zastupnici su važna sastavnica Internet sustava. Iako zastupnici ne rješavaju dio problema Interneta vezanih uz multimedijske sadržaje, značajan je njihov doprinos smanjenju kašnjenja tijekom dohvata Internet stranica. Zastupnici povezani u sustave raspodijeljene priručne memorije ostvaruju znatno bolje rezultate od izdvojenih zastupničkih računala. Većina upravljačkih protokola raspodijeljene priručne memorije, dostupnih na tržištu i prikazanih u znanstvenim radovima, nije prikladna za raspodijeljene zastupničke sustave s velikim brojem zastupnika. Protokoli Squid i Berkeley opisani u ovom radu, imaju dobra svojstva u manjim skupinama zastupnika. Povećanjem broja zastupnika u sustavu, kašnjenje navedenih protokola brzo raste. Squid protokol brzo se zasićuje, a Berkeley protokol ima manji rast kašnjenja, ali također ulazi u zasićenje. Analitički model upravljačkog protokola, stvoren u organizaciji AT&T Labs, pokazuje da je za ostvarenje proširivih sustava najpogodniji direktorijski protokol. Prednost je navedenog protokola otklanjanje porasta mrežnog opterećenja i porasta opterećenja zastupničkih računala.

U magistarskom radu su, na temelju suvremenih komunikacijskih standarda i protokola, definirane poruke potrebne za programsko ostvarenje direktorijskog protokola. Analizom izvornog kôda besplatnog zastupničkog sustava Squid prikupljene su informacije potrebne za programsko ostvarenje. Potom je, ugradnjom u Squid, programski ostvaren direktorijski protokol. Uporabom navedenog programskog ostvarenja, provedena su mjerenja svojstava protokola Squid i direktorijskog protokola. Osmišljen je mjerni sustav kojeg čini stvarni zastupnički sustav koji prima zahtjeve od korisnika. Izmjerena su svojstva osnovnog Squid sustava i Squid sustava s ugrađenim direktorijskim protokolom.

Rezultati mjerenja odgovaraju predviđanjima analitičkog modela. Kašnjenje direktorijskog protokola veće je od kašnjenja protokola Squid u sustavima koji imaju manje od pet zastupnika. Veće je kašnjenje u malim sustavima očekivano, jer direktorijski protokol uzrokuje dodatnu obradu svakog korisničkog zahtjeva. U sustavima koji imaju više od pet zastupnika, kašnjenje direktorijskog protokola znatno je manje od kašnjenja protokola Squid. Najznačajnija je prednost direktorijskog protokola neznatan rast kašnjenja pri povećanju broja zastupnika u sustavu. Navedeno svojstvo omogućuje izgradnju proširivih sustava raspodijeljene priručne memorije koji poslužuju stotine tisuća korisnika. Obzirom na stalni porast broja Internet korisnika i prometa, značaj proširivih zastupničkih sustava stalno raste.

6. Literatura

[1] T. Socolofsky, C. Kale: "**A TCP/IP Tutorial**", RFC 1180, January 1991.

[2] D. E. Comer: "**Internetworking With TCP/IP, Vol I: Principles, Protocols and Architecture**", Second Edition, *Prentice Hall*, New Jersey, 1991.

[3] J. Postel and J. Reynolds: "**File Transfer Protocol (FTP)**", RFC 0959, October 1985.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P.Leach, and T. Berners-Lee: "**Hypertext Transfer Protocol – HTTP/1.1**", RFC 2616, June 1999.

[5] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell: "**A Hierarchical Internet Object Cache**", *Proceedings of USENIX 1996 Annual Technical Conference*, January 1996.

[6] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, M. F. Schwartz, and D. P. Wessels: "**Harvest: A Scalable, Customizable Discovery and Access System**", *Technical Report CU-CS-732-94*, Department of Computer Science, University of Colorado, Boulder, March 1995.

[7] D. R. Hardy, M. F. Schwartz, and D. Wessels: "**Harvest User's Manual**", *Technical Report CU-CS-743-94*, Department of Computer Science, University of Colorado, Boulder, September 1995.

[8] "**Squid Web Proxy Cache**", http://www.squid.com

[9] D. Wessels and K. Claffy: "**Internet Cache Protocol (ICP), version 2**", RFC 2186, September 1997.

[10] D. Wessels and K. Claffy: "**Application of Internet Cache Protocol (ICP), version 2**", RFC 2187, September 1997.

[11] R. Malpani, J. Lorch, and D. Berger: **"Making World Wide Web Caching Servers Cooperate"**, *World Wide Web Journal*, Vol. I, Issue 1, Winter 1996.

[12] B. M. Duska, D. Marwood, and M. J. Feeley: **"The Measured Access Characteristics of World-Wide-Web Client Proxy Caches"**, *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[13] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy: **"On the scale and performance of cooperative Web proxy caching"**, *17th ACM Symposium on Operating Systems Principles (SOSP '99)*, *Operating Systems Review* 34(5):16-31, December 1991.

[14] V. Vallopillil and K. W. Ross: "**Internet Draft: Cache Array Routing Protocol v1.0**", February 1998. (http://www.web-cache.com/Writings/Internet-Drafts/draft-vinod-carp-v1-03.txt)

[15] S. Gadde, J. Chase, and M. Rabinovich: "**Directory Structures for Scalable Internet Caches**", *Technical Report CS-1997-18*, Department of Computer Science, Duke University, November 1997. (http://www.research.att.com/~misha/crisp/distrProxy/lazyCrisp.ps.gz)

[16] L. Fan, P. Cao, J. Almeida, and A. Z. Broder: "**Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol**", *Proceedings of the ACM SIGCOMM '98*, Vancouver, Canada, September 1998.

[17] B. Bloom: **"Space/time trade-off in hash coding with allowable errors"**, *Communications of ACM*, vol. 13, no. 7, pp. 422-426, July 1970.

[18] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay: "**Design Considerations for Distributed Caching on the Internet**", *Proceedings of International Conference on Distributed Computing Systems*, Austin, Texas, USA, May 1999.

[19] R. Rivest: **"The MD5 Message-Digest Algorithm"**, RFC 1321, April 1992.

[20] L. Zhang, S. Floyd, and V. Jacobson: **"Adaptive Web Caching"**, *2nd Web Caching Workshop*, Boulder, Colorado, USA, June 1997.

[21] L. Zhang, S. Michel, K. Nguyen, A. Rosenstein, S. Floyd, and V. Jacobson: "**Adaptive Web Caching: Towards a New Global Caching Architecture**", *3rd International WWW Caching Workshop*, Boulder, Colorado, USA, June 1998.

[22] S. Srbljić, Z. G. Vranešić, M. Stumm, and L. Budin: "**Analytical Prediction of Performance for Cache Coherence Protocols**", *IEEE Transactions on Computers*, Vol. 46, No. 11, November 1997.

[23] S. Srbljić, P.P. Dutta, T.B. London, D.F. Vrsalović, and J.J. Chiang: "**Scalable Distributed Caching System and Method**", *United States Patent* 5,933,849, issued August 3, 1999. (http://www.uspto.gov/)

[24] S. Srbljić, P.P. Dutta, T.B. London, D.F. Vrsalović, and J.J. Chiang: " **Scalable network object caching**", *United States Patent* 6,154,811, issued November 28, 2000. (http://www.uspto.gov/)

[25] A. Milanović, S. Srbljić, and J. Radej: "**Performance of Distributed Systems Based on Ethernet and Personal Computers**", *Proceedings of the IEEE International Symposium on Industrial Electronics*, Bled, Slovenia, June 1999, vol. 1, pp. 79-83.

[26] A. Milanović, S. Srbljić, and V. Sruk: **"Performance of UDP and TCP Communication on Personal Computers"**, *Proceedings of the 10th Mediterranean Electrotechnical Conference, MEleCon 2000*, Lemesos, Cyprus, May 2000, vol. 1, pp. 286-289.

[27] S. Srbljić, A. Milanović, and N. Hadjina: **"Performance Tuning of Large-Scale Distributed WWW Caches"**, *Proceedings of the 10th Mediterranean Electrotechnical Conference, MEleCon 2000*, Lemesos, Cyprus, May 2000, vol. 1, pp. 93-96.

[28] S. Srbljić, D. F. Vrsalović, and A. Milanović: **"Directory Based Multi-tier Internet Architectures"**, *Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2001)*, Orlando, Florida, USA, July 2001, vol. 12, pp. 217-223.

[29] M. Hamilton, A. Rousskov, and D. Wessels: **"Cache Digest specification – version 5"**, December 1998. (http://www.squid-cache.org/CacheDigest/cache-digest-v5.txt)

[30] P. Vixie and D. Wessels: **"Hypertext Caching Protocol (HTCP/0.0)"**, RFC 2756, January 2000.

7. Životopis

Rođen sam 5.8.1974. u Zagrebu, gdje sam pohađao osnovnu i srednju školu. Nakon što sam 1993. godine maturirao u XV gimnaziji, upisao sam Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu, u to doba Elektrotehnički fakultet. Tijekom studija primio sam tri pohvale "Josip Lončar" kao jedan od najboljih studenata na pojedinim godinama studija, te brončanu plaketu "Josip Lončar" za najbolji ukupni uspjeh tijekom studija na smjeru Računarska tehnika. Diplomirao sam 1998. godine s radom naslovljenim "Analiza prometa HTTP paketa na lokalnoj mreži". Nakon toga zaposlio sam se kao znanstveni novak na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu. Kao znanstveni novak sudjelujem na projektu 036014 "Računalna potpora rješavanju inženjerskih zadataka" čiji je voditelj prof. dr. sc. Leo Budin.

U ožujku 1998. godine upisao sam poslijediplomski studij na Fakultetu elektrotehnike i računarstva u Zagrebu, smjer Jezgra računarstva. Osim nastavnog dijela, pod vodstvom mentora prof. dr. sc. Siniše Srbljića bavim se istraživanjem prikladnosti osobnih računala i operativnih sustava za rad u realnom vremenu na Internetu. Osim toga, radim na primjeni upravljačkih protokola raspodijeljene priručne memorije u Internet sustavima. Od srpnja 1999. do siječnja 2000. boravio sam u istraživačkoj ustanovi AT&T Labs, Internet Platform Technology Organization, San Jose, California, SAD. U navedenoj ustanovi radio sam na programskom ostvarenju direktorijskog upravljačkog protokola raspodijeljene priručne memorije i mjerenju svojstava upravljačkih protokola. Nakon povratka iz SAD-a vratio sam se na mjesto znanstvenog novaka na fakultetu.

Sudjelovao sam na međunarodnim konferencijama s dva članka u području rada Internet sustava u realnom vremenu, te s dva članka u području upravljačkih protokola raspodijeljene priručne memorije.

8. Sažetak

Raspodijeljene priručne memorije značajno smanjuju kašnjenje dohvata Internet stranica. Učinkovitost raspodijeljenih priručnih memorija određuju njihovi upravljački protokoli. Magistarski rad opisuje programsko ostvarenje direktorijskog upravljačkog protokola i uspoređuje direktorijski i Squid protokol na osnovi rezultata mjerenja. U svrhu programskog ostvarenja u sustavu Squid proučeni su komunikacijski standardi koji se koriste u suvremenim zastupničkim sustavima. Analiziran je protokol koji sustav Squid koristi za upravljanje raspodijeljenim zastupničkim sustavima. Na osnovi komunikacijskih standarda i postojećeg upravljačkog protokola definirane su nove komunikacijske poruke za upravljanje raspodijeljenom priručnom memorijom. Nakon programskog ostvarenja direktorijskog protokola, uspoređena su njegova svojstva sa svojstvima protokola Squid. U svrhu usporedbe protokola ostvaren je mjerni sustav za mjerenje kašnjenja direktorijskog i Squid protokola. Programski su ostvareni mjerni programi i definirani uvjeti izvođenja mjerenja. Prikupljeni rezultati grafički su prikazani i analizirani. Na osnovi analize uspoređena su svojstva proširivosti direktorijskog i Squid protokola.

9. Summary

"Distributed Cache Management Protocol"

Distributed cache systems significantly reduce the latency of Web page download. Efficiency of distributed cache systems is affected by the distributed cache management protocols. The master thesis describes an implementation of a directory-based cache management protocol. The thesis also compares the performance of Squid protocol and directory-based protocol. In order to implement directory-based protocol in Squid, communication standards used in modern proxy systems were studied. The cache management protocol used by Squid proxy was analyzed. Using the knowledge of communication standards and existing cache management protocols new cache management communication messages were defined. Once the directory-based protocol was implemented, its performance was compared to the performance of Squid protocol. In order to compare the performance of the protocols, a measurement system was developed. The measurement system was used to measure the latency of Squid protocol and directory-based protocol. Measurement programs were implemented and the measurement environment was defined. The measurement results are presented and analyzed. Based on the analysis, scalability of Squid and directory-based protocols is compared.

10. Ključne riječi

priručna memorija, upravljački protokol, raspodijeljeni sustavi, zastupnik, Internet

cache, management protocol, distributed systems, proxy, Internet