Program za testiranje mrežnih aplikacija

Šafarić, Miroslav

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Miroslav Šafarić

PROGRAM ZA TESTIRANJE MREŽNIH APLIKACIJA

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Miroslav Šafarić

Matični broj: 41195/12–IZV

Studij: Primjena informacijske tehnologije u poslovanju

PROGRAM ZA TESTIRANJE MREŽNIH APLIKACIJA

ZAVRŠNI RAD

Mentor:
Doc.dr.sc.: Nikola Ivković

Varaždin, srpanj 2018.

Sadržaj

1. Uvod

Završni rad „Program za testiranje mrežnih aplikacija“ povezuje dva velika područja unutar domene informacijskih i računalnih znanosti: programiranje i mreže računala. Primarni produkt završnog rada je program koji testira ispravnost drugih mrežnih aplikacija, a to realizira korištenjem *eng. socekta*, koji su detaljnije opisani u poglavlju „Transportni sloj“.

Funkcionalnost programa svodi se na to da program iz datoteke čita određene parametre, te na temelju tih parametara, ovisno o transportnom protokolu, uspostavlja vezu ili samo šalje podatke prema odredišnom računalu, korištenjem socketa. Zatim zaprima odgovor aplikacije kojoj je poslan sadržaj, te taj odgovor i tok komunikacije sprema u *log* datoteku.

Programski jezik u kojem je program napisan zove se Python. Nastao je 1991., a njegov kreator je Guido van Rossum. Python je dobio ime po poznatoj TV seriji "*Monty Python's Flying Circus*". Kako navodi web stranica python.swaroopch.com[1], Python je objektno-orijentirani programski jezik koji je u isto vrijeme jednostavan i vrlo moćan. Zbog jednostavne sintakse, omogućava fokusiranje i bavljenje isključivo problemom kojeg programer želi riješiti, umjesto da se bavi samom problematikom programskog jezika i njegove sintakse. Iz tog razloga programi pisani u njemu su obično kraći od programa pisanih u drugim poznatim programskim jezicima, a sam programski jezik izrazito je pogodan za pisanje manjih skripti koje obavljaju jednostavne zadaće. Python je u današnje vrijeme našao vrlo široku primjenu: od biometrijskih aplikacija, preko pisanja kratkih skripti za različite potrebe održavanja sustava, do kompleksnih web aplikacija. Izuzetno je popularan u *open source*[2] krugovima, zbog modularnosti i portabilnosti. Program u ovom završnom radu pisan je u verziji 3 programskog jezika Python.

Prije detaljne analize programskog koda, potrebno je upoznati se s osnovnim konceptima i pravilima iz područja računalnih mreža. Kroz iduće veliko poglavlje „Arhitektura mreža“, predstavljeni su glavni principi rada računalnih mreža, protokola, te arhitekture i implementacije istih, koji omogućuju funkcionalnost ove, ali i funkcionalnost svih drugih aplikacija koje žele međusobno komunicirati putem Interneta i drugih računalnih mreža.

[1] https://python.swaroopch.com/about_python.html - pristupljeno 17.05.2018.
[2] „Open source“ krugovi – zajednica ljudi koji dijele mišljenje da bi sav programski kod morao biti javno dostupan i koja javno dijeli programska rješenja i aplikacije.

Nakon definiranja primarnih pojmova i principa rada računalnih mreža, završni rad u poglavlju „Analiza programskog rješenja“ opisuje sam kod aplikacije, te funkcionalnosti programskog jezika Python koje omogućuju razmjenu podataka između dvije aplikacije.

2. Arhitektura mreža

The Basic Reference Model for Open Systems Interconnection (skraćeno OSI model ili ISO/OSI) je dokument nastao 1983. godine kao produkt spajanja dva projekta koja su za cilj imala standardizaciju komunikacije između dva otvorena sustava koji međusobno žele razmijeniti informacije. Model se sastoji od 7 slojeva, pri čemu treba istaknuti da svaki sloj odvija određenu funkciju i pri tome koristi uslugu koji pruža niži sloj, a obavljajući svoju funkciju pruža uslugu višem sloju (izvor: http://mapmf.pmfst.unist.hr/~lada/rm/rm-pog2.pdf“).

Prema autorima Andrw S. Tanenbaum i David J. Wetherall[3], navedeni su principi koju su doveli do kreiranja ovih sedam slojeva su:

1. Kada god je neophodna nova apstrakcija, potrebno je kreirati novi sloj
2. Svaki sloj mora imati jasno definiranu funkciju
3. Funkcija svakog sloja mora biti izabrana u skladnu s međunarodno standardiziranim protokolima
4. Granice slojeva treba kreirati tako da se smanji protok informacija između slojeva
5. Broj slojeva mora biti dovoljno velik da se funkcije s bitno različitim namjenama ne bi svrstale u isti sloj, a opet dovoljno mali da arhitektura ne postane previše složena

Važno je naglasiti da OSI model ne predstavlja samu arhitekturu mreže zbog toga jer se modelom ne definiraju konkretne usluge i protokoli za svaki sloj, već on predstavlja smjernice u razvoju mrežnih protokola. Model također omogućuje projektiranje mreža prema slojevima, što olakšava izgradnju i planiranje većih i kompleksnijih mreža.

Prije dublje analize slojeva, potrebno je razjasniti nekoliko pojmova. Prema (Computer Networks, fifth edition – Andrw S. Tanenbaum, David J. Wetherall):

- Protokol – predstavlja dogovor između dvije strane o tome kako treba teći njihova komunikacija
- Računalna mreža – veći broj zasebnih, ali međusobno povezanih računala koji obavljaju određeni posao

[3] Computer Networks, fifth edition – Andrw S. Tanenbaum, David J. Wetherall, Pearson Education .Inc, 2011., (str. 63)

- Zaglavlje – upravljačke informacije ispred same poruke koje zajedno s porukom čine novu cjelinu u kontekstu određenog sloja ili protokola

U računalnoj znanosti, projektiranje sustava bazirano na slojevima je vrlo česta praksa. Kreiranje slojeva za svaku pojedinu funkciju sustava pokazalo se kao najbolja praksa prilikom dizajniranja, kako *hardwarea* tako i *softwarea*. Podjelom sustava na zasebne, individualne cjeline (slojeve) kreira se slijed u kojem sloj pruža funkcije višem sloju, a koristi usluge nižeg, a samim time se olakšava kontrola, nadgledanje i održavanje sustava. Ovim pristupom također dobivamo brži razvoj protokola i tehnologija na jednom sloju, jer su zadatci unutar sustava segmentirani.

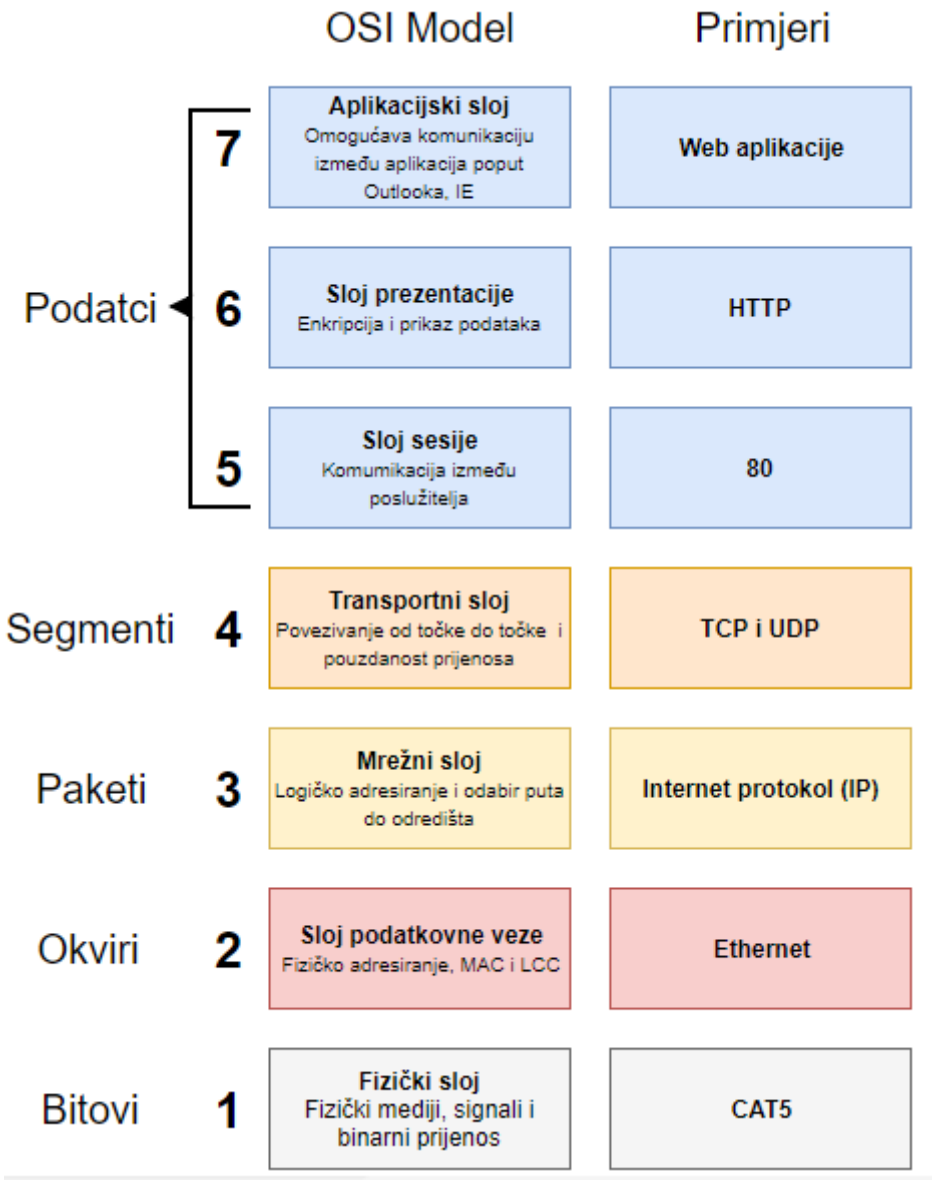
Prilikom dizajniranja sustava treba naglasiti da određeni sloj mora pružiti uslugu višem sloju, a da pritom ne prikazuje detalje unutrašnjeg stanja sloja, algoritme i postupke kojima je došao do rezultata koji prezentira višem sloju.

Prema autorima Andrw S. Tanenbaum i David J. Wetherall[4], slojevi koje OSI model definira su:

- Aplikacijski sloj
- Sloj prezentacije
- Sloj sesije
- Transportni sloj
- Mrežni sloj
- Sloj podatkovne veze
- Fizički sloj

Slika 1 prikazuje nam slojeve OSI modela na sredini, dok je na desnoj strani popis nekih od najpoznatijih protokola koji se koriste na pojedinom sloju. Poruka koja se šalje kroz računalnu mrežu se fragmentira, odnosno dijeli na manje dijelove koji se na odredištu spajaju u prvobitno stanje. Svakom od tih fragmenata (paketa) pridodaje se zaglavlje na pojedinom sloju. Na lijevoj strani se vidi kako se ti fragmenti nazivaju unutar pojedinog sloja.

[4] Computer Networks, fifth edition – Andrw S. Tanenbaum, David J. Wetherall, Pearson Education .Inc, 2011., str. 64)



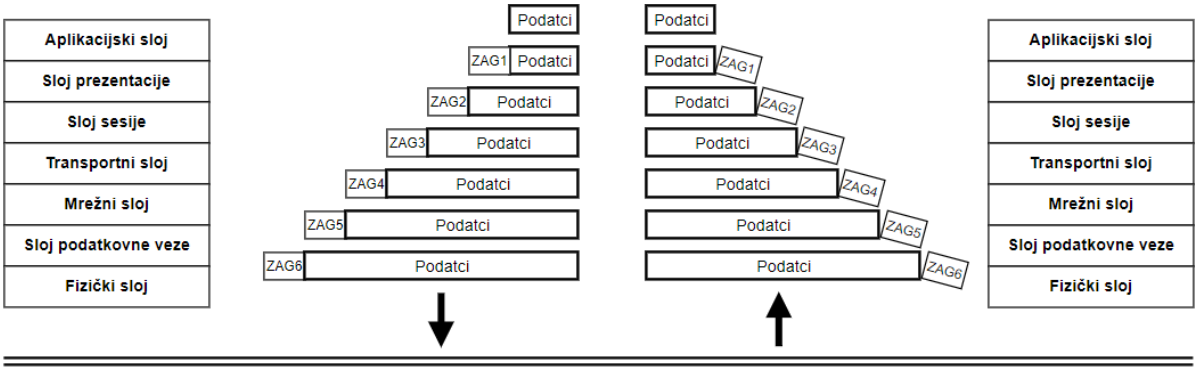
Slika 1: OSI model

Izvor: Izradio autor prema

https://i.pinimg.com/736x/22/31/b4/2231b42657f83924cfebf3e7410ebd39--osi-model.jpg

(pristupljeno 16.04.2018.)

Kako bi se lakše shvatila problematika, može se uzeti primjer slanja poruke sa slike 2. Na lijevoj strani nalazi se računalo A koje šalje poruku računalu B na desnoj strani. Prilikom slanja, poruka kod pošiljatelja prolazi kroz sve slojeve redom – aplikacijski sloj, sloj prezentacije, sloj sesije, transportni sloj, mrežni sloj, sloj podatkovne veze, fizički sloj. U trenutku pristizanja poruke na stranu primatelja, ona prolazi obrnutim redoslijedom – fizički sloj, sloj podatkovne veze, mrežni sloj, transportni sloj, sloj sesije, sloj prezentacije, aplikacijski sloj.



Slika 2: Primjer slanja poruke kroz OSI model

Izvor: Izrada autora prema https://tednetworking.blogspot.com/2016/12/normal-0-false-false-false-en-in-x-none_24.html

Slanje kreće od najvišeg sloja – Aplikacijskog. Kao primjer može se navesti elektronička pošta. Na početku poruka dobiva odgovarajuće zaglavlje aplikacijskog sloja, kako bi se znalo da je to elektronička pošta. Poruka se tada sastoji od korisničkog teksta, plus zaglavlja aplikacijskog sloja. Prilikom spuštanja poruke po slojevima, dodaju joj se zaglavlja određenog protokola na svakom sloju. Poruka se prilikom prijenosa više ne tretira kao jedna cjelina, nego kao skup fragmenata. Spuštajući se prema slojevima, poruka na najnižem sloju izgleda ovako:

PS zag. + SPV zag. + MS zag. + TS zag. + SS zag. + PS zag. + AS zag. + poruka

pri čemu je:

- PS zag. – zaglavlje fizičkog sloja
- SPV zag. – zaglavlje sloja podatkovne veze (npr. zaglavlje *Ethernet* protokola)
- MS zag. – zaglavlje mrežnog sloja (npr. zaglavlje IP protokola)
- TS zag. – zaglavlje transportnog sloja (npr. zaglavlje TCP protokola)
- SS zag. – zaglavlje sloja sesije
- PS zag. – zaglavlje prezentacijskog sloja
- AS zag. – zaglavlje aplikacijskog sloja (npr. zaglavlje SMTP protokola)

Promatrajući sliku 1, može se vidjeti da se na transportnom sloju ti fragmenti nazivaju segmenti, na mrežnom (kada im se pridoda IP zaglavlje) postaju paketi itd. Dakle, poruka se

više ne tretira kao u prvobitnom stanju, već je ona skup paketa, skup segmenata, skup bitova, ovisno o tome na kojem se sloju trenutno nalazi.

Nakon što poruka stigne na odredište, ponovo mora proći kroz svih 7 slojeva, ali ovaj put obrnutim redoslijedom. Za razliku od ishodišta, na odredištu se zaglavlja „otkidaju“. Nakon što sloj pročita zaglavlje namijenjeno njemu, on prosljeđuje poruku višem sloju bez tog zaglavlja. Taj proces se ponavlja sve dok poruka ne dođe do korisnika u istom formatu kao što je i bila poslana.

Na ovom primjeru vidi se da svaki sloj čita zaglavlje namijenjeno samo njemu, odnosno izvršava funkciju koja je specifična za taj sloj, te zatim prosljeđuje svoj rezultat (uslugu) višem sloju, bez da viši sloj mora brinuti ili imati informacije o tome kako je niži sloj došao do tog rezultata.

2.1. Aplikacijski sloj

Aplikacijski sloj je najviša razina u OSI modelu, a on pruža usluge izravno korisniku. Prema autoru Andrw S. Tanenbaum[5], slojevi ispod aplikacijskog sloja osiguravaju sigurni transport, ali sam korisnik nema utisak da navedeni slojevi stvarno nešto rade, jer nema direktne interakcije s njima. Aplikacijski sloj je onaj s kojim korisnik direktno vrši interakciju. Unutar aplikacijskog sloja razvile su se broje aplikacije koje koriste usluge svih slojeva niže razine. Kako bi aplikacije mogle raditi i unutar ovog sloja javila se potreba za protokolima. S obzirom na velik broj protokola, nabrojeni su samo najosnovniji i najkorišteniji protokoli:

- **DNS** (*eng. Domain Name System*) – protokol koji služi pretvaranju teksutalnih imena računala u mrežne (IP) adrese. S tehničke strane, računala mogu komunicirati korištenjem samo mrežnih adresa, međutim DNS pruža fleksibilnost i brojne dodatne usluge koje olakšavaju projektiranje i održavanje računalnih sustava. Neki od primjera su balansiranje mrežnog prometa (iza jednog naziva domene krije se više IP adresa prema kojima se prosljeđuju paketi), dostatnost samo zamjene zapisa na DNS serveru u slučaju ispada poslužitelja (kada se promet mora preusmjeriti na novi poslužitelj), olakšano izdavanje certifikata i drugo.
- **HTTP/HTTPS** (eng. *Hyper Text Transfer Protocol*) – prema službenom dokumentu dostupnom na web stranici w3.org[6], HTTP protokol je generički protokol koji se koristi

[5] Računarske mreže, prijevod četvrtog izdanja – Andrw S. Tanenbaum, Mikro knjiga Beograd, 2005., (str. 555)
[6] https://www.w3.org/Protocols/HTTP/1.1/rfc2616bis/draft-lafon-rfc2616bis-03.html#RFC2324 - pristupljeno 23.04.2018

primarno za distribuiranje *hiperlinkova*[7], datoteka, multimedije i ostalih sadržaja na Webu[8]. Ovo je jedan od najpoznatijih protokola jer putem Web preglednika korisnik izravno vidi njegovu najraširenuju zadaću i primjenu, a to je dohvaćanje web sadržaja. HTTP protokol zasniva se na modelu klijet – poslužitelj, gdje web preglednik predstavlja klijentsku stranu koja šalje zahtjev za sadržajem, dok poslužitelj odgovara na taj zahtjev i povratno šalje sadržaj.

- **SMTP** (eng. *Simple Mail Transfer Protocol*) – protokol za slanje elektroničke pošte. Služi za uspostavljanje veze između primateljevog i pošiljateljevog poslužitelja, te prosljeđuje poruke u odgovarajuće poštanske sandučiće

Ovo su tri najpoznatija protokola aplikacijskog sloja. Zbog velikog broja aplikacija, postoji i jako velik broj protokola, međutim nisu toliko značajni za razumijevanje aplikativnog djela ovog završnog rada.

2.2. Sloj prezentacije

Prezentacijski sloj je šesti po redu unutar OSI modela. Web stranica fossybytes.com[9] kao najvažnije zadaće sloja prezentacije ističe:

- **Prevođenje/Kodiranje** – prije slanja, podatci su u predstavljeni kao nizovi znakova i brojeva. Budući da je za slanje to neprihvatljiv oblik, potrebno je pretvoriti podatke u niz bitova kako bi se omogućila interoperabilnost između računala koja koriste različite vrste kodiranja
- **Enkripcija** – pruža uslugu kriptiranja kod strane pošiljatelja i dekriptiranja kod strane primatelja. Enkripcija pruža očuvanje integriteta i povjerljivosti podataka prilikom slanja preko nesigurnog kanala, kao što je Internet.
- **Kompresija** – omogućava sažimanje početne informacije, bez gubitka izvornog značenja. Razlozi zbog kojih se ovaj postupak izvršava mogu biti ograničenje prostora na disku primatelja, mala propusnost komunikacijskog kanala i drugo.
- **Konverzija** – omogućuje interoperabilnost između različitih sustava koji razmjenjuju podatke. Svako računalo i operacijski sustav tretira podatke na drugačiji način, pa je

[7] Hiperlink – u kontekstu weba predstavlja tekst koji vodi na neki drugi dokument (drugi (hiper) tekstualni dokument, sliku, video zapis, zvučni zapis i druge vrste datoteka), e-mail adresu ili neko drugo mjesto unutar web stranice.
[8] Web – skup dokumenata u elektroničkom obliku koji su međusobno povezani hiperlinkovima.
[9] https://fossbytes.com/presentation-layer-of-osi-model/ - pristupljeno 23.04.2018

zbog toga potrebna konverzija između formata, kako bi se omogućila komunikacija između različitih platformi.

Za razliku od slojeva niže razine koji se bave time kako premjestiti podatke s jednog kraja komunikacijskog kanala na drugi, ovaj sloj je fokusiran na sintaksu i semantiku tih podataka. Sloj prezentacije pokušava kreirati apstraktnu strukturu podataka zbog interoperabilnosti više platforma.

2.3. Sloj sesije

Autori Andrw S. Tanenbaum i David J. Wetherall[10] navode da sloj sesije omogućava korisnicima na različitim računalima da uspostave sesiju. Sesija je interakcija između dvije točke na krajevima komunikacijskog kanala, koje se odvijaju prilikom jednog povezivanja. Sesija se najčešće manifestira na način da jedna strana šalje zahtjev, druga prihvaća i ispunjava taj zahtjev, pa tako nastaje razmjena informacija koja traje tako dugo dok obije strane ne dogovore prekid komunikacije ili dok ne dođe do greške koja prekine komunikaciju.

Sloj sesije nudi usluge poput:

- Upravljanje dijalogom – vođenje računa o tome čiji je red za slanje poruke prilikom komunikacije dvije mrežne aplikacije.
- Rad sa žetonima (eng. *token*) – sprječavanje sudionika na istovremeno pokretanje kritičnih operacija.
- Sinkronizacije – provjeravanje niza podataka prilikom prijenosa, kako bi se u slučaju greške komunikacija nastavila od točke prije prekida.

[10] Computer Networks, fifth edition – Andrw S. Tanenbaum, David J. Wetherall, Pearson Education .Inc, 2011., (str. 67)

2.4. Transportni sloj

Transportni sloj je u sferi ovog završnog rada najvažniji od svih slojeva OSI modela, jer aplikacija koristi najviše usluge ovog sloja. Glavna zadaća transportnog sloja je prihvaćanje podataka od strane viših slojeva, prosljeđivanje tih podataka mrežnom sloju, prilikom čega osigurava da podatak ispravno stigne na odredište. U knjizi „Računarske mreže“[11] navedeno je da transportni sloj potpuno povezuje dva kraja: izvor i odredište. Povezivanje, poput aplikacije u ovom završnom radu, realiziraju međusobno slični programi koji koriste zaglavlja i upravljačke poruke, koje im nude protokoli transportnog sloja.

Implementacijom transportnog sloja u OSI model postiglo se rasterećenje samih aplikacija, jer one ne moraju znati mehanizme za uspostavu veze i transport podataka prema drugom računalu. Ovaj sloj također predstavlja apstrakciju mrežne veze s aplikacijom. Unutar ovog sloja postoje dva glavna i najraširenija protokola: TCP i UDP koji su detaljnije objašnjeni u idućim poglavljima. Prije detaljnije analize protokola, potrebno je definirati pojam priključak (eng. *port*).

Svako računalo posjeduje jedinstvenu adresu unutar mreže, kao što će to biti opisano u idućem poglavlju. Za komunikaciju između aplikacija, to nije dovoljna informacija i iz tog razloga potrebno je specificirati i port. Unutar transportnog sloja, port predstavlja i virtualni kraj konekcije, odnosno „mjesto“ na koje informacija mora pristići. Port se može zamisliti kao utičnica u koju se ušteka kraj kabela kako bi se zatvorio (spojio) strujni krug, a svojstvo virtualnosti znači da u tu utičnicu može biti uštekano više kablova (više klijentskih aplikacija može biti spojeno na jedan port).

Web stranica informatika.buzdo.com[12] definira port kao „brojčane vrijednosti temeljem kojih računalo po prihvatu podataka zna koju uslužnu programsku potporu (servise) mora aktivirati te na koji način razmjenjivati podatke“. Dakle, svaki proces u računalnom sustavu koji želi komunicirati s drugim procesima na udaljenom računalu, mora imati specificiran i dostupan port, kako bi se podatci između ta dva procesa mogli izmjenjivati. Broj koji određeni proces/servis koristi, može biti proizvoljno odabran, ali ni jedan drugi proces/servis na istom računalu ne smije koristiti taj broj. To znači da je broj porta jedinstven za svaku aplikaciju/servis na jednom računalu, a on može biti u rasponu od 1-65535. Za najčešće korištene servise postoje tzv.eng. *well known ports*. To je standardizirani popis portova za najčešće korištene servise, koji se mogu vidjeti unutar tablice 1 na idućoj stranici.

[11] Računarske mreže, prijevod četvrtog izdanja – Andrw S. Tanenbaum, Mikro knjiga Beograd, 2005., (str. 39)

[12] https://informatika.buzdo.com/s916-internet-tcp-udp-port-socket.htm - pristupljeno 24.04.2018.

Tablica 1: *„well known ports“*

Broj porta	**Ime procesa**	**Korišteni protokol**	**Opis**
20	FTP-DATA	TCP	*File transfer - data*
21	FTP	TCP	*File transfer – control*
22	SSH	TCP	*Secure Shell*
23	TELNET	TCP	*Telnet*
25	SMTP	TCP	*Simple Mail Transfer Protocol*
53	DNS	TCP i UDP	*Domain Name System*
67 (klijent prema serveru) 68 (server prema klijentu)	DHCPv4	UDP	*Dynamic Host Configuration Protocol version 4*
69	TFTP	UDP	*Trivial File Transfer Protocol*
80	HTTP	TCP i UDP	*Hypertext Transfer Protocol*
110	POP3	TCP	*Post Office Protocol 3*
123	NTP	TCP	*Network Time Protocol*
143	IMAP	TCP	*Internet Message Access Protocol*
443	HTTPS	TCP	Sigurna implementacija protokola HTTP
546 (klijent prema serveru) 547 (server prema klijentu)	DHCPv6	UDP	*Dynamic Host Configuration Protocol version 6*
3389	RDP	TCP	*Remote Desktop Protocol*

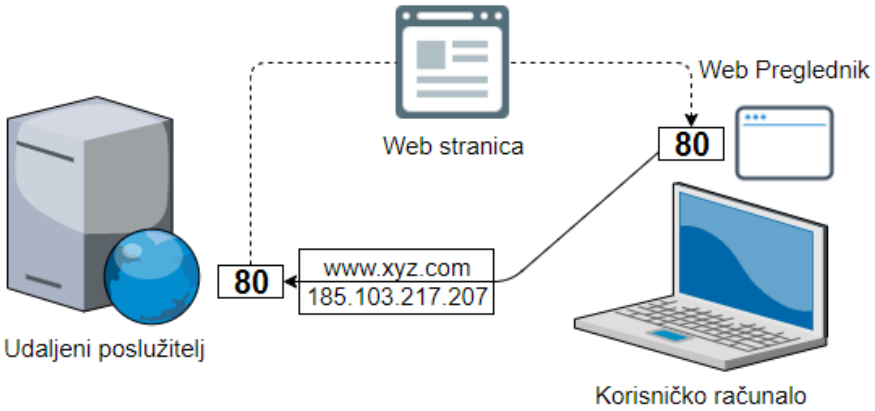
Izvor: Izrada autora prema http://slideplayer.com/6428944/22/images/45/Sockets+and+Ports+Well+Known+Port+Numbers+Demo.jpg, (pristupljeno 24.4.2018.)

Unutar tablice 1 također je vidljiv i pojam eng. *socket*. *Socket* predstavlja kombinaciju IP adresa-protokol-port, te je on zapravo krajnja točka prilikom komunikacije. Kao što vidi iz definicija, razlika između *socketa* i porta je ta što je port broj koji definira krajnju točku servisa,

dok je *socket* kranja instanca konekcije[13], odnosno *socket* je uređeni par koji sadrži više informacija, dok je port samo brojčana vrijednost koja govori samo o tome na kojem priključku proces čeka podatke.

Da bi lakše razumjeli portove može se navesti primjer pristupanja korisnika web stranici na udaljenom poslužitelju. Korisnik u svojem web pregledniku upisuje adresu stranice www.xyz.com. Ranije navedeni DNS protokol omogućuje da se sazna adresa poslužitelja koja se krije iza imena xyz.com. U ovom primjeru adresa je 185.103.217.207, pa tako web preglednik, nakon što je saznao adresu poslužitelja, želi uspostaviti konekciju i dohvatiti web stranicu. Tu se dolazi do praktične primjene portova. Na odredišnom računalo mora postojati servis koji željenu web stranicu može poslužiti korisnicima.

U definiciji portova spomenuto je da svaki servis mora otvoriti port i odabrati brojčanu vrijednost za taj port. Kod web servisa, najčešća vrijednost je broj 80 koji se koristi za HTTP protokol, te broj 443 koji se koristi za HTTPS protokol. Poslužitelj tako na portu 80 čeka zahtjeve klijenata koji uspostavljaju konekciju na tom portu i zatim preuzimaju web stranicu na svoje računalo, gdje web preglednik također na portu 80 prima podatke. Slijedeća slika pod brojem 3 prikazuje taj proces.



Slika 3: Primjer konekcije na port 80

Izvor: Izradio autor

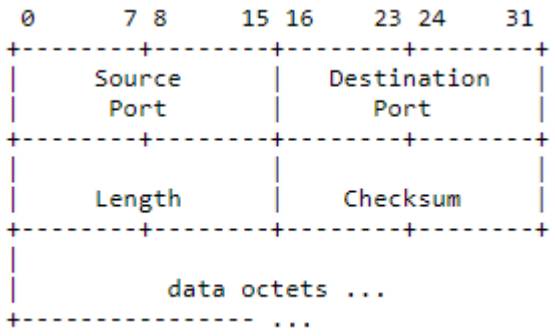
DNS servis, koji daje adresu odredišnog računala, također mora imati otvoren port 53, kako bi mogao zaprimiti zahtjev korisnika koji želi saznati adresu određene web stranice. DNS

[13] „A connection is defined by a pair of sockets“ - https://tools.ietf.org/html/rfc793 - pristupljeno 09.05.2018.

protokol počiva na protokolu transportnog sloja koji se naziva UDP, čiji su detalji i specifičnosti opisani u idućem poglavlju.

2.4.1. UDP (eng. User Datagram Protocol)

Stjepan Groš[14] navodi daj je UDP „vrlo jednostavan protokol prijenosnog sloja koji efektivno omogućava aplikacijama direktno korištenje usluga mrežnog sloja.“ To konkretno znači da protokol radi bez uspostavljanja direktne veze između primatelja i pošiljatelja i glavna odlika mu je brzina. U odnosu s drugim protokolima transportnog sloja, UDP ne uspostavlja vezu, nego se fokusira na čim brži prijenos segmenata. UDP protokol je opisan i definiram dokumentom RFC 768. Protokol prenosi segmente sastavljene od zaglavlja, veličine 8 bajtova, koje je nadodano na korisničke podatke. Zaglavlje je vrlo jednostavno i prikazano je na slici 4.



Slika 4: UDP zaglavlje

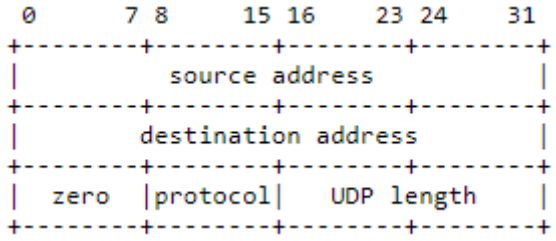
Izvor: https://tools.ietf.org/html/rfc768, (pristupljeno 24.04.2018.)

Zaglavlje se sastoji od slijedećih polja:

- Izvorišni broj porta – važan u slučaju slanja odgovora pošiljatelju, proces tada kopira izvorišni broj porta u odredišni, jer zna da drugi proces čeka na tom portu odgovor.
- Odredišni broj porta – isto kao i izvorišni, samo je on predstavalja broj porta primateljeve strane.
- Duljina (UDP paketa) – duljina paketa obuhvaća 8 bajtno zaglavlje UDP protokola i podatke koji se šalju, te definira ukupnu veličinu segmenta.

[14] http://www.zemris.fer.hr/~sgros/s tuff/mr/05.pdf - pristupljeno 24.04.2018.

- Kontrolni zbroj – računa se na način da se kreira tzv. pseudo-zaglavlje. Pseudo zaglavlje se kreira privremeno zbog izračuna kontrolnog zbroja, a uključuje IP adresu izvorišta i odredišta te ostala polja prikazana na slici 5. Polje kontrolnog zbroja nije obavezno i ima vrijednost 0 ako se nije izračunao. Isključivanje ovog polja se u praksi ne preporučuje, osim ako kvaliteta podataka nije od izuzetne važnosti (npr. digitalni zvuk).



Slika 5: Pseudo zaglavlje UDP protokola

Izvor: https://tools.ietf.org/html/rfc768, (pristupljeno 25.04.2018.)

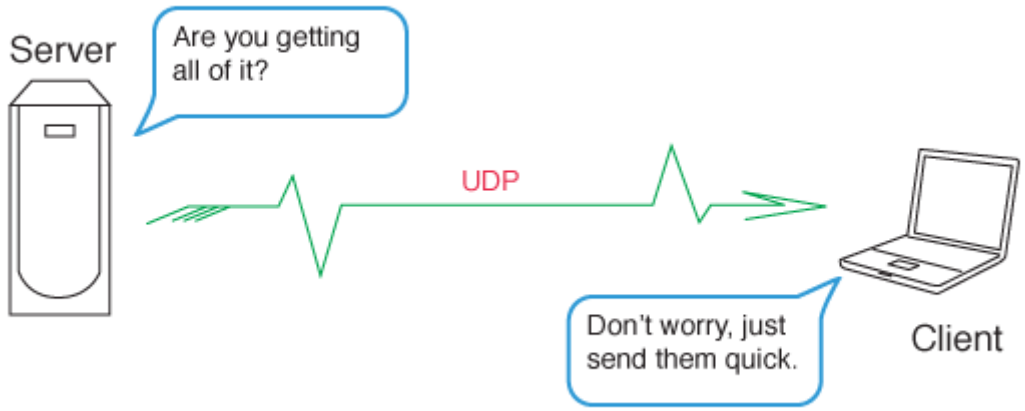
UDP zaglavlje predpostavlja da se izvorišna i odredišna adresa nalaze u zaglavlju mrežnog sloja (IP protokol), pa se zbog toga u zaglavlju nalaze samo brojevi portova kako bi optimizirali sam protokol i smanjili količinu dodatnih podataka koji se prenose. Ovaj pristup ima i nedostatke, pa tako autori u knjizi *Computer Networks*[15] jasno ističu da UDP protokol ne obavlja:

- kontrolu toka podataka
- kontrolu zagušenja komunikacijskog kanala
- kotrolu greške – ne kontrolira da li je došlo do pogreške, te ne radi ponovo slanje segmenta u slučaju da on nije stigao na odredište, jer ni ne zna što se dogodilo s segmetom

Navedene funkcije moraju obaviti korisnički procesi, a UDP protokol predstavlja samo sučelje procesu prema IP protokolu. Zbog svojih karakteristika, UDP protokol je izrazito koristan kod klijensko-serverski orijentiranih aplikacija. Klijet u malim vremenskim razmacima šalje zahtjev, a poslužitelj mu ogovara jednako tako kratim odgovorom. Ako dođe do gubitka odgovora, automatizirani proces generira novi zahtjev za slanje. Ovakav pristup pojednostavljuje programski kod, te uvelike smanjuje broj poruka koje dvije strane moraju razmijeniti, u odnosu na protokole koji zahtijevaju prethodnu uspostavu konekcije.

[15] Computer Networks, fifth edition – Andrw S. Tanenbaum, David J. Wetherall, Pearson Education .Inc, 2011., (str.: 561)

Ranije spomenuti DNS protokol upravo na ovakav način koristi usluge transportnog sloja, odnosno UDP protokola. Kada korisnik šalje zahtjev za IP adresom domene xyz.com, taj zahtjev je jedan paket male veličine, koji se šalje DNS poslužitelju na port 53. Nadređeni DNS poslužitelj odgovara IP adresom za određenu domenu, također putem UDP protokola. Ako korisnik ne zaprimi odgovor, generira novi zahtjev dok ne dobije adresu za željenu domenu. U ovom primjeru vidi se da nema potrebe za uspostavljanjem veze i njenim naknadim raskidanjem. Dovoljno je samo razmijeniti dvije kratke i jednostavne poruke. Iako DNS zbog brzine koristi UDP protokol, dok kontrolu greške, ostvaruje na slojevima više razine jer je točnost jako bitna kod navedenog protokola. Slika 6 trivialno prikazuje samu zamisao UDP protokola.



Slika 6: Trivialni prikaz UDP protokola

Izvor: https://solvethenetwork.com/wp-content/uploads/2016/10/udp.gif, (pristupljeno 25.04.2018.)

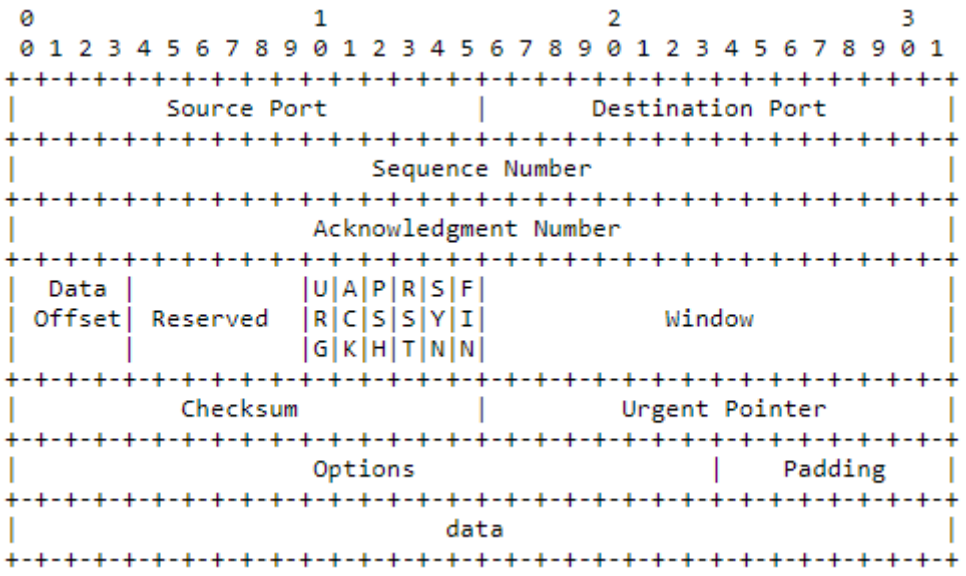
Osim kod DNSa, UDP protokol koristi se najčešće kod prijenosa multimedijskih sadržaja, pogotovo u realnom vremenu, gdje je važnije da se slika i ton dobiju što prije, od toga hoće li biti sitnih grešaka na slici uzrokovanih izgubljenim segmentima. Također se koristi i kod mrežnih računalnih igara, telefonije putem interneta (eng. *VoIP*) i drugo.

2.4.2. TCP (eng. Transmission Control Protocol)

UDP protokol svoju zadaću obavlja vrlo dobro, međutim za većinu mrežnih aplikacija njegova funkcionalnost nije dovoljna. Zahtjevi za pouzdanim prijenosom, uspostavom otvorene konekcije, slijednim slanjem paketa, te kontrolom toka i pojave grešaka nadilaze mogućnosti i dizajn UDP protokola. Iz tog razloga, TCP protokol razvio se u najpopularniji i najrašireniji protokol transportnog sloja.

Prema Stjepanu Grošu[16], TCP protokol nasljeđuje NCP (*eng. Network Control Program*) koji se koristio kao protokol transportnog sloja u ranim začetcima Interneta. U njegovom tekstu navodi se da je „primarna namjena protokola ostvarivanje pouzdane, dvosmjerne veze između bilo koja dva entiteta Internetu“. Protokol je službeno definiran dokumentima RFC 793, a njegove dopune, te ispravci nedostataka i odgovori na nove zahtjeve definirani su dokumentima RFC 1122 i RFC 1323.

Poput UDP zaglavlja, i TCP ima svoje zaglavlje, međutim zbog naprednijih i zahtjevnijih funkcionalnosti ono je dosta kompliciranje i sadrži puno veći broj polja, koje možemo vidjeti na slici 7.



Slika 7: TCP zaglavlje

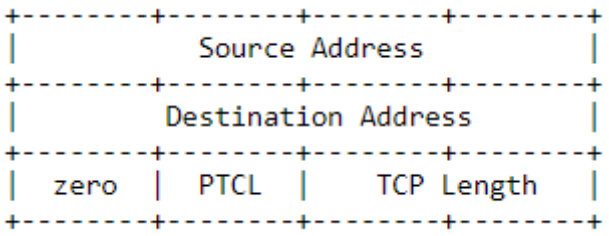
Izvor: https://tools.ietf.org/html/rfc793, (pristupljeno 25.04.2018.)

[16] http://www.zemris.fer.hr/~sgros/s tuff/mr/05.pdf - pristupljeno 25.04.2018.

Originalni dokument RFC 793 opisuje zaglavlje:

- Izvorišni port – važan u slučaju slanja odgovora pošiljatelju, proces tada kopira izvorišni broj porta u odredišni, jer zna da drugi proces čeka na tom portu odgovor
- Odredišni port – isto kao i izvorišni, samo je on predstavlja broj porta primateljeve strane
- Redni broj – broj prvog okteta podatka u segmentu (osim ako je zastavica SYN prisutna u zaglavlju)
- Broj potvrde – označava sljedeći očekivani oktet podataka, a samo polje je važeće ako je postavljena zastavica ACK
- Duljina TCP zaglavlja - broj 32-bitnih riječi u zaglavlju TCP segmenta, označava gdje počinju podatci
- Rezervirano – rezervirano za buduću uporabu, mora imati vrijednost 0
- Kontrolni bitovi:
    - URG – postavlja se vrijednost 1, kada se koristi polje „Pokazivač hitnosti“
    - ACK – postavlja se vrijednost 1, znači da je polje „Broj potvrde“ ispravno, ako je vrijednost 0, segment ne sadrži potvrdu, pa se polje „Broj potvrde“ zanemaruje
    - PSH – ako je vrijednost 1, podatci moraju biti odmah proslijeđeni (*PUSH*), odnosno da ih ne zadržava u međuspremniku
    - RST – koristi se za ponovnu uspostavu veze nastalu prekidom iz bilo kojeg neočekivanog razloga. Također se koristi za odbijanje neispravnih segmenata kojima se želi uspostaviti veza.
    - SYN – služi za uspostavljanje veze. Segment za uspostavljanje veze ima vrjenosti SYN=1 i ACK=0, dok dogovor na zahtjev ima vrijednosti SYN=1 i ACK=1. Ovim bitom se označuju segmenti za uspostavu konekcije, a ACK služi razlučivanju da li se radi o zahtjevu (ACK=0) ili prihvaćanju zahtjeva (ACK=1)
    - FIN – služi za zatvaranje veze i postavlja se na 1 kada pošiljatelj više nema segmenata koje mora poslati. Treba naglasiti da proces koji šalje zahtjev za raskidom i dalje može primati segmente.
- Veličina prozora – broj bajtova koji mogu biti poslanu pošiljatelju segmenta u kojem je zapisana ova vrijednost, uključujući i broj potvrde. Ako ima vrijednost 0, znači da su primljeni svi podatci zaključno s poljem *broj potvrde* koji ima vrijednost 1, ali da primatelj trenutno nije u mogućnosti zaprimiti nove podatke.

- Kontrolni zbroj – kontrolna suma izračunata na temelju TCP zaglavlja, podataka i pseudozaglavlja, a koristi se zbog veće pouzdanosti prijenosa.



Slika 8: Pseudo zaglavlje TCP protokola

Izvor: https://tools.ietf.org/html/rfc793, (pristupljeno 07.05.2018.)

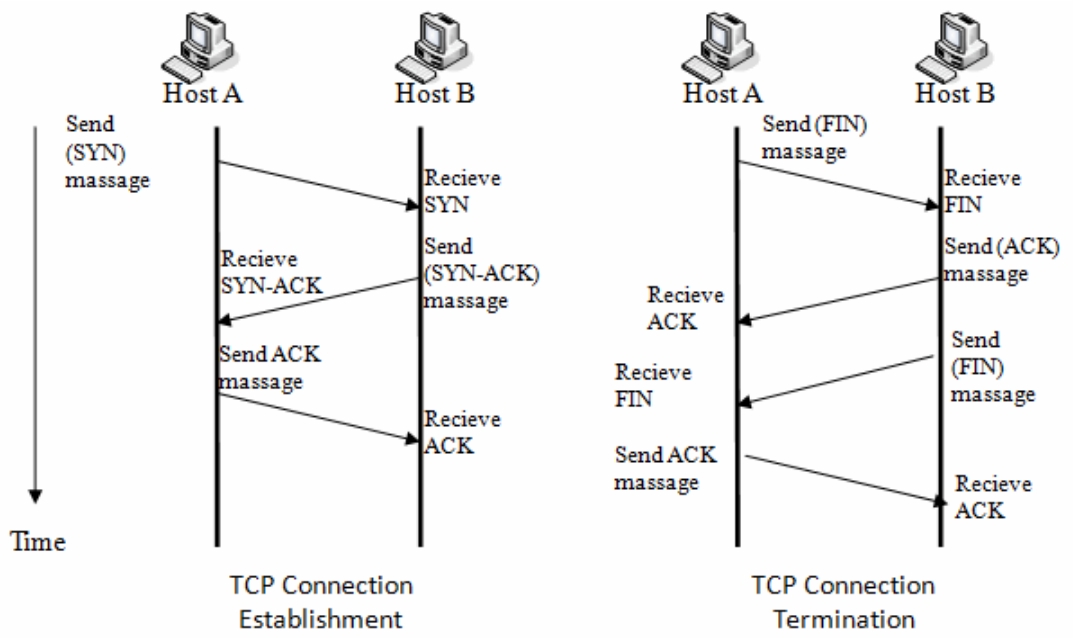
Pseudozaglavlje sadrži izvorišnu i odredišnu adresu računala, broj protokola i duljinu TCP zaglavlja plus duljinu podataka u oktetima. Ovo zaglavlje predstavlja i zaštitni mehanizam od krivo usmjerenih segmenata unutar mreže. Prilikom izračunavanja vrijednost polja, kontrolni zbroj se postavlja na 0, te se provjerava da li je dužina polja u kojem se nalaze podatci neparan broj i ako je, polje podataka dopunjava bajtom 0. Zatim algoritam koji izračunava vrijednost polja kontrolni zbroj zbraja 16 bitne riječi kao nepotpune komplemente[17] i nakon toga izračunava nepotpuni komplement tog zbroja. Primatelj nad cijelim segmentom mora napraviti identični postupak te dobiti vrijednost 0.

- Pokazivač hitnosti – ovo polje se koristi samo ako je postavljen URG kontrolni bit. Govori protokolu da taj segment ima prednost kod procesuiranja i mora čim prije stići na odredište. Često se koristi kod protokola *telnet* koji služi za dvosmjernu komunikaciju putem poruka u tekstualnom formatu korištenjem virtualnog terminala. Više detalja o implementaciji pokazivača hitnosti u TCP protokolu zapisano je u dokumentu RFC 6093 dostupnom na web stranici *https://tools.ietf.org/html/rfc6093*.
- Opcije – postoje 3 glavne opcije koje moraju biti implementirane prema RFC 793. Početak svake polja opcije je kod koji predstavlja samu opciju, a ako opcija prima parametre tada sadrži još jedan oktet sastavljen od broja koji predstavlja ukupnu duljinu opcije i njen parametar. Prema službenoj dokumentaciji, glavne opcije su:
    - eng. *End of Option List* - Označava kraj polja Opcije i početak podataka. Koristi se na kraju svih opcija, a ne iza svake opcije. Kod opcije je 00000000.

[17] Nepotpuni komplement – binarna operacija u kojoj se jedinice zamjenjuju nulom i obratno

- eng. *No-Operation* – nema posebno značenje i može se koristiti za poravnanje (eng. *offset*) iduće opcije na početak 32-bitne riječi
- eng. *Maximum Segment Size* – koristi se kako bi strane koje međusobno komuniciraju i razmjenjuju segmente mogle objaviti maksimalnu veličinu segmenta koje mogu prihvatiti. Kako navodi RFC 879, svaka strana može odrediti svoju maksimalnu veličinu segmenta. Ovo polje mora biti popunjeno prilikom inicijalnog zahtjeva za konekcijom (kada je SYN kontrolni bit postavljen). Ako se ova opcija ne koristi, sve veličine segmenata su dozvoljene prilikom prijenosa, a u maksimalnu veličinu segmenta ubrajaju se samo podatci bez TCP zaglavlja.

TCP konekcija se uspostavlja korištenjem tzv. trostrukog rukovanja (eng. *three way handshake*). Kroz primjer koji navodi web stranica[18], tvrtke *Mikrotik,* poznatog proizvođača mrežne opreme, objasnit će se proces uspostave i raskida konekcije. Na slici 9 prikazana su dva toka razmjene zaglavlja, na lijevoj strani vidi se primjer uspostave konekcije, a na desnoj strani primjer raskida konekcije.



Slika 9: Uspostava i raskid TCP veze

Izvor: https://wiki.mikrotik.com/images/f/fc/Image2001.gif, (pristupljeno 07.05.2018.)

Na lijevoj strani slike vidi se kako računalo A želi uspostaviti konekciju s računalom B. Računalo A tada u TCP zaglavlju šalje postavljen kontrolni bit SYN, te postavi vrijednost polja

[18] https://wiki.mikrotik.com/wiki/Manual:Connection_oriented_communication_(TCP/IP) - pristupljeno 07.05.2018.

redni broj u npr: 500. Računalo B zaprimi to zaglavlje i povratno šalje SYN-ACK, tj. postavi kontrolne bitove SYN i ACK te na taj način javlja računalu A da je spreman za uspostavu veze. Također, uz SYN-ACK, šalje i vrijednost 501 u polju *broju potvrde* (tu vrijednost je prethodno primio u polju redni broj), te postavlja novu vrijednost, npr. 600, u polju *redni broj*.

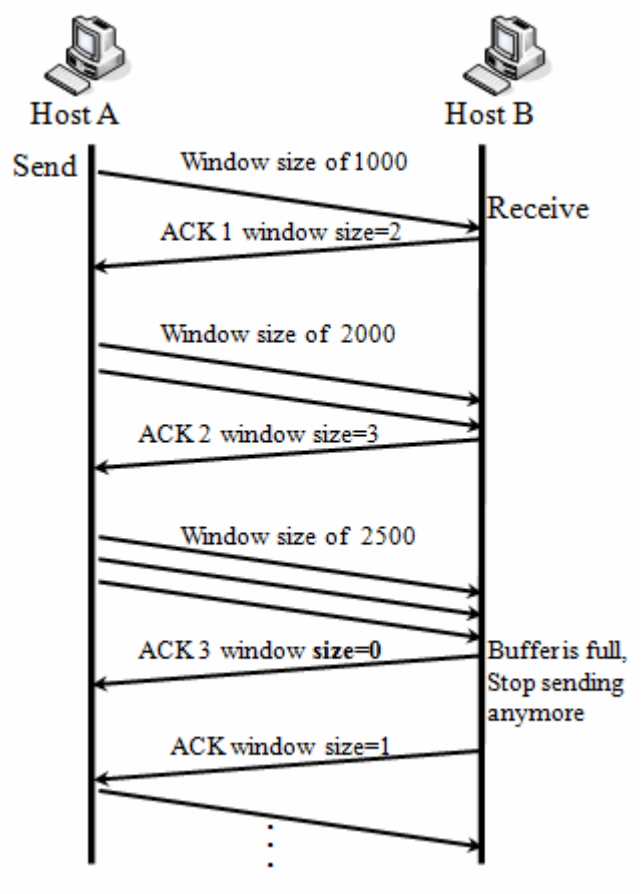
Računalo A zaprima taj segment, te odgovara s postavljenim kontrolnim bitom ACK i postavlja vrijednosti polja *broj potvrde* na 601. U trenutku kada računalo B zaprimi ACK računala A, veza je uspostavljena i računala mogu razmijeniti podatke. Kod protokola koji se oslanjaju na uspostavu veze, slanje potvrde o uspješnom primitku prethodnog segmenta predstavlja glavni mehanizam kontrole greške i kreiranja stabilne, perzistentne veze. Nakon slanja segmenta, pošiljatelj čeka potvrdu o primitku, a ukoliko je ne dobije nakon određenog vremenskog perioda, segment se šalje ispočetka.

Nakon uspješne uspostave veze, razmatra se primjer na desnoj strani gdje dolazi do raskida iste. Kod uspostave veze, proces povezivanja odvija se u tri koraka (eng. *three way handshake*), dok se prilikom raskida koristi četverostruku razmjenu poruka (eng. *four-way messages*). Veza između računala A i računala B zatvara se u trenutku kada oba računala zaprime ACK na segment, koji se odnosi na prije poslani segment s postavljenim kontrolnim bitom FIN. Prema slici 10, računalo A je poslalo sve podatke računalu B, te želi prekinuti međusobnu vezu. Računalo A šalje zaglavlje s postavljenim kontrolnim bitom FIN koji označava da je računalo A završilo s slanjem podataka. Računalo B ne terminira konekciju u istom trenutku kada zaprimi zaglavlje s postavljenim kontrolnim bitom FIN, već tada ulazi u CLOSE_WAIT stanje.

Računalo B na taj segment odgovara s ACK i potvrđuje da je spremno prekinuti vezu. Kada taj ACK stigne do računala A, ono ulazi u LAST_ACK stanje. Nakon razmijene ovih poruka, računalo B više ne zaprima podatke od računala A, ali može dalje slati podatke računalu A. Kada računalo B više nema podataka za slanje računalu A, ono šalje zaglavlje s postavljenim kontrolnim bitom FIN i time obavještava računalo A da su svi podatci poslani, te ulazi u TIME_WAIT stanje. Nakon toga računalo A odgovara s TCP zaglavljem u kojem je postavljen kontrolni bit ACK i u trenutku zaprimanja poruke kod računala B, veza se prekida.

Nakon upoznavanja s uspostavom i raskidom konekcije, promotra se kako izgleda sam proces slanja, te na koji način TCP protokol radi kontrolu toka podataka i uspijeva osigurati cijelokupni prijenos podataka bez gubitaka. Na web stranicama baze znanja tvrtke Mikrotik[19], nalazi se vrlo dobar i jednostavan primjer prijenosa podataka između dva računala.

[19] https://wiki.mikrotik.com/wiki/Manual:Connection_oriented_communication_(TCP/IP) - pristupljeno 08.05.2018.



Slika 10: Slanje podataka korištenjem TCP protokola

Izvor: https://wiki.mikrotik.com/images/3/34/Image2002.gif, (pristupljeno 08.05.2018.)

Primjer sa slike 10 izravno se naslanja na prethodni primjer, te se mora uzeti u obzir da su računala A i B međusobno uspostavila konekciju i da je korak trostrukog rukovanja uspješno završen. U ovom primjeru razmatra se situacija u kojoj računalo šalje podatke brže nego što to odredišno računalo može zaprimiti i procesuirati. Kod TCP prijenosa, svi zaprimljeni podatci spremaju se u međuspremnik (eng. *buffer*), pa tek nakon toga na mjesto koje pojedina aplikacija definira. Ranije je napomenuto da se spremanje podataka u međuspremnik može izbjeći korištenjem specijaliziranih polja za tu namjenu unutar TCP zaglavlja, ali u praksi svi podatci najčešće prolaze kroz neki međuspremnik.

Veličina tog međuspremnika je ograničena, te ako on postane krcat podatcima i više nema mjesta za primanje novih, dolazi do odbacivanja segmenata. Odbačeni segmenti moraju biti ponovo poslani, kako bi se ispunio zadatak TCP protokola koji govori o pouzdanom prijenosu podataka i otklanjanjem greške. Zbog retransmisije dolazi do pada performansi

samog prijenosa, pa je to uzrok sporijeg prijenosa u odnosu na UDP protokol. Dakle, zapunjenje međuspremnika dovodi do gubitka segmenata, pa TCP koristi protokol kontrole toka kako bi upravljao količinom podataka i smanjio broj odbačenih segmenata. To se realizira poljem *veličina prozora* unutar TCP zaglavlja koji je ranije spomenut kod opisa slike broj 8. U svakom segmentu, primatelj određuje vrijednost u polju *veličina prozora* koje predstavlja količinu podataka koju on može zadržati u međuspremniku. Kada se postavlja ili mijenja vrijednost u polju *veličina prozora,* šalje se i ACK, pa prema tome *veličina prozora* određuje količinu podataka koja može biti poslana bez zaprimanja novog ACK segmenta.

U slučaju da primatelj može zaprimiti veću količinu podataka od one koju šalje pošiljatelj, primatelj svakim novim segmentom povećava vrijednost polja *veličina prozora,* sve dok ta vrijednost ne dođe do veličine međuspremnika. Ako pošiljatelj zaprimi vrijednost 0, to znači da mora zaustaviti prijenos podataka, sve dok ne dobije pozitivnu vrijednost u polju *veličina prozora* od strane primatelja. Upravo je to ilustrirano na slici 11. U početku je *veličina prozora* postavljena na vrijednost 1000, pa je tako pošiljatelj poslao jedan segment veličine 1000 bajtova. Računalo B tada povećava vrijednost polja i šalje ACK, pa nakon zaprimanja tog paketa računalo A šalje dva paketa veličine 1000 bajtova. Nakon primitka i obrade pristiglih paketa, računalo B opet povećava vrijednost polja i šalje ACK, a pošiljatelj odgovara na to slanjem dva segmenta od 1000 bajtova i jednog segmenta od 500 bajtova.

U trenutku zaprimanja tih segmenata dolazi do popunjenja međuspremnika, zbog toga jer aplikacija ne može obraditi pristigle segmente brzinom većom nego oni stižu. Kako aplikacija još uvijek radi obradu pristiglih segmenata, računalo B šalje segment u kojem vrijednost polja *veličina prozora* postavlja na 0, što znači da računalo A mora pričekati s slanjem podataka. U trenutku kada se u međuspremniku oslobodi prostor, računalo B ponovo šalje segment u kojem polje *veličina prozora* postavlja na pozitivnu vrijednost i time obavještava pošiljatelja da može nastaviti slati podatke. Ovaj proces se ponavlja prilikom čitavog prijenosa, a određivanjem veličine prozora i brzine izmjene vrijednosti tog polja, upravljaju algoritmi za kontrolu zagušenja.

2.5. Mrežni sloj

Prethodno je na Slici 1 prikazano da se poslani podatci segmentiraju, te se im dodaje zaglavlje. Unutar mrežnog sloja, ti se segmenti nazivaju paketi. Prema tome, sysportal.carnet.hr[20] navodi da mrežni sloj pruža usluge povezivosti i najbolje putanje za paket kako bi on stigao do odredišta. Kako podatci kroz mrežu mogu putovati različitim putanjama, koristi se logičko adresiranje. Andrw S. Tanenbaum i David J. Wetherall[21] navode da mrežni sloj upravlja radom podmreže, te da je ključna zadaća ovog sloja odrediti način na koji paketi stižu do odredišta. Također navode da se putanje paketa mogu zasnivati na statičkim tablicama „ugrađenim“ u mrežu ili dinamičkim koje se određuju posebno za svaki pojedini paket ovisno o brojnim faktorima, poput zagušenja mreže.

Na temelju toga, može se vidjeti da se zadaće mrežnog sloja nameću same po sebi, reguliranje zagušenja, kontroliranje uskih grla i kolizije[22] paketa i najvažnije – provođenje paketa od ishodišta do odredišta. Posljednji zadatak je znatno drugačiji od onog koji obavlja sloj podatkovne veze, prenošenje paketa s jednog kraja medija na drugi.

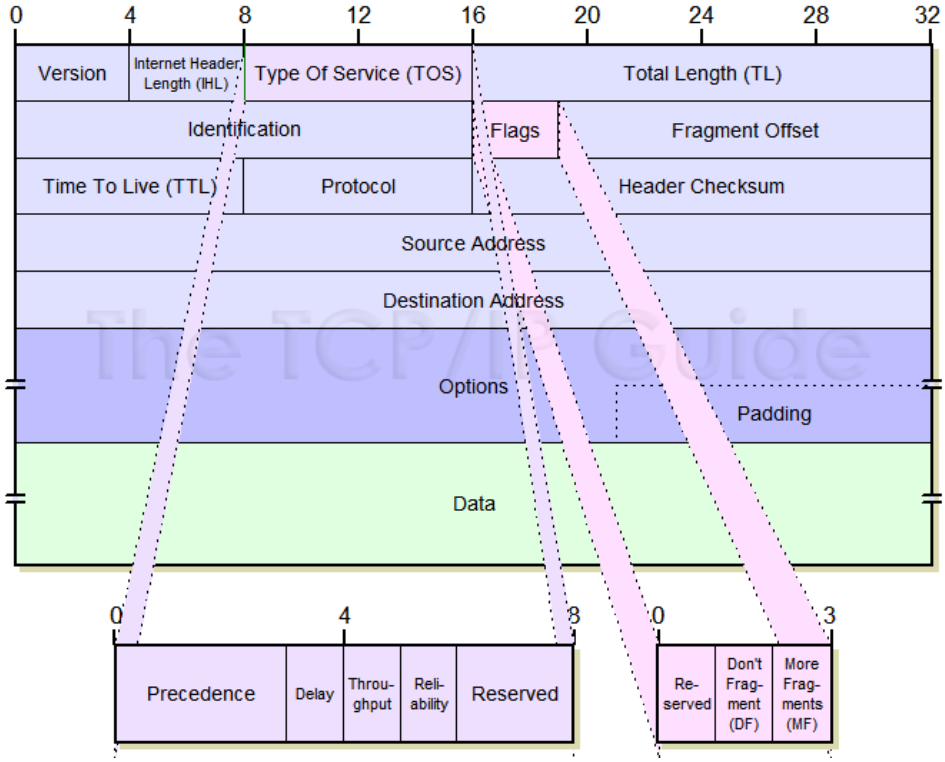
Mrežni sloj je izrazito širok i detaljan, pa će se spomenuti samo najbitnije stvari. Uz algoritme usmjeravanja i adresiranje, najvažniji segment mrežnog sloja predstavlja IP protokol. Prema mreze.layer-x.com[23], IP (*Internet Protocol* – RFC 791) je temeljni protokol internet razine TCP/IP arhitekture, kojeg koriste protokoli više razine. Protokol je sam po sebi bespojni, što znači da pošiljatelj i primatelj prije samog prijenosa ne uspostavljaju konekciju i dogovaraju početak i završetak iste, već pošiljatelj samo pošalje paket za koji ne dobi potvrdu primitka. Tu provjeru obavljaju protokoli viših razina. Iz tog razloga IP protokol naziva se i nepouzdanim protokolom.

Web stranica mreze.layer-x.com[24] navodi da su osnovne funkcije IP protokola:

- Definiranje sheme adresiranja na internetu
- Definiranje IP paketa
- Prosljeđivanje podataka između razine pristupa mreži i prijenosne razine
- Fragmentacija i sastavljanje paketa

[20] https://sysportal.carnet.hr/node/352 - pristupljeno 17.04.2018.
[21] Computer Networks, fifth edition – Andrw S. Tanenbaum, David J. Wetherall, Pearson Education .Inc, 2011., (str. 65)
[22] Kolizija - sudaranje paketa koje dovodi do grešaka i neispravnosti
[23] http://mreze.layer-x.com/s030100-0.html - pritupljeno 19.04.2018.
[24] http://mreze.layer-x.com/s030100-0.html - pritupljeno 19.04.2018.

Svakom paketu poslanom preko interneta dodaje se IP zaglavlje, prikazano na slici 11.



Slika 11: IP zaglavlje

Izvor: http://www.tcpipguide.com/free/diagrams/ipformat.png, (preuzeto: 19.4.2018.)

Zaglavlje IP protokola sastoji se od fiksnih 20 bajtova s varijabilnim dijelom (opcije). Prema službenom dokumentu RFC 791[25], te opisu autora Andrw S. Tanenbauma i David J. Wetheralla[26], polja u zaglavlju su:

- Verzija – predstavlja verziju protokola. U današnje vrijeme najrašireniji je IPv4 protokol. Postoji i razrađen je IPv6, međutim prelazak na isti zbog kompleksnosti i potrebe za redizajnom cijele arhitekture je spor, pa je i sama verzija IPv6 protokola vrlo slabo zastupljena, iako je prelazak na noviju verziju protokola neophodan zbog nedostatka slobodnih adresa verzije 4.
- IHL – eng. *Internet Header Length* polje postoji jer dužina zaglavlja nikad nije fiksa. U navedenom polju bilježi se prava dužina zaglavlja. Kako se veličina bilježi 32-bitnim riječima, najveća vrijednost polja je 15 (ograničavanje zaglavlja na 60 bajtova).

[25] Internet Protocol - Information Sciences Institute, str. 10-23, dostupno na https://tools.ietf.org/html/rfc791 - pristupljeno 19.04.2018.
[26] Computer Networks, fifth edition – Andrw S. Tanenbaum, David J. Wetherall, Pearson Education .Inc, 2011., (str. 457)

- Vrsta usluge – apstraktna vrijednost željene kvalitete usluge. Prilikom prijenosa moguće su različite kombinacije pouzdanosti i brzine prijenosa. U različitim mrežama može se koristiti za propuštanje paketa s većim prioritetom, prije paketa s manjim. Korištenje i kontrola nad parametrima ovisi o svakoj pojedinoj mreži.
- Ukupna dužina – obuhvaća sve što se nalazi u paketu, i zaglavlje i podatke. Maksimalna veličina je 65535 bajtova
- Identifikacija – služi odredišnom računalu za utvrđivanje koji fragment pripada kojem datagramu. Svaki fragmenti istog datagrama imaju jednaku vrijednost ovog polja
- Zastave (eng. *flags*) – Kontrolne „zastave“:

Tablica 2: Kontrolne zastave IP zaglavlja

0	1	2
0	DF	MF

Bit 0: rezervirano, mora biti 0

Bit 1: (DF) 0 = Smije se fragmentirati, 1 = Ne fragmentiraj

Bit 2: (MF) 0 = Zadnje fragmentiraj, 1 = Više fragmentiraj

Izvor: RFC 791 - https://tools.ietf.org/html/rfc791#page-11, (pristupljeno 19.4.2018.)

Kada se u zaglavlju nalaze bitovi za „Ne fragmentiraj“, pošiljatelj zna da će paket stići u jednom komadu, iako to značilo odstupanje od optimalne putanje do odredišta.

- Redni broj fragmenta – označava na kojem mjestu datagrama se fragment nalazi.
- Životni vijek – označava maksimalno vrijeme koje paket može provesti putujući mrežom. Ako polje sadrži vrijednost 0, paket se uništava, odnosno mrežni uređaju ga odbacuju. Prilikom svakog skoka paketa na mreži – dolaska do novog mrežnog uređaja koji mora proslijediti ili na drugi način obraditi paket – vrijednost polja se smanjuje za jedan. Maksimalna vrijednost pola je 255, što znači da jedan paket maksimalno može putovati mrežom 255 sekundi. Ovo polje ograničava beskonačno lutanje mrežom koje bi dovelo do zagušenja
- Protokol – označava protokol kojem mrežni sloj mora predati paket. Najčešće su to TCP i UDP, ali mogu biti i drugi procesi. Vrijednost se označava brojevima.
- Kontrolni zbroj zaglavlja – kontrolnim zbrojem zaglavlja provjerava se ispravnost samog zaglavlja koja može biti narušena neispravnim procesuiranjem od strane

usmjerivača. Vrijednost polja prilikom dolaska na odredište mora biti jednaka 0 ako nije došlo do greške. Kontrolni zbroj izračunava se pri svakom skoku jer se barem jedno polje uvijek mijenja (životi vijek)

- Izvorišna adresa – označava broj mreže i broj računala koje šalje paket
- Odredišna adresa - označava broj mreže i broj računala koja prima
- Opcije – vrijednosti koje se mogu i ne moraju pojaviti u zaglavlju. Primjer mogu biti sigurnosne opcije, vremenske oznake, strogo usmjeravanje itd.

U ovom opisu mrežnog sloja važno je još spomenuti IP adrese, vrijednosti koje se upisuju u polja „Izvorišna adresa“ i „Odredišna adresa“. Andrw S. Tanenbaum i David J. Wetherall[27] govore o tome da svako računalo na Internetu ima svoju IP adresu koja obuhvaća broj njihove mreže i broj samog računala. Kombinacija ovih dvaju brojeva je jedinstvena, što znači da na Internetu, u načelu, dva računala ne mogu imati jednaku IP adresu. Dužina svih IP adresa je 32 bita. Kod IP adresa je važno naglasiti da postoje javne i privatne IP adrese. Javne su one koje su dostupne na Internetu, dok se privatne koriste kao što im i samo ime govori – u privatnim mrežama. Pravilima u dokumentu RFC 1918[28] propisano je koje adrese su namijenjene za privatne mreže.

Kroz ovaj kratak pregled naglašene su osnovne funkcije i koncepti mrežnog sloja koji su potrebni kako bi se u kasnijim poglavljima razumjela funkcionalnost računalnog programa.

2.6. Sloj podatkovne veze

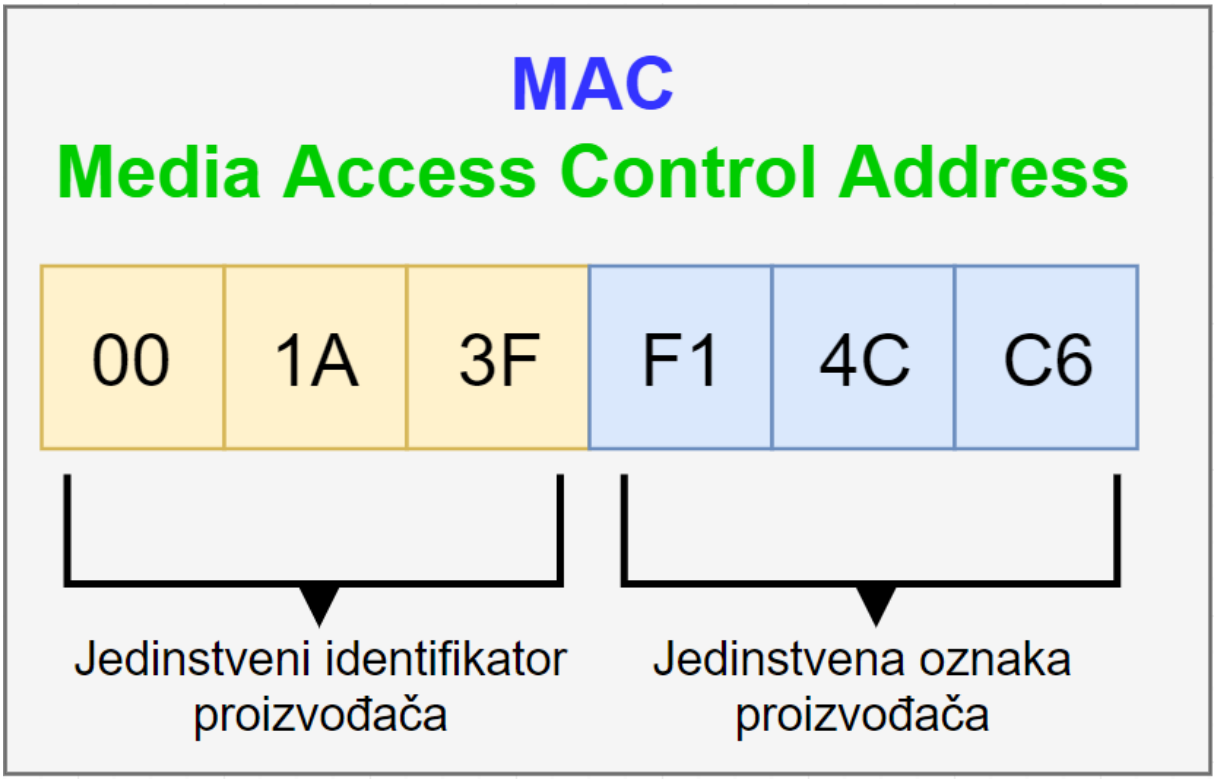
Sloj podatkovne veze omogućuje pouzdan prijenos podataka kroz fizički sloj. On brine o fizičkom adresiranju, topologiji mreže, kontroli protoka itd. Zadaće su mu: komunikacija s mrežnim i fizičkim slojem, segmentacija paketa u okvire koji su veličine koje fizički sloj može prenositi, rukovodi bitovima (postavlja start i stop bitove za prijenos) itd. Najvažniji protokoli su Ethernet, a važne su i fizičke adrese uređaja spojenih na mrežu – MAC adrese.

Prema autorima Andrw S. Tanenbaum i David J. Wetherall[29], glavni zadatak sloja podatkovne veze je da neposredno višem mrežnom sloju, pripremi transportnu liniju za prijenos podataka koja niz bitova prenosi bez greške. Sloj podatkovne veze također omogućuje pouzdan prijenos podataka kroz fizički sloj. Ovaj zadatak realizira se tako da se ulazni podatci, koje sloj

[27] Computer Networks, fifth edition – Andrw S. Tanenbaum, David J. Wetherall, Pearson Education .Inc, 2011., (str. 457)
[28] RFC 791 - https://tools.ietf.org/html/rfc1918 - pristupljeno 19.04.2018.
[29] Computer Networks, fifth edition – Andrw S. Tanenbaum, David J. Wetherall, Pearson Education .Inc, 2011., (str. 65)

podatkovne veze prima od mrežnog sloja, dijele na okvire (Slika 1, eng. *frames*) veličine od nekoliko stotina do nekoliko tisuća bajtova, koji se potom slijedno šalju do odredišta. Prilikom pravilnog prijenosa, primatelj za svaki okvir šalje povratnu informaciju pošiljatelju da je zaprimio poslani okvir. Ta povratna informacija naziva se eng. *acknowledgement frame*.

Kako navode autori Andrw S. Tanenbaum i David J. Wetherall[30] i web stranica poznate kompanije Juniper[31], unutar sloja podatkovne veze može se izdvojiti i podsloj za upravljanje pristupa mediju (eng. *medium access control*). Ovaj podsloj se specifično bavi problemom upravljanja pristupa zajedničkom kanalu. Podsloj omogućava adresiranje na fizičkoj razini te kontrolu pristupa kanalu kako bi uređaju koji sudjeluju u komunikaciji mogli pristupiti zajedničkom kanalu. Uređaji unutar istog kanala mogu se međusobno identificirati korištenjem MAC adrese – 12-tero znamenkasti heksadekadski zapis (dug 48 bitova).



Slika 12. Primjer MAC adrese

Izvor: Izrada autora prema https://i1.wp.com/thehappylearning.com/wp-content/uploads/2016/11/What-Is-MAC-Media-Access-Control-Address-Uses-Of-MAC-Address.png, (preuzeto 17.04.2018.)

[30] Computer Networks, fifth edition – Andrw S. Tanenbaum, David J. Wetherall, Pearson Education .Inc, 2011., (str. 275)

[31] https://www.juniper.net/documentation/en_US/junos/topics/concept/mac-qfx-series-understanding.html - pristupljeno 17.04.2018.

Najpoznatiji protokoli unutar sloja podatkovne veze su ARP (eng. *address resolution protocol*), Ethernet, IEEE 802.11 (WiFi), LACP (eng. *Link Aggregation Control Protocol*), L2TP (eng. *Layer 2 Tunneling Protocol*) itd.

2.7. Fizički sloj

Prema autorima Andrw S. Tanenbaum i David J. Wetherall[32], fizički sloj u OSI modelu ima zadaću prenošenja bitova kroz komunikacijski kanal, od primatelja do pošiljatelja. Navedeni niz bitova dobiva od njemu neposredno višeg sloja - sloja podatkovne veze. Glavna zadaća i briga prilikom projektiranja je ta da kada jedna strana pošalje bit 1, da druga strana primi bit 1, a ne bit 0. Glavni arhitekturalni dogovori između dvije strane moraju se postignuti oko napona koji predstavlja 0 ili 1, koliko dugo traje jedan bit prilikom prijenosa, da li se radi o jednosmjernoj ili dvosmjernoj komunikaciji, na koji način dolazi do uspostave konekcije, koje su vrste priključaka i što predstavlja koji kontakt.

Prema navedenim zahtjevima vidi se da se tu većinom radi o mehaničkim i električnim sklopovima, sklopovima za sinkronizaciju podataka, kao i fizičkim medijem za prijenos (bakar, optika, zrak i dr.). Dakle, fizički kontrolira električne i mehaničke funkcije koje se odnose na slanje i primanje komunikacijskih signala.

[32] Computer Networks, fifth edition – Andrw S. Tanenbaum, David J. Wetherall, Pearson Education .Inc, 2011., (str. 65)

3. Izrada programskog rješenja

3.1. Princip rada: pitanje-odgovor

Sljedeće poglavlje detaljno opisuje programski kod, te blokove od kojih je građen program. Dubljoj analizi programskog koda prethodi opis programa na visokoj razini, te opis funkcionalnosti pojedine funkcije i metode. Ova aplikacija radi na principu da pošalje poruku ili datoteku prema odredištu, zaprimi odgovor i zatvori socket. To znači da nije namijenjena za komunikaciju sa servisima koji šalju više odgovora, jer će se zaprimiti samo prvi.

Korisnik nakon pokretanja programa upisuje ime .csv datoteke u kojoj je pripremio podatke na temelju kojih će se obaviti slanje, te naziv datoteke u kojoj će se spremati informacije o komunikaciji i poslanim, odnosno primljenim porukama. CSV datoteka mora se nalaziti u istom direktoriju kao i sam program, a treba biti strukturirana na slijedeći način:

1. # - redni broj konekcije
2. eng. *host* – upisuje se IP adresa odredišnog računala
3. eng. *port* – upisuje se broj porta na koji program šalje poruku
4. eng. *protocol* – određuje koji protokol unutar transportnog sloja se koristi, a može poprimiti vrijednost TCP ili UDP
5. eng. *time(ms)* – određuje vrijeme čekanja u milisekundama u odnosu na prethodno slanje
6. eng. *message* – sadrži poruku koju korisnik programa želi poslati na odredište. Ako korisnik želi poslati datoteku, ime datoteke mora biti upisano unutar navodnih znakova s prefiksom f. Datoteka mora biti u istom direktoriju kao i program koji se pokreće

Pravilno strukturirana .csv datoetka izgledala bi:

```
#,host,port,protocol,time(ms),message

1,10.100.0.124,TCP,1000,“Pokusaj slanja na privatni IP“

2,10.100.0.8,UDP,5500,f“datoteka.txt“

3,185.142.221.14,UDP,160,“Pokusaj slanja UDP i javni IP“
```

Slika 13. Primjer .csv datoteke

Izvor: Izradio autor

Na temelju navedene .csv datoteke, program bi realizirao tri konekcije, pri čemu bi prilikom prve i treće bile poslane poruke, a kod druge bi se slala datoteka imena datoteka.txt.

Slanje prve poruke bilo bi pokrenuto nakon sekunde od pokretanja programa, dok bi datoteka bila poslana 5.5 sekundi nakon što je poslana prva poruka. Treća veza uspostavila bi se 160ms nakon slanja datoteke.

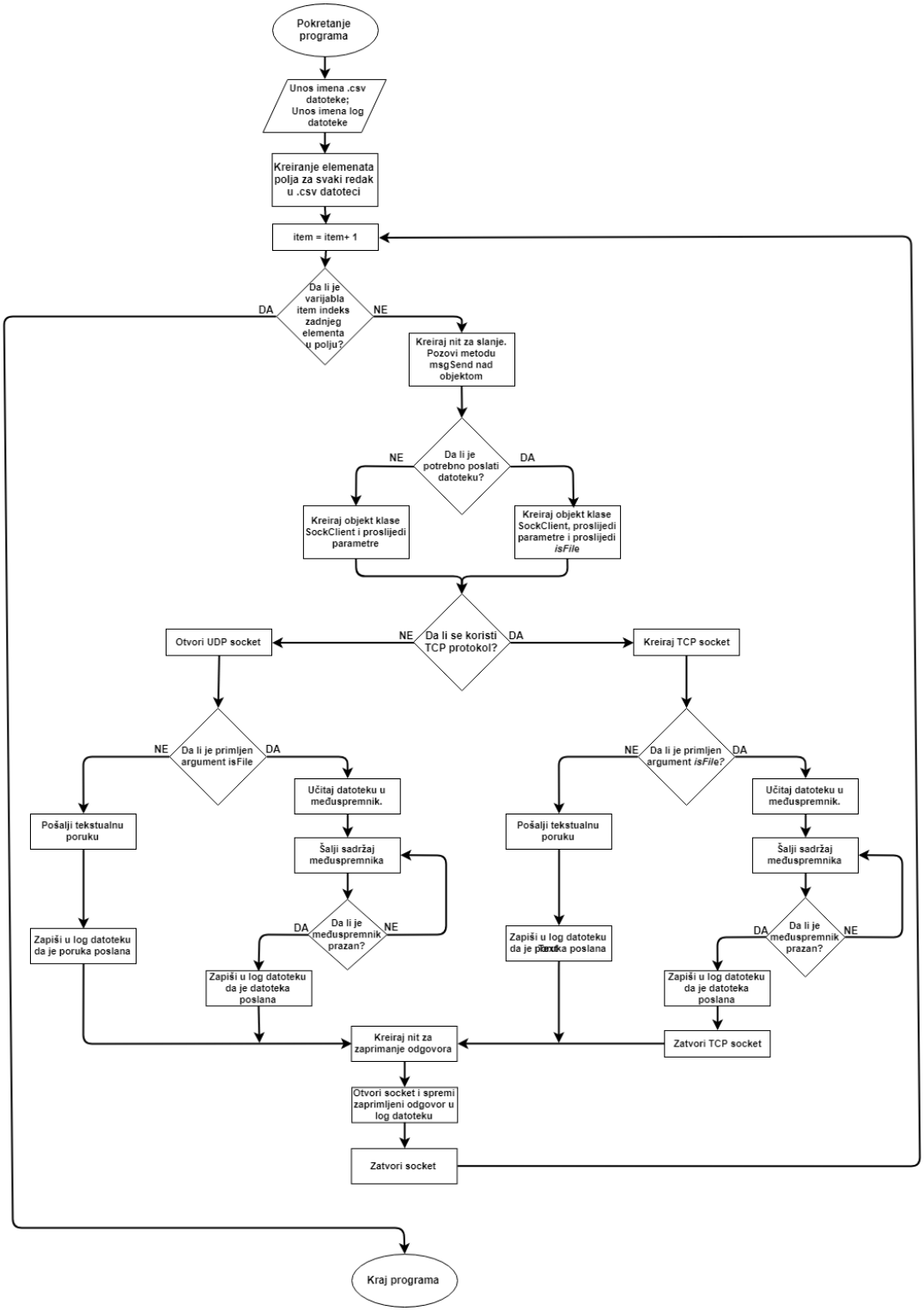
Kao što je napomenuto, program kao ulaz prima .csv datoteku, koju učitava u radnu memoriju i vrši analizu podataka koji se nalaze u istoj. Taj proces obavlja funkcija *inputData()*. Na temelju tih podataka poduzimaju se koraci koji vode do izlaznog produkta programa, a to slanje podataka od ishodnišnog sustava na kojem je program pokrenut, do odredišnog sustava, kojem je kranja točka definirana u .csv datoteci.

Poslije učitavanja datoteke i spremanje njenog sadržaja u polje, čita se sadržaj polja, odnosno svaki redak iz .csv zapisa. Za svaki element polja poziva se funkcija *checkstring()* koja provjerava se da li su u datoteci nalazi definirano ime datoteke koju je potrebno poslati. Ako je korisnik definirao datoteku koju želi poslati, ova funkcija koristi ugrađenu funkciju *open()* unutar programskog jezika *Python*. Datoteka se otvara za čitanje u binarnom obliku, te funkcija *checkstring()* vraća varijablu unutar koje je spremljen binarni zapis datoteke. Nakon toga se kreira objekt *socketTry* klase *SockClient*, te se prosljeđuju argumenti iz polja. Ako je potrebno poslati datoteku, umjesto vrijednosti u polju *message*, proslijeđuje se varijabla s binarnim zapisom datoteke.

Kreiranjem dviju dretvi *threadSend* i *threadRecv* pozivaju se metode *msgSend* i *msgRecieve* nad objektom *socketTry*. Pozivanjem prve metode, *msgSend*, pokreće se proces otvaranja socketa. Unutar nje se provjerava da li je definiran TCP ili UDP protokol, na temelju čega se kreira *socket* pripadajuće vrste. Također se provjerava da li je u pitanju slanje datoteke ili obične poruke, pri čemu treba naglasiti da proces slanja datoteke nešto drugačiji, te se javlja potreba za otvaranjem međuspremnika. Datoteka se tada učitava u međuspremnik i dio po dio se šalje na odredište, dok međuspremnik nije prazan. Njegova veličina može se definirati. Nakon slanja, socketi se zatvaraju. Potom se poziva metoda *dataLog* unutar iste klase *SockClient*. Ona služi za bilježenje događaja u datoteku čije je ime korisnik definirao na početku. S druge strane, metoda *msgRecieve* čeka odgovor od pošiljatelja, te taj odgovor bilježi u datoteku, također putem metode *dataLog*. I u ovom slučaju se *socketi* zatvaraju, ali nakon završetka slanja poruke za taj zapis.

Dakle, korisnik programa mora pripremiti .csv datoteku definiranog formata i opcionalno datoteke koje želi poslati, te ih mora smjestiti u isti direktorij kao i program. Pokretanjem programa unosi ime .csv datoteke i ime datoteke u koju želi spremiti informacije o slanju. Zatim kreće proces slanja prikazan na diagramu toka na slici broj 14, a korisnik može

unutar konzole promatrati tijek. Završetkom izvođenja programa, u .log datoteci nalaze se detaljnje informacije o izvođenju programa, poslanim i primljenim sadržajem.



Slika 14. Dijagram tijeka TCP-UDP aplikacije
Izvor: Izradio autor

3.1.1. Uvoz biblioteka (eng. *import libraries*)

Prije analize samog koda, potrebno je uvesti biblioteke koje su ugrađene u programski jezik python i čije klase i metode omogućuju sve funkcionalnosti ove aplikacije. Modul *time* omogućuje različite funkcije vezane uz manipulaciju vremena. Ostali su moduli objašnjeni u pododjeljcima koji slijede.

```
import socket
import csv
from threading import Thread
import logging
import time
import datetime
import sys
```

Programski kod 1: Uvoz biblioteka

Izvor: Izradio autor

3.1.1.1. Modul *socket*

Modul *socket* je direktna tranzicija socket modula iz UNIX sustava, prilagođena objektno orijentiranom pristupu python programskog jezika. Modul podržava nekoliko vrsta socketa (eng. address family), od kojih su najznačajniji:

1. AF_UNIX
    - Služi za međusobnu komunikaciju između procesa koji se izvršavaju lokalno, unutar istog računala
2. AF_INET
    - Služi za mrežnu komunikaciju između procesa, korištenjem uređenog para: IP adresa ; broj porta.
3. AF_INET6
    - Isto kao i AF_INET samo što se u ovom slučaju koristi IPv6
4. AF_TIPC
    - TIPC je mrežni protokol koji nije baziran na IP arhitekturi, a koristi se kod računala povezanih u klaster. Adrese su zapisane u tip podatka *tuple*, a ostala polja ovise o tipu adrese

3.1.1.2. Modul CSV

Modul CSV omogućuje korištenje najpopularnijeg i najraširenijeg formata datoteka za uvoz i izvoz baza podataka i proračunskih tablica. Format je standardiziran dokumentom RFC4180 u kojem su propisana pravila i definicije ove vrste zapisa. Iako je format zapisa definiran dokumentom RFC, ne postoji strogo zapisan standard, pa postoje razlike kod formata zapisa, ovisno o aplikacijama koje generiraju .csv datoteku.

Najčešće razlike u zapisu su vezane uz znak koji se koristi kao graničnik. Bez obzira na to što .csv znači vrijednosti odvojene zarezom (eng. *comma separated values*), broje aplikacije za izvoz tih vrijednosti koriste i druge znakove poput –, | ili ;. S obzirom na to da je format, bez obzira na graničnik, relativno identičan u svim varijantama, ovaj modul omogućuje efikasno manipuliranje podatcima zapisanim u ovom formatu, uglavnom koristeći objekte čitač (eng. *reader*) i pisač (eng. *writer*). U ovom završnom radu upravo se koristi objekt čitač (eng. *reader*), koji čita vrijednosti iz .csv datoteke.

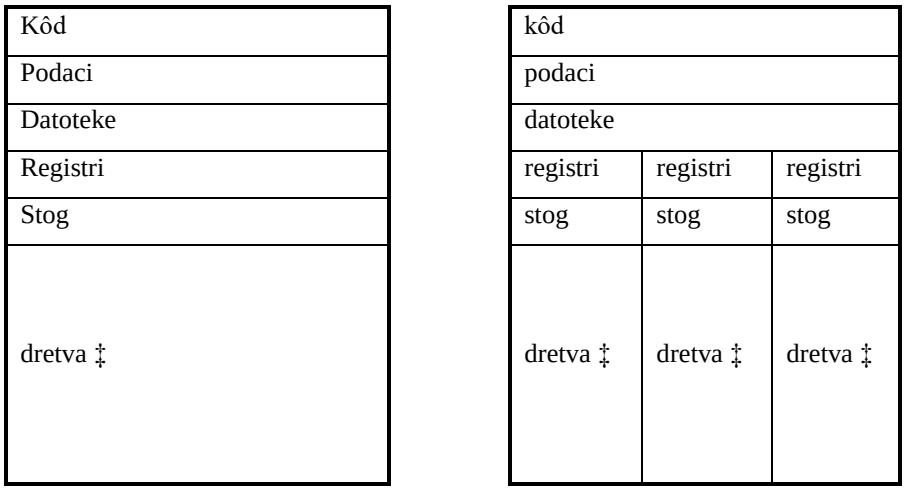
3.1.1.3. Modul *Thread*

Aplikacija istovremeno ostvaruje TCP i UDP konekcije. Ovaj modul upravo je zaslužan za tu funkcionalnost istovremenog obavljanja dva zadatka, jer omogućuje da se više funkcionalnosti aplikacije izvršava u istom vremenskom trenutku korištenjem dretvi[33] (eng. *thread*).

Dretva je programska cjelina koja može obavljati jedan zadatak, a sastoji se od niza instrukcija koje se izvode. Proces se sastoji se od jedne ili više dretvi. Budući da aplikacija treba izvršavati dva zadatka, u ovom slučajem ti zadatci su „uspostavi TCP-vezu“ i „šalji UDP“, implementacija dretvi je idealno rješenje za ostvarivanje željene funkcionalnosti. Danas, u vrijeme višejezgrenih procesora, teži se tome da se neki zadatak razbije na manje dijelove koje će obavljati mnogobrojne dretve na raspoloživim jezgrama. Na slici se može vidjeti razliku između jednodretvenih i višedretvenih procesa.

[33] Osim standardne riječi dretva u hrvatsko susrećemo i termin nit

Tablica 3: Slojeviti prikaz računalnog procesa



Izvor: Izradio autor (prema: http://www.csc.villanova.edu/~mdamian/threads/posixthreads.html), pristupljeno 27.04.2018.

3.1.1.4. Modul *Logging*

U ovom modulu, definirane su funkcije i klase za evidentiranje događaja odn. vođenje dnevnika (eng. *logging*) za postupak bilježenja rada funkcije, dretve ili procesa. Taj zapis se najčešće zapisuje u tekstualnom obliku u .txt ili .log datoteku. Jedna od najvećih prednosti korištenja *logova* je da prilikom pogreške u programom, možemo vidjeti gdje je on prestao s radom, te zbog koje pogreške je došlo do prestanka rada istog. Također, koristan je i za praćenje stanja programa ili sustava, kao i za kreiranje statistika. Najvažnije klase unutar modula su:

- Loggers – sučelje koje aplikacija direktno koristi
- Handlers – šalju zapise na zadano odredište
- Filters – služe za filtriranje što će se logirati
- Formatters – određuju oblik zapisa u datoteku

Razine logiranja predstavljaju količinu detalja i status zapisa koji se bilježe unutar log datoteke. U ovom programu koristi se razina INFO, a sama funkcionalnost logiranja detaljnije je definirana u poglavlju dataLog.

Prema dokumentaciji dostupnoj na https://docs.python.org/3/library/logging.html , razine su sljedeće:

Tablica 4: Numeričke vrijednosti razine logiranja

Razina	Vrijednost
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10
NOT SET	0

Izvor: https://docs.python.org/3/library/logging.html, pristupljeno 28.04.2018.

3.1.2. Klasa SockClient

Nakon inicijalizacije biblioteka, kreiramo klasu *SockClient* koja će služiti za kreiranje soketa i uspostavljanje konekcija, zaprimanje odgovora od strane poslužitelja i zapisivanje svih informacija o konekcijama u .log datoteku. Klasa sadrži inicijalizacijsku metodu *__init__* i još tri vlastito definirane metode: *msgSend*, *msgReceive* i *dataLog*.

Metoda msgSend služi za kreiranje konekcije prema poslužitelju i slanje poruke ili datoteke. Ona kreira socket ovisno o zaprimljenom argumentu (TCP ili UDP) te šalje poruku ili datoteku, ovisno o zaprimljenom argumentu. Druga metoda, *msgReceive*, služi za zaprimanje odgovora poslužitelja, dok je metoda *dataLog* zadužena za bilježenje svih informacija o konekcijama, Svaka od tih metoda detaljnije je opisana u vlastitom poglavlju.

Kako je cilj aplikacije mogućnost slanja putem TCPa i UDPa istovremeno, klasa *SockClient* je definirana kao *eng. child* klasa ugrađene klase *Thread* koja se nalazi unutar biblioteke *threading*. Prema tome, klasa *SockClient* nasljeđuje sve metode njezine matične klase *Thread* što nam kasnije omogućava kreiranje istovremenih konekcija. Inicijalizacijska metoda *__init__*, koristi se za kreiranje objekta te je detaljno opisana u idućem poglavlju.

```
class SockClient(Thread):
```

Programski kod 2: Inicijalizacija klase

Izvor: Izradio autor

3.1.2.1. Inicijalizacija

Poslije definiranja klase, potrebno je definirati što će se dogoditi prilikom njenog pozivanja, odnosno inicijalizacije. Praksa kreiranja objekta prilikom inicijalizacije je vrlo česta, kako prilikom instanciranja klase ne bi kreirali prazni objekt. Metoda *__init__* služi upravo tome. Ova metoda prima pet parametara: *host, port, socketType, delay, msg*.

U popisu argumenata, a i unutar cijele klase, može se vidjeti *self*. Kod objektno orijentiranog pristupa pythonu, *self* omogućava korištenje atributa istog objekta nad kojim se poziva neka metodu. U primjeru koda vidi se kako se svakom od argumenata pridružuje varijabla *self.ime_argumenta*. To znači da će svaki kreirani objekt imati one atribute, koji su predani klasi prilikom njegovog kreiranja. Ako ne bi koristili *self* , tada metoda koja bi se pozvala nad objektom ne bi znala koje podatke dohvatiti. Npr. kreirali bi objekte *slanjeA* i *slanjeB*. Tim objektima bi pridružili atribute *odrediste* i *poruka*. Ako ne bi koristili *self*, metoda za slanje *slanjeA.posalji()* ne bi znala koje atribute *odrediste* i *poruka* koristiti za slanje, da li od objekta *slanjeA* ili *slanjeB* i prema tome konekcija ne bi bila uspostavljena. Dakle, svakom objektu pridružujemo atribute koje dobiva preko argumenata, a *self* nam govori da atributi (konkretno podatci koji se nalaze u .csv datoteci) pridruženi prilikom kreacije objekta, pripadaju isključivom tom istom objektu.

Atribut *delay* je potrebno modificirati u adekvatni tip podatka, kako bi funkcije koje koriste ovaj podatak bile u mogućnosti prihvatiti i prepoznati njegovu vrijednost. Datoteka tipa csv ima zapisano vrijeme čekanja u milisekundama. Taj broj potrebno je pretvoriti u tip podatka eng. *float*, kako bi se moglo koristiti računsku operaciju dijeljenja, i potom podijeliti s 1000 da bi se dobilo vrijeme u sekundama. Ovaj korak potrebno je izvršiti kako bi kasnije metoda zadužena za realiziranje vremena čekanja između slanja, mogla primiti ispravan podatak.

```
def __init__(self, host, port, socketType, delay, msg):
            self.host = host
            self.port = port
            self.socketType = socketType
            self.msg = msg
            self.delay = float(delay) / 1000
```

Programski kod 3: Inicijalizacija __init_ metode

Izvor: Izradio autor

3.1.2.2. Metoda msgSend

Metoda *msgSend* zapravo je ključni dio ovog programa. Nakon pozivanja metode, prvo se izvršava metoda *sleep()* ugrađene klase *time*, koja se inicijalizira na početku programa. Ta metoda prima jedan parametar, a to je vrijeme u sekundama. Ova metoda omogućava ostvarivanje čekanja zapisanog u .csv datoteci, između slanja paketa i zbog nje je bilo potrebno modificirati argument *delay*.

Zatim slijedi glavna selekcija. Provjerava se da li je u .csv datoteci zapisan zahtjev za TCP ili UDP slanjem. Također, vrijednost varijable *socketType* pretvara se u mala slova, kako podatci iz .csv ne bi bili osjetljivi na velika i mala slova. Slijedi kreiranje objekta *clientSocket* tipa socket s parametrom AF_INET. *socket.SOCK_STREAM* nam definira da se radi o TCP socketu. Potom se poziva metoda *connect* koja prima parametre o poslužitelju na koji se šalje i o portu na koji se šalje, te se uspostavlja konekcija na odredište. U ovom slučaju ti parametri imaju vrijednost atributa *host* i *port* kreiranog objekta.

Nakon uspostave konekcije slijedi provjera o vrsti poruke koja se mora poslati. Ako se radi o običnoj poruci (tip string), poziva se metoda *send* nad objektom *clientSocket* koja prima parametar tipa string, a to je u ovom slučaju poruka koja se nalazi unutar .csv datoteke. S obzirom na to da je program pisan u pyhton verziji 3, potrebno je enkodirati string, pa se zbog toga koristi metoda *encode()*. Na samom kraju poziva se metoda *dataLog()* koja kao parametre prima status o smjeru slanja (*RECEIVED* ili *SENT*), te poruku. Metoda zatim zapisuje te informacije u datoteku .log. Taj postupak je opisan kasnije u poglavlju **dataLog**. Ako se radi o datoteci, kreira se varijabla dataBuffer u koju metoda *read()* sprema zapis iz te datoteke. Veličina buffera je postavljena na 1024KB. While petlja provjerava da li je varijabla dataBuffer prazna i sve dok nije, na zaslon ispisuje poruku o slanju, a potom se učitava novih 1024KB datoteke i postupak se ponavlja dok cijela datoteka nije poslana.

Nakon što je sav sadržaj poslan, na zaslon ispisuje poruku o završetku slanja. Pozivanjem metode *shutdown()* nad objektom *clientSocket*, zatvara se TCP konekcija, a metoda *close()* koja je poziva nad objektom *msg* zatvara datoteku koja je bila otvorena za slanje.

I u ovom slučaju se na kraju poziva metoda *dataLog()* koja kao parametre prima status o smjeru slanja (*RECEIVED* ili *SENT*), te poruku. Metoda zatim zapisuje te informacije u datoteku .log. Taj postupak je opisan kasnije u poglavlju **dataLog**.

```
def msgSend(self):
            time.sleep(self.delay)
            #if TCP
            if self.socketType.lower() == "tcp":
                  self.clientSocket = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
                  self.clientSocket.connect((self.host, int(self.port)))
                  #if txt
                  if type(self.msg) is str:
                        self.clientSocket.send(self.msg.encode())
                        self.dataLog("SEND", self.msg)
                  #if file
                  else:
                        dataBuffer = self.msg.read(1024)
                        while (dataBuffer):
                              print ("Sending...")
                              self.clientSocket.send(dataBuffer)
                              dataBuffer = self.msg.read(1024)

                        print ("Done sending")
                        self.clientSocket.shutdown(socket.SHUT_WR)
                        self.msg.close()
                        self.dataLog("SENT", "File was sent!")
```

Programski kod 4: metoda msgSend za TCP konekciju

Izvor: Izradio autor

U slučaju da se radi o zahtjevu za kreiranjem UDP socketa, kronologija je gotovo identična. Kreira se objekt *clientSocket* tipa socket s parametrom AF_DGRAM. s*ocket.SOCK_DGRAM* nam definira da se radi o UDP socketu. Slijedi provjera da li se radi o običnoj poruci (string) ili datoteci.

Razlika kod slanja obične poruke je ta da nije potrebno uspostaviti konekciju, kao kod TCP handshakea, već se koristi metoda *sendto* koja prima parametre poruku, poslužitelj i port, te nakon toga slijedi slanje poruke. Kod datoteke je situacija malo složenija, pa je opet potrebno kreirati varijablu u koju se zapisuje sadržaj datoteke (*dataBuffer = self.msg.read(1024)*). Slanje je identično kao kod običnog stringa gdje se metodom *sendto* šalje datoteka na odredište.

I u ovom slučaju se nakon uspješnog slanja zatvara socket (*metoda shutdown* nad objektom *clientSocket*), zatvara datoteka (metoda *close* nad objektom *msg*) i izvršava se zapisivanje u log datoteku.

```
#if UDP
      elif self.socketType.lower() == "udp":
            self.clientSocket = socket.socket(socket.AF_INET,
            socket.SOCK_DGRAM)
            #if txt
            if type(self.msg) is str:
                  self.clientSocket.sendto(self.msg.encode(), (self.host,
                  int(self.port)))
                  self.dataLog("SENT", self.msg)
            #if file
            else:
                  dataBuffer = self.msg.read(1024)
                  while (dataBuffer):
                        print ("Sending...")
                        self.clientSocket.sendto(dataBuffer, (self.host,
                        int(self.port)))
                        dataBuffer = self.msg.read(1024)

                  print ("Done sending")
                  self.clientSocket.shutdown(socket.SHUT_WR)
                  self.msg.close()
                  self.dataLog("SENT", "File was sent!")
```

Programski kod 5: metoda msgSend za UDP konekciju

Izvor: Izradio autor

3.1.2.3. Metoda msgReceive

Sljedeća metoda, *msgReceive* zadužena je za primanje informacija od strane poslužitelja na koji šaljemo poruku ili datoteku. Ova metoda je također definirana unutar klase *SockClient*. Metoda koristi objekt *clientSocket*, nad kojim poziva metodu *reciv()*. Ta metoda služi za prihvaćanje podataka poslanih od strane servera, a kao parametar prima veličinu međuspremnika u koji sprema odgovor. Također, primljenu poruku je potrebno dekodirati u odgovarajući string encoding tip, pa se zato poziva metoda *decode()*.

Pozivanjem metode *dataLog()*, koja kao parametre prima status o smjeru slanja (*RECEIVED* ili *SENT*) i poruku, realizira se zapisivanje te informacije u datoteku .log. Taj postupak je opisan kasnije u poglavlju **dataLog**.

```
def msgRecieve(self):
            repaly = self.clientSocket.recv(1024).decode()
            print(repaly)
            self.clientSocket.close()
            self.dataLog("RECIEVED", repaly)
```

Programski kod 6: metoda msgReceive

Izvor: Izradio autor

3.1.2.4. Metoda dataLog

Posljednja metoda unutar *SocketClient* klse je metoda *dataLog*. Ova metoda prima dva parametra, *direction* i *msg*. Prvi parametar predstavlja smjer u kojem je poruka ili datoteka bila poslana (*SEND* ili *RECIVE*), dok drugi parametar predstavlja samu poruku.

Navedena metoda upotrebljava funkciju logging.info unutar logging modula. Ova funkcija zapisuje predane informacije u .log datoteku. Ovdje je definirano samo ono što će se zapisati u datoteku i u kojem formatu, a sama konfiguracija vezana uz datoteku i razinu logiranja nalazi se unutar funkcije inputData. Zapis započinje datumom i vremenom kada je konekcija uspostavljena, zatim se ispisuje vrsta socketa (TCP ili UDP) i poruka, te na kraju odredište i smjer (da li je zapisana poruka primljena ili poslana)

```
def dataLog (self, direction, msg):
            self.direction = direction
            self.msg = msg

            logging.info(str(datetime.datetime.now())+ "|" +
            str(self.socketType) + "| MSG: " + str(self.msg) + " | FROM: " +
            str(self.host) + "| " + str(self.direction))
```

Programski kod 7: metoda dataLog za generiranje log datoteka

Izvor: Izradio autor

3.1.3. Funkcija inputData

Glavna zadaća funkcije inputData je da, na temelju unosa korisnika, otvori datoteku i spremi njezin sadržaj u polje. Na samom početku, funkcija traži od korisnika da unese ime .csv datoteke iz koje će se čitati podatci za slanje, te da odabere ime datoteke u koju će se spremiti log zapis. Unutar ove funkcije također definiramo i konfiguraciju logging modula. Logging.basicConfig definira osnovnu konfiguraciju za logiranje, a kao parametre prima ime

datoteke u koju zapisuje i jednu od razina zapisivanja koje su opisane u poglavlju Logging modul.

Varijabla importFile predstavlja objekt tipa file. To znači da je u toj datoteci spremljen sadržaj datoteke. Datoteka se otvara ugrađenom funkcijom open(). Ta funkcija kao parametre prima ime datoteke i mod, da li se iz datoteke čita, piše ili dodaje u nju. U ovom slučaju se samo čita iz datoteke, pa se koristi 'r'. Nad kreiranom objektom importFile poziva se:

```
def inputData():
      inputVariablesFile = input("Enter input file (messages and destinations)
      name: ")
      outputLogFile =  input("Enter output file (log) name: ")

      logging.basicConfig(filename=str(outputLogFile) +
      ".log",level=logging.INFO)

      importFile =  open(str(inputVariablesFile) + ".csv", "r")
      importValues = csv.reader(importFile)
      valueArray = []

      for row in importValues:
            valueArray.append(row)
      importFile.close()

      return valueArray
```

Programski kod 8: funkcija inputData

Izvor: Izradio autor

3.1.4. Funkcija checkString

Funkcija *checkSrting* provjerava da li je u .csv datoteci zapisano ime datoteke koju je potrebno poslati ili se radi o poruci (običnom stringu). Funkcija prima jedan argument, tipa string, koji predstavlja zapis unutar .csv datoteke. Taj zapis je formata *f"imeDatoteke"*. Prije same provjere, u varijablu *msgLen* sprema se duljina zapisa kako bi kasnije mogao provjeriti zadnji znak u zapisu.

Glavna selekcija provjerava da li je na nultoj poziciji slovo „f" koje označava da se radi o datoteci, te da li se na prvom i posljednjem mjestu nalaze navodnici. Ako je rezultat selekcije istinit, znači da se radi o datoteci koju je potrebno otvoriti i pripremiti za slanje.

U varijablu *fileName* sprema se ime datoteke. To ime započinje trećim znakom u zapisu (prvi je *f*, drugi je *„*), a završava pretposljednjim (posljednji je *„*) pa se zbog toga uzima samo

vrijednosti od 2. do predzadnjeg (-1) indeksa. Nakon toga poziva se ugrađena funkcija *open()*, koja otvara datoteku u binarnom zapisu (*rb*). Funkcija potom vraća vrijednost varijable *f*, koja je objekt tipa datoteke, odnosno to je sama datoteka u binarnom zapisu.

```
def checkString(sendMsg):
      msgLen = len(sendMsg) - 1
      if (sendMsg[0] is 'f') and (sendMsg[1] is '"') and (sendMsg[msgLen] is
      '"'):
            fileName = sendMsg[2:-1]
            f = open(fileName, "rb")
            return f
```

Programski kod 9: funkcija checkString

Izvor: Izradio autor

3.1.5. Funkcija Main

Glavna funkcija u programu zadužena je za pozivanje metoda iz klase, te funkcije check string za provjeru zapisa iz .csv datoteke. Nakon definicije glavne funkcije, poziva se metoda inputData() čija se vrijednost sprema u varijablu dataArray. U toj se varijabli tada nalaze svi zapisi iz datoteke, odnosno sve konekcije koje program treba ispitati. For petlja prolazi korz svaki zapis (osim prvog retka iz datoteke, jer on predstavlja zaglavlje). Najprije se u varijablu isFile sprema vrijednost iz datoteke u 6. stupcu koja govori da li se mora prenositi tekstualna poruka ili datoteka. Ako se radi o prijenosu datoteke, u konzolu se korisniku ispisuje obavijest, te se pokušava uspostaviti konekcija s udaljenim računalom pozivanjem SockClient metode.

SockClient metoda kao parametre prima odredišno računalo, broj porta na koji se mora spojiti, vrstu protokola, vrijeme čekanja između slanja slijednih paketa, te na koncu i samu datoteku. Konekcija se sprema u varijablu sockerTry, nad kojom se kasnije mogu izvršavati funkcije biblioteke socket. Ako se radi samo o tekstualnoj poruci, metoda prima iste parametre, ali se umjesto datoteke prosljeđuje tekstualna poruka koju nije potrebno fragmentirati prije slanja. Nakon kreiranja socket objekta, potrebno ih je inicijalizirati i izvršiti.

U try-catch bloku, koji provjerava greške i terminira program ako dođe do njih, kreiraju se dvije dretve: threadSend i threadRecv, te su nad njima pozvane metode start(), za pokretanje, i join() za kreiranje međusobne ovisnost dretvi. Na samom kraju u varijablu e spremaju se greške (iznimke) koje su se javile prilikom pokretanja dretve.

```
def Main():
	dataArray = inputData()

	for item in dataArray[1:]:
		isFile = checkString(item[5])
		if isFile:
			print("sendFile")
			socketTry = SockClient(item[1], item[2], item[3], item[4],
			isFile) #host/port/protocol/delayTime/msg

		else:
			#normalSend
			socketTry = SockClient(item[1], item[2], item[3], item[4],
			item[5]) #host/port/protocol/delayTime/msg

		try:
			threadSend = Thread(target=socketTry.msgSend, args=())
			threadRecv = Thread(target=socketTry.msgRecieve, args=())

			threadSend.start()
			threadSend.join()
			threadRecv.start()
			threadRecv.join()
		except Exception as e:
			print (e)
Main()
```

Programski kod 10: funkcija Main

Izvor: Izradio autor

3.2. Princip rada: beskonačni UDP

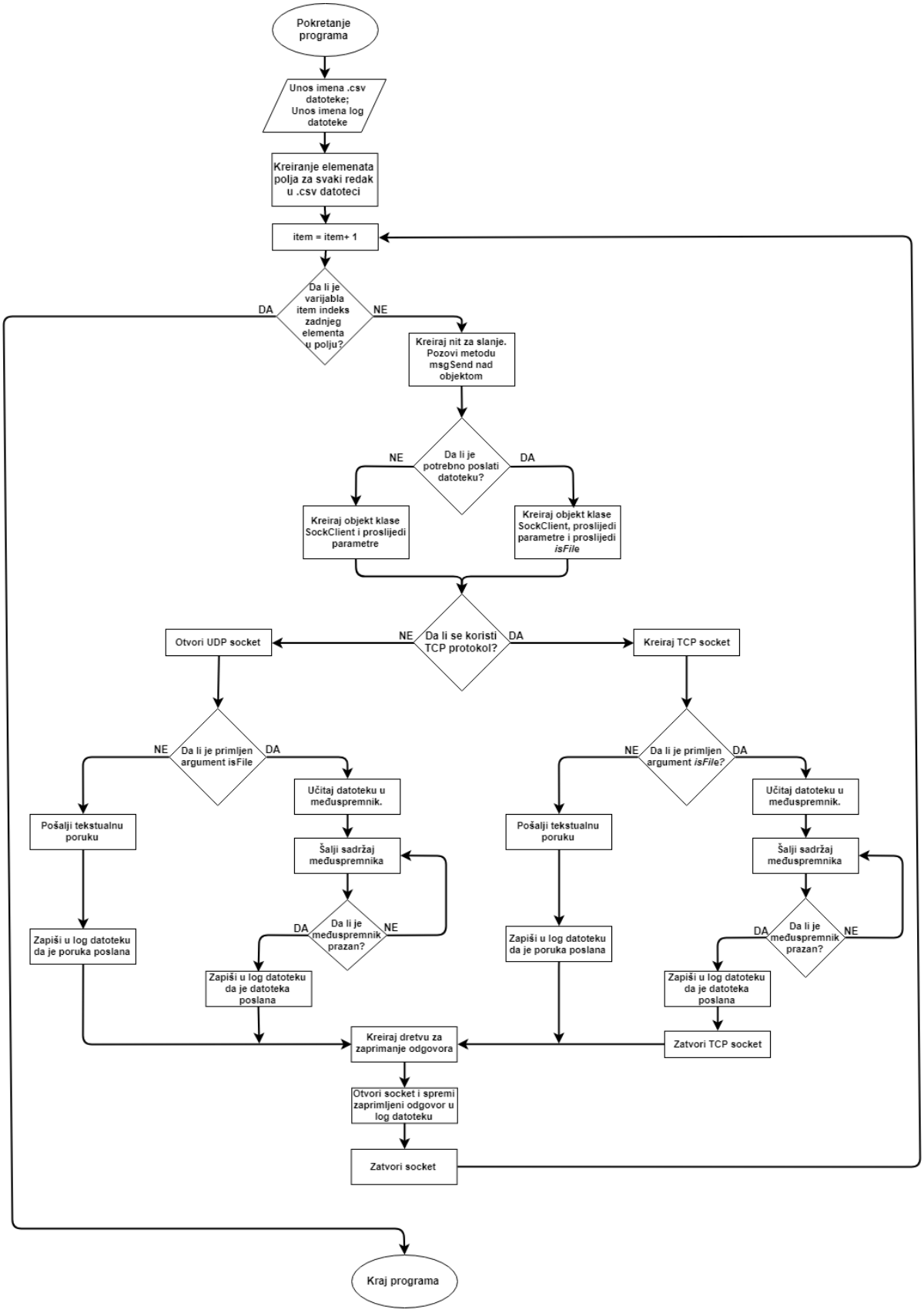
Nakon opisa primarne aplikacije u ovom završnom radu, slijedi prikaz koda i dijagrama toga aplikacije specifično namijenjene za korištenje UDP protokola. Iako su dijelovi koda slični ili identični prošloj aplikaciji, postoji jedna velika razlika prilikom korištenja UDP *socketa*. Kod ove aplikacije, jednom kad se UDP socket otvori, on se ne zatvara, već korištenjem dretvi on i dalje sluša odgovore od strane primatelja inicijalne poruke. Također, za svako sljedeće slanje, koristi se taj isti socket.

Glavna prednost ove aplikacije u odnosu na prethodnu je ta što može zaprimiti više odgovora od strane primatelja, te se sve šalje kroz isti socket, što znači da nije potrebno otvarati socket za svako novo slanje.

Sam početak programa je identičan prethodnom: korisnik unosi ime .csv datoteke koju koristi za testiranje i ime log datoteke u koju zapisuje dodatne informacije, zatim slijedi pozivanje funkcije *inputData()* koja retke iz .csv datoteke sprema u polje, potom slijedi prolazak kroz svaki element polja i provjera da li je potrebno poslati datoteku - funkcija *checkString()*. Nakon inicijalnih radnji, slijedi spremanje elemenata polja u datoteke, te nekoliko provjera.

Prva od navedenih provjera je da li se radi o UDP konekciji. Ako je korisnik u .csv datoteci naveo korištenje TCP protokola, program prekida s radom i upozorava korisnika da mora napraviti izmjene u .csv datoteci. Sljedeća provjera je da li postoji UDP socket. Ova provjera temelji se na inicijalnom postavljanju varijable *udpOpen* na vrijednost *False*, jer prilikom prvog prolaska kroz petlju UDP socket nije kreiran. To znači da se prilikom prvog prolaska kreira socket, a varijabla *udpOpen* se postavlja na *True*, te ne dolazi do novog kreiranja socketa prilikom ostvarivanja preostalih konekcija definiranih u datoteci. Zatim slijedi provjera slanja datoteke ili obične poruke, koja je identična prethodnom programu. Nakon te provjere slijedi slanje poruke.

Dio programa koji slijedi zapravo je ključna funkcionalnost i najveća prednost nad prethodim programom. Provjerava se da li u globalnom prostoru postoji vrijednost *endlessListener*. Metoda *globals()* omogućava provjeru svih varijabli, metoda, klasa itd. u globalnom djelokrugu programa, odnosno u globalnoj simboličkoj tablici. Kod prvog prolaska kroz petlju, prilikom dolaska na ovaj upit, nigdje nije definirana varijabla *endlessListener*, pa tako ovaj upit vraća *False*. Tada slijedi postavljanje varijable *endlessListener*, koja je zapravo dretva unutar koje se izvršava funkcija *msgRecieve* kojoj se prosljeđuje parametar *udp_socket*. Sada se unutar globale simboličke tablice pojavljuje vrijednost *endlessListener*, pa prilikom sljedećeg prolaska kroz petlju neće biti potrebe ze kreiranjem nove dretve koja poziva funkciju msgRecieve. Sama funkcija identična je istoimenoj metodi iz prethodnog programa, a ona služi za zaprimanje odgovora nakon uspješnog slanja poruke. Jedina i najveća razlika je ta što se ova funkcija vrti u beskonačnoj petlji, pa tako može zaprimiti više odgovora i koristi se za zaprimanje svih odgovora na uspostavljene konekcije. Na slici 15 prikazan je dijagram toka aplikacije.



Slika 14. Dijagram tijeka TCP-UDP aplikacije
Izvor: Izradio autor

3.2.1. Funkcija msgRecieve

Funkcija prikazana u Programski kod 11 gotovo je identična metodi msgRecieve iz Prethodnog programa. Razlika je u tome što se ovdje koristi while petlja koja se vrtu u beskonačnost jer je kao uvjet postavljena provjera ugrađene konstante *True*. To govori da se petlja mora vrtjeti sve dok je vrijednost ugrađene konstante *True* jednaka *True*, a to je zauvijek.

```
def msgRecieve(socket):
      print ('Opening UDP listener for responses...')
      while True:
            repaly = socket.recv(1024).decode()
            print("Server replied: %s" %(repaly))
            dataLog("RECIEVED", repaly)
```

Programski kod 11: funkcija msgRecieve

Izvor: Izradio autor

3.2.2. Prikaz ključnih dijelova programskog koda

Programski kod 12 prikazuje dijelove koda koji su drugačiji u odnosu na prethodnu aplikaciju, odnosno izostavlja prikaz identičnih fukncija i procesa samog slanja poruke. Prikazana je provjera da li se radi o slanju putem UDP protokola, te izlazak iz programa u suprotom slučaju.

Iz isječka koda također je vidljivo da se na početku postavlja varijabla udpOpen na vrijednost False, kako bi se prilikom prvog prolaska kroz petlju „okinuo“ dio koda koji kreira socket, te da bi spriječili kreiranje novih socketa za sljedeća slanja. Taj dio vidljiv je kod upita if udpOpen == False. Klučni dio oko korištenja dretvi, koji je ranije opisan, nalazi se na samom dnu prikaza.

```
udpOpen = False
dataArray = inputData()
for item in dataArray[1:]:
      ...
      if protocol.lower() != 'udp':
            print ("Wrong protocol, must use UDP! Edit csv file and try
            again")
            sys.exit()
      else:
            if udpOpen == False:
                  udp_socket = socket.socket(socket.AF_INET,
                  socket.SOCK_DGRAM)
                  udpOpen = True
            if isFile:
                  ...
            else:
                  ...
      if 'endlessListener' in globals():
            if endlessListener.isAlive():
      else:
            print ("Starting listener thread:")
            endlessListener = threading.Thread(target=msgRecieve,
            args=(udp_socket,))
            endlessListener.start()
      print ("sleeping")
      time.sleep(delayTime)
```

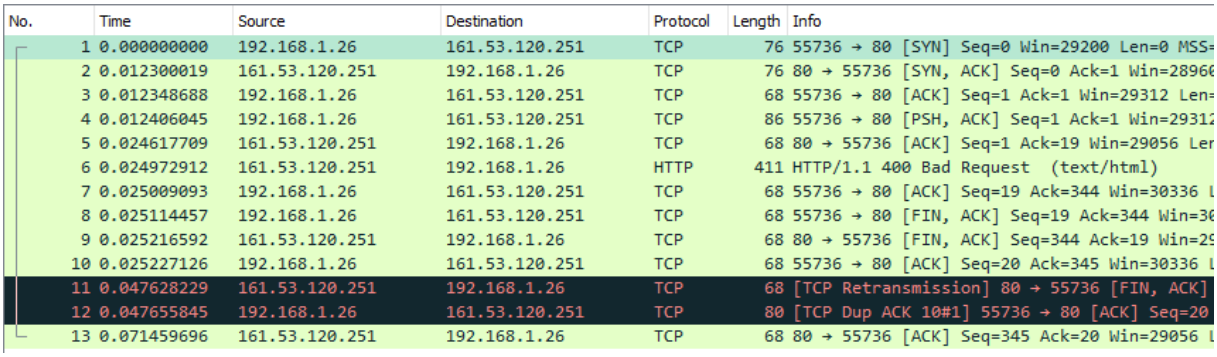
Programski kod 12: Isječci iz programa za UDP

Izvor: Izradio autor

4. Prikaz prometa – Wireshark

Slike u ovom poglavlju prikazuju pakete koje generiraju programi izrađeni u ovom završnom radu, te aplikacije koje zaprimaju poruke, odnosno datoteke. Za snimanje prometa i komunikacije korišten je alat Wireshark.

Slika 15 prikazuje promet aplikacije koja koristi TCP protokol. Iz slike je vidljivo da se paketi šalju na IP adresu 161.53.120.251 iz akoje se nalazi web stranica www.foi.hr



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.26	161.53.120.251	TCP	76	55736 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=
2	0.012300019	161.53.120.251	192.168.1.26	TCP	76	80 → 55736 [SYN, ACK] Seq=0 Ack=1 Win=28960
3	0.012348688	192.168.1.26	161.53.120.251	TCP	68	55736 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=
4	0.012406045	192.168.1.26	161.53.120.251	TCP	86	55736 → 80 [PSH, ACK] Seq=1 Ack=1 Win=2931
5	0.024617709	161.53.120.251	192.168.1.26	TCP	68	80 → 55736 [ACK] Seq=1 Ack=19 Win=29056 Le
6	0.024972912	161.53.120.251	192.168.1.26	HTTP	411	HTTP/1.1 400 Bad Request (text/html)
7	0.025009093	192.168.1.26	161.53.120.251	TCP	68	55736 → 80 [ACK] Seq=19 Ack=344 Win=30336 L
8	0.025114457	192.168.1.26	161.53.120.251	TCP	68	55736 → 80 [FIN, ACK] Seq=19 Ack=344 Win=3
9	0.025216592	161.53.120.251	192.168.1.26	TCP	68	80 → 55736 [FIN, ACK] Seq=344 Ack=19 Win=2
10	0.025227126	192.168.1.26	161.53.120.251	TCP	68	55736 → 80 [ACK] Seq=20 Ack=345 Win=30336 L
11	0.047628229	161.53.120.251	192.168.1.26	TCP	68	[TCP Retransmission] 80 → 55736 [FIN, ACK]
12	0.047655845	192.168.1.26	161.53.120.251	TCP	80	[TCP Dup ACK 10#1] 55736 → 80 [ACK] Seq=20
13	0.071459696	161.53.120.251	192.168.1.26	TCP	68	80 → 55736 [ACK] Seq=345 Ack=20 Win=29056 L

Slika 15. TCP stream prema www.foi.hr
Izvor: Pomoć mentora

Na slici 16 prikazan je UDP promet prema lokalnom poslužitelju na adresi 127.0.0.7 na portu 4444.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.7	UDP	57	45128 → 4444 Len=13
2	0.005165379	127.0.0.7	127.0.0.1	UDP	67	4444 → 45128 Len=23
3	0.005665906	127.0.0.1	127.0.0.7	UDP	57	38797 → 4444 Len=13
4	0.015788605	127.0.0.7	127.0.0.1	UDP	67	4444 → 38797 Len=23
5	0.016270369	127.0.0.1	127.0.0.7	UDP	57	43789 → 4444 Len=13
6	0.036544657	127.0.0.7	127.0.0.1	UDP	67	4444 → 43789 Len=23
7	0.037081419	127.0.0.1	127.0.0.7	UDP	57	58564 → 4444 Len=13
8	0.057196962	127.0.0.7	127.0.0.1	UDP	67	4444 → 58564 Len=23
9	0.057782234	127.0.0.1	127.0.0.7	UDP	57	56726 → 4444 Len=13
10	0.077687121	127.0.0.7	127.0.0.1	UDP	67	4444 → 56726 Len=23

Slika 16. UDP promet prema lokalnom poslužitelju
Izvor: Pomoć mentora

5. Zaključak

Svakodnevna i neprekidna potreba za povezivanjem elektroničkih uređaja, pojava „interneta stvari“ i neprestano širenje računalnih mreža, zahtjeva adekvatne alate i načine testiranja novoimplementiranih rješenja. Svaki sustav potrebno je detaljno testirati prije puštanja u produkciju, pa tako i aplikacije koje komuniciraju putem računalnih mreža. Ova aplikacija predstavlja najosnovniji test za te aplikacija, a to je provjera ima li konekcije između strana koje žele uspostaviti komunikaciju, je li odredište u mogućnosti primiti podatke koji su joj namijenjeni, vremenski slijed poruka te sadržaj komunikacije.

Završni rad „Program za testiranje mrežnih aplikacija“ dotaknuo se najvažnijih osnova računalnih mreža. Kroz uvodna poglavlja, objašnjeno je na koji način komuniciraju računala, te na koji način mrežne aplikacije razmjenjuju podatke, te koji protokoli i mehanizmi omogućavaju tu komunikaciju. Primjerom aplikacije za slanje podataka putem računalne mreže, prikazan je objektno-orijentirani pristup programskom jeziku Python, te primjena biblioteke socket.

Ovaj jednostavni program iz prve ruke prikazuje prednosti programskog jezika Python: lako čitljiv kod, kratkoća samog programa, jednostavna i intuitivna sintaksa, ali nadasve bogat funkcijama i mogućnostima

6. Literatura

Knjige:

[1] Computer Networks, fifth edition – Andrw S. Tanenbaum, David J. Wetherall, Pearson Education .Inc, 2011

[2] Računarske mreže, prijevod četvrtog izdanja – Andrw S. Tanenbaum, Mikro knjiga Beograd, 2005

Web stranice:

[1] https://python.swaroopch.com/about_python.html, pristupljeno 17.5.2018

[2] https://www.w3.org/Protocols/HTTP/1.1/rfc2616bis/draft-lafon-rfc2616bis-03.html#RFC2324, pristupljeno 23.04.2018

[3] https://fossbytes.com/presentation-layer-of-osi-model/, pristupljeno 23.04.2018

[4] https://informatika.buzdo.com/s916-internet-tcp-udp-port-socket.htm, pristupljeno 24.4.2018

[5] https://tools.ietf.org/html/rfc793, pristupljeno 9.5.2018

[6] http://www.zemris.fer.hr/~sgros/s tuff/mr/05.pdf, pristupljeno 24.04.2018

[7] https://wiki.mikrotik.com/wiki/Manual:Connection_oriented_communication_(TCP/IP), pristupljeno 7.5.2018

[8] https://sysportal.carnet.hr/node/352, pristupljeno 17.4.2018

[9] http://mreze.layer-x.com/s030100-0.html, pritupljeno 19.04.2018

[10] https://tools.ietf.org/html/rfc791, pristupljeno 19.04.2018

[11] https://tools.ietf.org/html/rfc1918, pristupljeno 19.4.2018

[12] https://www.juniper.net/documentation/en_US/junos/topics/concept/mac-qfx-series-understanding.html, pristupljeno 17.04.2018

[13] http://www.linfo.org/csma_cd.html, pristupljeno 17.04.2018.

7. Popis slika

8. Popis tablica

9. Popis programskih kodova