Fakulteta za matematiko, naravoslovje
in informacijske tehnologije

Računalništvo in informatika

Projektni seminar

P2P chat aplikacija

Tehnična dokumentacija

Avtorji:
Andrej Erjavec
Mojca Kompara

Mentor:
doc. dr. Aleksandar Tošić

23. avgust 2024

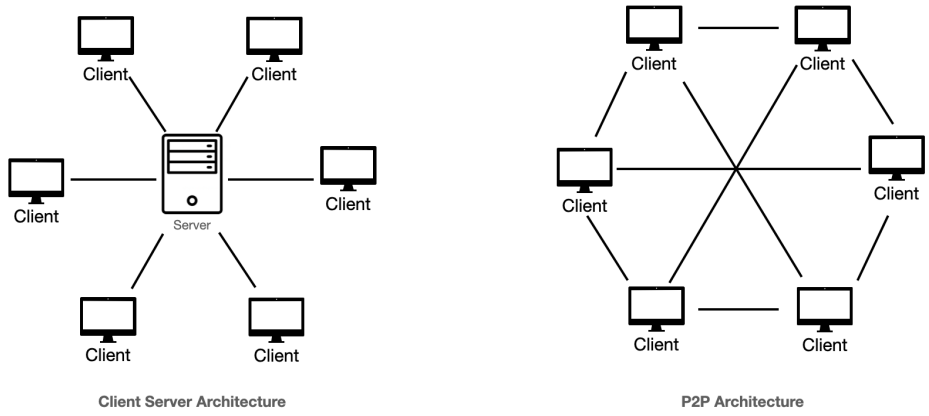
1 Opis problema

Klasične aplikacije za komunikacijo večinoma temeljijo na centralizirani arhitekturi, kar pomeni, da je procesiranje, posredovanje in hranjenje sporočil naloga centralnega strežnika. Naprave klienti torej ves promet usmerjajo preko tega strežnika. Takšen prisot ima svoje prednosti, med katerimi je je prenos večine procesorskeag dela in hrambe podatkov na strežnik, ki je v primerjavi z klineti precej zmogljivejši in s tem razbremenitev sistemskih virov klientov. Kljub temu pa ima centralizirana arhitektura tudi svoje slabosti - v prvi vrsti strežnik oziroma manjša skupina le-teh predstavlja šibko točko sistema, saj lahko njegov izpad ohromi celoten sistem. V tem primeru govorimo o pojavu *single point of failure (SPOF)*. Na tem mestu je pomembno omeniti tudi vprašanje varnosti in zasebnosti podatkov, ki ju je v primeru centralizacije težko zagotavljati, saj obstaja možnost manipulacije s podatki s strani entitet, ki si lastijo strežniške sisteme.

V primeru aplikacij za komunikacijo je rešitev omenjenih težav implementacija komunikacije v decentralizaranem peer-to-peer (P2P) omrežju, kjer si klienti sporočila izmenjujejo neposredno med seboj. V sklopu projektnega seminarja smo tako implementirali aplikacijo za izmenjavo tekstovnih sporočil v peer-to-peer omrežju.

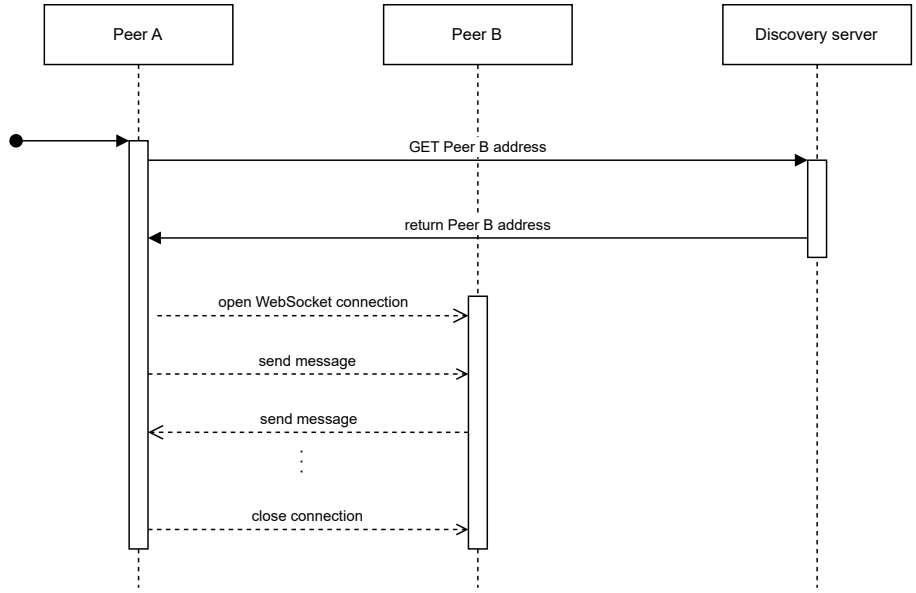
2 Analiza in načrtovanje sistema

Peer-to-peer komunikacija je po definiciji vrsta komunikacije, kjer si naprave spoorčila izmenjujejo neposredno med seboj brez posredovanja vmesnega centralnega strežnika. Naprave sodelujoče v omrežju s tujko imenujemo *peers* in vsaka izmed njih hkrati igra vlogo odjemalca in strežnika podatkov. Tako je tudi v primeru naše aplikacije - strežnik mora neprestano poslušati in sprejemati nova sporočila, odjemalec pa na strežnike drugih klientov sporočila pošilja. Ker je bila želja implementirati dostavljanje sporočil v realnem času, je za ta namen uporabljena tehnologija WebSocket - torej vsaka naprava igra vlogo WebSocket odjemalca in WebSocket strežnika hkrati.



Slika 1: Primerjava centralizirane in P2P arhitekture omrežja

Ena izmed glavnih težav P2P arhitekture je odkrivanje naprav v omrežju. Komunikacija med dvema napravama lahko steže le, če sporočevalec pozna IP naslov in vrata prejemnikovega strežnika. Ker se naprave v splošnem ne poznajo med seboj, rešimo problem odkrivanja tako, da žrtvujemo del decentraliziranosti in uvedemo strežnik, ki hrani trenutne IP naslove in vrata vseh registriranih naprav, te pa ob vsaki povezavi v omrežje strežniku sporočijo svoj naslov. Ko *peer A* želi poslati sporočilo *peer B*, strežnik vpraša po naslovu za *peer B* in nato pošlje sporočilo na ta naslov.



Slika 2: Sekvenčni diagram komunikacije med dvema peeroma

3 Definicija zahtev

Cilj je bil implementirati aplikacijo za neposredno komunikacijo, ki omogoča neposredno izmenjavo sporočil v lokalnem omrežju in jo bo v prihodnje enostavno nadgraditi in omogočiti tudi komunikacijo med napravami v različnih omrežjih z uporabo hole-punching protokola. Na podlagi tega so bile definirane naslednje zahteve sistema:

- V istem lokalnem omrežju morajo biti naprave sposobne neposredne komunikacije.
- Naprave v različnih omrežjih morajo biti sposobne komunikacije, vendar se v tem primeru ovrže potreba po striktno neposredni komunikaciji.
- Naprave morajo same hraniti zgodovino klepetov in sporočil.
- Aplikacija mora omogočati obveščanje uporabnika o novo prejetih sporočilih.
- Strežnik mora hraniti izključno podatke o naslovih naprav in ne sme hraniti sporočil.
- Sistem mora biti implementiran tako, da zagotavlja potrebno podlago za kasnejšo implementacijo hole-punching protokola.

4 Tehnologije

Želja je bila implementriati aplikacijo za mobilne naprave. Prvi pogoj, ki ga je bilo potrebno upoštevati je bil, da mora vsaka naprava delovati kot WebSocket strežnik in odejamalec hkrati. Uporaba spletnih tehnologij kot je React Native zato ni prišla v poštev, saj v brskalnikih lahko poganjamoo le implementacijo WebSocket odjemalca, ne pa tudi strežnika. Logična alternativa je bila implementacija mobilne apikacije za naprave z operacijskim sistemon Android. Ta je bila implementirana v programskem jeziku Kotlin, za hranjenje sporočil in pogovorov pa je bila uporbljena lokana podatkovna baza SQLite in knjižnica `Room`. Kot implementacija WebSocket protokola sta bila uporabljena `WebSocketClient` in `WebSocketServer` iz knjižnice `org.java_websocket`.

Strežniški del je bil implementiran v Node.js z uporabo knjižnice `Express` pri čemer sta bila za implementacijo WebSocket protokola uporabljena `WebSocket` in `WebSocketServer` iz paketa `ws`. Celoten strežniški del je bil nato zgrajen kot Docker vsebnik.

4.1 WebSocket

WebSocket je komunikacijski protokol, ki zagotavlja kanal za sočasno dvosmerno komunikacijo. Protokol je kompatibilen s HTTP protokolom in teče na eni sami TCP povezavi. Za razliko od HTTP protokola WebSocket podpira full-duplex komunikacijo, kar pomeni, da strežnik lahko pošilja sporočila klientom ne da bi ti morali prej poslati zahtevato. Zaradi tega se protokol pogosto uporablja za implementacijo aplikacij, ki zahtevajo prikaz podatkov v realnem času na primer aplikacije za neposredno komunikacijo in večigralske igre. V komunikaciji sta udeležena odjemalec in strežnik, protokol za vzpostavitev povezave pa se začne z rokovanjem (angl. handshake), kjer se obe strani dogovorita o protokolu za komunikacijo. Rokovanje začne odjemalec, ko pošlje strežniku sporočilo glavo HTTP protokola za nadgradnjo (angl. HTTP upgrade Header). Če strežnik podpira WebSocket protokol, odgovori s kodo 101 in povezava je vzpostavljena. Po vzpostavitvi povezave si odjemalec in strežnik lahko pošiljata sporočila preko iste povezave dokler je eden izmed njiju ne zapre.

4.2 Kotlin

Kotlin je statični visokonivojski programski jezik, razvit leta 2011 s strani podjetja JetBrains. Za razliko od klasičnih tipiziranih jezikov Kotlin uporablja *type inference*, kar pomeni, da prevajalnik avtomatsko zaznava tipe in s tem nekoliko poenostavi sintakso. Vključuje tako koncepte objektno-orientiranega kot tudi funkcijskega programiranja. Kotlin je bil primarno zasnovan za Java Virtual Machine (JVM) in iz tega razloga je tudi popolnoma kompatibilen z Java knjižnicami. Poleg tega pa se lahko prevede tudi v JavaScript za ustvarjanje spletnih aplikacij ter v native kodo za Android in iOS. Danes se v veliki meri uporablja za razvoj aplikacij za platformo Android.

4.3 SQLite in Room

Vsa sporočila in pogovori so hranjeni na strani klientov in sicer v podatkovni bazi SQLite. SQLite je tip vgrajane podatkovne baze (angl. embedded database) in hkrati knjižnica za upravljanje s podatkovno bazo. Za razliko od klasičnih podatkovnih baz kot je PostgreSQL, je ta namenjena uporabi na vgrajenih napravah (angl. embedded devices) med katere sodijo tudi pametni telefoni. SQLite za delovanje ne potrebuje strežnka, DBMS sistema in administracije, temveč je celotna baza shranjena v eni sami datoteki, dostop do katere urejajo pravice datotečnega sistema naprave same. Vzporedni dostopi do baze s strani več procesov je implementiran z zaklepanjem dostopa med pisanjem. Sintaksa poizvedb se zgleduje po PostgreSQL, vendar SQLite ne zahteva preverjanja tipov.

Pogosta praksa pri razvoju Android aplikacij s funkcijo lokalne hrambe podatkov je uporaba SQLite baze v kombinaciji s knjižnico Room, ki zagotavlja nivo abstrakcije

nad bazo. Room ustvari povezavo med podatkovnimi razredi in tabelami v bazi, omogoča preverjanje poizvedb med prevajanjem kode ter omogoča migracije konfiguracije podatkovne baze. Knjižnica sestoji iz treh glavnih komponent: razreda poadtkovne baze, podatkovnih razredov in objektov za dostop do podatkov (DAO), kjer definiramo metode, preko katerih izvajamo poizvedbe, vstavljamo in brišemo podatke v bazi.

```
@Dao
interface ChatMessageDao {
    AndrejErjavec
    @Insert
    suspend fun insert(chatMessage: ChatMessage)

    AndrejErjavec
    @Query("SELECT * FROM chat_messages")
    fun getMessages(): LiveData<List<ChatMessage>>

    AndrejErjavec
    @Query("SELECT * FROM chat_messages WHERE conversationId = :conversationId")
    fun getMessagesForConversation(conversationId: String): LiveData<List<ChatMessage>>

    AndrejErjavec
    @Query("DELETE FROM chat_messages")
    suspend fun deleteAll()
}
```

Slika 3: DAO razred s poizvedbami za tabelo tekstovnih sporočil

```
@Entity(tableName = "chat_messages",
        foreignKeys = [ForeignKey(
    entity = Conversation::class,
    parentColumns = ["username"],
    childColumns = ["conversationId"],
    onDelete = ForeignKey.CASCADE
)])

data class ChatMessage (
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    var conversationId: String? = null,
    var sender: String? = null,
    var receiver: String? = null,
    var message: String? = null,
    var timestamp: String? = null,
)
```

Slika 4: Podatkovni razred za tekstovno sporočilo z definiranimo relacijo na tabelo pogovorov

4.4 Node.js

Node.js je odprtokodno okolje za razvoj strežniških aplikacij v jeziku JavaScript. Vsebuje knjižnice za interakcijo z datotečnim sistemom gostiteljskega operacijksega sistema in kreiranje HTTP srežnika ter izvajanje HTTP zahtev.

4.5 Docker

Docker je programska platforma, ki omogoča virtualizacijo na nivoju operacijskega sistema. Na voljo je na platformah Linux, Windows in MacOS. Docker uporablja koncept virtualizacije s vsebniki (angl. containers), kar zagotavlja precej nižjo porabo sistemskih virov v primerjavi z virtualizacijo z virtualnimi napravami. Vsebnik namreč za delovanje ne potrebuje lastnega operacijskega sistema kakor to velja za virtualne naprave, ampak do zahtevanih nižjenivojskih funkcionalnosti in virov dostopa preko jedra operacijskega sistema gostitelja (angl. kernel). Jedro zagotavlja vmesnik za virtualizacijo, dostop do datotečnega sistema in omrežja gostitelja ter skrbi za upravljanje s spomonim in procesorjem. Virtualizirane aplikacije so grajene v obliki vsebnikov, ki poleg same aplikacije vsebujejo tudi vse knjižnice in ostale programske pakete, ki so potrebni za delovanje aplikacije. Zaradi abstrakcije, ki je dosežena s pakiranjemm aplikacij v samostojne vsebnike in dodatno abstrakcijo omrežja ter datotečnega sistema je Docker uporabljen za zagotavljanje delovanja aplikacij neodvisno od konfiguracije strojne in programske opreme gostitelja.

5 Implementacija

5.1 Strežnik za odkrivanje

Strežnik ima v sistemu primarno naslednje tri naloge.

5.1.1 Registracija klientov

Ob vsaki registraciji, ki jo klient opravi, strežnik shrani njegove trenutne podatke.

- `username` - uporabniško ime, ki si ga izbere uporabnik v aplikaciji
- `publicAddress` - javni IP naslov naprave
- `publicPort` - vrata, ki jih usmerjevalnik dodeli za komunikacijo s strežnikom (javna vrata)
- `privateAddress` - IP naslov klienta v privatnem omrežju
- `privatePort` - vrata na katerih posluša WebSocket strežnik klienta
- `socketId` - unikatni identifikator WebSocket povezave, ki jo klient vzpostavi s strežnikom

Klient strežniku preko WebSocket protokola posreduje podatke o uporabniškem imenu, privatnem IP naslovu in vratih lastnega WebSocket strežnika. Podatek o javnem IP naslovu in javnih vratih pa pridobi strežnik sam iz povezave. Ta dva podatka pri trenutni implementaciji nista uporabljena in se shranita za podporo kasnejši implementaciji hole-punching protokola. Unikatna identifikatorja sta uporabniško ime in socketId. Podatki o klientih so na strežniku hranjeni v JSON formatu.

5.1.2 Pridobivanje podatkov o klientih

Strežnik lahko na zahtevo klienta posreduje seznam vseh registriranih naprav in njihove naslove oziroma naslov določenega kilenta. Te podatke je potrebno pridobiti vsakič, ko želimo vzpostaviti povezavo za komunikacijo, saj je lahko IP naslovi naprav spreminjajo v primeru, da niso statično alocirani.

5.1.3 Posredovanje sporočil med klienti

V primeru, da sporočevalec ne more vzpostaviti povezave s strežnikom prejemnika, posreduje sporočilo strežniku za odkrivanje in ta ga naprej posreduje prejemniku. Za iskanje pravega prejemnika je uporabljen socketId. Tega sporočevalec posreduje skupaj z vsebino sporočila, strežnik pa najde odprto WebSocket povezavo, ki ima enak id kot posredovani socketId in preko nje pošlje sporočilo. Vsebina sporočil pri tem ni

shranjena na strežniku.

Struktura sporočil, ki se pošilja preko WebSocket povezave je na strežniku definirana na sledeči način.

$$message = \langle topic, code, data \rangle$$

- $topic = register \mid chat\_request \mid forward\_message \mid all\_users \mid user\_data$
- $code = success \mid failure$
- *data* je JSON objekt s podatki, ki jih želimo poslati, zakodiran kot string.

5.1.4 Namestitev strežnika

Strežniški del aplikacije lahko namestimo kot Docker vsebnik. Za namestitev najprej potrebujemo Docker in Docker Compose.

Najprej kloniramo git repozitorij

```
$ git clone https://github.com/AndrejErjavec/P2P-chat-server.git
$ cd P2P-chat-server
```

Zgradimo in zaženemo Docker vsebnik aplikacije

```
$ docker compose up -d oziroma docker-compose up -d
```

Strežnik privzeto teče na vratih 5757. Izpise, ki jih podaja strežnik si je mogoče ogledati z ukazom

```
$ docker logs p2p-chat-server
```

5.1.5 Konfiguracija posredniškega strežnika (reverse proxy)

V kolikor imamo pred strežnik za odkrivanje postavljen posredniški strežnik (angl. reverse proxy) moramo v njegovi konfiguraciji dodati dodatne parametre. Ob registraciji strežnik pridobi podatek o javnem IP naslovu klienta preko HTTP glave iz polja `X-forwarded-for`. Posredniški strežnik je potrebno nastaviti tako, da v prejeto zahtevo vključi ta podatek.
Če uporabljamo Nginx reverse proxy je portebno v konfiguracijo za strežnik dodati naslednjo vrstico:

```
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

5.2 Mobilna aplikacija

Koda mobilne aplikacije je dostopna na GitHubu: `https://github.com/AndrejErjavec/P2P-chat-client`

5.2.1 WebSocket strežnik in odjemalec

Aplikacija skupno vsebuje tri instance WebSocket. Ob zagonu se vzpostavi instanca WebSocket strežnika, ki v ozadju posluša za sporočila ostalih klientov. Poleg tega se vzpostavi še instanca WebSocket odjemalca, ki se poveže na naslov strežnika za odkrivanje. Ta se uporabi ob vsaki registraciji in posluša za posredovana sporočila s strani strežnika za odkrivanje. Obe instanci tečeta v ozadju kot storitev (angl. Service) in sta dostopni tudi ostalim aktivnostim. Tako tudi minimizirana aplikacija še vedno lahko prejema sporočila. Ko strežnik prejme sporočilo drugega kilenta, shrani odprto povezavo. Ta se uporabi, da se izognemo odpiranju nove povezave v kolikor želimo komunicirati s klientom, ki je z našo napravo že vzposatvil povezavo. Naprava,

ki prva začne pogovor torej inicializira instanco WebSocket odjemalca, se poveže na prejemnikov strežnik in odpre povezavo. Ko je povezava odprta, lahko prejemnik sporočila pošilja nazaj preko iste povezave.

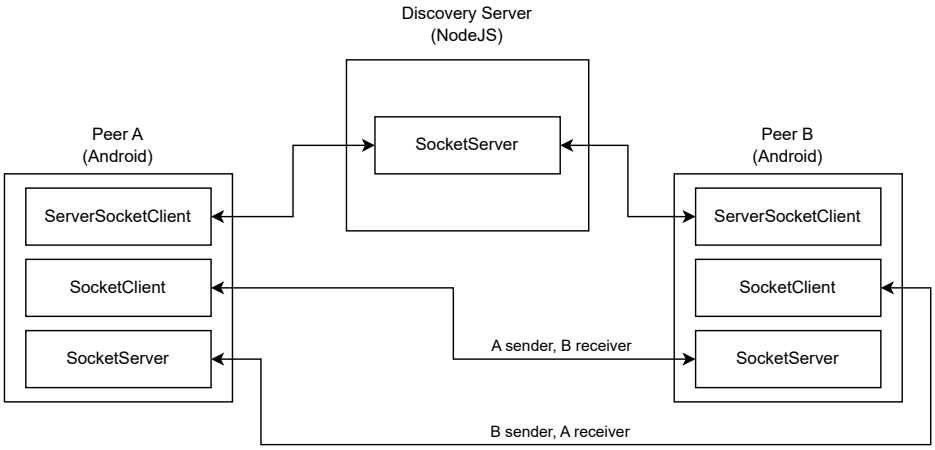
5.2.2 Registracija

Ob prvem zagonu se prikaže zaslon za registracijo, kamor v vnosno polje uporabnik vnese svoje uporabniško ime. Naprava sporoči strežniku svoje podatke, uporabniško ime pa se shrani tudi lokalno in s tem se uporabniku ni potrebno ročno registrirati ob vsakem zagonu aplikacije. Avtomatska registracija se zgodi ob vsakem naslednjem zagonu aplikacije, da naprava strežniku sporoči svoj trenutni IP naslov.

5.2.3 Klepet

Glavna aktivnost je zaslon s seznamom preteklih pogovorov s povezavo na zaslon, kjer lahko ustvarimo nov pogovor. Aplikacija na tem zaslonu ustvari zahtevo na strežnik na topic `all_users`, ta pa vrne seznam vseh registriranih naprav z njihovimi IP naslovi. S klikom na uporabnika se ustvari nov klepet v podatkovni bazi in prikaže zaslon za klepet.

Ob pošiljanju sporočila aplikacija vedno najprej poskuša poslati le-tega direktno na privatni IP naslov prejemnika. V kolikor prejemnik ni v istem lokalnem omrežju in spoorčila ni mogoče poslati direktno, se sporočilo pošlje na strežnik za odkrivanje in ta ga posreduje ciljni napravi. Na prejemnikovi napravi se v vsakem primeru ob prejetem sporočilu prikaže obvestilo (angl. notification), ki uporabnika obvesti o novem sporočilu. Pri pošiljatelju in prejemniku se ob prvem poslanem sporočilu na zaslonu klepetov prikaže ikona z uporabniškim imenom naprave, s katero je bil vzpostavljen pogovor.



Slika 5: Arhitektura aplikacije. Dvosmerne povezave predstavljajo WebSocket povezave. Naprava, ki prva pošlje sporočilo se poveže na strežnik prejemnika. Prejemnik shrani povezavo in ob pošiljanju sporočila preveri ali povezava že obstaja. V kolikor obstaja, pošlje poročilo preko te iste povezave. Če pa je bila povezava zaprta oziroma ne obstaja, ustvari novo povezavo s svojega WebSocket odjemalca na prejemnikov strežnik.

6 Razdelitev dela

- **Andrej:** načrtovanje, implementacija strežnika za odkrivanje, implementacija aplikacije, tehnična dokumentacija
- **Mojca:** načrtovanje, oblikovanje grafične podobe, implementacija aplikacije, uporabniška dokumentacija

7 Izboljšave in prihodnje delo na aplikaciji

Implementirana aplikacija ponuja vso potrebno funkcionalonost za izmenjavo sporočil v lokalnem omrežju in med različnimi omrežji. Kljub temu pa je odprtih še veliko možnosti za nadaljnje izboljšave. Na prvem mestu je implementacija hole-punching protokola, ki bi omogočil neposredno komunikacijo brez posredovanja strežnika tudi napravam, ki niso v istem lokalnem omrežju. Ker je peer-to-peer komunikacija izpostavljena takoimenovanim man-in-the-middle napadom bi bilo potrebno implementirati tudi enkripcijo sporočil.

7.1 Hole-punching protokol

Hole-punching je tehnika s katero omogočimo neposredno komunikacijo med napravami, povezanimi v različna omrežja, brez potrebe po odpiranju vrat požarnega zidu na usmerjevalnikih. Težava je, da usmerjevalniki omogočajo vzpostavljanje izhodnih povezav, vhodne povezave pa blokirajo. Vhodne povezave na določena vrata je mogoče dovoliti z odpiranjem vrat požarnega zidu. To je sicer mogoče v domačih omrežjih v kolikor ponudnik interneta naročniku omogoči dostop do nastavitev usmerjevalnika, a odprta vrata pogosto predstavljajo šibko točko omrežja, zato odpiranje vrat z vidika varnosti ni priporočljiv pristop. Težava nastane tudi v primeru, da je naprava, v tem primeru mobilni telefon, povezana v mobilno omrežje, kjer uporabnik nima nadzora nad požarnim zidom.

Omenjene omejitve je mogoče zaobiti z uporabo hole-punching protokola. Ta izkorišča UPnP protokol, ki je omogočen pri večini usmerjevalnikov za domačo uporabo. Ideja UPnP je sledeča: Predpostavimo arhitekturo odjemalec-stražnik, ki komunicirata med seboj. Ko z naprave na lokalnem omrežju izvedemo poizvedbo na strežnik, povezan na Internet, usmerjevalnik avtomatsko odpre zunanja vrata, preko katerih pričakuje odgovor strežnika. Zunanja vrata se mapirajo na IP naslov in notranja vrata naprave, s katere je bila poslana poizvedba, tako, da odgovor strežnika ob prispetju doseže ciljno napravo. Hole-punching izkoristi dejstvo, da so po poslani poizvedbi vrata odprta in lahko v teoriji preko njih pošlje sporočilo tudi druga naprava kot tista, na katero je bila naslovljena poizvedba.

Ker je hole-punching zahteven problem, delovanje rešitve pa je močno odvisno od specifične konfiguracje usmerjevalnikov, ostaja njegova implementacija izziv za prihodnje nadgradnje aplikacije.

V nadaljevanju je opisan proces protokola za TCP povezavo. Za poenostavitev predpostavimo, da napravi za povezavo uporabljata le javne IP naslove in vrata ter ne privatnih. Obe udeleženi napravi imata tako kot v primeru naše aplikacije tri socket instance: povezavo s strežnikom za odkrivanje, povezavo s prejemnikom in strežnik, ki posluša za nove povezave. Proces protokola je sledeči:

Napravi A in B se registrirata s povezavo na strežnik S. Pri tej poizvedbi usmerjevalnik odpre zunanja vrata, preko katerih pričakuje odgovor, strežnik pa shrani podatek o javnem IP naslovu in zunanjih vratih, preko katerih je bila opravljena registracija.

1. Naprava A želi poslati sporočilo napravi B. A vzpostavi povezavo s strežnikom za odkrivanje S in ga vpraša po naslovu naprave B.
2. S odgovori napravi A z javnim IP naslovom in javnimi vrati, preko katerih je naprava B opravila registracijo. Hkrati S tudi napravi B pošlje podatke naprave A.

3. A in B preko istih lokalnih vrat, s katerih je bila opravljena registracija z S pošiljata sporočila na naslove drug drugega. Hkrati obe napravi poslušata za nove povezave preko svojega strežnika.

4. Obe napravi pošiljata sporočila dokler ena izmed njiju ne uspe vzpostaviti povezave. Ko je povezava uspešno vzpostavljena, se napravi autenticirata in tako preverita, da je povezava res vzpostavljena med pravima napravama.

7.2 Enkripcija sporočil

V izogib man-in-the-middle (MITM) napadom ostaja odprt problem tudi implementacija šifriranja sporočil. Eded od možnih pristopov je uporaba ECDSA enkripcije, ki temelji na kriptografiji z eliptičnimi krivuljami.

Programska koda aplikacije je razdeljena na dva dela: strežnik in mobilna aplikacija. Koda je dostopna na GitHubu.
Strežnik: https://github.com/AndrejErjavec/P2P-chat-server
Mobilna aplikacija: https://github.com/MuciferTheCat/P2P-app