Mrežna igra Potapanje brodova u programskom jeziku C++

Kristijan, Perković

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Kristijan Perković

Mrežna igra Potapanje brodova u programskom jeziku C++

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Kristijan Perković

Matični broj: 44097/15-R

Studij: Informacijski sustavi

Mrežna igra Potapanje brodova u programskom jeziku C++

ZAVRŠNI RAD

Mentor:

prof. dr. sc. Danijel Radošević

Varaždin, srpanj 2018.

Kristijan Perković

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima, osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovom završnom radu bavit ću se obradom svih ključnih dijelova koji su potrebni za razumijevanje izrade mrežne aplikacije. Uređaji u aplikaciji će se povezivati pomoću IP adresa pa ću iz tog razloga reći nešto o internetu, kako bih pokazao o kojem se točno dijelu interneta radi te koji dio zapravo trebam programirati. Kako bi se povezala dva računala, potrebni su *socketi* (utičnice), pomoću kojih će se uspostaviti veza pa je potrebno znati nešto o njima i njihovim vrstama. No, kako se ovdje radi o mrežnoj aplikaciji, objasnit ću još što su to mrežne aplikacije i navesti njihove vrste te primjere za svaku vrstu. Nakon objašnjene teorije, krenut ću na dio programiranja same mrežne aplikacije i navesti probleme koji se mogu pojaviti prilikom programiranja.

Ključne riječi: klijent-poslužitelj, *socket*, mrežna igra, internet, NAT, protokoli.

Sadržaj

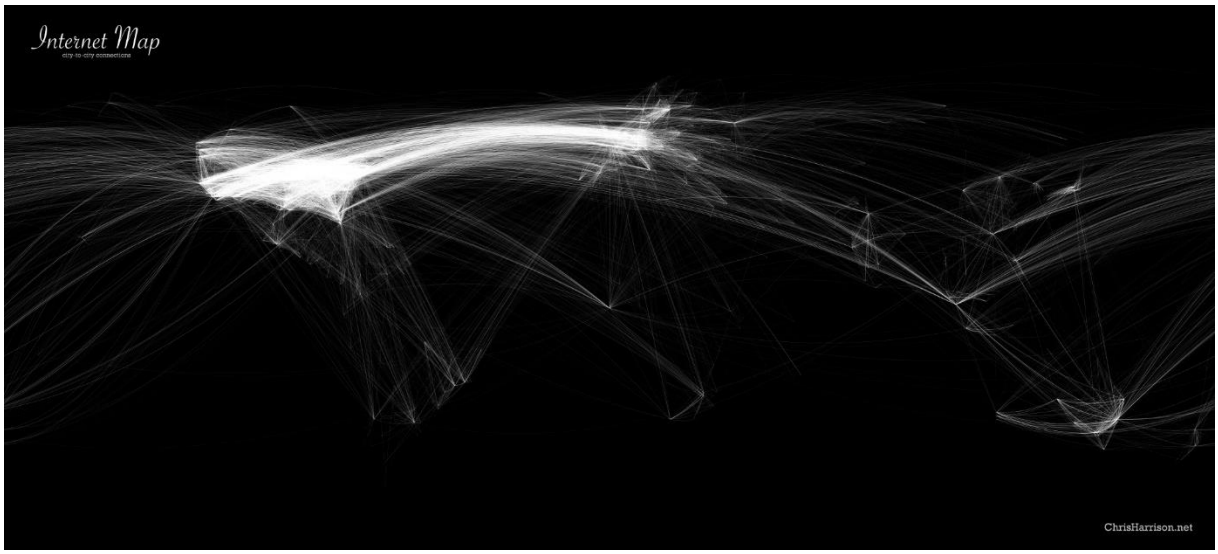
1. Uvod

U današnje vrijeme internet se koristi svakodnevno. Ako želimo pregledati novosti, posjetit ćemo *web* stranice kao što su dnevnik.hr, gol.hr ili jutarnji.hr. Prilikom posjete liječniku, više se ne izdaje papirnati recept za lijek, nego odlazimo u ljekarnu i dajemo svoju zdravstvenu iskaznicu. Na temelju podataka s iskaznice, ljekarnik pronalazi naše podatke te nam daje preporučeni lijek. Iz ovog primjera se može zaključiti kako su liječnik i ljekarnik umreženi. No to je samo jedan od brojnih primjera umrežavanja.

Umrežavanje mladih danas se svodi na korištenje aplikacija poput Facebooka, Instagrama ili Vibera, koje omogućavaju komunikaciju ljudi putem interneta. Iako su to aplikacije koje koristi gotovo svaka mlada osoba, uz njih se koriste i mrežne igre. Mrežne igre su popularne u današnjem svijetu te su skoro sve igre koje izlaze na tržištu igara, mrežne. Iz tog ću razloga u ovom radu, govorit o onom dijelu interneta koji je potreban kako bi se razumio sam rad, o mrežnim aplikacijama, objasnit ću mrežnu arhitekturu klijent-poslužitelj i kako se povezuju dva računala pomoću socketa, programirat ću mrežnu igru u programskom jeziku C++ i prikazati rad mrežne aplikacije „Potapanje brodova“. Još ću reći nešto o samim problemima prilikom programiranja mrežnih aplikacija i navesti nekoliko rješenja, pomoću kojih se mogu riješiti navedeni problemi.

2. Internet

Kako bi lakše razumjeli što je internet, navest ću samu definiciju interneta. Kermek navodi da je „ Internet fizička mreža koja povezuje milijune računala koristeći isti protokol za dijeljenje/prijenos informacija“ [14]. Neke osobe internet nazivaju mrežom svih mreža. Proučit ću malo bolje prvu definiciju. Prvi pojam na koji nailazimo je „Fizička mreža“. Od čega se sastoji ta fizička mreža? Dakle, fizička mreža se sastoji od računala koja su povezana komunikacijskim poveznicama (optika, bakreni vodovi, radiovalovi, sateliti) te prespojnici i usmjernici (prespajanje paketa). Drugi ključni pojam bi bio „Protokol“. Ivković navodi da „ protokoli određuju format i redoslijed poruka koje se šalju i primaju te akcije koje se poduzimaju prilikom slanja i primanja poruka“. [8]



Slika 1. Internet svjetske veze grad-grad [2]

Slika 1 prikazuje kako su povezani gradovi u svijetu preko interneta. Pogleda li se slika malo bolje, uočava se kako je najveći broj poveznica između europskih i sjevernoameričkih gradova. No, ako se zanemari broj poveznica i samo pogleda sliku kao sliku, može se reći kako komunikacijske poveznice interneta prekrivaju sve kontinente.

2.1. Povijest Interneta

1969. godine Advanced Research Project Agency osnovala ARPANET na osnovu kojeg je kasnije razvijen internet. Prva četiri slova dolaze od samog naziva agencije za napredna istraživanja, dok nastavak NET predstavlja mrežu. Iste godine 29.10. ostvareno je prvo povezivanje. Povezala su se četiri računala sa četiri sveučilišta. Povezana sveučilišta bila su UCLA, UCSB, SRI (Stanford) i Utah. Sedamdesetih godina veličina interneta se udvostručavala svake godine. Jedan od razloga brzog rasta broja umreženih računala je decentralizacija, koja je pojednostavila dodavanje novih računala u mrežu. Osamdesetih

godina vlada Sjedinjenih Američkih Država uzela je veću ulogu u razvoju interneta te je 1986. godine kreiran NSFNET za akademska istraživanja, dok je ARPANET ostavljen za vojna i vladina računala. Do devedesetih godina uspješno su povezana sva sveučilišta i koledži, a kontrola nad internetom je prenesena na neprofitne organizacije (1992. godina). [14]

2.2. Struktura interneta

Glavno pitanje koje se postavlja jest: „Kako se može znati koje je koje računalo?“. Odgovor na ovo pitanje je jednostavan te glasi da svako računalo na mreži ima svoju vlastitu IP adresu koja ga jednoznačno određuje. IP adrese mogu biti u dvije verzije, IPv4 ili IPv6. U početku se koristila samo IP verzije 4, ali kako raste broj stanovnika i opadaju cijene računalne opreme, broj računala u mreži se povećava te samim time IPv4 ne može generirati dovoljan broj IP adresa. Za primjer, ako se uzme IP adresa 192.168.178.43, vidi se kako se IP adresa sastoji od 4 skupine brojeva, što predstavlja IPv4. Svaki broj može imati vrijednost od 0 do 255, što odgovara 2^8, odnosno 256 različitih vrijednosti. Daljnja računica o broju dostupnih adresa je jednostavna . Svaka adresa se sastoji od $8 \cdot 4 = 32$ bita, te je moguće generirati $2^{32} = 4\,294\,967\,296$ različitih IP adresa. Ako se uzme u obzir da je broj stanovnika na zemlji 7.49 milijardi, može se zaključiti kako je broj stanovnika veći od samog broja IP adresa. Iz tog razloga se uvela nova verzija IPv6.

Nakon što smo uspjeli odrediti koje je koje računalo, između njih se može poslati poruka. Poruka koja se šalje između računala cijepa se u pakete. Svakom od paketa pridružuje se izvorišna i odredišna IP adresa te broj paketa. Paketi se šalju neovisno. [14]

Ranije sam naveo kako su usmjernici dio interneta. Njihova uloga je da usmjeravaju pakete od izvorišta do odredišta. Taj zadatak obavljaju na način da pročitaju odredišnu IP adresu te proslijede paket na drugi usmjernik koji je bliži odredištu. Kada paketi stignu na odredište, spajaju se u početnu poruku. Tada do izražaja dolazi broj paketa, koji određuje redoslijed paketa za spajanje, te se ujedno provjerava koji je paket stigao na odredište, a koji nije. [14]

2.3.Slojevitost protokola

S obzirom na veličinu mreže, koristi se velik broj različitih tehnologija i protokola. Iz tog se razloga mreže mogu podijeliti na lokalne, takozvane LAN, i na mreže dalekog dosega, takozvane WAN mreže. Protokole možemo podijeliti prema slojevima u kojima se nalaze. [8]

Navest ću dva modela, Internetov model (TCP/IP model) i ISO/OSI referentni model, te napraviti usporedbu slojeva navedena dva modela koja je prikazana u tablici ispod. Usporedba je napravljena prema izvoru [8].

ISO/OSI model	TCP/IP
7. Aplikacijski sloj	5. Aplikacijski sloj
6. Sloj prikaza	
5. Sloj razgovora	
4. Transportni sloj	4. Transportni sloj
3. Mrežni sloj	3. Mrežni sloj
2. Poveznički sloj	2. Poveznički sloj
1. Fizički sloj	1. Fizički sloj

Ivković navodi da fizički sloj predstavlja bitove koji se prenose putem komunikacijskih poveznica.[8]

Ivković navodi da poveznički sloj služi za prijenos podataka od jednog mrežnog uređaja do drugog mrežnog uređaja, dok Milak navodi da služi i za otkrivanje i ispravljanje grešaka na fizičkom sloju. Na ovom sloju se koriste protokoli kao što su Ethernet, IEEE 802.11, PPP.[8][13]

Mrežni sloj služi za dostavljanje paketa od računala pošiljatelja do računala primatelja. Na strani pošiljatelja, segment se učahuruje u datagram. Veliki datagram se cijepa na više manjih datagrama. Glavnim funkcijama mrežnog sloja, usmjeravanjem i prosljeđivanjem, paketi se prenose kroz mrežu. Usmjeravanjem se određuje putanja kojom će paket proći od izvora do odredišta. Prosljeđivanjem se paket premješta s ulaza na odgovarajući izlaz na usmjerniku. Kada paket dođe do računala ponovno se sastavlja u veliki datagram te prosljeđuje transportnom sloju. U ovom sloju koriste se protokoli: IP, RIP, OSPF, BGP, ICMP.[8][10]

Transportni sloj prenosi podatke od jednog procesa do drugog procesa te tako omogućava logičku komunikaciju između procesa aplikacija, koje se izvode na različitim računalima. Poruka koja se šalje, dijeli se na manje pakete koji se nazivaju segmenti, koji se u mrežnom sloju kasnije učahuruju u datagrame. Strana koja prima segmente sastavlja segmente u poruku koju prosljeđuje aplikacijskom sloju. Aplikacije na raspolaganju imaju dva protokola: UDP i TCP. Kao što postoje dvije vrste protokola, postoje i dvije vrste socketa pomoću kojih se razmjenjuju podaci. Svaki socket određen je uređenom četvorkom, koja se sastoji od izvorišne IP adrese, broja izvorišnog porta, odredišne IP adrese i broja odredišnog porta. Korištenje TCP protokola dobro je, ako ne smijemo izgubiti niti jedan paket koji se šalje, jer TCP će zatražiti ponovno slanje zagubljenog paketa. Takav je pristup pogodan za aplikacije gdje nije bitna brzina, nego točnost podataka. S druge strane, UDP nema kontrolu je li se koji

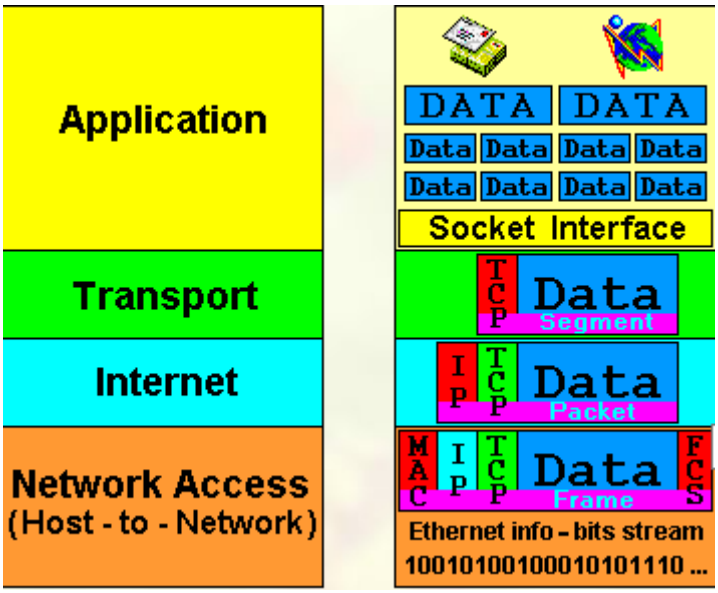
paket zagubio, ali je zato pogodan za multimedijalne aplikacije, gdje nije bitno je li se zagubio koji paket, već je bitna sama brzina komunikacije. Ako želimo pouzdan prijenos preko UDP-a, taj dio dodajemo u aplikacijskom sloju.[8][9]

Sloj razgovora služi za sinkronizaciju, provjeru stanja te za oporavak od pogrešaka u razmjeni podataka.[8]

Sloj prikaza omogućava aplikacijama da interpretiraju značenje podataka, na primjer enkripcija, kodiranje ili kompresija.[8]

U TCP/IP modelu se ne nalazi sloj sesije i prezentacijski sloj, te ako se žele dodati moraju se implementirati u sklopu aplikacijskog sloja.[8]

Aplikacijski sloj služi za prikaz podataka ljudima u razumljivom obliku te koristi protokole za slanje i primanje podataka kroz internet. U svrhu razmjene podataka, nižem sloju se isporučuje broj porta i IP adresa, kako bi se znalo kamo treba paket dospjeti. Također, aplikacijski sloj treba odrediti koji će se protokol koristiti u transportnom sloju. U transportnom sloju se mogu koristiti dva protokola: TCP i UDP. U primjeru mrežne aplikacije koristit će se UDP. Na slici 2 se može vidjeti kako se Socket Interface nalazi između aplikacijskog i transportnog sloja, što je glavna tema ovog rada.



Slika 2. Enkapsulacija [1]

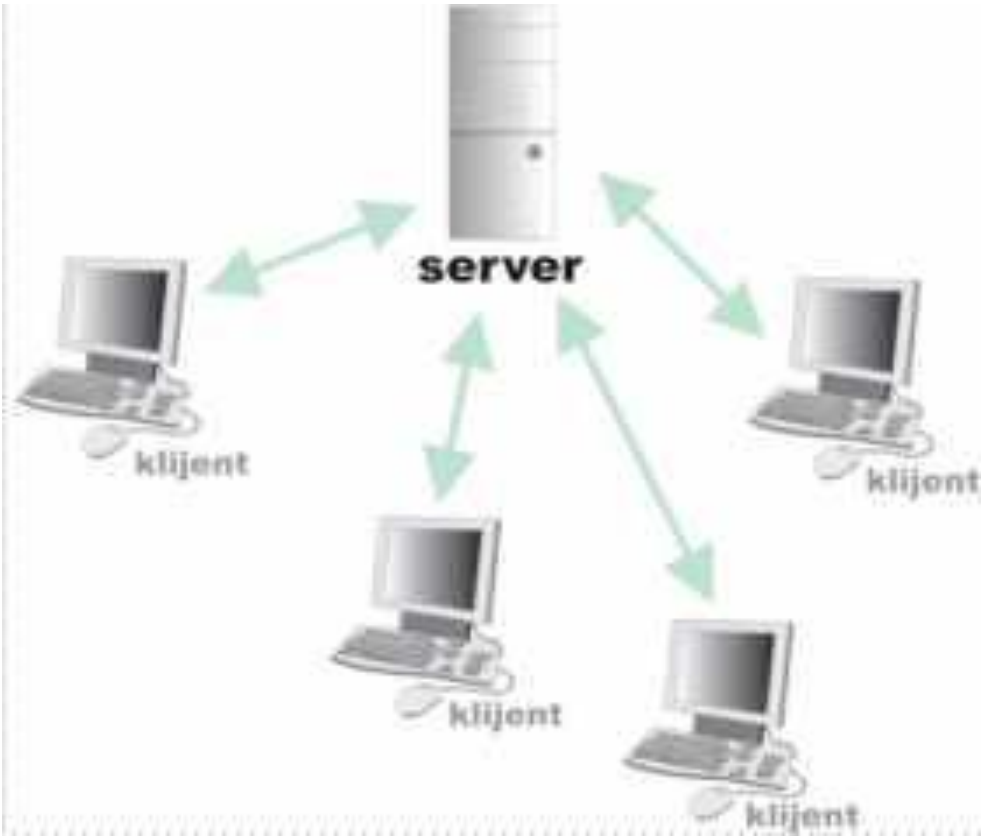
3. Mrežne aplikacije

Kao što sam naveo u uvodu, mrežne se aplikacije sve više koriste u svakodnevnom životu. Predmet ovog rada jesu prvenstveno mrežne igre. Kao i u prethodnom poglavlju,

krećem sa samom definicijom. Androić navodi da „mrežne aplikacije su aplikacije koje su pokrenute od strane različitih računala spojenih na mrežu“ [3]. Odnosno, Radovan navodi da su „mrežne aplikacije uglavnom softverski sustavi koji rade na raznim računalima, pri čemu razmjenjuju informacijske sadržaje sa sustavima na drugim računalima, koristeći pri tome usluge prijenosa koje im pruža računalna mreža“ [4]. Mrežne aplikacije se izvršavaju na završnim čvorovima mreže, koje nazivamo domaćinima, odnosno na računalima na kojima se nalazi aplikacija. Prema osnovnoj strukturi mrežne aplikacije dijelimo na dvije skupine. Jedna skupina su aplikacije klijent-poslužitelj, a druga skupina su *peer-to-peer* (P2P). Tradicionalne mrežne aplikacije su tipa klijent-poslužitelj te takve aplikacije, kao što i sam naziv govori, imaju klijentski i poslužiteljski dio. Klijentski dio šalje zahtjeve, dok poslužiteljski dio odgovara na te zahtjeve. Klijentski dio aplikacije obično se nalazi na osobnim računalima, ali u novije vrijeme i na pametnim telefonima te ne mora uvijek biti aktivan. Poslužiteljski dio mrežne aplikacije obično se nalazi na većim računalima, koja su stalno aktivna, i povezan na mrežu te imaju stalnu IP adresu. Ta IP adresa treba biti poznata klijentskom dijelu aplikacije, kako bi mogao stupiti u komunikaciju sa serverskim dijelom te mu uputiti zahtjev. Kod nekih mrežnih usluga odvija se intenzivan promet i prenose se velike količine podataka. Takve aplikacije su sustavi za distribuciju video sadržaja, sustavi društvenog umrežavanja (poznatije pod nazivom društvene mreže), tražilice i sustavi za internetsko trgovanje. Kod mrežnih aplikacija, gdje je intenzivan promet, koriste se grozdovi (clusters) domaćina, odnosno poslužitelj je podijeljen na nekoliko tisuća fizičkih domaćina, koji su raspoređeni po svijetu, na kojima rade kopije istog poslužitelja koji pruža neku određenu uslugu. Na takvim mrežnim aplikacijama poslužiteljski dio prima veliku količinu podataka (zahtjeva), koje treba obraditi, tako da jedan fizički poslužiteljski dio ne bi mogao udovoljiti svim zahtjevima, koji istodobno dolaze od velikog broja klijenata.[4]

Kao primjer mrežne aplikacije tipa klijent-poslužitelj, navodim igru „Fortnite“. U navedenoj igri, svaki igrač mora imati na svome računalu instaliranu klijentsku aplikaciju, pomoću koje se povezuje na poslužitelj. Igrači odabiru kakvu će vrstu igre igrati, te ih na temelju tih podataka, poslužitelj povezuje s ostalim igračima, koji su odabrali istu vrstu igre. Igrači ne mogu komunicirati izravno jedan s drugim, već preko poslužitelja. Kod mrežnih aplikacija tipa *peer-to-peer* (P2P), klijent i poslužitelj su samo trenutne uloge istog sustava u nekoj mrežnoj komunikaciji. Domaćini na kojima rade mrežne aplikacije tipa P2P su obično računala krajnjih korisnika. Aplikacije tipa P2P uspostavljaju izravnu komunikaciju i razmjenu sadržaja između istovrsnih sustava. Aplikacije ovog tipa pogodne su za razmjenu datoteka između velikog broja korisnika. Kod P2P aplikacija, na svim domaćinima koji sudjeluju u procesu međusobne razmjene sadržaja, nalazi se jednak softver i svi domaćini sudjeluju u procesu međusobne razmjene. Kao primjer P2P aplikacije, navodim aplikaciju BitTorrent, koja

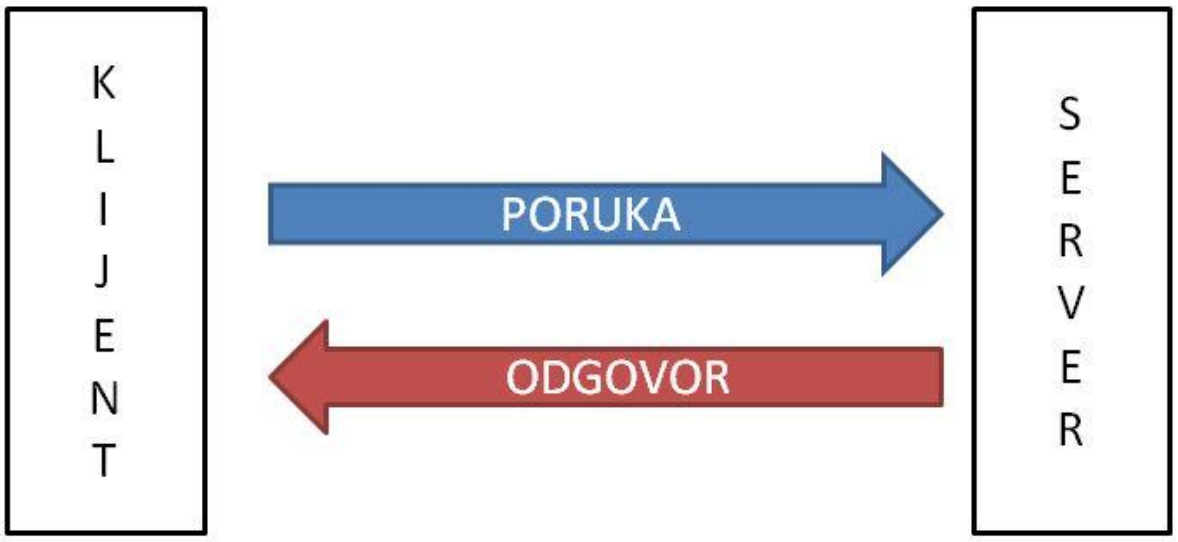
služi za distribuiranje datoteka. Svi korisnici mogu preuzeti aplikaciju na službenoj stranici BitTorrenta (jednak softver na svim domaćinima).[4]



Slika 3. Arhitektura klijent- poslužitelj [5]

Slika 3 prikazuje složenu arhitekturu klijent-poslužitelj, gdje poslužitelj komunicira s više klijenata, što odgovara načinu prikaza igre „Fortnite“. Klijentska računala su računala igrača koja se povezuju na server. U ovom radu ću napraviti jednostavnu mrežnu aplikaciju

„Potapanje brodova“, gdje komuniciraju dva računala od kojeg je jedno računalo poslužitelj, dok je drugo računalo klijent.



Slika 4. Jednostavna arhitektura klijent-poslužitelj [6]

Slika 4 prikazuje arhitekturu primjera mrežne aplikacije, koja će se prikazati kasnije u radu. U komunikaciji sudjeluje jedan klijent i jedan poslužitelj koji međusobno komuniciraju.

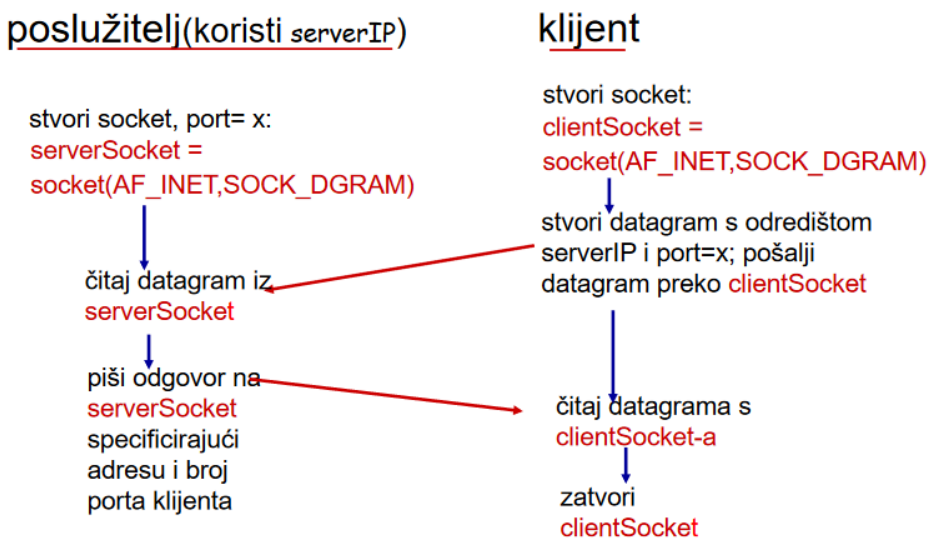
4. Socket

Nakon što sam napisao i objasnio koji dio interneta mi treba za sam rad i koju ću to arhitekturu implementirati, sada ću objasniti sockete i krenuti s implementacijom same mrežne igre.

Kao i u svakom prethodnom poglavlju, ovdje ću također započeti s definicijom. Ivković navodi da su „Socket vrata između aplikacijskog procesa i transportnog protokola“ [12]. Ovu definiciju je lako odrediti i prema slici 2 iz poglavlja Slojevitost protokola. Također sam spomenuo kako postoje dvije vrste socketa: TCP i UDP.

4.1. UDP

Kod UDP protokola se ne uspostavlja veza prije slanja paketa, već pošiljatelj prilikom slanja svakog paketa navodi IP adresu i broj porta primatelja. Primatelj iz primljenog paketa doznaje podatke o pošiljatelju, odnosno IP adresu i broj porta s kojega je primio paket. UDP ne pruža pouzdan prijenos podataka te se podaci mogu izgubiti ili doći u krivom redoslijedu. [12]

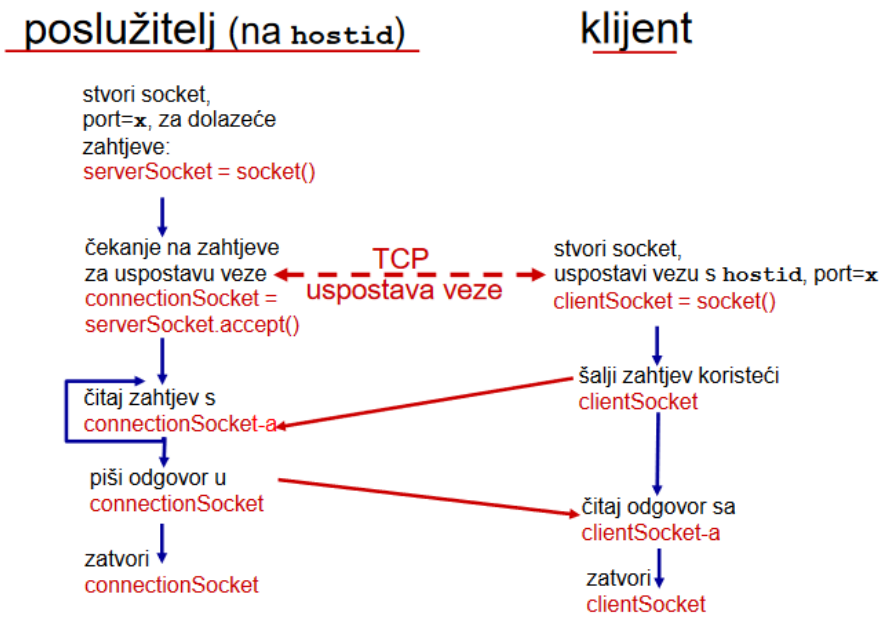


Slika 5. Prijenos podataka putem UDP socketa [12]

4.2. TCP

Kod socket programiranja s TCP-om, server treba biti pokrenut prije primanja paketa, kako bi stvorio socket, koji će služiti za čekanje „javljanja„ klijenta. Kako bi se klijent „javio“, poslužitelju mora stvoriti TCP socket, zadajući pri tome IP adresu i broj porta poslužitelja. Kada se stvori TCP socket, klijent uspostavlja TCP vezu s poslužiteljem. Tada je poslužitelj spreman za „javljanje“ klijenta, i kada mu se „javi“ poslužitelj, stvara novi socket, pomoću kojega će

komunicirati s klijentom. Takav pristup omogućuje poslužitelju da istovremeno može komunicirati s više različitih klijenata. [12]



Slika 6. Prijenos podataka putem TCP socketa [12]

4.3. Implementacija komunikacije putem UDP socketa

Prije samog početka programiranja u projekt sam uključio winsock2.h i povezao projekt s libws2_32.a. Bez navedenog headera i biblioteke ne bih mogao koristiti funkcije koje služe za stvaranje socketa, brisanje socketa, slanje paketa, primanje paketa te za ostale funkcije koje su potrebne za kreiranje mrežne aplikacije.

U programskom jeziku C++, kako bi se kreirala mrežna aplikacija, prvo je potrebno kreirati socket. To se postiže funkcijom `int socket(int domain, int type, int protocol)`. Parametri u ovoj funkciji određuju koji tip IP adrese će se koristiti, koja vrsta paketa se prenosi te koji protokol se koristi za prijenos.

```
opisnikUticnice=socket(AF_INET,SOCK_DGRAM,0);
```

Za prvi parametar se odabire AF_INET, koji definira kako se radi o IPv4, drugi parametar definira kako će se koristiti datagram, a kao treći parametar se može odabrati 0, što znači da se automatski odabire odgovarajući protokol na osnovu prva dva parametra. Odnosno, kao zadnji parametar se mogao navesti IPPROTO_UDP jer znamo da će se koristiti UDP.

Nakon kreiranja socketa, potrebno ga je povezati sa željenim portom. Port ćemo povezati sa socketom funkcijom bind().

```
int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
```

sockfd - identifikacijski broj socketa, koji se dobije kao povratna vrijednost funkcije socket(), odnosno u konkretnom slučaju, to je vrijednost koja je spremljena u varijabli opisnikUticnice.

my_addr - je kazaljka na strukturu podataka, koja sadrži polja za definiranje obitelji protokola, IPv4 i broj porta.

Navedena struktura:

```
struct sockaddr_in {
      short int sin_family; // obitelj protokola, AF_INET
      unsigned short int sin_port; // broj porta, 16 bitni broj
      struct in_addr sin_addr; // Internet adresa 32 bitni broj
      unsigned char sin_zero[8]; // polje vel. 8 bajtova
};
```

U konkretnom primjeru:

```
servAddr.sin_family=AF_INET;
servAddr.sin_addr.s_addr=inet_addr(IpAdresa.c_str());
servAddr.sin_port=htons(6000);
```

Kao zadnji parametar funkcije bind(), addrlen je veličina varijable na koju je prenesena kazaljka u prethodnom argumentu funkcije. Izgled funkcije u konkretnom primjeru:

```
rc=bind(opisnikUticnice,(struct sockaddr *) &servAddr,
sizeof(servAddr));
```

Nakon što je socket povezan s portom, mogu se slati i primati podaci. Za primanje podataka se koristi funkcija recvfrom(), dok se za slanje koristi funkcija sendto().

```
ssize_t recvfrom(int sockfd, void *buf, size_t len,  , struct sockaddr
*from, socklen_t *fromlen);
ssize_t sendto(int sockfd, const void *buf, size_tlen, int flags, const
struct sockaddr *to, socklen_t tolen);
```

Opis parametara funkcija:

- sockfd - identifikacijski broj *socketa,* koji se dobije kao povratna vrijednost funkcije socket().

- buff – kazaljka na međuspremnik u kojem su podaci koji se šalju sa sendto() funkcijom, odnosno međuspremnik u koji se zapisuju podaci koje prima recvfrom() funkcija.
- len – veličina podataka u bajtovima koji se nalaze u međuspremniku. Kod slanja sa sendto() se upisuje broj bajtova koji se šalje, a kod recvfrom() veličina međuspremnika.
- flags – zastavice, za potrebe ovog rada uvijek ću upisivati 0.
- from – kazaljka na strukturu u koju funkcija upisuje adresu i port izvora poruke.
- To - kazaljka na strukturu u koju se upisuje adresa i port odredišta.
- fromlen – veličina strukture u kojoj su zapisani adresa i port izvora, a upisuje ju funkcija.
- tolen – upisuje se veličina strukture u koju je zapisana adresa i port odredišta .

Konkretan primjer iz mrežne igre „Potapanja brodova“:

```
sendto(opisnikUticnice,meduspremnik,sizeof(meduspremnik),0,(struct
sockaddr*)&cliAddr,cliLen);

cliLen=sizeof(cliAddr);
n=recvfrom(opisnikUticnice,meduspremnik,MAX_MSG,0,(struct sockaddr
*)&cliAddr,&cliLen);
```

Ako se socket više ne koristi, može se pozvati funkciju close(), u koju se kao parametar prosljeđuje identifikacijski broj socketa.

Kod implementacije arhitekture klijent-server tipičan redoslijed poziva funkcija je:

Server	Klijent
socket()	socket()
bind()	
recvfrom()	sendto()
sendto()	recvfrom()
close()	close()

Cijelo poglavlje je pisano prema [11].

5. Mrežna aplikacija „Potapanje brodova“

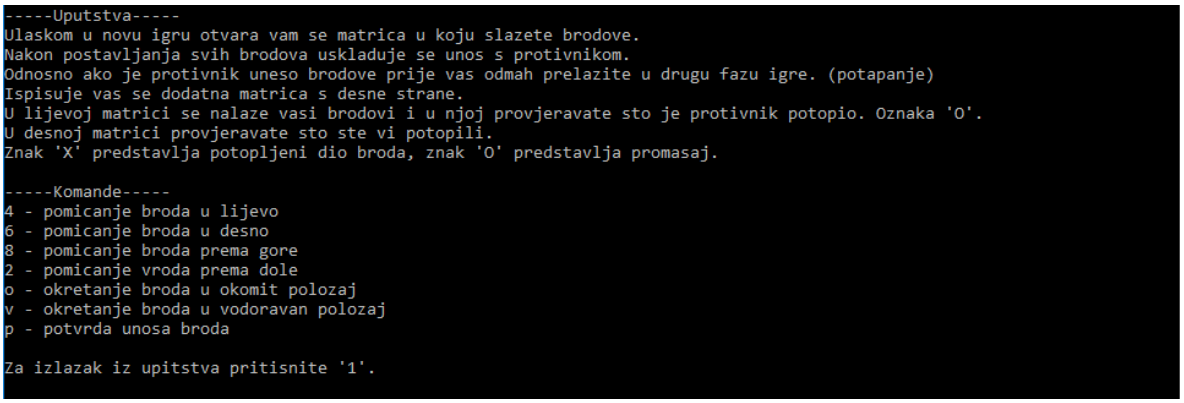
U prethodnim sam poglavljima govorio o 5 slojeva, odnosno o 7 slojeva, ovisno o modelu. No, ne trebam programirati sve slojeve kako bih napravio mrežnu aplikaciju, već trebam programirati samo aplikacijski sloj, dok svim ostalim slojevima upravlja operacijski sustav.

Mrežna igra „Potapanje brodova“ zamišljena je kao jednostavna mrežna aplikacija, koja ima arhitekturu klijent-poslužitelj. Iz ranije navedene teorije, zaključak je kako se različite aplikacije pokreću na računalima. Iako su izbornici jednaki i početak same igre izgleda jednako, u pozadini se „vrte“ različiti kodovi. Igra „Potapanje brodova“ je društvena igra koju igraju ljudi širom cijelog svijeta, ali na papiru. Nacrtaju se dvije matrice, a zatim se u jednoj od njih nacrtaju brodovi. Brodovi mogu biti duljine pet ćelija, četiri ćelije, dva broda duljine tri ćelije te jedan brod duljine dvije ćelije. Nakon toga igrači jedan drugoga ispituju nalazi li se na nekom polju dio broda, i ako se doista nalazi oni obilježavaju tu ćeliju znakom 'X', kako bi znali da se na tom polju nešto nalazi. Ako su pogodili polje, na kojem se nalazi dio broda, nastavljaju pogađati, a ako su pogriješili protivnik dobiva pravo ispitivati što se nalazi na nekom polju (u nekoj ćeliji). Igra završava kada jedan igrač uspije potopiti sve brodove svog protivnika.

Ova igra je implementirana u programskom jeziku C++ i započinje izbornikom koji je prikazan na slici 7.

Slika 7. Izbornik aplikacije

Odabere li se opcija 2, otvaraju se upute prikazana na slici 8.



Slika 8. Upute za igranje

Dok se nalazimo u uputama, pritiskom na tipku 1 ponovno se vraćamo u izbornik. Odabirom opcije 1, u izborniku se dobiva naša IP adresu, koju kažemo protivniku te unesemo njegovu IP adresu. Iako sam pokušao u potpunosti izbjeći komunikaciju između igrača, nisam uspio. Jedini razlog, zbog kojega se ovaj korak nalazi u igri, je dinamičko dodjeljivanje IP adrese.

Slika 9. Unos protivnikove IP adrese

Naša IP adresa se dohvaća pomoću funkcije:

```
string VratiIpAdresu(){

	string line;

	ifstream IPFile;

	int offset;

	char* search0 = "IPv4 Address. . . . . . . . . . . :";

	system("ipconfig > ip.txt");

	IPFile.open ("ip.txt");

	if(IPFile.is_open())

	{

	while(!IPFile.eof())

	{

	getline(IPFile,line);
```

```
if ((offset = line.find(search0, 0)) != string::npos)

{

line.erase(0,39);

IPFile.close();

return line;

}}}}
```

Pomoću funkcije `system("ipconfig > ip.txt");` izvršava se naredba ipconfig te se dobiveni rezultat sprema u tekstualnu datoteku ip.txt kao što je prikazano na slici 10.

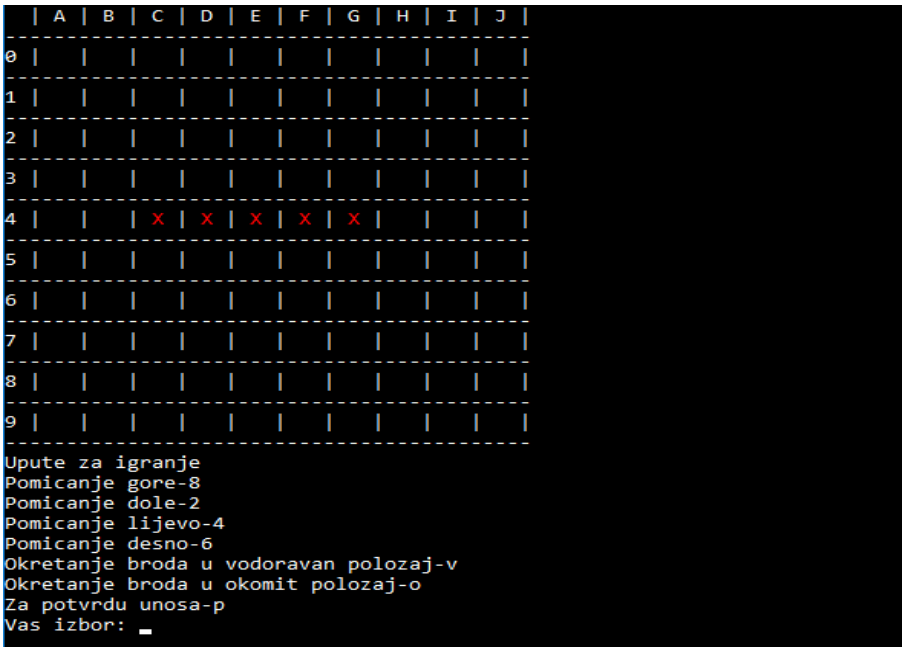
```
Windows IP Configuration


Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . : sd.scvz.hr

Wireless LAN adapter Lokalna veza* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Lokalna veza* 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : fritz.box
   Link-local IPv6 Address . . . . . : fe80::18f1:4077:638c:4a93%15
   IPv4 Address. . . . . . . . . . . : 192.168.178.43
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.178.1

Ethernet adapter Bluetooth mre§na veza:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

Slika 10. Rezultat naredbe ipconfig

Daljnjim pretraživanjem tekstualne datoteke i rezanjem znakovnog niza, dobiva se IP adresa u obliku znakovnog niza. Kod funkcije je preuzet s [7].

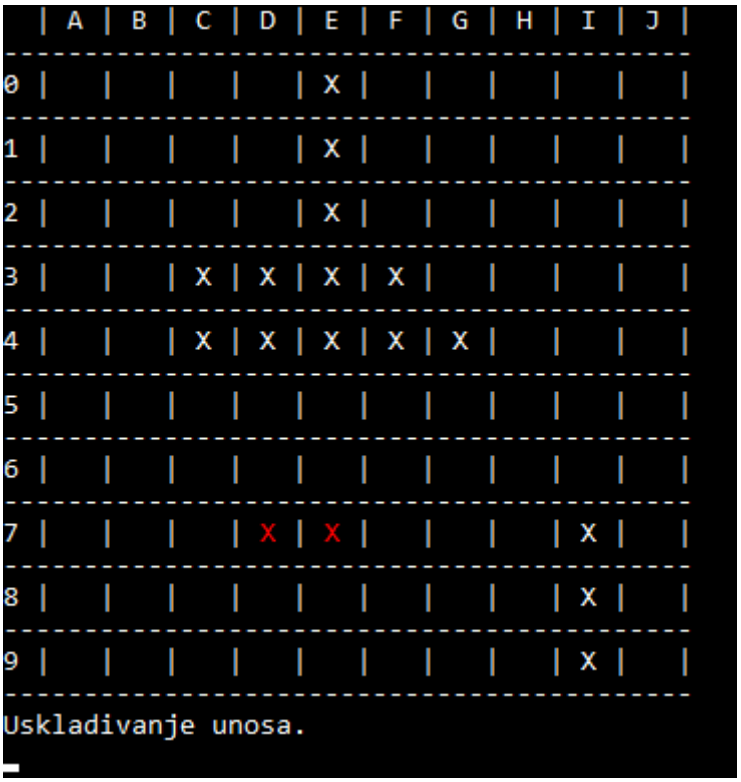
Nakon unosa protivnikove IP adrese, ispisuje se matrica s brodom duljine 5. Kako bi se prepoznao brod, koji se trenutno unosi, on je obojan crvenom bojom kao što je prikazano na slici 11. Brodovi se uvijek pojavljuju na sredini matrice, pa se s te pozicije razmještaju po matrici prema zadanim uputama, kojima se može pristupiti iz glavnog izbornika. Za svaki

slučaj, upute su još navedena ispod matrice kako bi bila dostupna igračima u svakom trenutku. Kako je prikazano na slici 11, unos broda duljine pet ćelija, na isti način se unose i svi ostali brodovi. Jedina razlika prilikom unosa je provjera postojanja nekog broda na toj poziciji. Ukratko rečeno, na istoj poziciji se ne smiju nalazi dva dijela broda, to jest brodovi se ne smiju preklapati. Ako se pokuša odabrati 'p', kako bi se potvrdio unos broda, a na tome polju se nalazi već neki drugi brod, ponovno će se ispisati matrica s postojećim rasporedom brodova. Ako se brodovi pokušaju okrenuti vodoravno ili okomito na mjestima gdje brod nema dovoljno ćelija, za okret će se samo ponovno ispisati postojeći raspored brodova. Nakon što su objašnjeni načini unosa brodova u matricu, nastavljam s opisom ostatka igre.



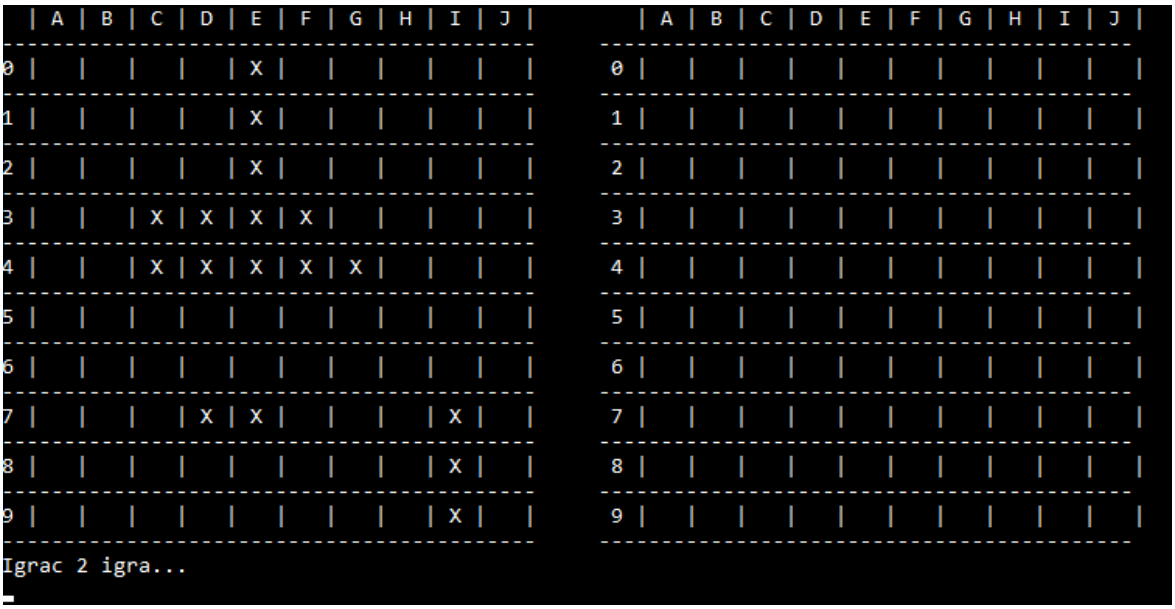
Slika 11. Početak unosa brodova

Nakon unosa svih brodova, dobiva se prikaz kao na slici 12. Ovo je samo jedan od mogućih rasporeda brodova.

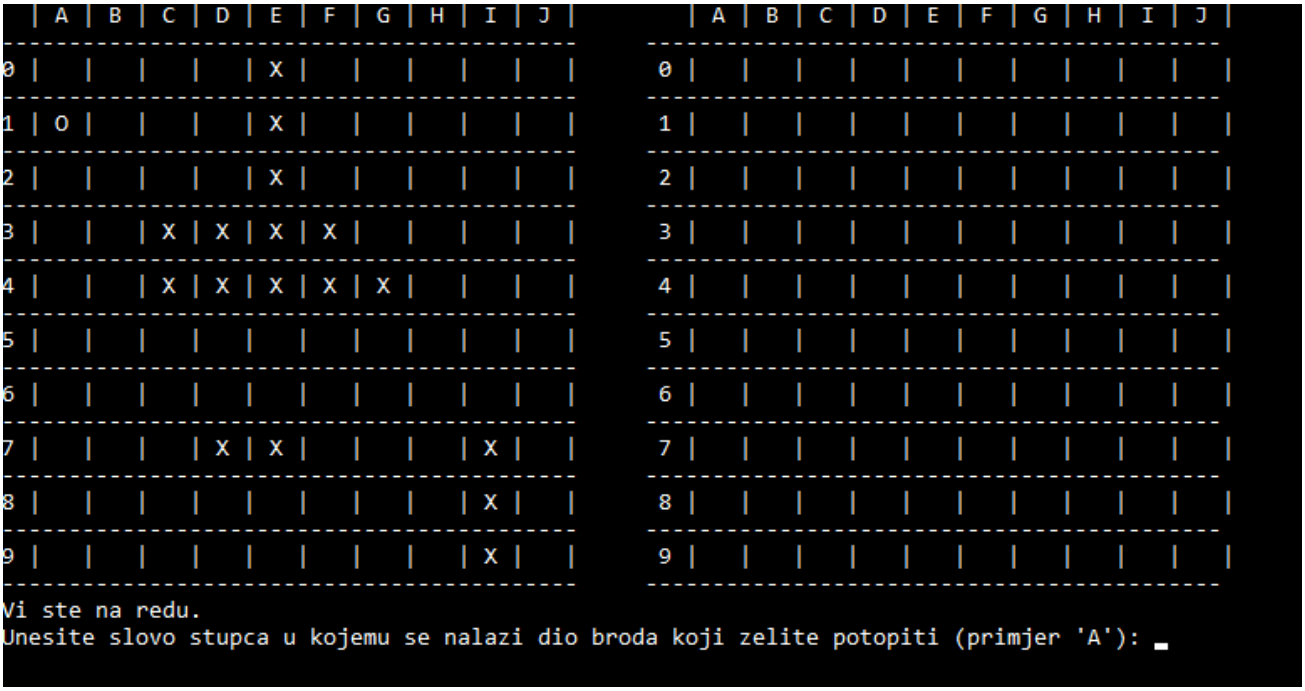


Slika 12. Uneseni brodovi

Iz razloga što naš protivnik može unositi brodove kada i mi, potrebno je uskladiti unose jer unos nikada neće završiti u istom trenutku. Nakon usklađivanja unosa, ispisuje nam se dodatna matrica u kojoj se prikazuju rezultati našeg potapanja. Ako se u desnoj matrici pojavi znak 'O', nismo potopili protivnikov dio broda. U slučaju da se pojavi oznaka 'X', potopili smo protivnikov dio broda te nastavljamo potapanje. U lijevoj matrici provjeravamo što je protivnik nama potopio. U matrici se pojavljuje znak 'O' na mjestu na kojemu je protivnik pokušao potopiti brod.

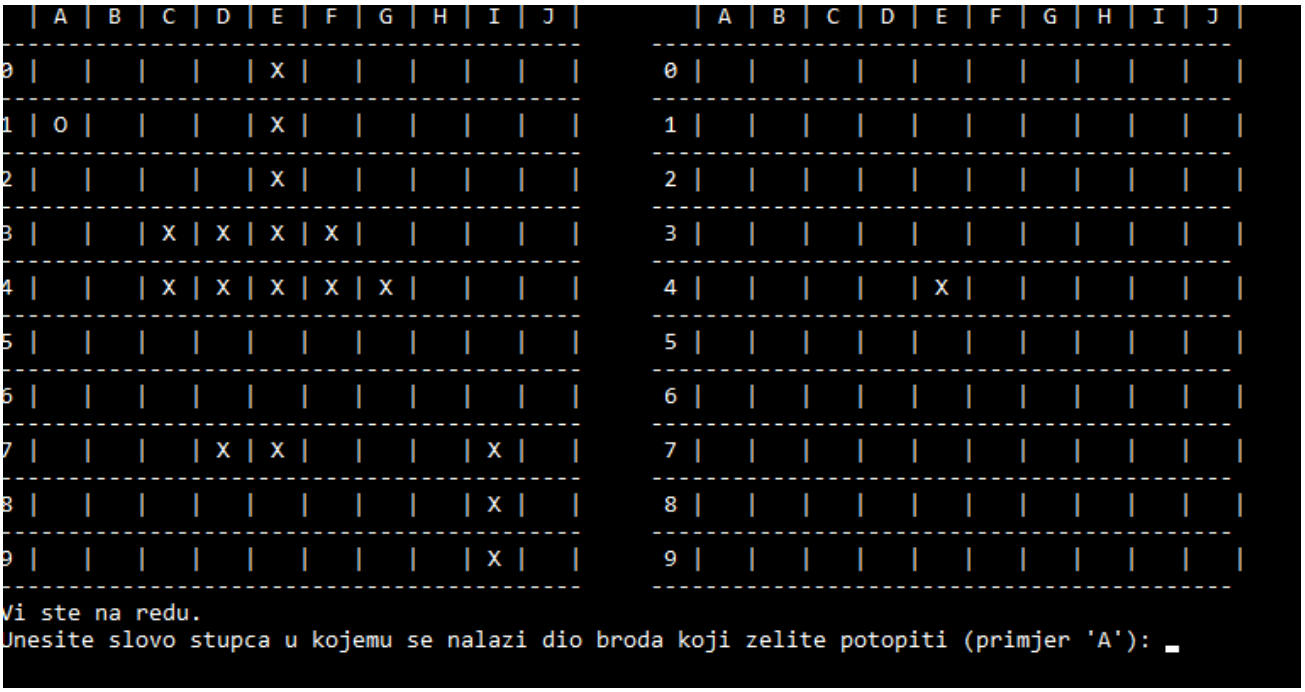


Slika 13. Ispis dodatne matrice za provjeru potapanja



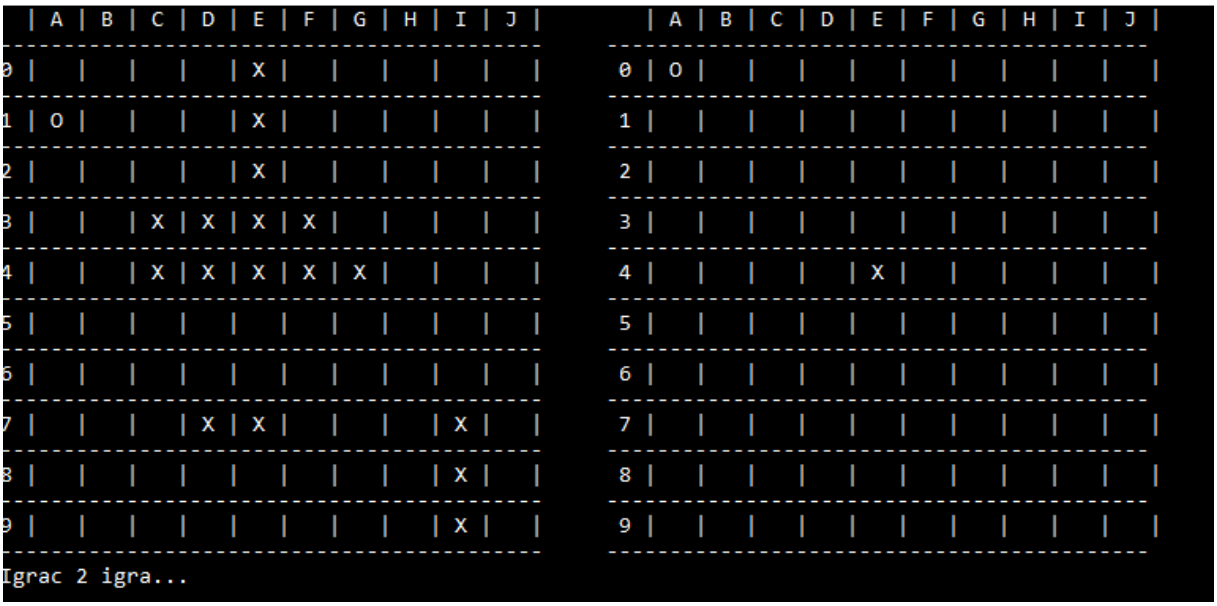
Slika 14. Protivnik je promašio brod

Na slici 14 se može primijetiti kako se u lijevoj matrici pojavila oznaka 'O' na polju A1, na kojemu se ne nalazi niti jedan naš brod te nam se nudi mogućnost potapanja protivnikovih brodova.



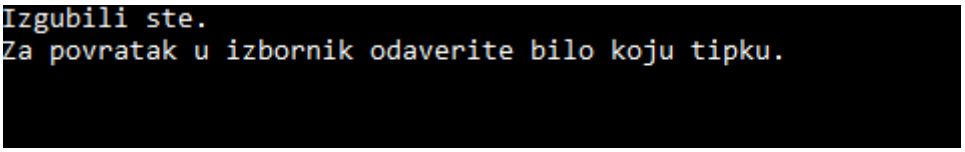
Slika 15. Potopili smo dio protivnikova broda

Unosom koordinata matrice E4 u desnoj matrici, može se primijetiti kako se pojavio znak 'X' što znači da smo potopili dio broda našeg protivnika.



Slika 16. Promašili smo

Kod protivnika smo pokušali potopiti brod na polju A0, ali tamo se ne nalazi niti jedan brod te se pojavio znak 'O' u desnoj matrici i protivnik ponovno može potapati naše brodove. Igra se tako izmjenjuje dokle god jedan igrač ne potopi sve brodove protivnika. Nakon završetka igre ispisuje se „Izgubili ste“ ili „Cestitamo pobijedili ste u igri potapanje brodova“.



Slika 17. Obavijest da ste izgubili

Po odabiru bilo koje tipke, prebacujemo se u izbornik odakle možemo započeti novu igru, pregledati upute ili izaći iz igre.

6. Problemi prilikom programiranja

U igri „Potapanje brodova“ omogućio sam igračima da mogu raditi unos brodova istovremeno, što je dovelo do problema prepoznavanja kada je koji igrač završio. Kada bi jedan igrač završio unos svojih brodova, trebao bi na neki način „javiti“ drugome igraču da je završio. No postoji i problem kako će protivnik znati na koju IP adresu treba poslati poruku da je završio.

6.1. Problem slanja poruke

Problem oko slanja poruke sam pokušao prvo riješiti na način kako i radi sam UDP protokol. IP adresu klijentskog dijela sam određivao pomoću dobivenog paketa, dok sam u klijentskom dijelu upisivao IP adresu serverskog dijela direktno u kod. No takav način mi nije odgovarao za usklađivanje unosa.

Zatim sam problem pokušao riješiti upisivanjem IP adrese serverskog dijela i IP adrese klijentskog dijela odmah u kod. Ovakav pristup bi radio u nekoliko pokušaja igranja, odnosno dokle god se ne bi promijenila barem jedna IP adresa. Razlog promjene IP adrese je dinamičko dodjeljivanje, te ako se kojim slučajem računalo odspoji s interneta i ponovno spoji može se dogoditi da ne dobije istu IP adresu.

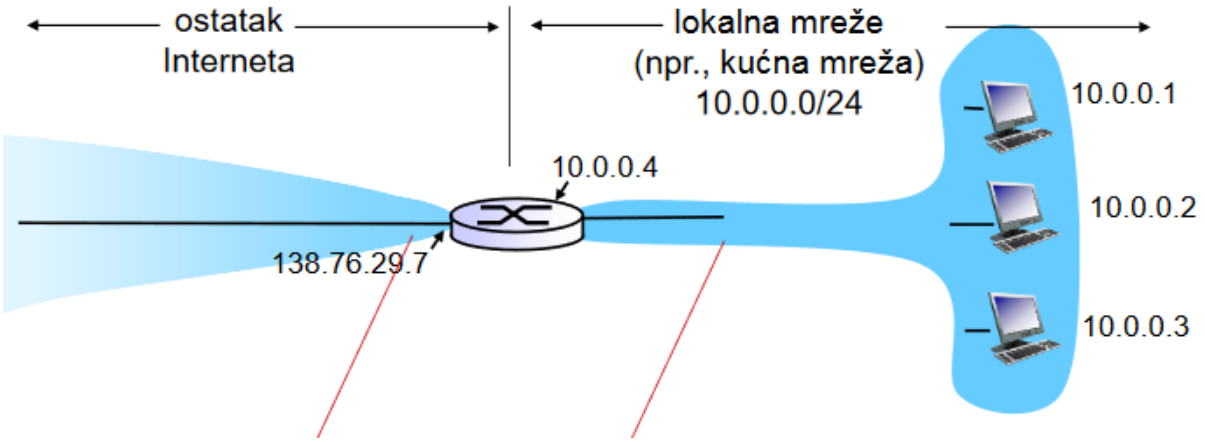
Nakon neuspjeha s izravnim upisivanjem IP adrese, odlučio sam se za komunikaciju igrača, kako bi si igrači međusobno poslali svoje IP adrese koje im se ispišu na zaslon. Adrese koje im se ispisuju na zaslon, dobivene su pomoću naredbe ipconfig. Na taj se način zna kome se šalju podaci.

6.2. Usklađivanje unosa

Kada se misli na usklađivanje unosa, moraju se uzeti u obzir kako obje strane, i klijentska i poslužiteljska, mogu prve završiti s unosom. Ne postoji pravilo po kojemu bi se odredilo koja je strana prva završila s unosom. Iz tog sam se razloga morao poigrati sa slanjem paketa između klijentske i poslužiteljske strane aplikacije. Napravio sam kod u kojemu je nakon unosa brodova, poslužiteljska strana neprestano primala pakete i slala klijentskoj strani, dok je klijentska strana neprestano slala pakete prema poslužiteljskoj, dokle god se ne bi promijenilo stanje u varijabli na obje strane, u koju se spremala primljena poruka, odnosno poruka koja se šalje. Ovaj kod je imao problema iz razloga što sam na početku koristio pristup da serverska strana prvo mora primiti paket, kako bi znala kamo mora poslati paket s promijenjenom vrijednosti. No kada smo primijenio pristup s IP adresama, ovaj kod bio je dobar.

6.3. Problemi mrežnih aplikacija

Nakon programiranja, kada sam mislio kako su riješeni svi problemi i da aplikacija radi savršeno, jer je bila testirana na dva računala i radila sve što treba, dogodio se problem koji ću opisati u nastavku. Aplikacija je testirana na dva računala koja su spojena na isti usmjernik, no pri pokušaju testiranja aplikacije na dva računala koja se nalaze u dva različita grada, aplikacija nije radila. Kroz daljnje proučavanje, došao sam do zaključka kako aplikacija zapravo radi, ali problem do kojeg dolazi je kompleksan. U ranijem poglavlju sam naveo koliko se IPv4 adresa može generirati i zaključio kako je broj adresa manji od broja stanovnika na planetu Zemlji. Iz tog razloga je uvedena tehnologija NAT (Network Address Translation), čija kratica na hrvatskom jeziku znači mrežni prevoditelj adresa. Kako bi se nadoknadio manjak broja IP adresa, jednoj lokalnoj mreži se dodjeljuje jedna IP adresa koja je vidljiva vanjskom svijetu. IP adrese unutar lokalne mreže su nevidljive ostatku svijeta, što znači da u jednom trenutku u više lokalnih mreža može postojati ista IP adresa. Kada neko računalo iz lokalne mreže šalje paket prema vanjskom svijetu, njegova izvorišna IP adresa je lokalna IP adresa, nakon što paket dođe do usmjernika, čija je IP adresa poznata vanjskoj mreži, izvorišna IP adresa preslikava se u poznatu IP adresu, što znači da će svi paketi koji se šalju iz lokalne mreže imati istu izvorišnu IP adresu. NAT tehnologija u ovome slučaju radi preslikavanje iz jedne IP adrese u drugu IP adresu, no radi promjene IP adrese mijenja se i PORT s kojega se šalje paket. Kada paket, koji služi kao odgovor, treba doći iz vanjske mreže u lokalnu NAT, „pregledava“ preslikavanja te sukladno tome određuje na koju će lokalnu IP adresu proslijediti paket. Ova tehnologija je dobra po pitanju sigurnosti jer vanjska mreža ne može vidjeli lokalne IP adrese, dok se adrese unutar lokalne mreže mogu mijenjati, a da se pri tome ne mora obavijestiti vanjska mreža o tome.[10]



Slika 18. Primjer NAT-a [10]

Kao primjer uzimam računalo s lokalnom IP adresom 10.0.0.1, koje šalje paket na uređaj s IP adresom 192.168.178.1 i port 2000. Paket s tog računala, kada je tek poslan, ima izvorišnu IP

adresu 10.0.0.1 i uzimam za primjer port 2500. Kada paket dođe do usmjernika, njegova izvorišna IP adresa se mijenja u 138.76.29.7 te se port mijenja recimo u port 3000, a to se preslikavanje zapisuje u NAT tablicu preslikavanja. Paket odlazi do uređaja s IP adresom 192.168.178.1 na port 2000, taj uređaj vraća odgovor na IP adresu 138.76.29.7 na port 3000. U NAT tablici preslikavanja se pronalazi zapis o preslikavanju. Odredišna IP adresa se treba preslikati u 10.0.0.1, a port u 2000 te se prosljeđuje paket. Kako bi se izbjegao ovakav način preslikavanja i nedostatak IP adresa, uvodi se IPv6 koji će imati dovoljan broj IP adresa za sve uređaje spojene na internet. Proces izmjene IP adresa iz IPv4 u IPv6 je već započeo, ali iz razloga što internet „radi“ svaki dan 24 sata taj proces nije jednostavan.

6.4. Problem do kojeg može dovesti NAT

Do problema dolazi ako se server nalazi u lokalnoj mreži iza NAT-a. Uzimaju se IP adrese kao i ranije, recimo da server ima lokalnu IP adresu 10.0.0.1 te mu klijent želi poslati poruku. U ovom slučaju dolazi do problema jer klijent „vidi“ samo IP adresu 138.76.29.7 i ako pošalje na nju paket, u NAT tablici preslikavanja ne postoji zapis te se paket odbacuje. Za ovaj problem postoji rješenje statičke konfiguracije NAT-a, koja uvijek prosljeđuje pakete koji dolaze na adresu 138.76.29.7 i port 2000 na adresu 10.0.0.1 i port 20000. Kao drugo rješenje se nudi da se između servera i klijenta postavi uređaj koji ima javnu IP adresu, što znači da je vidljiva i serveru i klijentu. Klijet i server se povezuju na uređaj s javnom IP adresom te on služi kao posrednik između njih [10].

U slučaju primijenjenom na ovom radu, aplikacija radi ako se klijentski i poslužiteljski dio izvode na uređajima u istoj lokalnoj mreži. No ne radi ako se proba testirati na računalima koja nisu u istoj lokalnoj mreži. Razlog je NAT jer računala nemaju javne IP adrese te se međusobno „ne vide“. Problem se ne može riješiti na način posrednika, jer nemam uređaj s javnom IP adresom. Odnosno imam barka.foi.hr, ali aplikacija je programirana za Windows platformu, a ne za Linux koji se nalazi na tome računalu.

7. Zaključak

Mrežne aplikacije se koriste u svakodnevnom životu. Stvari koje gledamo svakodnevno izgledaju nam jednostavne, ali s mrežnim aplikacijama nije isto. Prije samog početka programiranja treba dobro razumjeti cijelu strukturu interneta. Većina ljudi miješa pojmove internet i web. Govore za web da je internet, jer si ne mogu predočiti što je zapravo internet. Za one koji još uvijek nisu sigurni znaju li točno što je internet, a što web reći ću ukratko da je internet sklopovlje, dok je web softver. Internet koristi veliki broj protokola za prijenos podataka koji su raspoređeni u slojeve pomoću kojih se lakše može razumjeti primjena protokola. Ja sam u ovome radu koristio protokol UDP za slanje i primanje podataka te UDP socket preko kojega sam slao i primao podatke. Naveo sam također da to nije jedina vrsta socketa te rekao da postoje i TCP socketi. Složenost programiranja mrežnih aplikacija ću objasniti na jednostavnom primjeru. Programer može naučiti programirati jednostavne mrežne aplikacije te ih testira na svojoj lokalnoj mreži isto kao što sam i ja prilikom programiranja testirao. Implementirao je igru „Potapanje brodova“ te je poželio zaigrati igru protiv svojeg prijatelja koji živi u drugome gradu, ali igra ne radi, iako je sve radilo u lokalnoj mreži. Recimo da je programer završio tečaj programiranja te nije upoznat sa strukturom interneta, on bi u ovoj situaciji bio bespomoćan, iako zna sve programirati. Ako programer nije učio o strukturi interneta vjerojatno nije čuo za NAT te ne može pronaći problem jer ova vrsta greške u programiranju nije sintaktička niti će ju kompajler prepoznati već je logička i teško ju je pronaći. Za pronalaženje ovakve vrste greške potrebno je puno vremena što govori o samoj složenosti programiranja mrežnih aplikacija. No iako je programiranje same mrežne aplikacije složeno treba uzeti u obzir da se zapravo programiraju dva dijela, a to su poslužiteljski i klijentski dio. Ova dva dijela nisu jednaka jer klijentski dio šalje podatke prema poslužiteljskom koji ih obrađuje i prosljeđuje rezultat ponovno na klijentski dio. U ovome rad sam objasnio dvije vrste mrežnih aplikacija P2P i klijent-poslužitelj te naveo njihove razlike. Uz sve navedeno kao najvažniji dio bih izdvojio funkcije koje su potrebne za stvaranje komunikacije između dva računala. No morate razumjeti da su to funkcije koje se koriste u programskom jeziku C++ te možda neće raditi u nekom drugom programskom jeziku.

Kao završni zaključak bih rekao da iako je programiranje mrežnih aplikacija složeno da će se njihova proizvodnja povećavati s vremenom, radi što većeg povezivanja svijeta, što možemo vidjeti iz dosadašnjeg trenda.

8. Literatura

[1] Drago Radić, „Informatička abeceda“ Split-Hrvatska [Slika], Dostupna: https://informatika.buzdo.com/s918-internet-tcp-ip-skup-protokola.htm, Pristupano: 13.07.2018.

[2] Habermann Chair, Dostupno: http://www.chrisharrison.net/index.php/Visualizations/InternetMap, [Slika], Pristupano: 13.07.2018.

[3] Darko Androić, Osnovna mrežna tehnologija, [Na internetu], Dostupno: http://www.phy.pmf.unizg.hr/~dandroic/nastava/ramr/poglavlje_1_1.html, Pristupano: 10.07.2018.

[4] Mario Radovan, Računalne mreže 2, 2011. [Na internetu], Dostupno: https://www.inf.uniri.hr/~mradovan/rm2docs/RM2p3.docx , Pristupano: 13.07.2018.

[5] Miroslav Mihaljišin, Računarske mreže, [Na internetu], Dostupno: http://razno.sveznadar.info/3_4_net/Rracunarske-mreze.pdf, Pristupano: 10.07.2018.

[6] Toni Petrina, Mreže računala – 1 klijent 1 server (1.dio), [Na internetu], Dostupno: https://programabilan.wordpress.com/2011/11/12/mreze-racunala-1k1s, Pristupano: 14.07.2018.

[7] Samuel Adams, How to get my own IPV4 address, 2012. [Na internetu], Dostupno: http://www.cplusplus.com/forum/windows/82534/, Pristupano: 01.07.2018.

[8] Nikola Ivković, „Poglavlje 1 Uvod“, Nastavni materijal na predmetu Mreže računala 1 [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i infirmatike, Varaždin, 2016.

[9] Nikola Ivković, „Poglavlje 3 Transportni sloj“, Nastavni materijal na predmetu Mreže računala 1 [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i infirmatike, Varaždin, 2016.

[10] Nikola Ivković, „Poglavlje 4 Mrežni sloj“, Nastavni materijal na predmetu Mreže računala 1 [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i infirmatike, Varaždin, 2016.

[11] Nikola Ivković, „Kratak uvod u sockete za izradu raspodijeljenih aplikacija koje koriste UDP“, Nastavni materijal na predmetu Mreže računala 2 [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i infirmatike, Varaždin, 2017.

[12] Nikola Ivković, „Socket programiranje: uvod kroz primjere“, Nastavni materijal na predmetu Mreže računala 2 [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i infirmatike, Varaždin, 2017.

[13] Tea Milak, „Analiza značajki i primjene protokola mrežnog sloja TCP/IP skupine protokola“, 2017. [Na internetu]. Dostupno: https://repozitorij.fpz.unizg.hr/islandora/object/fpz:1120/preview , [pristupano 01.08.2018.]

[14] Dragutin Kermek, „Internet i Web. Povijest Interneta i Weba. Razvoj Interneta i Weba.“, Nastavni materijal na predmetu Web dizajn i programiranje [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, 2018.

9. Popis slika