UNIVERZITET U NOVOM SADU

FAKULTET TEHNIČKIH NAUKA

Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija
Dekanat: 021 350-413 450-810; Centrala: 021 350-122
Računovodstov: 021 58-220; Studentska služba: 021 350-763
Telefon/fax: 021 58-133; ftndean@uns.ac.rs

Studijski program
Geodezija i geomatika

Seminarski rad

UDP slanje i prijem poruka

Profesor: dr Vladimir Bulatović
Student: Mehmed Batilović, br. indeksa: o375
E-mail: batilovicm@gmail.com

Novi Sad, maj 2014

Sadržaj

1 Uvod 3
2 Modeli slojevite računarske komunikacije 4
2.1 *OSI* referentni model 4
2.2 *TCP/IP* model 5
3 *IP* protokol 6
4 *UDP* protokol 8
4.1 Uvod 8
4.2 Uloga i krakateristike *UDP* protokola 8
4.3 *UDP* portovi 10
4.4 Zaglavlje *UDP* datagrama 11
4.5 Enkapsulacija i deenkapsulacija 12
4.6 Multipleksirane i demultipleksiranje 13
4.7 Reasembliranje *UDP* datagrama 14
4.8 *UDP* klijent-server procesi i zahtevi 14
4.9 Prednosti *UDP* protokola 15
5 Klijent - server komunikacija u programskom jeziku *Java* 16
6 Literatura 22

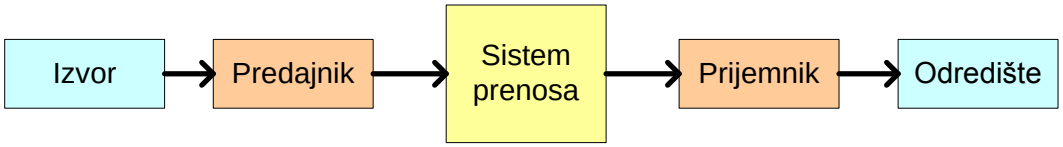
Rezime: Tema ovog rada je *UDP* protokol, slanje i prijem poruka. U radu je kratko opisano šta su računarske mreže i protokoli, kako se vrši prenos podataka u mreži, *OSI* refrentni model, *TCP/IP* model i *IP* protokol, a sve u cilju razumevanja *UDP* protokola. Detaljno je opisan *UDP* protokol. Na primeru klijent-server komunikacije, implementiranom u programskom jeziku *Java*, ilustrovana je primena *UDP* protokola.

Ključne reči: Protokol, *UDP, OSI, TCP/IP, IP, Port*, Datagram, *Socket.*

1 Uvod

Računarska mreža se može posmatrati kao komunikacioni sistem, gde se informacija generisana na predajnoj strani (izvorištu poruke) dostavlja na željenom odredištu [1]. Osnovni elementi komunikacionog sistema su (Sl. 1-1):

- Izvor – generiše podatke za prenos;
- Predajnik – transformiše generisane podatak u oblik pogodan za prenos;
- Sistem prenosa – može biti jednostavna linija ili komplesna mreža koja spaja izvor i odredište;
- Prijemnik – prihvata signal iz sistema za prenos i transformiše ga u oblik pogodan za odredište;
- Odredište – prihvata prenete podatke [1].



Sl. 1-1: Komunikacioni sistem;

Postoji nekoliko načina za prenos podataka u mrežama. Prvi način, veza između izvorišta i odredišta uspostavlja se kroz čvorove mreže, na način da se zauzima kompletan spojni put. Drugi način, poruka se deli u manje celine – pakete, a kroz mrežu se paketi mogu preusmeravati po različitim spojnim putevima. Ovaj način prenosa se koristi kod Interneta. Treći način, paketni prenos podataka, ali svi paketi prolaze isti spojni put [1].

Prenos podataka kroz mrežu se obavlja po protokolima – utvrđenim pravilima koja su poznata svim učesnicima u komunikaciji. Protokol predstavlja standard (konvenciju) za ostvarivanje i kontrolu veze i prenosa podataka između dve krajnje tačke. Posao komuniciranja je toliko složen da je neophodno razviti protokole u više slojeva. Svaki sloj je namenjen za jedan odgovarajući posao. Kod prvobitnih računarskih mreža, umrežavanje se vršilo zavisno od proizvođača računarske opreme. Uvođenjem standarda putem kojih se komunicira po logički jasno definisanim slojevima, omogućeno je kombinovanje hardvera i softvera od različitih proizvođača, što je sve zajedno dovelo do povećanja kvaliteta usluge u mrežama i smanjenja cene opreme [2].

2 Modeli slojevite računarske komunikacije

2.1 *OSI* referentni model

OSI (*Open Systems Interconection*) model (Tabela 2.1-1) je apstraktni opis dizajna protokola računarskih mreža, predstavljen u obliku sedam slojeva [2]. Razvijen je 1984. godine od strane Međunarodne organizacije za standarde (*International Organization for Standardization*) [3]. Sve današnje mreže su bazirane na *OSI* modelu. *OSI* model definiše 7 slojeva (Tabela 2.1-1):

Slojevi	Jedinica	Protokoli
Aplikacija Mrežni procesi vezani za aplikaciju	Podatak	*HTTP, FTP, Telnet, DNS, DHCP, POP/SMTP*
Prezentacija Enkripcija i kodiranje podataka	Podatak	
Sesija Uspostavljanje sesije krajnjih korisnika	Podatak	*NetBIOS, PAP, CHAP, SSH*
Transport Veza, pouzdanost, transport	Segment Datagram	*TCP, UDP*
Mreža Logičko adresiranje i rutiranje	Paket	*IP, ICMP, ARP, RARP*
Sloj veze Fizičko adresiranje, pristup medijumu	Okvir	*PPP, HDLC, Frame Relay*
Fizički sloj Transmisija signala	Bit	*Token Ring* *IEEE 802.11*

Tabela 2.1-1: *OSI* referentni model.

- Fizički Sloj - Fizički sloj je zadužen za prenos bitova (nula i jedinica) putem komunikacionog kanala. Ovaj sloj definiše pravila po kojima se bitove prenose, koji električni napon je potreban, koliko bitova se šalje po sekundi i fizički format korišćenih kablova i konektora;

- Sloj veze - Sloj veze upravlja prenosom putem fizičkog sloja i omogućava prenos oslobođen grešaka na ovom i fizičkom sloju. Zadatak sloja veze jeste da zaštiti slojeve višeg nivoa og grešaka nastalih pri prenosu podataka. Takođe, s obzirom na to da je jedinica prenosa fizičkog sloja bit, sloj veze upravlja i formatom poruke (definiše početak i kraj poruke);

- Mrežni sloj - Zadatak mrežnog sloja jeste određivanje jedne ili više putanja kojima će poruka biti prosleđena od izvorišta do odredišta. Mrežni sloj je zadužen da u svakom čvoru mreže (stanici do odredišta) odredi koji je sledeći računar kome poruka treba biti prosleđena;

- Transportni sloj - Zadatak ovog sloja jeste obrada poruka na krajnjim tačkama – izvorištu i odredištu. Ovaj sloj uspostavlja, održava i prekida virtuelne veze za prenos podataka između izvorišta i odredišta. Transportni sloj je zadužen za nabavku mrežne adrese odredišta, podelu podataka u

segmente pogodne za slanje, prilagođavanje brzine prenosa mogućnostima strane sa slabijim preformansama, osiguravanje prenosa svih segmenata, eliminisanje dupliranih segmenata i sl. Takođe ovaj sloj može izvršiti i dodatnu kontrolu grešaka pri prenosu (dodatnu u smislu daje ona već izvršena na sloju veze);

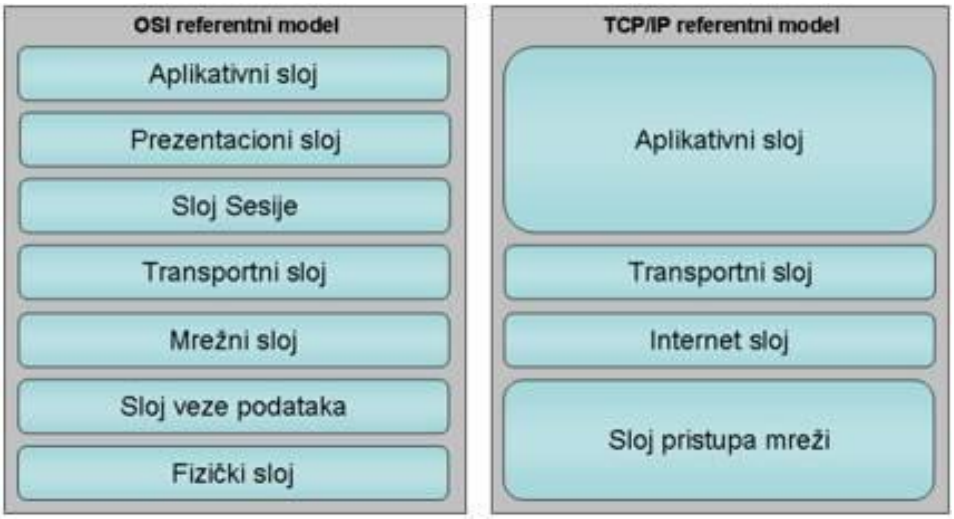
- Sloj sesije - Sloje sesije je zadužen za uspostavljanje, održavanje i prekid logičkih sesija između krajnjih tačaka. Svrha sesija jeste definisanje stanja (ili faza) svakog dijaloga radi definisanja validnih akcija u svakom od stanja. Na osnovu toga se vrši upravljanje transportnim slojem i provera podataka dobijenih od njega. Dodatna uloga sesija jeste obračunavanje sesija;

- Sloj prezentacije - Sloj prezentacije formatira podatke za prezentaciju korisniku. Zadatak ovog sloja jeste da uskladi format podataka između učesnika u komunikaciji i sloju aplikacije dostavi ove podatke u formatu koji on zahteva. Na primer, sloj prezentacije može originalne podatke dobijene od sloja aplikacije kompresovati radi efikasnijeg prenosa. Ovakve podatke sloj prezentacije na strani drugog učesnika ne može direktrno proslediti sloju aplikacije već je pre toga neophodno izvršiti dekompresiju;

- Sloj aplikacije - Sloj aplikacije predstavlja interfejs mreže ka korisniku. Osnovna uloga ovog sloja je da omogući pristup mreži korisničkim programima [2].

2.2 *TCP/IP* model

Ovaj model je razvijen za potrebe Interneta i jednostaviniji je od *OSI* modela. Jednostavnost ovog model ogleda se u apstraktrnom gledanju na najviša tri sloja *OSI* modela tako da *TCP/IP* model propisuje samo sloj aplikacije naspram slojeva aplikacije, prezentacije i sesije kod *OSI* modela (Sl. 2.2-1). Takođe, fizički sloj i sloj veze podataka su objedinjeni u sloj pristupa mreži (Sl. 2.2-1). Funkcije slojeva su identične kao kod *OSI* modela. Danas se u većini slučajeva koristi *TCP/IP* model kao referentni [1].



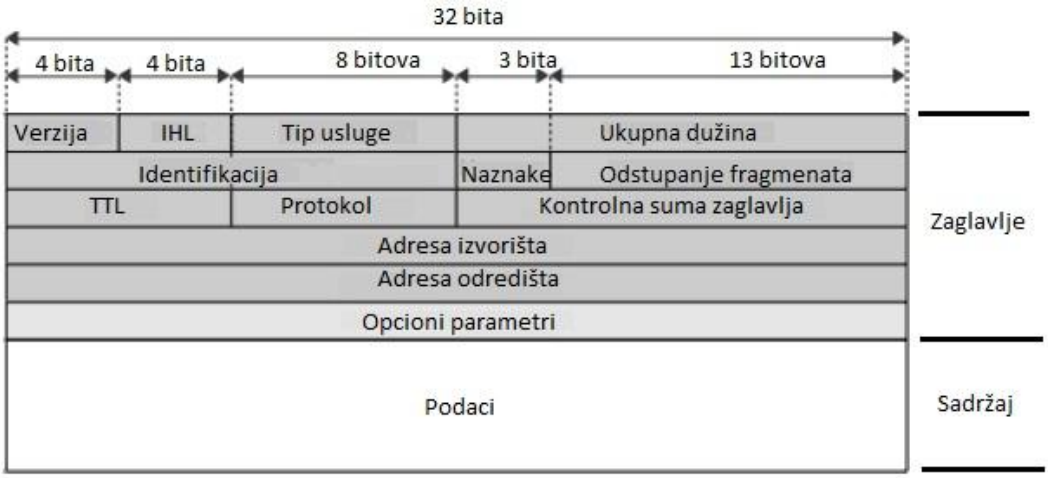
Sl. 2.2-1: *OSI* referentni model i *TCP/IP* referentni model.

3 *IP* protokol

Trenutno je najzastupljeniji *IP* protokol cetvrte verzije (*Internet Protocol version 4, IPv4*). Dve osnovne fukcije *IP* protokola jesu adresovanje računarskih mreža i njihovih članova i fragmetovanje podataka [2].

U postupku adresovanja *IP* protokolu se prosleđuje odredišna adresa (*IP* adresa), odnosno kome treba dostaviti podatke [2]. Dva uređaja na internetu nikada ne mogu imati istu *IP* adresu. *IP* adresa je 32-bitni broj. Umesto toga, u praksi se koristi ekvivalenti dekadni zapis sa 4 broja razdvojena tačkom. Svaki broj je u intervalu od 0 do 255 što odgovara jednom *byte*-u (8 bita). Primer *IP* adrese: 01111111 00000000 0000000 00000001 – binarni zapis, 127.0.0.1 – dekadni zapis. *IP* adresa se sastoji iz dva bitna dela: adresa mreže (mrežnog segmenta) i adresa hosta (računara sa dodeljenom *IP* adresom) [4].

Podaci se smeštaju u pakete koji se nazivaju datagramima. Paketi *IP* protokola – datagrami – sastoje se od dva osnovna dela zaglavlja paketa i podataka koji se njime prenose (Sl. 3-1). Zaglavlje paketa omogućava funkcionisanje *IP* protokola [2].



Sl. 3-1: Struktura datagrama *IP* protokola.

Osnovna polja zaglavlja čine (Sl. 3-1):

- Verzija – Odnosi se na verziju *IP* protokola;
- Dužina *IP* zaglavlja (*IHL*) – Bitska dužina zaglavlja u jedinicama od po 32 bita. Koristi se za određivanje bitske pozicije na kojoj se zaglavlje završava;
- Tip usluge – Sadrži parametre na osnovu kojih se određuje željeni kvalitet usluge, odnosno, proritet pod kojim će se paket obrađivati;
- Ukupna dužina – Sadrži podatke o ukupnoj dužini paketa, izraženoj u bajtovima;
- Identifikacija, naznake i odstupanje paketa – Ova tri parametra ukupne dužine 32 bita, koriste se za potrebe fragmentovanja podataka;
- Preostalo vreme postojanja (*TTL*) – Ova opcija ograničava vreme koje paket može da provede u putovanju kroz mrežu;

- Protokol – Sadržaja ovog polja određuje koji je protokol transportnog sloja inicirao slanje podataka, odnosno kom protokolu transportnog sloja će podaci na odredištu biti isporučeni;
- Kontrolna suma zaglavlja – Ovo polje omogućava da se na prijemnoj strani odredi da li je tokom prenosa došlo do oštećenja zaglavlja, odnosno do izmene bitova koji njemu pripadaju;
- Adresa izvora – U ovo polje se upisuje adresa mrežnog interfejsa izvorišta sa koga je paket poslat;
- Adresa odredišta – U ovo polje se upisuje mrežna adresa odredišta kome je paket poslat [2].

Fragmentovanje podataka omogućava da se paketi prenose posredstvom mreža koje imaju različita ograničenja u pogledu najveće moguće dužine jedinice podataka. U ovom postupku datagrami se dele na framente koji imaju istu strukturu kao i originalni datagrami ali su manje veličine. Na prijemnoj strani *IP* protokol od dobijenih fragmenata ponovo formira originalni datagram i nastavlja sa njegovim prosleđivanjem konačnom odredištu [2].

4 *UDP* protokol

4.1 Uvod

Trasportni sloj u referentnim *OSI* i *TCP/IP* modelima računarskih mreža zadužen je za prenos podataka između dve komunikacione strane [1]. Opšta uloga transportnog sloja je da omogući komunikaciju sloja iznad i sloja ispod. Konkretna uloga transportnog sloja jeste da prihvati podatke od aplikacije na strani pošiljaoca i dostavi ih aplikaciji odredišta starajući se o prenosu, kontroli i ispravljanju grešaka pri prenosu i o garantovanju isporuke. Nakon prihvatanja podataka transportni sloj ima zadatak da ih prevede u oblika pogodan za transport [2]. Funkcije transportnog sloja su:

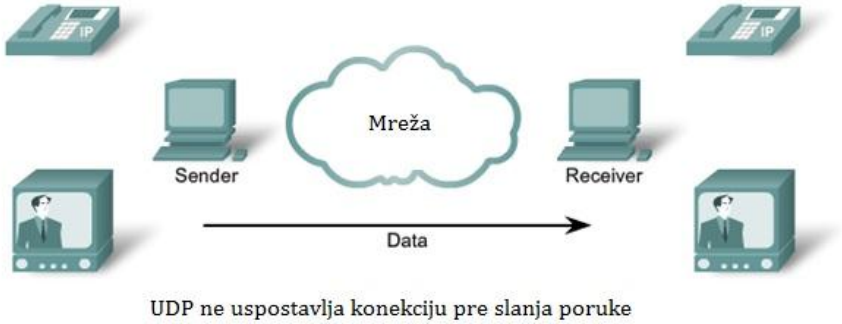
- Ostvarivanje virtuelne veze za prenos podataka;
- Prevođenje podataka u format pogodan za prenos;
- Segmentacija podataka radi efikasnijeg korišćenja komunikacionog kanala;
- Isporuka podataka na strani primaoca u identičnom obliku u kom su poslati;
- Omugućavanje optimalne brzine prenosa podataka u skladu sa propusnom moći i učestalošću grešaka na komunikacionom kanalu i prihvatnoj moći primaoca [1].

Dva najčešće korišćena protokola transportnog sloja *OSI* i *TCP/IP* modela su:

- *UDP* (*User Datagram Protocol*) - Nepouzdani protokol, bez uspostavljanja direktne veze. Omogućava aplikacijama da šalju kapsulirane *IP* datagrame za koje ne moraju prethodno da uspostavljaju vezu;
- *TCP* (*Transmission Control protocol*) – Protokol sa uspostavljanjem direktne veze. Obezbeđuje pouzdan tok bajtova s kraja na kraj veze kroz nepouzdanu raznovrsnu mrežu [5].

4.2 Uloga i krakateristike *UDP* protokola

UDP protokol je jedan od ključnih protokola transportnog nivoa koji pruža usluge aplikacijama koje ne zahtevaju prethodno uspostavljanje veze, niti pouzdan prenos (Sl. 4.2-1). *UDP* protokol je razvio *David Reed* 1980. godine, a kasnije je ovaj protokol ušao u standard i opisan je u dokumentu *RFC 768.* Za aplikacije koje komuniciraju preko mreže, ukoliko koriste usluge *UDP*-a, kaže se da razmenjuju datagrame [6].



Sl. 4.2-1: *UDP* protokol.

Ključni protokoli sloja aplikacije koji koriste usluge *UDP* protokola su:

- *Domain Name System (DNS);*
- *Simple Network Management Protocol (SNMP);*
- *Dynamic Host Configuration Protocol (DHCP);*
- *Routing Information Protocol (RIP);*
- *Trivial File Transfer Protocol (TFTP);*
- Online igre [7].

Neke aplkacije, kao što su online igre ili *VoIP*, mogu tolerisati manje gubitke nekih podataka. Ako bi ove aplikacije koristile *TCP*, moglo bi doći do velikih kašnjenja jer *TCP* mora vršiti detekciju izgubljenih podataka i vršiti retransmisiju. Ova kašnjenja bi bila mnogo više neprihvatljiva za aplikaciju nego što su to mali gubici informacija. Neke aplikacije, kao što je *DNS*, će jednostavno ponovo poslati zahtev za koji nisu dobile odgovor, i zato ne moraju imati *TCP* garanciju o pristizanju poruke [7].

UDP poseduje sledeće karakteristike:

- S kraja na kraj (*end-to-end*). *UDP* je transportni protokol koji može da razlikuje višestruke aplikativne programe koji rade na datom računaru. Servis *end-to-end* omogućava aplikativnom programu da šalje i prima individualne poruke, od kojih svaka putuje u odvojenom datagramu. Aplikacija može izabrati da ograniči komunikaciju na jedan drugi aplikativni program ili da komunicira sa više aplikacija;

- Bez konekcije (*connectionless*). Interfejs koga *UDP* obezbeđuje aplikacijama sledi beskonekcionu paradigmu. Aplikacija koja koristi *UDP* ne mora da unapred uspostavlja komunikaciju pre slanja podataka, niti mora da informiše mrežu kada završi. Aplikacija može da generiše i šalje podatke u bilo koje vreme;

- Orijentisan na poruke (*message-oriented*). Aplikacija koja koristi *UDP* šalje i prima individualne poruke. Kada neka aplikacija zahteva da *UDP* šalje blok podataka, *UDP* postavlja podatke u jednu poruku za prenos. *UDP* ne deli poruku na nekoliko paketa, i ne kombinuje poruke za isporučivanje - svaka poruka koju aplikacija pošalje se prenosi preko Interneta i isporučuje primaocu. Svaka poruka koju *UDP* isporuči aplikaciji primaocu će biti potpuno ista kao što ju je poslao pošiljalac. Veličina *IP* datagrama formira apsolutno ograničenje sa strane *UDP* poruke. *UDP* postavlja kompletnu poruku u korisnički datagram i zatvara korisnički datagram u Internet datagram, a *IP* mora obaviti fragmentaciju pre nego što se datagram može poslati;

- Najbolja praksa (*best-effort*). *UDP* nudi aplikacijama istu semantiku isporuke najbolje prakse kao *IP*. *UDP* koristi *IP* za sve isporuke, i obezbeđuje aplikacijama potpuno istu semantiku isporuke kao *IP*. To znači da poruke mogu biti: izgubljene, duplirane, isporučene sa zakašnjenjem, isporučene van

reda i prekinute. *UDP* ne stvara namerno probleme isporuke. On jednostavno koristi *IP* za slanje poruka, i ne detektuje niti rešava probleme isporuke;

- Arbitrarna interakcija (*arbitrary interaction*). *UDP* omogućava aplikaciji da šalje mnogim aplikacijama, prima od mnogih aplikacija, ili komunicira sa tačno jednom drugom aplikacijom;

- Nezavisno od operativnog sistema (*operating system independent*). *UDP* obezbeđuje način identifikovanja aplikativnih programa koji ne zavise od identifikatora koje koristi lokalni operativni sistem [8].

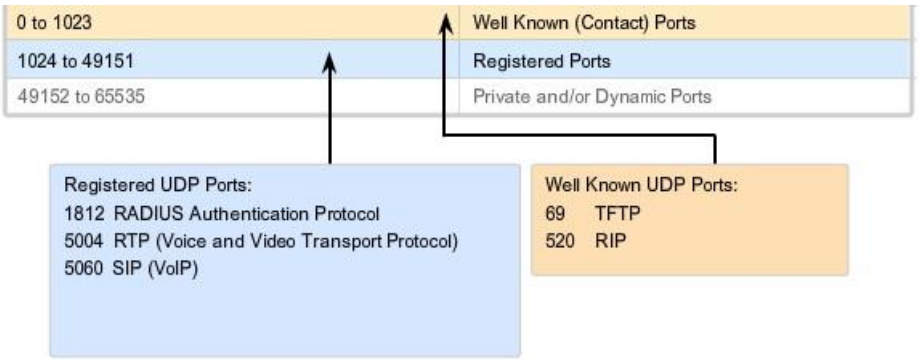
4.3 *UDP* portovi

Da bi se razlikovali datagrami za svaku aplikaciju, *UDP* zaglavlja sadrže polja koja mogu jedinstveno identifikovati ove aplikacije. Ovaj jedinstveni identifikator se naziva broj porta [7]. Port je zapravo 16-to bitni ceo broj koji se pridružuje svakoj aplikaciji koja komunicira preko mreže [6]. U zaglavlju svakog datagrama nalazi se broj porta izvorišta i odredišta. Serverski procesi imaju statičke brojeve porta, kijentski port brojevi se dinamički određuju za svaku konverzaciju. Telo koje propisuje standare vezane za adresiranje portova je *IANA.* Postoje različite vrste brojeva porta: *well known*, registrovani portovi i dinamički portovi (Sl. 4.3-1) [7].

Well Known (dobro poznati) port-ovi od 0 do 1023 – ovi brojevi su rezervisani za servise i aplikacije (Sl. 4.3-1). Najčešće se koriste za aplikacije kao što su: *HTTP*, *POP3/SMTP* i *Telnet* [7].

Registrovani portovi (od 1024 do 49151) – ovi brojevi portova su dodeljeni korisničkim procesima ili aplikacijama (Sl. 4.3-1). Ovi procesi su primarno individualne aplikacije koje je korisnik izabrao da instalira radije nego uobičajene aplikacije koje imaju *well known* portove [7].

Dinamički ili privatni portovi od 49152 do 65535 – takođe poznati i kao „*ephemeral ports*“ (Sl. 4.3-1). Ovi brojevi porta se dinamički dodeljuju korisničkim procesima ili aplikacijama koje zahtevaju konekciju [7].



Sl. 4.3-1: *Well known* i registorvani UDP portovi.

Kombinacija broja porta i *IP* adrese identifikuje tačnu aplikaciju koja je pokrenuta na odredištu. Ova kombinacija naziva se *Socket*.

4.4 Zaglavlje *UDP* datagrama

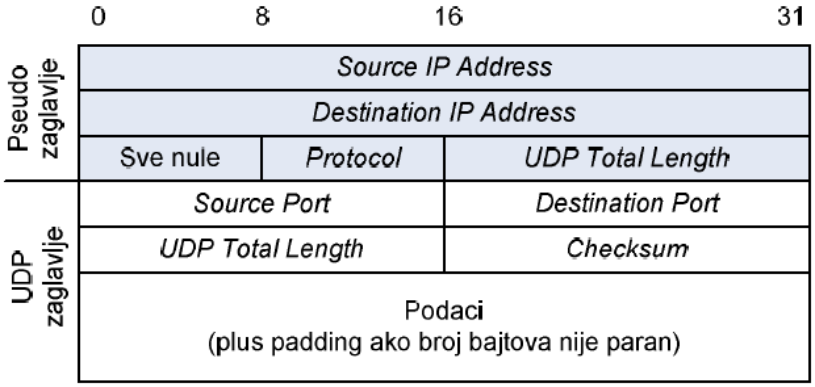
Zaglavlje *UDP* datagrama se sastoji od sledećih polja (Sl. 4.4-1):

- *Source port* - Izvorišni broj porta usluge je opciono polje. Kada se koristi, označava broj porta procesa koji šalje podatke. Na nju će doći odgovor kada ne postoji druga informacija. Ako se polje ne koristi popuni se nulama;
- *Destination port* – Odredišni broj porta usluge;
- *Length* - Dužina *UDP* datagrama u bajtovima uključujući zaglavlje i podatke. Minimalna dužina *UDP* datagrama je 8 bajtova;
- *Checksum* - Kontrolna suma, računa se na osnovu pseudo zaglavlja iz *IP* i *UDP* zaglavlja i podataka;
- *Data* – Podaci [7].

16	*32*
Source port	*Destination port*
Length	*Checksum*
Data	
Struktura UDP zaglavlja u 32b redovima	

Sl.4.4-1: Struktura UDP zaglavlja.

Kod *UDP*-a, kontrolna suma uključuje: pseudo zaglavlje, *UDP* zaglavlje i podatke. Pseudo zaglavlje je deo zaglavlja *IP* datagrama u kojem je *UDP* datagram enkapsuliran (Sl. 4.4-2). Uključivanje *IP* zaglavlja u kontrolnu sumu *UDP* datagrama obezbeđuje veću zaštitu od isporuke *UDP* datagrama pogrešnom hostu ili pogrešnom protokolu [9].

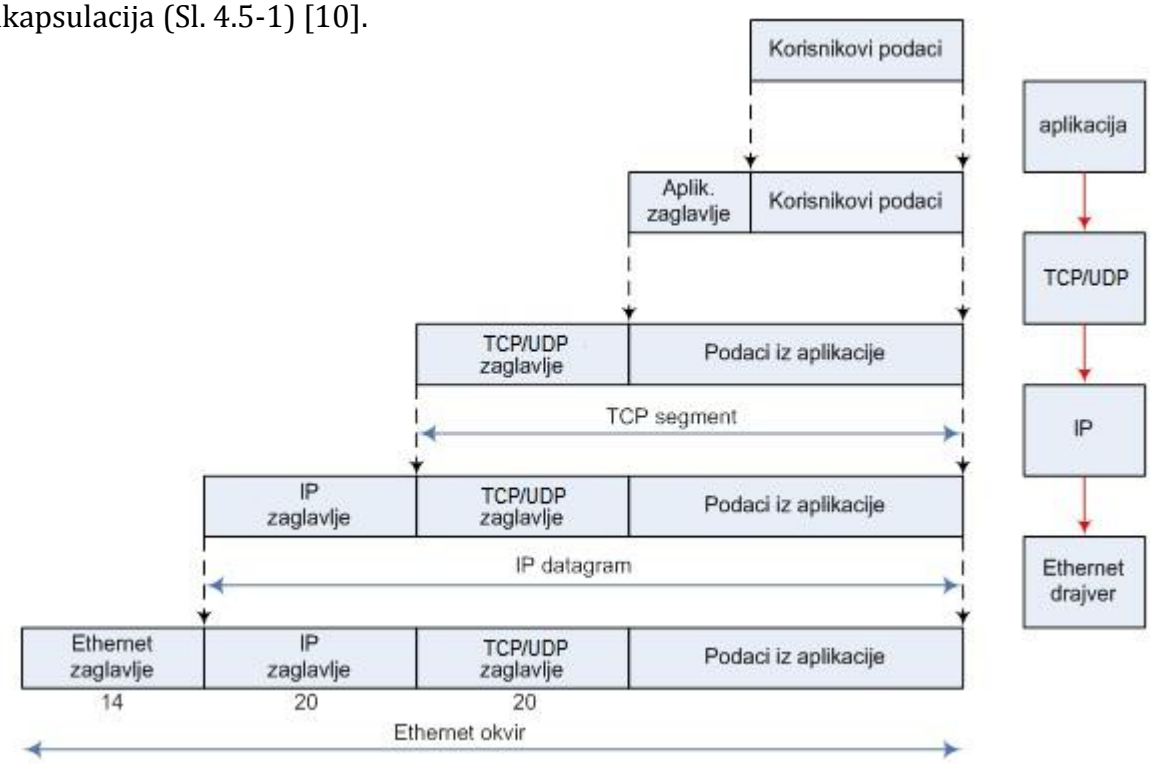


Sl.4.4-2: Pseudo zaglavlje kod *UDP*-a.

Naime, ako je greška nastala u *IP* zaglavlju i to baš u polju za odredišnu *IP* adresu, i pri tome nije otkrivena kontrolnom sumom *IP* datagrama, *UDP* datagram, koji je sam za sebe ispravan, biće isporučen pogrešnom hostu. Polje za protokol iz *IP* zaglavlja je uključeno da bi se obezbedilo da primljeni paket pripada *UDP*-u. U polju za protokol zaglavlja *IP* datagrama koji enkapsulira *UDP* datagram, upisana je vrednost 17. Ako se tokom prenosa ova vrednost promeni, datagram može biti isporučen nekom drugom protokolu (npr. *TCP*) [9].

4.5 Enkapsulacija i deenkapsulacija

Kada aplikacija šalje podatke preko *TCP/IP* modela, podaci prolaze kroz sve slojeve, dok ne postanu niz bita koji kao signali putuju preko mreže. Svaki sloj dodaje informacije, nadovezujući zaglavlja na informacije koje dobija. Ovaj proces naziva se enkapsulacija (Sl. 4.5-1) [10].



Sl. 4.5-1: Enkapsulacija.

Jedinica podataka koju *UDP* prosleđuje *IP*-u naziva se *UDP* datagram. Jedinica podataka koju *IP* prosleđuje mrežnom interfejsu naziva se *IP* datagram (paket). Niz bita koji putuju preko *ethernet*a naziva se okvir (eng. *frame*). Brojevi ispod zaglavlja su tipične dužine zaglavlja u bajtovima [10].

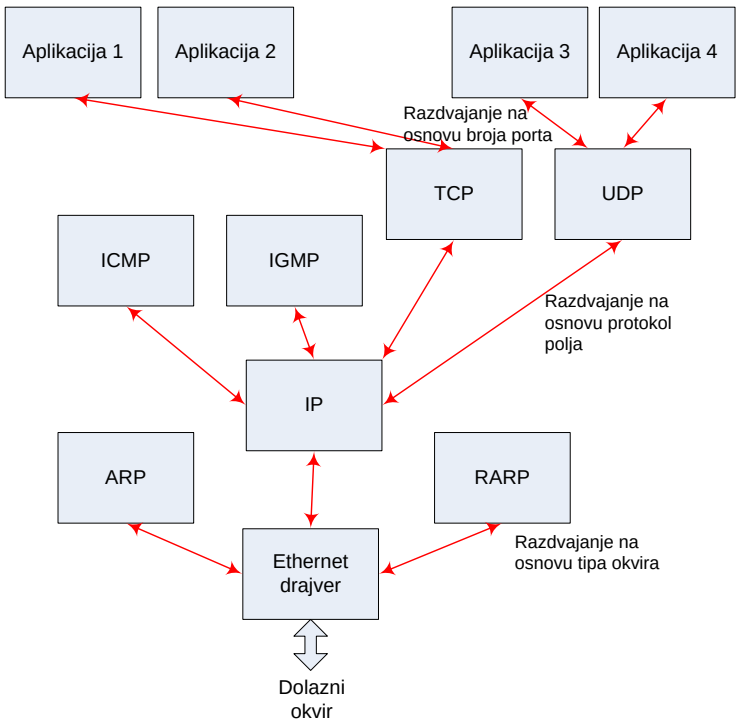
Protokoli transportnog sloja (*TCP* ili *UDP*) prosleđuju podatke *IP*-u. Zbog toga *IP* mora u svom zaglavlju upisati nekakav identifikator, na osnovu kog će znati kom sloju i kom protokolu unutar njega pripadaju podaci. Zato *IP* u svoje zaglavlje smešta 8-bitnu vrednost, koja se zove protokol polje [10].

Isto tako, mnogo različitih aplikacija može koristiti *TCP* i *UDP* u isto vreme. Protokoli transportnog sloja čuvaju u zaglavlju identifikator kako bi identifikovali aplikaciju. I *TCP* i *UDP* koriste 16-bitni broj porta za identifikovanje aplikacije čije podatke šalju, a u zaglavlju pamte i broj porta na izvoru, i broj porta na odredištu [10].

Konačno, na najnižem sloju situacija je slična – mrežni interfejs u *ethernet* zaglavlju beleži 16-bitni identifikator protokola sa sloja iznad – tzv. tip okvira.

Kada neki *ethernet* okvir stigne na odredišni računar, počinje proces inverzan prethodno navedenoj enkapsulaciji – deenkapsulacija. Paket prolazi svoj put, i na

svakom sloju se oslobađa zaglavlja tog sloja, sve dok ne dođe na nivo aplikacije kao „čist“ podatak (Sl.4.5-2) [10].

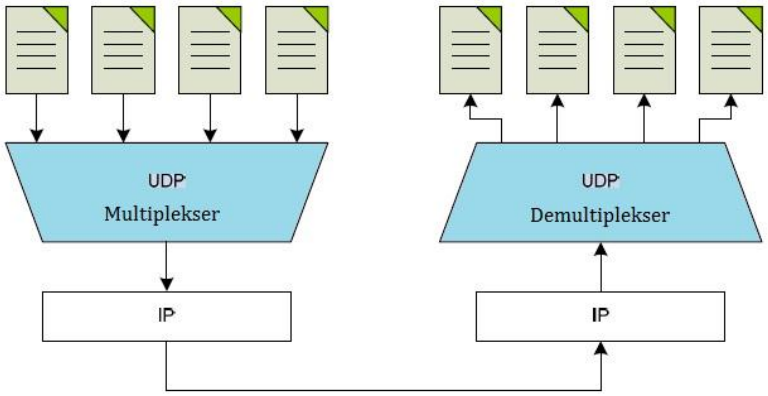


Sl. 4.5-2: Deenkapsulacija.

4.6 Multipleksirane i demultipleksiranje

Na računaru na kome se izvršava *TCP/IP* postoji samo jedan *UDP*. Međutim, može postojati nekoliko aplikacionih procesa koji u isto vreme žele da koriste usluge *UDP*-a. Da bi se razrešile ovakve situacije, na nivou *UDP*-a koristi se koncept multipleksiranja i demultipleksiranja (Sl. 4.6-1) [9].

Multipleksrianje se koristi na strani pošiljaoca kada više od jednog aplikacionog procesa želi da pošalje datagrame (Sl. 4.6-1). *UDP* prihvata poruke od različitih procesa (koji se identifikuju svojim brojevima porta), svakoj poruci pridodaje zaglavlje i kreirani korisnički datagram predaje *IP*-u [9].

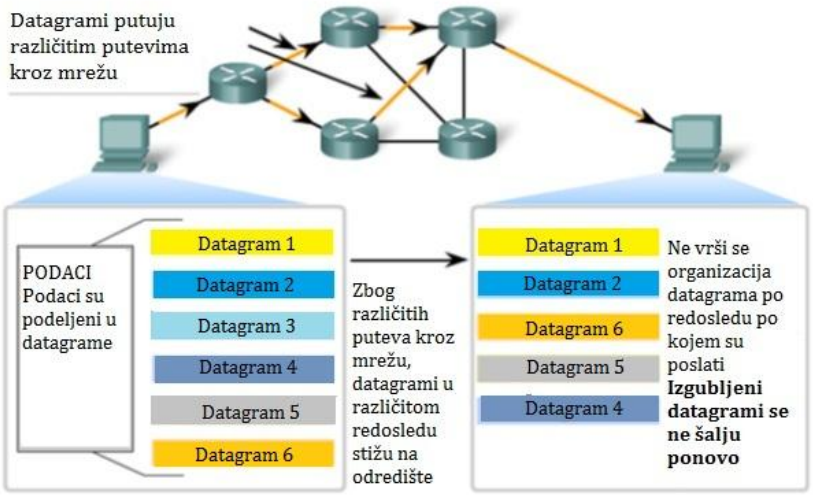


Sl. 4.6-1: Multipleksiranje i demultipleksiranje.

Demultipleksiranje se koristi na strani prijemnika kada postoji više od jednog aplikacionog procesa koji očekuju datagrame (Sl. 4.6-1). *UDP* prima datagram od *IP*-a. Nakon provere grešaka i odstranjivanja zaglavlja, *UDP* isporučuje poruku odgovarajućem aplikacionom procesu shodno broju porta [9].

4.7 Reasembliranje *UDP* datagrama

Mnoge aplikacije koje koriste *UDP* šalju male količine podataka koje mogu da stanu u jedan datagram. Međutim, neke aplikacije će poslati veće količine podataka koji moraju biti podeljeni u više datagrama. Kada se pošalje više datagrama na odredište, moguće je da do njega dođu različitim putevima kroz mrežu i tako stignu u pogrešnom redosledu (Sl. 4.7-1). *UDP* nema način da datagrame sortira po prvobitnom redosledu. Zbog te činjenice, *UDP* jednostavno reasemblira podatke redosledom kojim su stigli i takve prosleđuje aplikaciji [7].



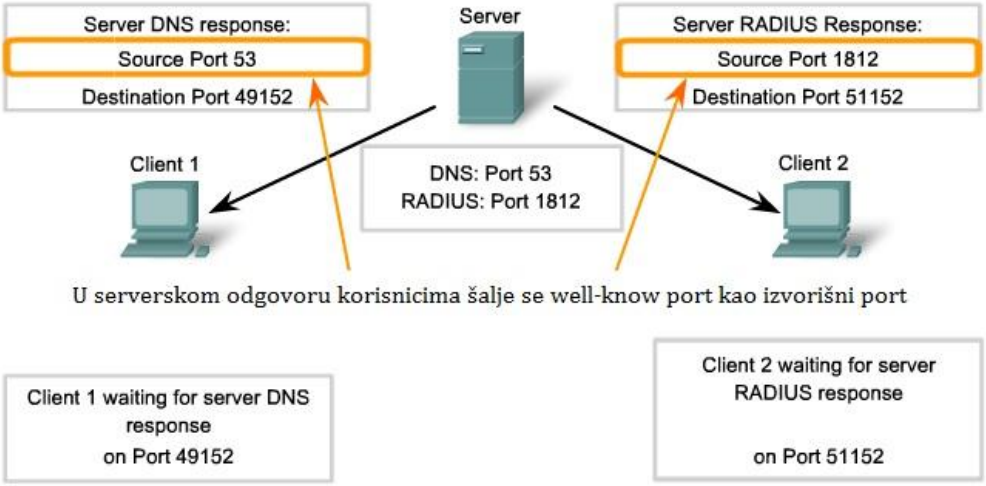
Sl. 4.7-1: UDP kao nepouzdan protokol.

4.8 *UDP* klijent-server procesi i zahtevi

UDP bazirane aplikacije imaju dodeljene *well known* ili registrovane brojeve porta. Kada ove aplikacije ili procesi rade, oni će prihvatati podatke koji su adresirani na broj porta koji im je pridružen. Kada *UDP* primi datagram za neki od ovih portova, prosleđuje podatke aplikacije odgovarajućoj aplikaciji bazirano na broju porta [7].

Klijent-server komunikacija je inicirana od strane klijentske aplikacije koja traži podatke od serverskog procesa (Sl. 4.8-1). *UDP* klijentski proces odabira slučajni broj porta unutar dinamičkog opsega brojeva portova i koristi taj broj kao izvorišni broj porta za komunikaciju. Odredišni port obično će biti *well known* port ili registrovani broj porta dodjeljen serverskom procesu [7].

Kada podaci budu spremni za slanje i brojevi portova budu definisani, UDP može formirati datagram i proslediti ga mrežnom sloju na adresiranje i slanje na mrežu.



Sl. 4.8-1: Slanje zahteva na odgovarajući port i odgovor servera.

Treba imati na umu da jednom kada su brojevi izvorišnog i odredišnog porta izabrani, isti par portova će se koristi u zaglavljima svih datagrama korištenih u komunikaciji. Za podatke koji se vraćaju od servera ka klijentu, brojevi porta izvorišta i odredišta su obrnuti [7].

4.9 Prednosti *UDP* protokola

Nekada je bolje za prenos poruka koristiti *UDP* protokol umesto *TCP* protokola. Neki od tih slučajeva su:

- Prenos podataka aplikacija koje same osiguravaju pouzdani prenos, ili kada aplikacija dopušta manje gubitke;
- Slanje upita jednog računara drugom računaru, uz mogućnost ponavljanja upita ako odgovor ne stigne nakon isteka određenog vremenskog intervala;
- Kada je potrebno poslati manji blok podataka, veličine jednog paketa pa je jednostavnije i brže prenositi samo podatke, bez dodatnih kontrola, a u slučaju pogrešnog prijema podatke poslati ponovo [7].

5 Klijent - server komunikacija u programskom jeziku *Java*

Da bi dva programa mogla da komuniciraju preko mreže, oba programa moraju da obrazuju po jedan *socket*. *Socket* je kombinacija *IP* adrese i porta. Program koji konstruiše *socket* koji čeka na zahtev za uspostavljanje komunikacije naziva se server, ovaj pasivni *socket* koji se u njemu koristi se naziva serverski *socket*. Drugi program koji se povezuje sa serverom naziva se klijent, a njegov aktivni *socket* kojim se šalje zahtev za uspotavljanje komunikacije sa serverom naziva se klijentski *socket*. Osnovna ideja modela klijent-server je da se server nalazi negde u mreži i da čeka na zahtev za uspostavljanje komunikacije od nekog klijenta [11].

Programski jezik *Java* podržava komunikaciju razmenom *UDP* datagrama pomoću sledećih klasa :

- *DatagramPacket;*
- *DatagramSocket* [12].

Klasa *DatagramPacket* sadrži nekoliko konstruktora koji se mogu koristiti za keriranje *UDP* datagrama. Na primer: *DatagramPacket(byte[] buf, int length, InetAddress address, int port);.*

Ključne metode *DatagramPacket* klase su :

- *byte[] getData()* - Vraća bafer podataka;
- *int getLength()* - Vraća dužinu podataka koji se šalju ili dužinu primljenih podataka;
- *void setData(byte[] buf)* – Postavlja bafer podataka za paket;
- *void setLength(int length)* – Postavlja dužinu za paket [12].

Klasa *DatagramSocket* podržava različite metode koje se mogu koristiti za prenos ili prijem datagrama preko mreže. Takođe sadrži nekoliko konstruktora za kreiranje *socket*-a. Na primer: *DatagramSocket(int port);* [12].

Dve ključne metode su:

- *void send(DatagramPacket p)* - Šalje datagram iz ovog *socket*-a;
- *void receive(DatagramPacket p)* - Prima datagram iz ovog *socket*-a [12].

U nastavku je dat primer klijent-server komunikacije preko *UDP* protokola. Implementiran je u programskom jeziku *Java.*

Server program

```
package Server;

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

/**
 * Klasa modeluje UDP server. Server nakon pokretanja i prijema podataka sa
klijenta
 * vraca klijentu kopiju primljenih informacija (echo server).
 *
 * @author Mehmed Batilovic
 */
public class Server {
    // Soket na koji se uspostavlja komunikacija
    private DatagramSocket socket;
    // Podrazumevani port za osluskivanje
    private static final int DEFAULT_PORT = 9000;
    // Podrazumevana velicina bafera podataka
    private static final int DEFAULT_BUFFER_SIZE = 1024;
    // Bafer podataka
    private byte[] buffer;

    /**
     * Konstruktor bez parametara. Inicijalizuje server postavljanjem
     * parametara na podrazumevane vrednosti.
     * @throws SocketException ukoliko nije moguce kreirati soket sa
     * podrazumevanim portom kao parametrom.
     */
    public Server() throws SocketException
    {
        socket = new DatagramSocket(DEFAULT_PORT);
        buffer = new byte[DEFAULT_BUFFER_SIZE];
    }

    /**
     * Metoda pokrece server. Po pokretanju, server neprekidno ceka na
     * zahteve i nakon prijema prosledjuje odgovor klijentu.
     * @throws IOException ukoliko dodje do greske u komunikaciji.
     */
    public void run() throws IOException
    {
        System.out.println("Server running on port " + DEFAULT_PORT);
        // Izvrsavanje servera u beskonacnoj petlji bez prekida
        while(true)
        {
            // Prijemni datagram
            DatagramPacket in = new DatagramPacket(buffer, buffer.length);
            // Prijem datagrama
            socket.receive(in);

            // Kreiranje datagrama za slanje
            DatagramPacket out = new DatagramPacket(
                in.getData(),
                in.getLength(),
                in.getAddress(),
```

```
                in.getPort());
            // Slanje datagrama
            socket.send(out);
        }
    }

    /**
     * Metoda zaustavlja server i zatvara soket ukoliko je prethodno bio
     * inicijalizovan.
     */
    public void stop()
    {
        if(socket != null)
        {
            socket.close();
        }
    }

}
```

```
import java.io.IOException;
import java.net.SocketException;

/**
 * Klasa sa <code>main</code> metodom koja sluzi za pokretanje servera.
 *
 * @author Mehmed Batilovic
 */
public class ServerRunner {

    public static void main(String[] args)
    {
        try
        {
            Server server = new Server();
            server.run();
        }
        catch(SocketException exception)
        {
            System.out.println("Socket error on server: " +
exception.getMessage());
        }
        catch(IOException exception)
        {
            System.out.println("IO error on server: " + exception.getMessage());
        }
    }

}
```

Klijent program

```
package Klijent;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

/**
 * Klasa modeluje jednostavan UDP klijent. Klijent nakon pokretanja i preuzimanja
korisnickog
 * ulaza sa konzole prosledjuje preuzete podatke serveru a potom prikazuje odgovor
primljen sa
 * servera.
 *
 * @author Mehmed Batilovic
 */
public class Klijent {
    // Soket preko kojeg se uspostavlja komunikacija
    private DatagramSocket socket;
    // Podrazumevani naziv host-a na kojem se izvrsava server
    private static final String DEFAULT_HOST_NAME = "localhost";
    // Host adresa servera
    private InetAddress host;
    // Podrazumevani port na kojem server osluskuje zahteve
    private static final int DEFAULT_SERVER_PORT = 9000;
    // Podrazumevana velicina bafera podataka
    private static final int DEFAULT_BUFFER_SIZE = 1024;
    // Bafer podataka
    private byte[] buffer;
    // Ulazni tok za preuzimanje korisnickog ulaza sa konzole
    private BufferedReader userInput;

    /**
     * Konstruktor bez parametara. Inicijalizuje klijenta postavljanjem
     * parametara na podrazumevane vrednosti.
     * @throws IOException ukoliko je doslo do greske prilikom inicijalizacije
     * ulaznog toka.
     */
    public Klijent() throws IOException
    {
        socket = new DatagramSocket();
        host = InetAddress.getByName(DEFAULT_HOST_NAME);
        buffer = new byte[DEFAULT_BUFFER_SIZE];
        userInput = new BufferedReader(new InputStreamReader(System.in));
    }
```

```
    /**
     * Metoda pokrece klijenta. Po pokretanju klijent neprekidno ceka na
     * korisnicki ulaz i nakon preuzimanja linije sa ulaza prosledjuje podatke
       serveru.
     * Nakon toga na konzoli se prikazuje eho prosledjenih podataka dobijen od
       servera.
     * Ukoliko korisnik kao ulaz unese 'exit' rad klijenta se zaustavlja.
     * @throws IOException ukoliko dodje do greske u ulaznom toku ili komunikaciji
       sa serverom.
     */
    public void run() throws IOException
    {
        String input = null;

        while(true)
        {
            System.out.print("Enter your message: ");
            input = userInput.readLine();
            // Ako korisnik ukuca 'exit', klijent se zaustavlja
            if(input.toLowerCase().equals("exit"))
            {
                stop();
                break;
            }

            // Kreiranje datagrama koji se salje
            DatagramPacket out = new DatagramPacket(
                input.getBytes(),
                input.length(),
                host,
                DEFAULT_SERVER_PORT
            );
            // Slanje datagrama
            socket.send(out);

            // Prijemni datagram
            DatagramPacket in = new DatagramPacket(buffer, buffer.length);
            // Prijem datagrama
            socket.receive(in);
            // Prikaz podataka koji su stigli kao odgovor od servera
            System.out.println("Server echo:"+new String(in.getData()) + "\n");
        }
    }

    /**
     * Metoda zaustavlja klijenta i zatvara soket ukoliko je prethodno bio
     * inicijalizovan.
     */
    public void stop()
    {
        if(socket != null)
        {
            socket.close();
        }
    }

}
```

```java
package Klijent;
import java.io.IOException;

/**
 * Klasa sa <code>main</code> metodom koja sluzi za pokretanje klijenta.
 *
 * @author Mehmed Batilovic
 */
public class KlijentRunner {

    public static void main(String[] args)
    {
        try
        {
            Klijent client = new Klijent();
            client.run();
        }
        catch(IOException exception)
        {
            System.out.println("IO error on server: " + exception.getMessage());
        }
    }

}
```

6 Literatura

[1] Mladen V, Aleksandar J. Uvod u računarske mreže. Beograd: Univerzitet Singidunum; 2008.

[2] Mladen V, Aleksandar J. Računarske mreže. Beograd: Univerzitet Singidunum; 2011.

[3] Wikipedia [home page on the internet]. OSI model. [updated 2013 Jun 16; cited 2014 May 14]. Avaliable from: http://sh.wikipedia.org/wiki/OSI_model

[4] Physics [home page on the internet]. Mrežni sloj. [cited 2014 May 14]. Avaliable from: http://physics.kg.ac.rs/fizika/informatika/RacunarskeMreze/mrezniSloj.PPT

[5] Sinergija [home page on the internet]. Protokoli transportnog sloja. [cited 2014 May 15]. Avaliable from: http://predmet.sinergija.edu.ba/pluginfile.php/4564/mod_folder/content/1/RM11_ProtokoliTransportnogSloja.pdf?forcedownload=1

[6] Gimpi [home page on the internet]. Mrežni protokoli II. [cited 2014 May 15]. Avaliable from: http://www.gimpi.ni.ac.rs/SharedFiles/Download.aspx?pageid=34&fileid=3196&mid=65

[7] Fsk [home page on the internet]. Transportni sloj *TCP/IP* model. [cited 2014 May 16]. Avaliable from: http://www.nastava.fsk.unsa.ba/index.php/sadrzaj/item/download/1071_7506d7e1fef55dc32e8707942e032b91.html

[8] Ffuis [home page on the internet].*UDP* protokol. [cited 2014 May 15].Avaliable from: http://www.ffuis.edu.ba/media/faculty/documents/2011/06/08/RAUNARSKE_MREE-UDP_UserDatogramProtokol.ppt

[9] Elfak [home page on the internet]. *TCP/IP* model. [cited 2014 May 16]. Avaliable from: http://es.elfak.ni.ac.rs/rmif/Materijal/TCP-IP.pdf

[10] Singidunum [home page on the internet]. Lekcija 4 – Internet. [cited 2014 May 16]. Avaliable from: http://www.crnarupa.singidunum.ac.rs/ARHIVA/Master/Master studije/Savremene informaciono komunikacione tehnologije/Zastita u racunarima i racunarskim mrezama/Lekcija 4 - Internet.doc

[11] Dejan Ž. Java programiranje. Beograd: Univerzitet Singidunum; 2011.

[12] Buyya [home page on the internet]. Socket programing. [cited 2014 May 18]. Avaliable from: http://www.buyya.com/java/Chapter13.pdf