TEHNIČKI FAKULTET - RIJEKA
Diplomski studij računarstva

PROJEKT - NAPREDNE RAČUNALNE MREŽE

PEER-TO-PEER KOMUNIKACIJA PREKO NAT UREĐAJA

Tibor Jaklin

1. UVOD

Projekt opisan u ovom izvješću izveden je kao implementacija teorije opisane u znanstvenom radu "*Peer-to-Peer Communication Across Network Address Translators*" [1].

Cilj ovog projekta je izraditi mrežnu aplikaciju koja uspostavlja izravnu komunikaciju između dva uređaja na različitim privatnim mrežama. Aplikacija je arhitekture *peer-to-peer* (*P2P*). Ispravan rad *P2P* aplikacija općenito otežavaju kućni router uređaji (*Residential Gateways*) iz razloga što koriste *Network Address Translation* (*NAT*) tehnologiju. Kako bi se uspješno implementirala *P2P* aplikacija, potrebno je koristiti tehniku koja će omogućiti uspostavu komunikacije preko *NAT* uređaja. Takvih tehnika postoji nekoliko, a u ovom je projektu korištena *Hole Punching* tehnika.

Opis izrade projekta i objašnjenje spomenutih pojmova slijedi u nastavku ovog rada.

1.1. Client-server i peer-to-peer arhitekture

Prilikom dizajniranja mrežne aplikacije, programer se odlučuje na jednu od dvije najraširenije arhitekture u svijetu mrežnih aplikacija: *client-server* arhitekturu ili *peer-to-peer* (*P2P*) arhitekturu.

Client-server arhitektura određuje uloge računala koja međusobno komuniciraju. Jedno računalo određuje kao *server* računalo, čija je uloga poslužiti ostala računala, koja se nazivaju klijenti.

Client-server aplikacije rade na način da klijentsko računalo pošalje zahtjev prema *server* računalu, koje je gotovo uvijek dostupno, kako bi ono primilo i obradilo zahtjev. Na temelju primljenog zahtjeva *server* vraća neku informaciju nazad klijentskom računalu. Specifično za ovu arhitekturu mrežnih aplikacija je potreba da *server* računalo ima dodijeljenu javnu *IP* adresu, kako bi svaki klijent koji je spojen na javnu mrežu, internet, mogao koristiti uslugu tog *server* računala.

Iznimno popularne *web* aplikacije, koje koriste *client-server* arhitekturu, imaju velik broj klijenata pa tako i velik broj zahtjeva za usluživanjem. Kako bi *web* aplikacija mogla efikasno poslužiti sve svoje klijente, potrebno je da *server* računalo, ili više *server* računala, koja izvršavaju aplikaciju raspolažu sa dovoljno resurasa za usluživanje velikog broja zahtjeva. U praksi su cijene nabave i cijene održavanja takvih računala vrlo visoke.

Druga popularna arhitektura u web aplikacijama je *peer-to-peer* arhitektura. Za razliku od *client-server* arhitekture, u *P2P* arhitekturi sva računala koja komuniciraju mogu istovremeno funkcionirati kao klijent i kao *server*, to jest, mogu istovremeno pružati neki sadržaj i primati neki drugi sadržaj. Prilikom dizajniranja *P2P* aplikacija, programer ne mora uložiti u nabavu i održavanje skupocijenih *servera* jer su svi klijenti istovremeno i *serveri*. Iz tog razloga, mreža se zove *peer-to-peer*, a uređaji koji su dio te mreže se zovu *peer* uređaji.

Primjer ovakve *web* aplikacije bila bi aplikacija internetske telefonije, gdje dva korisnika (ili više njih) mogu održavati necentraliziranu glasovnu komunikaciju.

1.2. Javne i privatne IP adrese te komunikacija privatne i javne mreže

U prethodnom potpoglavlju spomenuto je kako se *client-server web* aplikacije oslanjaju na dostupnost *server* računala. Iz tog razloga *server* računalo koristi javnu *IP* adresu, za razliku od klijentskih računala koja internetu obično pristupaju sa privatnih mreža, na kojima klijenti imaju privatne *IP* adrese.

IP adresa je jedinstveni identifikator svakog uređaja (točnije, uređajeve mrežne kartice) koji se nalazi na mreži. Uređaji koji su spojeni na neku mrežu komuniciraju na način da preko komunikacijskog kanala razmijenjuju pakete. Paketi u sebi nose neke informacije specifične za vrstu razgovora kojeg uređaji vode, ali također moraju nositi i informacije o pošiljatelju i primatelju paketa kako bi se osiguralo ispravno dostavljanje paketa. Za identifikaciju pošiljatelja i primatelja koriste se spomenute *IP* adrese u kombinaciji sa posebnim brojevima, zvanim *port* brojevima.

Moguće je da će uređaj htjeti voditi nekoliko komunikacija preko mreže istovremeno, koristeći jednu komunikaciju za svaki program koji se na uređaju trenutno izvršava. Spomenuto je da se za identifikaciju uređaja na mreži koristi *IP* adresa, a kako bi se identificirao i točan program (od nekoliko trenutno aktivnih programa) koji šalje pakete koristi se *port* broj. Tako je, recimo, paket kojeg program na uređaju šalje u komunikaciji sa nekim *web serverom* identificiran sa: **127.0.0.1:12345**.

Ovaj zapis predstavlja *IP* adresu (**127.0.0.1**) i port broj (**12345**). Neki drugi program, koji komunicira sa različitim *web serverom* može imati *IP* adresu i *port*: **127.0.0.1:12346**.

Iz ovih je primjera vidljivo da je *IP* adresa uređaja ista, dok je *port* različit za svaki program koji se izvršava na tom istom uređaju i komunicira na mreži. Sada kad je ideja *IP* adresa i *port* brojeva jasna, može se opisati razlika između privatnih i javnih *IP* adresa.

Ako je *IP* adresa nekog uređaja javna, to znači da je jedinstvena u cijelom svijetu i da će bilo koji drugi uređaj, kojem je dostupna javna mreža, moći kontaktirati taj prvi uređaj.

Ako je *IP* adresa nekog uređaja privatna, to znači da adresa nije jedinstvena u cijelom svijetu, već je samo jedinstvena u privatnoj mreži u kojoj se nalazi. U svijetu u istom trenutku može biti mnogo uređaja koji u vlastitim privatnim mrežama koriste neku određenu privatnu *IP* adresu. Primjerice, mobilni telefon koji je spojen na neku kućnu (privatnu) mrežu ima adresu **192.168.1.50**. U susjednoj zgradi postoji druga kućna, privatna, mreža na koju je spojen laptop sa adresom **192.168.1.50**. Kako se uređaji nalaze na različitim, nepovezanim mrežama, nikad neće doći do konflikta.

Uređaji koji se nalaze na privatnoj mreži svejedno mogu kontaktirati neki *web server* na javnoj mreži. To im omogućuje kućni *router* uređaj, koji ima dvije mrežne kartice pa tako i dvije *IP* adrese. Jedna mrežna kartica *router* uređaja komunicira sa uređajima na privatnoj kućnoj mreži, a druga mrežna kartica komunicira sa uređajima na javnoj mreži. Stoga, kućni *router* ima jednu privatnu *IP* adresu i jednu javnu *IP* adresu.

Kada neki korisnički uređaj sa privatne mreže želi kontaktirati *web server* na javnoj mreži, poruku će poslati kućnom *router* uređaju koji će poruku dalje proslijediti preko javne mreže prema *web serveru*. Prije nego kućni *router* uređaj proslijedi poruku, on će izmijeniti *IP* adresu pošiljatelja iz privatne *IP* adrese tog korisničkog uređaja, u javnu *IP* adresu od *routera*. Također je moguće da će izmijeniti i *port* broj pošiljatelja. Izmjena *port* broja će se objasniti u sljedećem poglavlju.

Web server će po primitku paketa vratiti neki drugi paket, kao odgovor, nazad na *IP* adresu i *port* broj s kojeg je primio prvobitni paket, a to su IP adresa i *port* broj kućnog routera. Po

primitku paketa, *router* će inverznim procesom izmijeniti adresu i *port* broj primatelja kako bi paket prosljedio na korisnički uređaj.

Tehnologija koja kućnom *routeru* omogućuje pretvorbu privatnih adresa u javne, i obrnuti proces, naziva se *Network Address Translation* (*NAT*).

2. NAT

Opisana tehnologija *NAT* omogućuje komunikaciju uređaja na privatnoj mreži sa uređajem na javnoj mreži.

Kako *NAT* tehnologija na router uređaju vrši pretvorbu privatne adrese u javnu?

Pretpostavimo nekoliko različitih slučajeva:

- **Slučaj 1:** U slučaju da korisnički uređaj na privatnoj mreži sa novopokrenutog procesa pošalje paket prema nekom uređaju na javnoj mreži, *router* će tu komunikaciju zabilježiti u tablicu uspostavljenih komunikacija te će zamaskirati paket.

  Router pamti komunikacije kao spoj *IP* adrese i *port* broja pošiljatelja i *IP* adrese i *port* broja primatelja.

  Maskiranje će se izvršiti na način da će privatnu *IP* adresu pošiljatelja pretvoriti u javnu *IP* adresu *routera*. *Port* broj će maskirati samo ako *router* uređaj već koristi taj broj.

- **Slučaj 2:** U slučaju da je korisnički uređaj na privatnoj mreži preko nekog procesa već uspostavio komunikaciju sa nekim uređajem na javnoj mreži, svaki budući paket koji će slati sa istog procesa prema istom primatelju će *router* maskirati na način kako je maskirao u prvom slučaju.

- **Slučaj 3:** U slučaju da je korisnički uređaj na privatnoj mreži preko nekog procesa već uspostavio komunikaciju sa nekim uređajem na javnoj mreži, a zatim s istog tog procesa pošalje paket čiji je primatelj neki uređaj na novoj, nepoznatoj adresi, *router* će postupiti ovisno o tipu *NAT*-a kojeg koristi. Tipovi *NAT* implementacija će se opisati u nastavku.

- **Slučaj 4:** U slučaju da je korisnički uređaj na privatnoj mreži preko nekog procesa već uspostavio komunikaciju sa nekim uređajem na javnoj mreži, a zatim *router* primi paket koji je namijenjen tom procesu korisničkog uređaja, ali je pošiljatelj s neke nove, nepoznate adrese, *router* će postupiti ovisno o tipu *NAT*-a kojeg koristi. Tipovi *NAT* implementacija će se opisati u nastavku.

U praksi postoji nekoliko različitih mogućih implementacija *NAT*-a, gdje su neke implementacije restriktivnije od drugih.

Tipovi NAT implementacija:

I. **Full Cone NAT:** U trećem slučaju, router će primljeni paket prihvatiti kao da je dio već uspostavljene komunikacije stoga će *IP* adresu i *port* pošiljatelja zamaskirati maskom na jednak način kao i u drugom slučaju.

   U četvrtom slučaju, paket će proslijediti korisniku, ne obazireći se na nepoznatu adresu pošiljatelja.

II. **Address Restricted Cone NAT:** U trećem slučaju, router će primljeni paket prihvatiti kao da je dio već uspostavljene komunikacije stoga će *IP* adresu i *port* pošiljatelja zamaskirati maskom na jednak način kao i u drugom slučaju.

U četvrtom slučaju, paket će proslijediti korisniku, ali samo u slučaju da je *IP* adresa pošiljatelja jednaka onoj s kojom je već uspostavljena komunikacija.

III. **Port Restricted Cone NAT:** U trećem slučaju, router će primljeni paket prihvatiti kao da je dio već uspostavljene komunikacije stoga će *IP* adresu i *port* pošiljatelja zamaskirati maskom na jednak način kao i u drugom slučaju.

U četvrtom slučaju, paket će proslijediti korisniku, ali samo u slučaju da su *IP* adresa i *port* broj pošiljatelja jednaki onima s kojima je već uspostavljena komunikacija.

IV. **Symmetric NAT:** Ovaj je *NAT* najrestriktivniji.

U trećem slučaju, router će primljeni paket smatrati kao dio nove komunikacije stoga će *IP* adresu i *port* pošiljatelja zamaskirati novom maskom.

U četvrtom slučaju, odbaciti će svaki paket koji dolazi s nepoznate *IP* adrese i *port* broja.

Ovisno o implementaciji *NAT*-a koju neki kućni *router* koristi, moguće je otežati, ali i onemogućiti korištenje *P2P* aplikacija.

Primjerice, *Full Cone NAT* je namajnje ograničavajući *NAT* te omogućava korištenje *P2P* aplikacije, bez potrebe da aplikacija koristi posebne tehnike za zaobilaženje ograničenja.

Symmetric NAT je najograničavajući pa kod ovog tipa nije moguće korištenje *P2P* aplikacija.

2.1. Nedostaci korištenja NAT-a

Pretpostavimo da većina kućnih *routera* koristi *Port Restricted Cone NAT*.

Zatim pretpostavimo da neka *web* aplikacija na *serveru* s javnom *IP* adresom, koja koristi *client-server* arhitekturu, pokuša uspostaviti vezu sa nekim uređajem na udaljenoj privatnoj mreži. *Server* u tome ne bi uspio, a razlog tome je da *web server* ne može znati kako doći do udaljene privatne mreže nekog klijenta, jer privatne *IP* adrese imaju smisla samo u privatnim mrežama.

Kao što je ranije spomenuto, moguće je da u istom trenutku postoji mnogo privatnih adresa na kojima su spojeni uređaji sa istom privatnom *IP* adresom koju ima i taj određeni klijent.

U hipotetskom slučaju da *web server* zna na kojem kućnom *router* uređaju se nalazi privatna mreža sa korisničkim računalom koje želi kontaktirati, *server* bi mogao poslati paket na javnu adresu tog *routera*. Pregledom informacija o pošiljatelju i primatelju na paketu, *router* bi zaključio da paket nije dio već uspostavljene komunikacije pa bi ga odbacio.

U *P2P web* aplikacijama sudjeluju uglavnom korisnički uređaji, poput laptopa i pametnih telefona, a oni su gotovo isključivo spojeni na privatne mreže. Lako je zaključiti da uspostava komunikacije između dvije privatne mreže koje koriste *NAT* nije trivijalna te da je nužno upotrijebiti posebne tehnike kako bi se ostvarila.

Tehnike koje se bave problemom uspostave veze preko *NAT* uređaja nazivaju se *NAT Traversal* tehnike, a ovisno o tipu *NAT*-a koji se koristi, problem uspostave veze iziskuje upotrebu specifične *NAT Traversal* tehnike.

2.2. NAT Traversal

Neke od često korištenih tehnika za ostvarivanje *NAT Traversal*-a su:

- ***STUN*:** Cilj ove tehnike je informirati korisničke uređaje na privatnim mrežama o javnoj *IP* adresi njihovog *routera*, koju koriste kako bi komunicirali sa javnom mrežom. *STUN* se obično implementira kao *web* aplikacija koja radi na javno dostupnom *serveru*. Klijenti pošalju upit na *server*, a on im da informaciju o *IP* adresi i *port* broju, s kojih je primio klijentov upit.
- ***TURN*:** Cilj ove tehnike je uspješno povezati dva uređaja u slučaju da se nalaze iza *Symmetric NAT*-a. Način na koji *TURN* povezuje te uređaje se ne može opisati kao *P2P* jer *TURN server* služi kao *proxy* za slanje paketa između dva uređaja.
- ***Hole Punching*:** Ova tehnika omogućuje uspostavljanje prave, necentralizirane, *P2P* komunikacije između uređaja. Ideja *Hole Punching* tehnike je "probiti rupu" u *NAT*-u na način da se dva klijentska uređaja najprije spoje sa *STUN serverom* koji je dostupan na javnoj *IP* adresi. U trenutku kad su uređaji spojeni sa *STUN serverom*, *STUN* će svakog klijenta informirati o adresi onog drugog klijenta, kako bi se oni međusobno izravno kontaktirali. Ako se klijenti nalaze iza *NAT*-a koji nije tipa *Symmetric NAT*, uspjeti će uspostaviti izravnu vezu sa svojim *peer*-om.

  Način na koji će to uspjeti je slijedeći:

  Prvi klijent, nazovimo ga *A*, koji je od servera primio informacije o drugom klijetu, *B*, će poslati paket sa istog procesa izravno prema *B*. Kućni router klijenta *A*"će taj paket prihvatiti kao dio već postojeće veze, kao što je opisano u tipovima *NAT*-a.

  Kućni *router* klijenta *B* će primljeni paket odbaciti jer dolazi sa nepoznate adrese. Tada će klijent *B* poslati paket izravno prema *A*. Kućni router klijenta *B* će također taj paket prihvatiti kao dio već postojeće veze.

  Paket sa klijenta *B* sada uspješno prolazi kućni *router* klijenta *A* i veza je uspostavljena.

Ovaj projekt ostvaruje *P2P* vezu između dva uređaja koristeći *Hole Punching* tehniku u kombinaciji sa *STUN serverom*.

3. IMPLEMENTACIJA

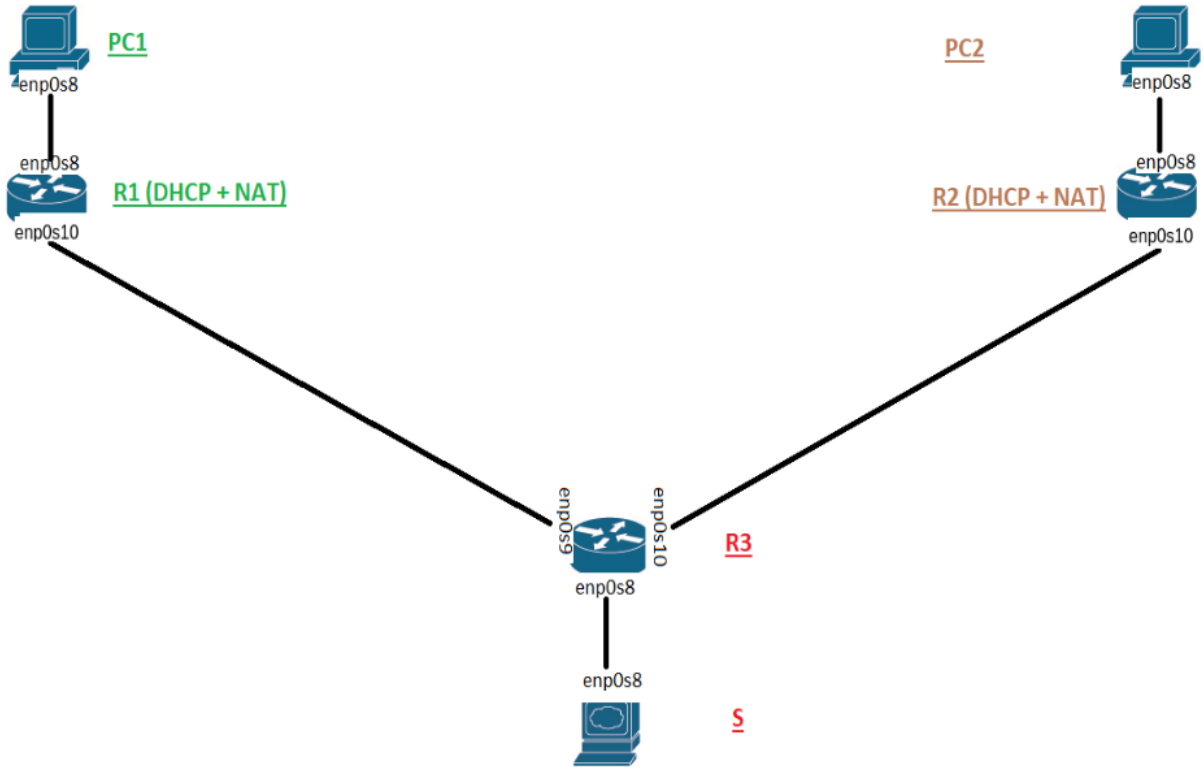
Autori rada na kojem se ovaj projekt temelji objašnjavaju svu nužnu teoriju kako bi se shvatila problematika komunikacije uređaja iza *NAT* uređaja. Također, detaljno objašnjavaju *Hole Punching* tehniku te opisuju kako su je oni odlučili implementirati te kako su testirali efikasnost implementirane tehnike.

U implementaciji ovog projekta korištena je mrežna topologija slična onoj u radu, a testiranje *Hole Punching* tehnike izvršeno je koristeći isti pristup kao i u radu.

Za simulaciju mrežne topologije koristi se *Oracle Virtualbox* program pomoću kojeg se pokreće nekoliko virtualnih računala, koja koriste *ubuntu 16.04 server* operacijski sustav.

Neka virtualna računala glume korisničke uređaje na privatnim mrežama dok druga računala glume kućne router uređaje i javni server. (Kućni router uređaji koriste *DHCP* tehniku pomoću paketa *isc-dhcp*), a *NAT* tehnologiju ostvaruju koristeći *iptables* program. Pomoću *iptables* moguće je postaviti *NAT* da funkcionira kao jedan od četiri navedenih tipova *NAT*-a. Programi koji implementiraju *Hole Punching* tehniku i uspostavljaju komunikaciju između uređaja napisani su koristeći *python* programski jezik.

3.1. Implementacija Topologije



Slika 1: Topologija mreže

Tablica 1: Tablica adresiranja

Device	Interface	IP Address	Subnet Mask	Default Gateway
R3	enp0s8	192.168.3.1	255.255.255.0	N/A
	enp0s9	192.168.101.1	255.255.255.0	N/A
	enp0s10	192.168.102.1	255.255.255.0	N/A
R1	enp0s8	192.168.1.1	255.255.255.0	N/A
	enp0s10	192.168.101.2	255.255.255.0	N/A
R2	enp0s8	192.168.2.1	255.255.255.0	N/A
	enp0s10	192.168.102.2	255.255.255.0	N/A
S	enp0s8	192.168.3.10	255.255.255.0	192.168.3.1
PC1	enp0s8	192.168.1.10	255.255.255.0	192.168.1.1
PC2	enp0s8	192.168.2.10	255.255.255.0	192.168.2.1

Na *slici 1* prikazana je mrežna topologija, a u *tablici 1* predstavljene su informacije o mrežnim karticama koje koriste računala iz topologije.

Uređaji **PC1** i **PC2** su dva privatna računala koja se nalaze u zasebnim privatnim mrežama. Svaka od dvije privatne mreže ima vlastiti router uređaj - **R1**, odnosno, **R2** - koji koriste implementaciju *Port Restricted NAT*-a koji je izveden koristeći *iptables* program.

Osim opisanih privatnih mreža, u mrežnoj topologiji postoji još jedan *router* uređaj, **R3** te server uređaj, **S**, koji izvršava *STUN web* aplikaciju. **R3** povezuje privatne mreže iza **R1** i **R2** sa javnim serverom na uređaju **S**.

3.1.2. Postavljanje NAT-a pomoću iptables programa

Program *iptables* korišten je za implementaciju svakog od navedenih tipova *NAT*-a. To se ostvaruje postavljanjem, takozvanih, pravila ili *rules* koji rade kao svojevrsni filtri paketa koji prolaze preko mrežnih kartica *linux* računala. Kombinacijom tih pravila moguće je imitirati rad svakog od četiri tipa *NAT*-a koji su spomenuti u ovom radu. Za vrijeme izrade projekta, isprobano je korištenje *Hole Punching* tehnike za zaobilaženje svakog od tipova *NAT*-a. Zaključak je da *Hole Punching* tehnika uspjeva na svim tipovima *NAT*-a osim *Symmetric NAT*-a.

Iptables pravila za imitaciju svakog od opisanih tipova *NAT*-a nalazi se u dodatku.

3.2. Implementacija koda

Kao što je opisano u teoriji *Hole Punching* tehnike, ključno je da klijenti uspostave vezu sa *STUN serverom* koristeći isti mrežni socket, ili istu *IP* adresu i *port* broj kojom će kasnije kontaktirati *peer* uređaj. Adresu kojom klijent uspostavlja vezu sa *STUN serverom* će *server* proslijediti *peer*-u, koji će zatim slati poruke na primljenu adresu.

Ukoliko će poruke klijenta prema *peer*-u koristiti drugačiju kombinaciju *IP* adrese i *port* bro-

ja, komunikacija se neće moći ostvariti.

Razlog zašto bi poruke klijenta ka *peer*-u mogle koristiti drugačiju kombinaciju *IP* adrese i *port* broja je moguće neispravno postavljanje *NAT* uređaja. Primjerice, *Symmetric NAT* je *NAT* postavljen na način da *Hole P*unching tehnika nikako neće uspjeti u uspostavljanju veze s drugom privatnom mrežom jer će se svaka poruka sa istog socketa prema imalo drugačijoj adresi primatelja tumačiti kao različita komunikacija, a svaka se komunikacija maskira na jedinstven način.

Potrebno je izraditi *STUN server* kako bi klijenti mogli saznati adrese željenih *peer*-eva.

Izrađena je *STUN server python* skripta te *python* skriptu klijentske aplikacije.

Kako su u autori referiranog rada opisali i implementirali *Hole Punching* tehniku koristeći *UDP* i *TCP* transportne protokole, ovaj će projekt također koristiti *UDP* i *TCP*.

3.2.1. Opis UDP server skripte:

Server izrađuje dvije varijable tipa *dictionary*. Prva je *Clients*, u koju sprema informacije o klijentima koji su ga kontaktirali. Druga je *Requests*, koja sprema klijentske zahtjeve za dohvaćanje informacija o adresi određenih *peer*-eva.

Nakon izrade dvaju *dictionary*-a, *server* izrađuje mrežni *socket* koji postavlja da koristi *UDP* protokol te *socket* izvrši *bind* na željenu *IP* adresu i *port* broj.

Kada je *socket* uspješno postavljen, počinje se izvršavati beskonačna petlja u kojoj *socket* sluša upite klijenata. Kada klijentov upit konačno stigne, izvršiti će se procesuiranje primljene poruke, tj. upita, kako bi se izvukle bitne informacije o pošiljatelju i njegovom zahtjevu.

Upit koji klijent šalje je formatiran kao ***userA->userB***, a označava da je ime klijenta ***userA*** te da traži informacije o *peer*-u, imena ***userB***.

U ovom trenutku *server* sprema sve primljene informacije u dva *dictionary*-ja. Klijentovo ime i adresu sprema u *Clients*, a klijentovo i *peer*-ovo ime u *Requests*.

U tom trenutku *server* je gotov sa procesuiranjem primljenog upita te može krenuti u izvršavanje drugog bitnog dijela programa. U tom dijelu programa *server* pokušava uslužiti sve primljene zahtjeve. To uspjeva na način da prolazi redak po redak kroz *Requests* te za svaki pročitani upit provjerava zna li adresu *peer*-a čije se informacije traže u upitu. Ukoliko ima sve potrebne informacije, izraditi će paket koji je formatiran **127.0.0.1:6789**, a sadrži *IP* adresu i *port* broj *peer*-a. Izraditi će i paket sa *IP* adresom i *port* brojom klijenta koji će biti namijenjen *peer*-u. Tada oba paketa šalje određenim računalima. Upravo obrađene zahtjeve smatra uspješno riješenima pa ih miče iz *Requests*.

3.2.2. Opis UDP klijent skripte:

Klijentska skripta, iz terminala preko kojeg se pokreće, prima dva ulaza koji određuju ime klijenta i ime *peer*-a. Klijent stvori mrežni *socket* kojim će slati poruke na *server*, čiju *IP* adresu i *port* broj unaprijed zna. Poruku, tj. upit, koji šalje na *server* formatiran je kao ***userA->userB***, koristeći informacije primljene kao ulaz iz terminala. Stvoreni mrežni *socket* će imati *IP* adresu klijentove mrežne kartice, a *port* broj je nasumičan broj kojeg će mu dodijeliti operacijski sustav. Klijent šalje poruku na *server* te ulazi u beskonačnu *while petlju* koja preko

izrađenog *socketa* sluša primljene pakete. Prvi paket koji će primiti biti će *serverov* odgovor na ranije poslani upit. *Serverova* poruka sadrži informacije o traženom *peer*-u, a formatirana je **192.168.1.10:57342**. Po primitku poruke, klijent je procesuira pa spremi bitne informacije. Slijedeći korak je uspostavljanje izravne veze sa *peer* uređajem. Klijent izrađuje novu poruku, sadržaja "***Hey userB***", koju sa istog mrežnog *socketa* šalje na *peer*-ovu adresu.

Nakon nekoliko trenutaka, ako je mrežna topologija ispravno postavljena, klijent će primiti poruku od svog *peer*-a koja sadrži "***Hey userA***".

U tom se trenutku smatra da je izravna komunikacija uspostavljena.

3.2.3. Opis TCP server skripte:

Server program koji koristi *TCP* kao transportni protokol radi na vrlo sličan način kao i *server* program koji koristi *UDP*. Jedina je razlika u tome što mrežni *socketi* koji su postavljeni na korištenje *TCP*-a mogu u nekom trenutku održavati vezu sa samo jednim udaljenim uređajem. U slučaju *UDP*-a, *serverov* je jedini mrežni *socket* mogao istovremeno održavati nekoliko veza sa klijentima. U ovoj je verziji *server* program osmišljen tako da jedan mrežni *socket* stalno sluša zahtjeve klijenata, a jednom kad primi zahtjev nekog klijenta, stvoriti će novi, jedinstveni *socket* koji će služiti samo za komunikaciju sa tim klijentom. Ako će, primjerice, *server* primiti upite od tri klijenta, morati će stvoriti tri posebna *socketa* za održavanje veze sa tim klijentima. *Server* izrađuje dvije varijable tipa *dictionary* u koje sprema informacije o klijentima i njihovim zahtjevima.

Tada sluša klijentovske zahtjeve za uspostavu veze. Kada uspostavi vezu sa klijentom, klijent će poslati upit o željenom *peer*-u. Primljeni upit će *server* procesuirati te će iz njega zaključiti ime klijenta i ime *peer*-a čije informacije klijent traži. Informacije dobivene procesuiranjem će spremiti u svoje *dictionary*-je. Također, nakon spremanja informacija, označiti će vezu sa ovim klijentom kao "neposluženom", što znači da klijentov zahtjev još nije poslužen.

Slijedeći korak je provjeriti svaki zahtjev u *Requests* te ga pokušati poslužiti. Ukoliko su informacije o *peer*-u koje se traže u *Requests* retku dostupne, *server* će izraditi poruku za klijenta koji je tražio informaciju pa će mu poruku poslati.

3.2.4. Opis TCP klijent skripte:

Klijent program koji koristi *TCP* kao transportni protokol je prilično kompliciraniji za implementaciju od onog koji koristi *UDP*. Činjenica da mrežni socketi koji koriste *TCP* mogu istodobno održavati samo jednu vezu je ograničavajuće jer tehnika *Hole P*unching zahtjeva da klijent koristi istu *IP* adresu i *port* broj za komunikaciju sa *STUN serverom* i sa *peer* uređajem. Ukoliko koristi neku drugu kombinaciju adrese i *port* broja, komunikacija se neće moći uspostaviti. Srećom, moguće je stvoriti više neovisnih mrežnih *TCP socketa* kojima se *IP* adresa i *port* broj ograničava na neki određeni. Stoga, iako jedan mrežni *socket* može održavati samo jednu vezu, klijent će moći komunicirati sa *STUN serverom* i sa *peer* uređajem koristeći više *socketa* koji dijele istu *IP* adresu i *port* broj.

U kodu, aplikacija uzima ulaz iz terminala kako bi saznala ime klijenta i *peer*-a. Nakon toga, šalje zahtjev za uspostavu veze sa *serverom*, kojem će nakon uspješnog spajanja poslati upit o željenom *peer*-u. Ubrzo će primiti odgovor sa *servera*, koji sadrži tražene informacije o adresi *peer*-a. Pomoću primljenih informacija sastaviti će novu poruku koju će preko novog mrežnog *socketa*, ali iste *IP* adrese i *port* broja, poslati na *peer*.

Istovremeno, klijentska aplikacija ima aktivno nekoliko socketa:

- Mrežni *socket* kojim aplikacija želi uspostaviti vezu sa *serverom*
- Mrežni *socket* kojim aplikacija želi uspostaviti vezu sa *peer*-om
- Mrežni *socket* koji sluša na ulazne zahtjeve za uspostavu veze sa nekim *peer*-om

Ukoliko klijent ne primi odgovor na poruku poslanu *peer*-u, s vremenom će primiti zahtjev za uspostavu veze sa tim istim *peer*-om, kojeg mu šalje *peer*. Veza sa *peer*-om će se uspješno uspostaviti, bilo da je vezu pokušao pokrenuti klijent ili *peer*.

4. LITERATURA

[1] - Ford, Srisuresh, Kegel: "*Peer-to-Peer Communication Across Network Address Translators*" , s interneta

https://pdos.csail.mit.edu/papers/p2pnat.pdf

[2] - James F. Kurose, Keith W. Ross: "*Computer Networking: A Top-Down Approach*", Pearson, 2012

[3] - Brian Linkletter: "*How to use virtualbox to emulate a network*", s interneta

https://www.brianlinkletter.com/how-to-use-virtualbox-to-emulate-a-network/

[4] - Gentoo Forums: "*How to use iptables for implementing different types of NAT*", s interneta

https://forums.gentoo.org/viewtopic-t-826825.html

[5] - Python web mjesto: "*Lib/socket.py*", s interneta

https://docs.python.org/3/library/socket.html

DODATAK A

Ovaj dodatak sadrži *iptables* pravila za postavljanje različitih tipova *NAT*-a na **R1** *routeru*. **R2** koristi ista pravila, ali sa drugačije *IP* adrese.

Full Cone NAT:

iptables -t nat -A POSTROUTING -o enp0s10 -p udp -j SNAT --to-source 192.168.101.2

iptables -t nat -A POSTROUTING -o enp0s10 -p tcp -j SNAT --to-source 192.168.101.2

iptables -t nat -A PREROUTING -i enp0s10 -p udp -j DNAT --to-destination 192.168.1.10

iptables -t nat -A PREROUTING -i enp0s10 -p tcp -j DNAT --to-destination 192.168.1.10

Address Restricted Cone NAT:

iptables -t nat -A POSTROUTING -o enp0s10 -p udp -j SNAT --to-source 192.168.101.2

iptables -t nat -A POSTROUTING -o enp0s10 -p tcp -j SNAT --to-source 192.168.101.2

iptables -t nat -A PREROUTING -i enp0s10 -p udp -j DNAT --to-destination 192.168.1.10

iptables -t nat -A PREROUTING -i enp0s10 -p tcp -j DNAT --to-destination 192.168.1.10

iptables -A FORWARD -i enp0s10 -p udp -m state --state NEW -j DROP

iptables -A FORWARD -i enp0s10 -p tcp -m state --state NEW -j DROP

iptables -A FORWARD -i enp0s10 -p udp -m state --state ESTABLISHED,RELATED -j ACCEPT

iptables -A FORWARD -i enp0s10 -p tcp -m state --state ESTABLISHED,RELATED -j ACCEPT

Port Restricted Cone NAT:

iptables -t nat -A POSTROUTING -o enp0s10 -p udp -j SNAT --to-source 192.168.101.2

iptables -t nat -A POSTROUTING -o enp0s10 -p tcp -j SNAT --to-source 192.168.101.2

iptables -t nat -A PREROUTING -i enp0s10 -p udp -j DNAT --to-destination 192.168.1.10

iptables -t nat -A PREROUTING -i enp0s10 -p tcp -j DNAT --to-destination 192.168.1.10

iptables -A FORWARD -i enp0s10 -p udp -m state --state NEW,RELATED -j DROP

iptables -A FORWARD -i enp0s10 -p tcp -m state --state NEW,RELATED -j DROP

Symmetric NAT:

iptables -t nat -A POSTROUTING -o enp0s10 -j MASQUERADE --random

iptables -A FORWARD -i enp0s10 -o enp0s8 -m state --state RELATED,ESTABLISHED -j ACCEPT

iptables -A FORWARD -i enp0s8 -o enp0s10 -j ACCEPT