Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Denis Fidel

Implementacija glasovne komunikacije z uporabo WiFi Direct tehnologij

diplomsko delo

interdisciplinarni univerzitetni
študijski program prve stopnje
računalništvo in matematika

Mentor: prof. dr. Nikolaj Zimic
Somentor: asist. dr. Mattia Petroni

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Standard IEEE 802.11 je doživel številne različice in nadgraditve. Ena izmed takšnih je tudi WiFi Direct, ki omogoča povezovanje naprav brez uporabe vstopne točke. Najpogosteje se uporablja pri prikazovanju slik iz mobilnih telefonov z operacijskim sistemom Android na modernejše televizorje.

V diplomski nalogi izdelajte aplikacijo, ki bo omogočala govorno komunikacijo preko protokola WiFi Direct. Aplikacija naj bo samostojna in neodvisna od strežnikov in interneta. Aplikacija naj bo enostavna za uporabnika in naj deluje brez dodatnih sistemskih nastavitev, kot je na primer nastavitev IP številke.

Zahvaljujem se mentorju prof. dr. Nikolaju Zimicu in somentorju asist. dr. Mattii Petroniju za pomoč, odzivnosti in dostopnost v času izdelave diplomske naloge. Zahvaljujem se tudi družini in prijateljem za vso podporo v času študija in pomoč pri izdelavi diplomske naloge. Posebna zahvala gre tudi mami Tatjani in sestri Marijani za lektoriranje diplomske naloge.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
APK	Android Package Manger	upravitelj Android paketov
VoIP	Voice over Internet Protocol	telefonija preko internetnega protokola
iOS	iPhone Operating System	Applov operacijski sistem
AP	Access Point	vstopna točka
P2P	Peer-to-peer	točka-točka
GO	Group Owner	upravitelj Android paketov
SSID	Service Set Identifier	identifikator omrežja
WPS	Wi-Fi Protected Setup	Wi-Fi zaščitena namestitev
RTP	Real-time Transport Protocol	transportni protokol v realnem času
API	Application Programming Interface	aplikacijski vmesnik
TCP	Transmission Control Protocol	protokol za nadzor prenosa
UDP	User Datagram Protocol	protokol za uporabniška sporočila
HTTP	Hypertext Transfer Protocol	protokol za prenos obogatenega besedila
DHCP	Dynamic Host Configuration Protocol	omrežni protokol za dinamično nastavitev gostitelja

Povzetek

Naslov: Implementacija glasovne komunikacije z uporabo WiFi Direct tehnologij

Avtor: Denis Fidel

Večina že razvitih aplikacij, ki uporabnikom omogočajo zvočno komunikacijo, ne deluje brez povezave v internetno omrežje. Zato smo se v okviru diplomske naloge odločili za implementacijo aplikacije, ki bo uporabnikom omogočala zvočno komunikacijo preko protokola Wi-Fi Direct. Aplikacija bo omogočala izmenjavo zvočnih sporočil v realnem času, namenjena pa bo napravam z operacijskim sistemom Android. Problema smo se lotili tako, da smo najprej implementirali aplikacijo, ki omogoča pošiljanje zvoka s pomočjo shranjevanja v datoteko. Po uspešnem razvoju opisane aplikacije smo to nadgradili v končni izdelek. Obe različici aplikacije smo kasneje uporabili za primerjavo zakasnitev.

Ključne besede: Wi-Fi Direct, Android, Walkie-Talkie, glasovna komunikacija, mobilna aplikacija.

Abstract

Title: Implementation of voice communication using WiFi Direct technology

Author: Denis Fidel

The majority of existing applications that enable audio communication does not work without an internet connection. This led us to explore the possibilities for implementing an application that would enable users to use audio communication over Wi-Fi Direct protocol. The application, intended for devices with Android operational system, will enable real-time transfers of audio messages. The first step in addressing the problem was to implement an application that enables sending audio by saving to a file. After a successful development, the application was upgraded and developed into the end product. Later, we used both versions of the application to compare the delays.

Keywords: Wi-Fi Direct, Android, Walkie-Talkie, voice communication, mobile aplication.

Poglavje 1

Uvod

V današnjem času se pojavlja vse več mobilnih aplikacij, ki so namenjene medsebojni komunikaciji uporabnikov. Pojavljajo se aplikacije za medsebojno izmenjavo krajših in daljših tekstovnih sporočil, izmenjavo slikovnega gradiva in nenazadnje tudi zvočnih podatkov. Večina do sedaj implementiranih aplikacij uporabnikom ponuja komunikacijo na daljavo na podlagi dostopa do svetovnega spleta.

V tej diplomski nalogi se bomo osredotočili na zvočno komunikacijo uporabnikov. Implementirali bomo aplikacijo za operacijski sistem Android, ki bo za prenos podatkov med napravami uporabljala protokol Wi-Fi Direct. Aplikacija bo omogočala tako prenos zvoka v realnem času kot tudi pošiljanje zvočnega posnetka v obliki datoteke. Oba načina bosta delovala brez uporabe svetovnega spleta ali oddaljenega strežnika za dostop do internetnega omrežja.

Aplikacija, ki jo bomo implementirali v okviru te diplomske naloge bo temeljila na vzorčni aplikaciji. Namen vzorčne aplikacije je pošiljanje tekstovnih sporočil preko protokola Wi-Fi Direct. Poleg implementacije vzpostavitve neposredne povezave preko protokola Wi-Fi Direct vzorčna aplikacija vsebuje uporabniški vmesnik za pisanje, pošiljanje ter prikazovanje krajših tekstovnih sporočil. Za potrebe naše aplikacije bomo tega ustrezno prilagodili in dodali potrebno kodo za omogočanje snemanja, pošiljanja in predvajanja

zvoka.

V nadaljevanju diplomskega dela bomo v poglavju 2 podrobneje predstavili cilje diplomske naloge in predstavili aplikacije, ki že obstajajo na trgu in so sorodne naši aplikaciji. V poglavju 3 bomo opisali protokol Wi-Fi Direct in njegovo delovanje, sledilo pa mu bo poglavje o operacijskem sistemu Android, ki bo vsebovalo tudi osnove aplikacij za omenjen operacijski sistem. Poglavje 5 bo namenjeno predstavitvi orodij, ki bodo uporabljena za implementacijo zgoraj opisane aplikacije. Pred poglavjema, ki bosta namenjena sklepnim ugotovitvam in zaključku pa bomo v poglavju 6 podrobneje predstavili implementacijo obeh različic naše aplikacije. Na kratko bomo prikazali tudi scenarij uporabe naše aplikacije in težave, ki smo jih imeli pri implementaciji.

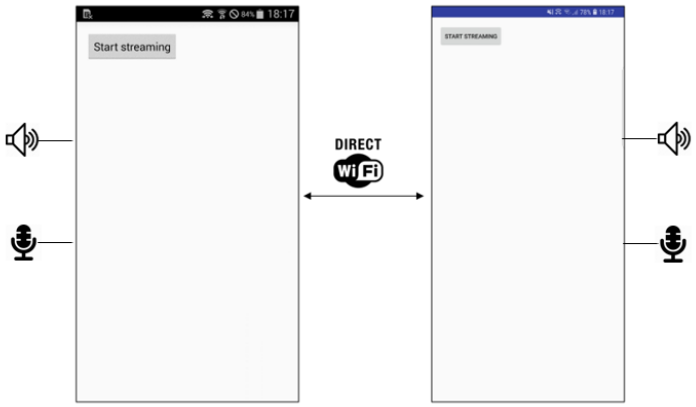
Poglavje 2

Podobne aplikacije in cilji diplomske naloge

Glavni cilj diplomske naloge je izdelava aplikacije za prenos zvoka med dvema napravama. Prenos zvoka bo potekal v realnem času preko protokola Wi-Fi Direct, ki omogoča neposredno povezovanje naprav. Po uspešno vzpostavljeni povezavi preko Wi-Fi Direct bo na uporabnikovo zahtevo naprava posnela zvok iz mikrofona in ga v realnem času poslala drugi napravi, s katero bo povezana. Na ta način je končnim uporabnikom omogočena komunikacija, ki spominja na delovanje Walike-Talkie naprav. Izgled aplikacije lahko vidimo na Sliki 2.1.

V tem poglavju si bomo podrobneje ogledali nekoliko bolj znane aplikacije, ki ravno tako omogočajo prenos zvoka med napravami. Med najbolj znanimi so npr. Skype, Viber, FaceTime, WhatsApp. Glavna razlika med našo aplikacijo in prej naštetimi bo, da bomo v našem primeru za prenos podatkov (t.j. okvirjev) uporabili protokol Wi-Fi Direct, kjer za delovanje ne potrebujemo internetne povezave, medtem ko vse naštete aplikacije za prenos potrebujejo dostop do internetnega omrežja. Za uporabo naštetih aplikacij je potrebna tudi ustrezna avtentikacija, medtem ko bo naša aplikacija delovala brez zahteve po preverjanju pristnosti.

Aplikacija bo torej implementirana z namenom, da bosta lahko dve na-



Slika 2.1: Na sliki je prikazan izgled aplikacije v dveh Android napravah, ki komunicirata preko Wi-Fi Direct [26].

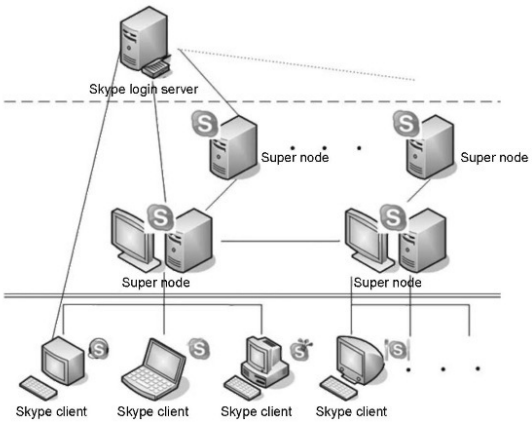
pravi, ki sta na prostem oddaljeni največ 100 metrov in nista nujno povezani z zunanjim strežnikom za dostop do internega omrežja, vzpostavili neposredno povezavo in si med seboj v realnem času pošiljali zvok. Cilj je, da aplikacijo implementiramo na dva načina. V prvem načinu bomo implementirali pošiljanje podatkov v realnem času, torej brez predhodnega shranjevanja. Drugi način bo omogočal pošljanje zvoka s predhodnim shranjevanjem podatkov v datoteko, ki jo po končanem snemanju pošljemo drugi napravi. Namen implementacije obeh načinov je kasnejša primerjava zakasnitve ter primerjava kakovosti prejetega zvoka glede na način implementacije.

2.1 Skype

Skype je aplikacija, ki omogoča avdio in video klice med več napravami (računalniki, pametnimi telefoni, tablicami...). Na trgu se je pojavila leta 2003, leta 2010 pa je presegla mejo 660 milijonov uporabnikov. Skype je eden od primerov t.i. VoIP aplikacije [22] (angl. *voice over internet protocol*). VoIP je metodologija in skupina tehnologij za zagotavljanje govornih komunikacij in multimedijskih sej preko internetnega protokola (IP). Temelj

VoIP telefonije je prenos podatkov preko internetnega protokola (IP) ter pretvorba analognega signala v digitalnega. VoIP poleg pretvorbe signala vključuje še signalizacijo, nastavitev kanala in kodiranje [13], [17].

Aplikacija Skype za delovanje uporablja protokol Skype [18], ki temelji na arhitekturi odjemalec-strežnik, njegove specifikacije pa niso javno dostopne. Skype velja za prvi protokol, ki je implementiral povezovanje točka-točka (angl. *peer-to-peer*) pri IP telefoniji. Omenjen način povezovanja bo na nek način uporabljen tudi pri naši aplikaciji. Ravno tako kot pri Skypeu se namreč tudi pri naši aplikaciji uporabnika lahko povežeta v načinu točka-točka, in sicer preko protokola Wi-Fi Direct. Razlika je v tem, da lahko uporabniki Skypea komunicirajo tudi na občutno večjih razdaljah, medtem ko bo naša aplikacija delovala samo v okviru omejitev protokola Wi-Fi Direct. Tako kot ostale naštete aplikacije tudi Skype za prenos večpredstavnostnih datotek uporablja protokol UDP [9].



Slika 2.2: Na sliki lahko vidimo Skypeovo decentralizirano topologijo [19].

V primerjavi z ostalimi aplikacijami je posebnost Skypea uporaba decentraliziranega strežniškega modela. Kot je razvidno iz Slike 2.2 so v Skypeu naprave in strežniki razdeljeni na tri tipe: strežnik za avtentikacijo, strežniki Supernode in navadne naprave. Po uporabnikovi uspešni prijavi, je uporabnik povezan na enega izmed Supernode strežnikov, preko katerega se nato

poveže v omrežje. Opazimo torej, da je centraliziran samo strežnik za prijavo v storitev, vsi ostali pa so decentralizirani. Tovrstnih težav nam pri implementaciji naše aplikacije ne bo potrebno reševati, saj aplikacija ne bo zahtevala prijave v storitev. Bo pa delovanje naše aplikacije centralizirano, saj ena od naprav deluje kot upravitelj skupine (angl. *group handler*) in s tem nadzoruje prenos podatkov med napravami.

2.2 Viber

Ravno tako kot Skype je tudi Viber [8], [21] primer VoIP aplikacije. Ustanovljen je bil s strani japonskega podjetja Rakuten, in sicer kot neposredna konkurenca Skypeu. Podobno kot Skype tudi Viber za delovanje potrebuje dostop do svetovnega spleta in posledično omogoča komunikacijo pri večjih razdaljah.

Najbolj opazna razlika v primerjavi z našo aplikacijo bo pri uporabi UDP protokola, ki ga Viber uporablja za potrebe realizacije VoIP komunikacije. Viber namreč za zagotavljanje dostave paketov v realnem času (ang. *real-time*) uporablja protokol UDP, medtem ko naša aplikacija v obeh podprtih načinih delovanja uporablja protokol TCP. TCP je bil v naši aplikaciji uporabljen predvsem zaradi zagotavljanja dostave paketov in boljše kvalitete storitve (angl. *quality of service*).

2.3 FaceTime

Applova različica aplikacije, ki ponuja prenos zvoka in slike v realnem času se imenuje FaceTime in omogoča podobne funkcionalnosti kot že omenjene aplikacije. Obstaja tudi posebna različica FaceTime Audio, ki omogoča samo prenos avdio podatkov in je podobna zgoraj naštetim aplikacijam. Posebnost, ki jo opazimo pri Applovi rešitvi je, da ta ne omogoča skupinskih pogovorov in komunikacije več kot dveh naprav hkrati. Prednost njihove rešitve je delovanje v načinu brez povezave z zunanjim strežnikom, ki omogoča dostop

do svetovnega spleta. To prednost ima tudi naša aplikacija. Od iOS 7 dalje je namreč omogočena tudi komunikacija preko mobilnih omrežij (LTE in 3G) [11].

2.4 WhatsApp Messenger

WhatsApp Messenger je brezplačna VoIP storitev v lasti Facebooka, ki poleg glasovnih omogoča tudi video klice, pošiljanje besedil in slik ter ostalih dokumentov. Večpredstavnostna sporočila (angl. *multimedia messages*) so poslana s prenosom zvočne, slikovne ali video datoteke na HTTP strežnik in nadaljnjim pošiljanjem povezave do želenih podatkov. WhatsApp za izmenjavo sporočil med dvema uporabnikoma uporablja mehanizem *store and forward* [20], ki v osnovi deluje tako, da sporočilo shrani na WhatsApp strežnku, kjer ta počaka na uporabnikov prenos. Ko je sporočilo preneseno, izgine iz strežnika in ga ni mogoče ponovno prenesti. V naši aplikaciji bo delovanje implementirano nekoliko drugače, saj bo naprava ob uspešno vzpostavljeni povezavi ter uspešno posnetem zvoku, le-tega posredovala drugi napravi, ki bo posnetek prejela takoj in ga takoj tudi predvajala [23].

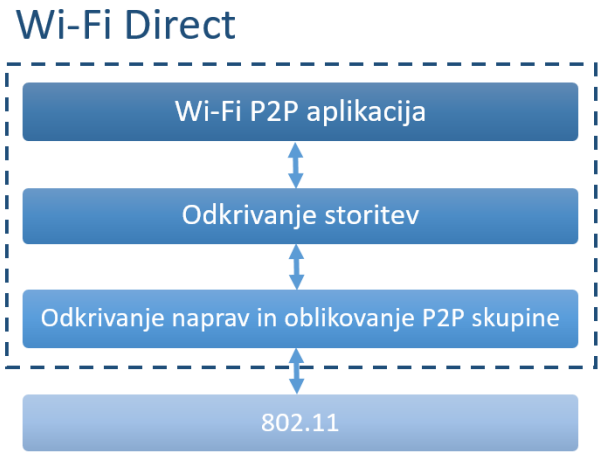
Poglavje 3

Wi-Fi Direct

Leta 2009 je organizacija Wi-Fi Alliance ugotovila, da se pametne naprave vse več uporabljajo za shranjevanje in prenos večpredstavnostnih datotek. To je vodilo k uvedbi protokola Wi-Fi Direct, ki je bil postavljen na temeljih tradicionalnih Wi-Fi standardov. Nov standard je nadgradil prednosti tradicionalnih 802.11 standardov kot so zmogljivost, varnost in enostavnost uporabe. Dodal je funkcionalnosti, ki uporabniku omogočajo enostaven in hiter prenos podatkov brez zahteve po dostopu do infrastrukturnega omrežja, tj. do omrežja vodenega s strani vstopne točke. Že omenjenim prednostim tradicionalnih standardov so bile dodane nove zmogljivosti, kot so samodejno odkrivanje naprav, prepoznavanje storitev naprave, s katero se povezujemo, boljši nadzor nad porabo energije in povezovanje naprav brez vzpostavljene povezave z internetnim omrežjem.

V nasprotju s predhodnim povezovanjem v infrastrukturno omrežje, preko katerega se povežemo z drugo napravo, lahko pri protokolu Wi-Fi Direct naprave povežemo neposredno. Zaradi omogočanja neposredne povezave je standard Wi-Fi Direct poznan tudi pod imenom Wi-Fi Peer-To-Peer. Standard omogoča sočasno delovanje z ostalimi Wi-Fi standardi, kar pomeni da je lahko naprava hkrati povezana z vstopno točko in drugo napravo preko protokola Wi-Fi Direct. Poleg povezovanja v paru z drugo napravo protokol Wi-Fi Direct omogoča tudi povezovanje z več napravami hkrati [1].

Standard Wi-Fi Direct lahko poleg mobilnih telefonov, tablic in računalnikov uporabljajo tudi naprave kot so televizije, tiskalniki, kamere, skenerji ipd. Na začetku obstoja standarda je bil tipičen primer njegove uporabe povezovanje računalnika s tiskalnikom, z namenom pošiljanja datoteke v tisk. Eden izmed prvih standardov, ki je uporabljal protokol Wi-Fi Direct je bil tudi standard Miracast [14]. Wi-Fi Direct se pogosto uporablja tudi v Androidovih in BlackBerry napravah za medsebojno deljenje datotek (angl. *file sharing*) [25].



Slika 3.1: Sklad protokola Wi-Fi Direct.

V naslednjih poglavjih bomo najprej predstavili arhitekturo protokola Wi-Fi Direct, nato pa si bomo podrobneje ogledali nekatere izmed njegovih novosti. Pogledali bomo postopek odkrivanja naprav, ki je nadgradnja postopka odkrivanja naprav pri tradicionalnih standardih 802.11. Podrobneje bomo opisali tudi postopek oblikovanja P2P skupine, ki je nadomestila tradicionalno omrežje z vstopno točko. Nekaj več bomo povedali tudi o postopku odkrivanja storitev, ki jih ponuja naprava s katero se povezujemo. Največja novost protokola Wi-Fi Direct v primerjavi s starejšimi standardi je odkrivanje storitev, saj pri standardih 802.11 omenjena funkcionalnost ni bila potrebna. V tradicionalnih omrežjih se je namreč naprava z vstopno

točko vedno povezala z istim namenom, ki bil povezati se v svetovni splet. Omenjene novosti lahko vidimo na Sliki 3.1 in jih bomo podrobneje predstavili v nadaljevanju.

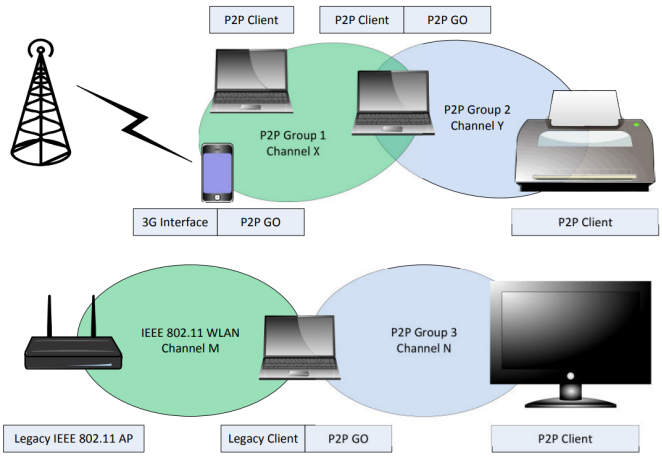
3.1 Arhitektura

Kot posledica nadgradnje tradicionalnih standardov 802.11 mora naprava, ki podpira Wi-Fi Direct protokol, delovanje omogočati v dveh različnih vlogah. Omogočati mora delovanje v vlogi vstopne točke (angl. *access point - AP*) in v vlogi odjemalca. Oba načina delovanja sta bila podprta že pri starejših standardih. Novost Wi-Fi Direct protokola je, da sta ti vlogi določeni dinamično in da lahko naprava teoretično opravlja naloge obeh vlog hkrati. Iz Slike 3.1 je razvidno, da protokol Wi-Fi Direct temelji na standardih 802.11.

Naprave v okviru standarda Wi-Fi Direct komunicirajo znotraj vzpostavljene P2P skupine (angl. *P2P Group*), ki ponuja enake funkcionalnosti kot tradicionalno Wi-Fi omrežje z vstopno točko. Vzpostavitev skupine si bomo podrobneje pogledali v razdelku 3.3. Naprava, ki v tako vzpostavljeni skupini opravlja naloge vstopne točke se imenuje *P2P Group Owner* (P2P GO), medtem ko naprave v vlogi odjemalca imenujemo *P2P Clients*. Kot že omenjeno sta vlogi dodeljeni dinamično, zato morajo naprave po postopku odkrivanja za vzpostavitev P2P skupine najprej doseči dogovor o opravljanju vloge P2P GO. Ko je vloga P2P GO dodeljena, je skupina vzpostavljena in odjemalci se lahko pridružijo skupini tako kot pri tradicionalnih omrežjih z vstopno točko. Naprave, ki delujejo kot odjemalci, lahko s P2P GO komunicirajo samo v primeru, da podpirajo ustrezen 802.11 standard in zahtevane varnostne mehanizme. Naprave, ki tem pogojem ne zadostujejo, vidijo napravo P2P GO kot navadno vstopno točko in ne delujejo v okvirju protokola Wi-Fi Direct.

Podobno kot vstopna točka v tradicionalnih omrežjih mora naprava, ki deluje kot P2P GO oddajati pakete *beacon*, s katerimi oglašuje SSID omrežja. Poleg tega pa mora podpirati mehanizme za delo v načinu majhne porabe

energije in omogočati delovanje prototokolu DHCP, ki skrbi za dodelitev IP naslovov med odjemalci. Ko je P2P skupina vzpostavljena, prenos vloge P2P GO med napravami ni več mogoč. Zato morajo v primeru, da P2P GO zapusti skupino, naprave ponovno vzpostaviti skupino [24].



Slika 3.2: Primer dolovanja naprav v različnih vlogah, ki jih omogoča Wi-Fi Direct [24].

Zgornji del Slike 3.2 prikazuje primer delovanja odjemalcev (P2P clients) v dveh P2P skupinah in hkratnem delovanju naprave v vlogi P2P GO in *P2P Client*. Spodnji del slike prikazuje delovanje Wi-Fi Direct ob tradicionalnem brezžičnem omrežju (IEE 802.11 WLAN).

3.2 Odkrivanje naprav

Novost protokola Wi-Fi Direct je torej odkrivanje naprav in oblikovanje P2P skupine. Pred vzpostavitvijo P2P skupine se mora izvesti postopek odkrivanja naprav, ki je nekoliko drugačen kot pri standardih 802.11.

Postopek odkrivanja naprav pri protokolu Wi-Fi Direct se začne s standardnim preiskovanjem in odkrivanjem omrežij v bližini, tako kot je pri odkrivanju navadnih Wi-Fi omrežij z vstopno točko. V postopku preiskovanja

naprava poleg omrežij, ki jih oglašujejo tradicionalne vstopne točke, vidi tudi oglaševane P2P skupine v bližini. Po preiskovanju se postopek nadaljuje z izvedbo algoritma *Discovery*. Algoritem se prične z izbiro frekvenčnega kanala, na katerem bo naprava poslušala prihajajoče zahteve za povezavo (angl. *Probe Requests*). V nadaljevanju algoritma se izmenjujeta stanji iskanja in poslušanja. V stanju iskanja naprava izvaja aktivno preiskovanje s pošiljanjem *Probe Requests* preko kanalov 1, 6 in 11 na frekvenčnem pasu 2,4 GHz in preko izbranih kanalov na frekvenčnem področju 5 GHz. V stanju poslušanja preko prej izbranega kanala sprejema *Probe Requests* in odgovarja s *Probe Response*. Ko se napravi uspešno odkrijeta v skladu z opisanim algoritmom, se prične postopek oblikovanja skupine, ki je opisan v naslednjem poglavju. Postopek preiskovanja lahko vidimo na Sliki 3.3, in sicer v razdelku *Discovery* [24].

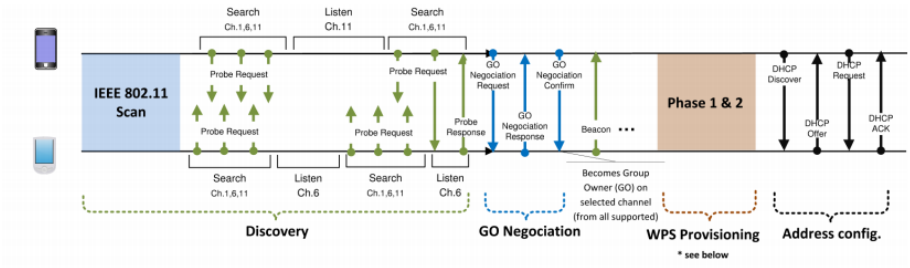
3.3 Oblikovanje skupine

Za oblikovanje skupine je potreben dogovor o prevzemu vloge *P2P Group Owner*. Obstaja več načinov, da se naprave dogovorijo, katera bo imela omenjeno vlogo. Podrobneje si bomo ogledali *Standard* način in *Autonomous* način. Vzpostavljanje skupine v naši aplikaciji bomo pogledali v poglavju 6 [24].

3.3.1 *Standard* način

V *Standard* načinu se po uspešnem postopku odkrivanja naprav, opisanem v prejšnjem podpoglavju, začne faza *GO Negotiation*. Ta je implementirana z uporabo tristranskega rokovanja (angl. *three-way handshake*), v katerem si po vrsti sledijo *GO Negotiation Request, Response* in *Confirmation*. Rezultat omenjene faze je dosežen dogovor o napravi, ki bo postala *P2P Group Owner* in določitev kanala, na katerem bo delovala P2P skupina. Po uspešni dodelitvi vloge *P2P Group Owner* sledi faza *WPS Provisioning*, v kateri naprave vzpostavijo varno komunikacijo z uporabo *Wi-Fi Protected Setup*. Postopek

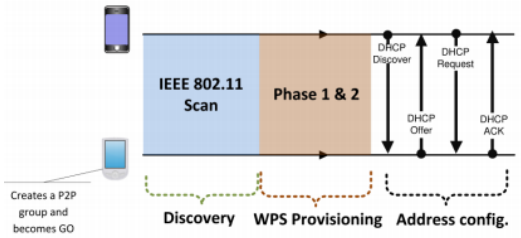
se zaključi z *Address config* fazo, v kateri je napravam s pomočjo protokola DHCP dodeljen IP naslov. Delovanje *Standard* načina lahko vidimo na Sliki 3.3 [24].



Slika 3.3: Primer oblikovanja skupine v *Standard* načinu [24].

3.3.2 *Autonomous* način

Nekoliko enostavnejši je *Autonomous* način. V tem primeru naprava sama vzpostavi P2P skupino in ima posledično že vlogo P2P GO. Na izbranem kanalu začne oddajati *beacon* pakete, ostale naprave pa po že znanem postopku odkrijejo P2P skupino ter se priključijo z izvedbo faz *WPS Provisioning* in *Address config*. V tem načinu torej ni potrebe po fazi *GO Negotiation*, saj naprava samodejno postane *P2P Group Owner*. Delovanje *Autonomous* načina lahko vidimo na Sliki 3.4 [24].



Slika 3.4: Primer oblikovanja skupine v *Autonomous* načinu [24].

3.4 Odkrivanje storitev

Po uspešnem oblikovanju P2P skupine, ali že prej se začne postopek odkrivanja storitev (angl. *Service discovery*). Umestitev omenjenega postopka v Wi-Fi Direct sklad lahko vidimo na Sliki 3.1. Odkrivanje storitev je bilo pri protokolu Wi-Fi Direct dodano z namenom prepoznavanja storitev, ki jih ponuja naprava, s katero se povezujemo. Naprava lahko na podlagi pridobljenih informacij o storitvi, ki jo ponuja druga naprava, prekine ali nadaljuje postopek oblikovanja P2P skupine. Postopek je implementiran z izmenjevanjem poizvedb (angl. *queries*), s katerimi na posamezni napravi upravlja protokol *Generic Advertisement Protocol*, ki je del standarda 802.11u. Podrobnejše delovanje je opisuje vir [24].

Poglavje 4

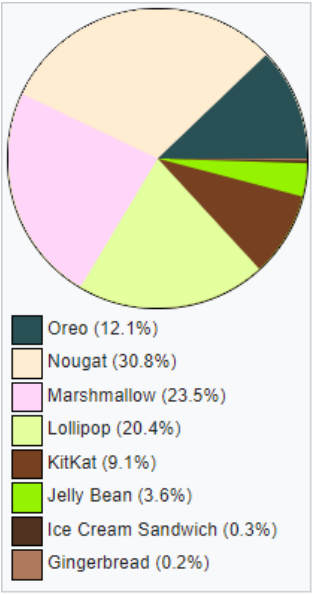
Android

Android je operacijski sistem, razvit s strani Googla, ki temelji na jedru operacijskega sistema Linux. Prva komercialna različica je izšla leta 2008, ko je Google izdal prvo napravo z nameščenim operacijskim sistemom Android. V začetku je bil razvit predvsem za naprave na dotik, in sicer pametne telefone in tablice, kasneje pa so nastale tudi različice za TV, avtomobile in ure. Leta 2011 je postal najbolj prodajan operacijski sistem za pametne telefone, za katerega je danes razvitih že več kot 3 milijone aplikacij. Na Sliki 4.1 lahko vidimo najpopularnejše različice operacijskega sistema Android v juliju 2018. Najnovejša različica je bila izdana avgusta 2018 in se imenuje Android *Pie* [3].

V nadaljevanju poglavja si bomo pogledali arhitekturo operacijskega sistema Android, nato pa se bomo podrobneje posvetili osnovam Android aplikacij.

4.1 Arhitektura operacijskega sistema Android

V tem poglavju si bomo podrobneje pogledali zgradbo operacijskega sistema Android, ki je razdeljen na štiri plasti. Na vrhnji plasti so aplikacije, sledijo jim aplikacijska ogrodja in knjižnice, najnižjo plast pa predstavlja jedro operacijskega sistema Linux. Zaporedje omenjenih plasti lahko vidimo na Sliki 4.2. [3]



Slika 4.1: Diagram prikazuje pregled deleža uporabe najpopularnejših različic operacijskega sistema Android. [3].

Na vrhu se nahaja aplikacijska plast, ki ponuja vrsto aplikacij, med katerimi so nekatere na napravo že nameščene skupaj z operacijskim sistemom. Medtem ko jih je večina nameščenih dodatno, glede na uporabnikove zahteve in potrebe. Primeri že nameščenih aplikacij so spletni brskalnik, aplikacije za elektronsko pošto, predvajanje glasbe, upravljanje imenika ter pošiljanje in prebiranje SMS sporočil. Operacijski sistem omogoča sočasno delovanje več aplikacij, kar pomeni, da je mogoče na primer predvajati glasbo in hkrati brati elektronsko pošto. Vse aplikacije za operacijski sistem Android so napisane v programskem jeziku Java. V opisano plast sodi denimo tudi aplikacija, ki jo bomo implementirali v okviru te diplomske naloge [12].

Aplikacijski plasti sledi aplikacijsko ogrodje (angl. *Application framework*), ki v obliki javanskih razredov punuja različne storitve aplikacijam na višji plasti. Sem sodi na primer javanski razred *WifiP2pManager*, ki je bil v naši aplikaciji uporabljen z namenom upravljanja z *Wi-Fi peer-to-peer* povezavo. Pri razvijanju novih aplikacij imajo razvijalci možnost uporabe

Slika 4.2: Slika prikazuje plasti operacijskega sistema Android [3].

vseh že implementiranih javanskih razredov (API-jev), ki so že bili uporabljeni v do sedaj razvitih aplikacijah. Namen nastanka omenjenih API-jev je bila poenostavitev ponovne uporabe le-teh pri izdelavi novih Android aplikacij [10], [16].

Plast, ki se nahaja pod aplikacijskim ogrodjem predstavljajo knjižnice napisane v programskem jeziku C ali C++. Razvijalec do njih dostopa preko javanskih vmesnikov, ki so del aplikacijskega ogrodja. Primer tovrstne knjižnice je denimo `MediaCodec` (MPEG-4 in MP3 format), ki omogoča kodiranje ali dekodiranje podatkov. Pomemben del opisane plasti je tudi Android izvajalno okolje (angl. *Android Runtime*). Vanj je vključena večina knjižnic, v katerih so omogočene funkcionalnosti, ki jih ponujajo izvorne knjižnice programskega jezika Java. Kot lahko vidimo na Sliki 4.2 je del Android izvajalnega okolja tudi Dalvik navidezna naprava (angl. *Dalvik virtual machine*), ki ima pomembno vlogo v delovanju Android aplikacij. Vsaka aplikacija v operacijskem sistemu Android namreč deluje v svojem procesu z lastnim primerkom Dalvik navidezne naprave [10].

Najnižjo plast operacijskega sistema Android predstavlja jedro, ki temelji na jedru operacijskega sistema Linux, zato je pogosto tudi imenovano kot slednje. Od leta 2018 Android uporablja verzije 4.4, 4.9 ali 4.14 Linuxovega jedra, dejanska uporabljena verzija pa je od naprave do naprave različna. Jedro operacijskega sistema skrbi za osnovne sistemske storitve, kot so varnost, upravljanje pomnilnika, upravljanje procesov, način delovanja gonilnika ipd. Jedro torej deluje kot abstraktni sloj med strojno opremo in ostalim skladom operacijskega sistema. Primer je gonilnik omrežne kartice, preko katerega operacijski sistem komunicira s strojno opremo kartice [3], [12], [16].

4.2 Osnove Android aplikacij

Aplikacije so del najvišje plasti operacijskega sistema Android in so najpogosteje napisane v programskem jeziku Java. Pogosto se uporabljata tudi jezika C++ in Kotlin. Celotna koda aplikacije je prevedena in arhivirana v datoteki *Android package* (.apk), na podlagi katere je aplikacija tudi nameščena na vseh napravah s podprtim operacijskim sistemom Android.

Vsaka aplikacija je s strani operacijskega sistema Android podprta z več varnostnimi mehanizmi, del katerih je tudi ločena izvedba posamezne od ostalih aplikacij. Ta je realizirana z zagonom vsake aplikacije v svoji Dalvik navidezni napravi, ki smo jo omenili v podpoglavju 4.1. Za varnost je poskrbljeno tudi z načelom, imenovanim *principle of least privilege*, ki pravi, da ima vsaka aplikacija dostop samo do tistih komponent, ki ji potrebuje za svoje delovanje in nič več.

V nadaljevanju bomo pogledali tri osnovne dele vsake Android aplikacije. V prvem delu bomo predstavili osnovne komponente. Sledila bo predstavitev datoteke *manifest file*, v kateri definiramo uporabljene komponente in zahtevane funkcije naprave, ki jih potrebujemo za delovanje aplikacije. V poglavju 4.2.3 bomo na kratko predstavili tretji del aplikacije, to so njeni viri (angl. *Resources*) [15].

4.2.1 Komponente aplikacije

Vsaka Android aplikacija je sestavljena iz več osnovnih komponent. Komponente aplikacij so bistveni gradniki Android aplikacij, saj predstavljajo vstopno točko (angl. *entry point*) skozi katero lahko sistem ali uporabnik vstopi v aplikacijo. Delimo jih na štiri različe tipe:

- aktivnosti (angl. *activities*),
- storitve (angl. *services*),
- sprejemnike dogodkov (angl. *broadcast receivers*) in
- ponudnike vsebin (angl. *content providers*).

Aktivnosti so namenjene interakciji z uporabnikom, saj so predstavljene kot uporabniški vmesnik, ki je prikazan na zaslonu naprave. Ob zagonu vsake aplikacije se na začetku zažene glavna aktivnost aplikacije (angl. *main activity*), preko katere lahko uporabnik dostopa do vseh ostalih delov aplikacije. Poleg aktivnosti obstajajo tudi storitve, katerih namen je ohranjanje izvajanja aplikacije v ozadju. Za njih je značilno, da z omogočajo izvedbo dolgotrajnejših operacij. Tretji tip osnovnih komponent so sprejemniki oddajnih signalov, ki aplikaciji omogočajo odziv na sistemska obvestila. Kot zadnji tip delujejo ponudniki vsebin, ki upravljajo s skupnimi podatki aplikacije (npr. podatki shranjeni v podatkovni bazi).

Posebnost sistema Android je, da lahko vsaka aplikacija poleg svojih zažene tudi komponente drugih aplikacij. Na ta način bi lahko naša aplikacija za zajemanje zvoka iz mikrofona uporabila drugo aplikacijo, ki omogoča enako funkcionalnost. Kot smo že omenili, v operacijskem sistemu Android vsaka aplikacija deluje kot ločen proces, zato neposreden zagon komponente iz druge aplikacije ni mogoč. Lahko pa posamezne komponente zažene operacijski sistem, kateremu namero zagona komponente druge aplikacije sporočimo z uporabo razreda `Intent` [2].

4.2.2 Datoteka *Manifest*

Pred zagonom komponent aplikacije mora operacijski sistem vedeti, da te obstajajo oziroma so definirane. Potrebne informacije pridobi v datoteki `AndroidManifest.xml`, v kateri razvijalec aplikacije navede vse uporabljene komponente v aplikaciji. Poleg deklaracije komponent aplikacije ima datoteka *Manifest* še nalogo:

- prepoznavanja uporabnikovih dovoljenj, ki jih aplikacija zahteva (npr. dovoljenje za dostop do internetnega omrežja),
- določanja minimalne verzije operacijskega sistema, ki ga aplikacija potrebuje za pravilno delovanje,
- deklaracije strojne in programske opreme, ki jo aplikacija uporablja (npr. uporaba mikrofona).

Več o datoteki *Manifest file* in njenih nalogah lahko najdemo na spletni strani [15].

4.2.3 Viri (angl. *Resources*)

Poleg kode, Android aplikacije za delovanje potrebujejo tudi vire (angl. *resources*), kot so slike, zvočne datoteke in vse, kar je povezano z vizualno predstavitvijo aplikacije. S pomočjo spreminjanja XML datotek viri omogočajo urejanje menijev, barv, animacij in kar je najpomembnejše, postavitev uporabniškega vmesnika posamezne aktivnosti. Z uporabo virov lahko torej spreminjamo izgled aplikacije brez poseganja v izvorno kodo. Uporaba dodatnih virov omogoča tudi optimizacijo aplikacije za delovanje v različnih napravah. Več o virih Android aplikacije je opisano na strani [15].

Poglavje 5

Uporabljena orodja

Za razvoj aplikacije so uporabili Googlovo razvojno okolje Android studio, ki smo ga izbrali pred Microsoftovim Visual studijem predvsem zaradi obširne dokumentacije, ki je dostopna na spletnih straneh [4]. V nadaljevanju poglavja bomo podrobneje predstavili Android studio, njegove zmogljivosti, osnove delovanja ter izgled.

5.1 Android studio

Za razvoj aplikacije smo uporabili razvojno okolje Android studio, ki temelji na programski opremi IntelliJ IDEA, razviti s strani podjetja JetBrains. Na voljo za prenos je za operacijske sisteme Windows, MacOS in Linux. Nastalo je kot nadomestitev razvojnega okolja Eclipse, ki je bilo primarno razvojno okolje za Android aplikacije [7].

Najnovejša različica razvojnega okolja Android studio 3.1.4 zagotavlja veliko zmožnosti, med katerimi so tudi:

- **Vizualni urejevalnik postavitve** (angl. *visual layout editor*), ki omogoča hitro in enostavno urejanje postavitve po principu *drag-and-drop*, brez potrebe po ročnem urejanju XML kode. Urejevalnik omogoča tudi predogled izgleda v različnih napravah in različnih verzijah operacijskega sistema Android ter dinamično urejanje postavitve za različne

velikosti zaslona.

- **APK analizator** (angl. *Android Package Manager Analyzer*), ki omogoča neposreden vpogled v sestavo APK-ja po končanem postopku izgradnje (angl. *build*) aplikacije. Uprava APK analizatora omogoča enostavnejše in hitrejše razhroščevanje.

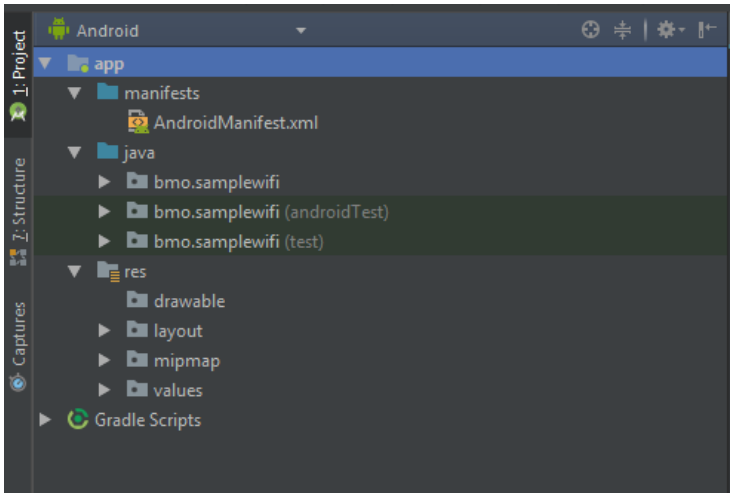
- **Zagon aplikacije z orodjem Android Emulator**, ki simulira delovanje Android naprave. Android Emulator je namenjen testiranju aplikacije, ne da bi za to potrebovali fizično napravo. Prednost njegove uporabe je simulacija katere koli različice operacijskega sistema Android, kar omogoča testiranje aplikacije na različnih napravah.

- **Pameten urejevalnik kode**, ki omogoča samodejno dokončanje kode za vse podprte programske jezike (Java, Kotlin, C++) [5].

5.1.1 Struktura projekta

Pri razvoju Android aplikacije je v Android studiu ta predstavljena kot projekt, ki je podoben projektom v ostalih razvojnih okoljih kot recimo Eclipse. Vsak projekt v Android studiu je sestavljen iz enega ali več modulov, ki vsebuje datoteke z izvorno kodo in vire. Moduli so razdeljeni v tri tipe, med katerimi je največkrat uporabljen *Android app module*.

Na Sliki 5.1 je prikazana struktura projekta v razvojnem okolju Android studio. Vse datoteke potrebne za gradnjo (podrobneje opisano v podpoglavju 5.1.3) so vidne znotraj mape `Gradle Scripts`, medtem ko je vsak *app module* sestavljen iz podmap `manifests`, `java` in `res`. V mapi manifest je shranjena datoteka `AndroidManifest.xml`, ki smo jo omenili v poglavju 4.2.2. Mapa `java` vsebuje javansko izvorno kodo aplikacije, medtem ko mapa `res` vsebuje datoteke z viri aplikacije, o katerih smo govorili v poglavju 4.2.3 [6].



Slika 5.1: Slika prikazuje strukturo projekta v Android Studiu.

5.1.2 Uporabniški vmesnik

Izgled Android studia in njegovo glavno okno lahko vidimo na Sliki 5.2. Glavno okno Android studia je sestavljeno iz več logičnih področij. Omenjeno razvojno okolje nam omogoča, da področja poljubno skrijemo in s tem pridobimo več prostora na področju, ki ga trenutno uporabljamo. Glavno okno je torej razdeljeno na sledeča področja, ki so prikazana na Sliki 5.2, kjer je pod zaporedno številko:

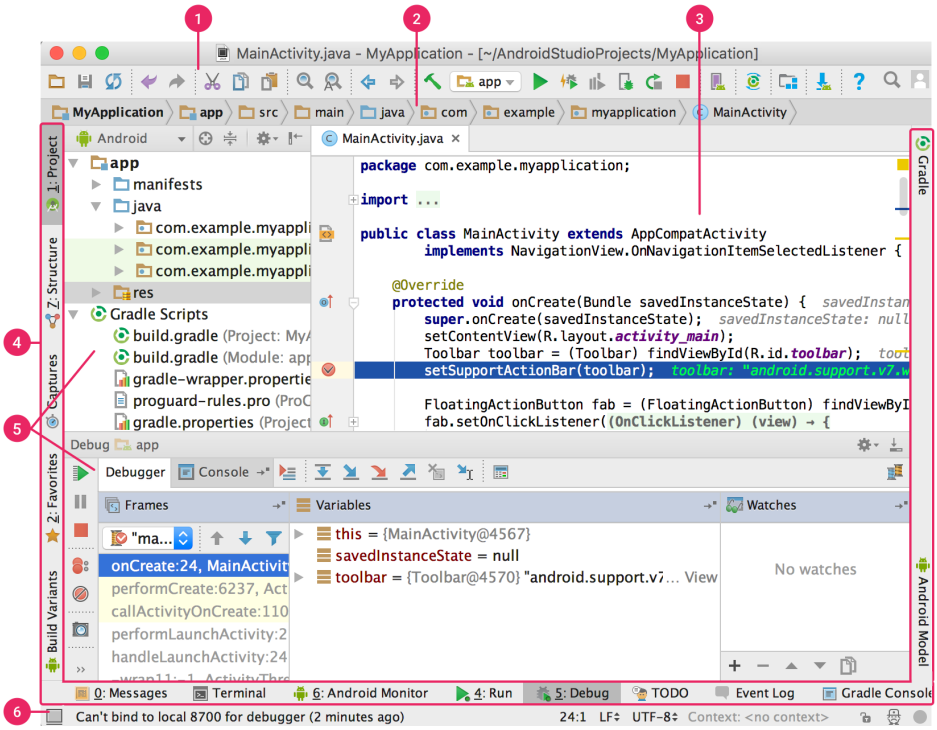
1. Orodna vrstica, ki omogoča izvedbo številnih dejanj, vključno z zagonom aplikacije in zagonom Android orodij.

2. Navigacijska vrstica, ki nam pomaga upravljati s projektom in z odpiranjem datotek za urejanje.

3. Okno urejevalnka, v katerem spreminjamo kodo.

4. Orodna vrstica, ki vsebuje gumbe, preko katerih aktiviramo želeno orodje.

5. Okno posameznega orodja.



Slika 5.2: Glavno okno razvojnega okolja Android studio [6].

6. Statusna vrstica, ki prikazuje status, obvestila in sporočila projekta ter razvojnega okolja [6].

5.1.3 Gradnja aplikacije

Po končanem urejanju izvorne kode in virov aplikacije in pred zagonom na napravi, jo je potrebno ustrezno prevesti in zapakirati s pomočjo APK. V Android studiu je postopek izveden s pomočjo orodja Gradle, ki je za namene Android aplikacij razvil tudi Android vtičnik. Postopku prevajanja in pakiranja na kratko rečemo tudi gradnja aplikacije (angl. *build*). Orodje Gradle nam med drugim omogoča, da na podlagi iste izvorne kode zgradimo aplikacijo za različne tipe naprav in različne verzije operacijskih sistemov Android [6].

Poglavje 6

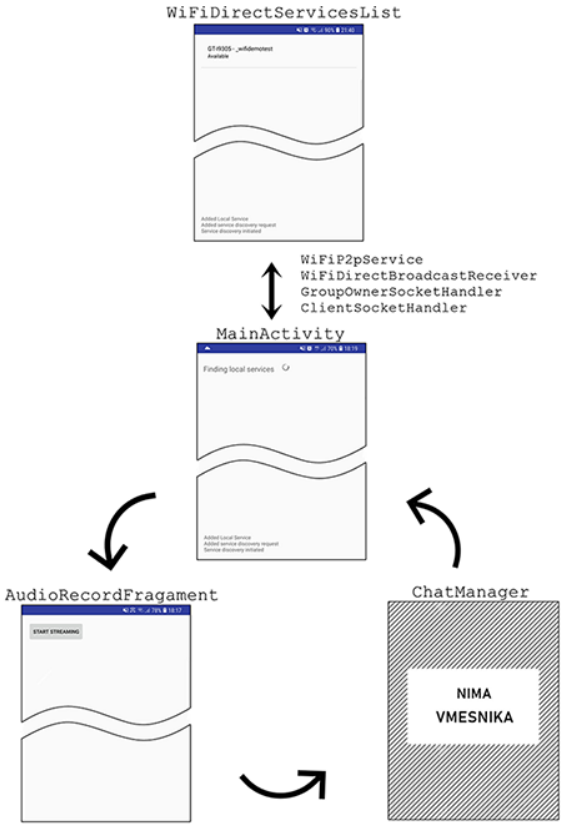
Implementacija aplikacije

6.1 Opis implementirane aplikacije

V okviru diplomske naloge bomo implementrirali Android aplikacijo (v nadaljevanju aplikacija), ki bo omogočala pošiljanje zvoka med dvema napravama preko protokola Wi-Fi Direct. Po uspešno vzpostavljeni povezavi, ki je bila implementirana že v okviru vzorčne aplikacije, opisane v uvodnem poglavju, je na zaslonu aplikacije prikazan gumb, ki ob kliku povzroči snemanje zvoka iz mikrofona. Do ponovnega klika na gumb se posneti podatki preko *socketa* pošiljajo napravi, s katero smo povezani, ta pa zvok sproti predvaja. Aplikacija torej spominja na komunikacijo pri Walkie-talkie napravah.

Poleg delovanja v načinu pošiljanja zvoka v realnem času, pa smo aplikacijo implementirali še s pomočjo shranjevanja posnetega zvoka v datoteko. Tako kot v prej opisanem načinu, je tudi v tem primeru vzpostavljanje povezave Wi-Fi Direct enako kot pri vzorčni aplikaciji. Razlika med aplikacijama je ta, da so pri drugem načinu podatki po uspešnem zajemanju zvoka iz mikrofona najprej shranjeni v datoteko, ki jo nato preko *socketa* pošljemo drugi napravi. Opis implementacije obeh različic aplikacije bomo podrobneje predstavili v podpoglavjih 6.4 in 6.5.

Apikacija je v Android studiu lahko sestavljena iz več javanskih razredov. Vse uporabljene razrede v naši aplikaciji lahko vidimo na Sliki 6.1.



Slika 6.1: Na sliki je prikazan diagram razredov naše aplikacije.

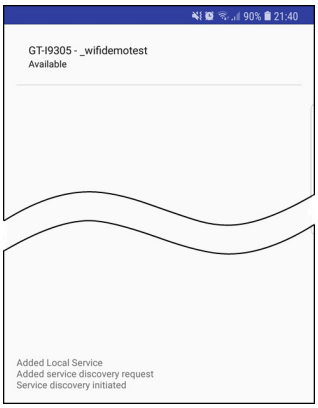
Glavna aktivnost, v kateri je implementirana vzpoztavitev Wi-Fi Direct povezave, je implementirana v razredu `MainActivity`, ki kot pomožne razrede za vzpostavitev povezave uporablja `WiFiDirectBroadcastReceiver`, `WiFiP2pService`, `GroupOwnerSocketHandler` in `ClientSocketHandler`. Najpomembnejši med pomožnimi razredi je razred `ChatManager`, v katerem je implementirano zajemanje zvoka iz mikrofona in pošiljanje v *socket*. V omenjenem razredu je implementirano tudi branje in predvajanje podatkov, kar bomo podrobneje pogledali v podpoglavju 6.4.

Poleg naštetih razredov pa aplikacijo sestavljata tudi fragmenta `WiFiDirectServicesList` in `AudioRecordFragment`, ki ju bomo poleg glavne

aktivnosti predstavili v naslednjem podpoglavju.

6.2 Opis aktivnosti in fragmentov aplikacije

Ob zagonu aplikacije se zažene glavna aktivnost aplikacije, katere primarna naloga je vzpostavitev Wi-Fi Direct povezave s pomočjo pomožnih razredov, ki smo jih našteli v prejšnjem poglavju. Po uspešnem odkrivanju naprav v bližini, glavna aktivnost zažene fragment `WiFiDirectServicesList`, ki omogoča prikazovanje seznama vseh naprav, s katerimi se lahko uporabnik poveže. Prikaz seznama naprav lahko vidimo na Sliki 6.2. Ob kliku na eno od prikazanih naprav se iz fragmenta kliče ustrezna metoda, ki poveže napravi preko protokola Wi-Fi Direct.



Slika 6.2: Izgred fragmenta `WiFiDirectServicesList.java`.

Po uspešno vzpostavljeni povezavi med napravama, glavna aktivnost zažene fragment `AudioRecordFragment`. Ta je zadolžen za prikazovanje gumba *Start streaming* in izvajanje kode ob kliku na le-tega. Njegov izgled smo že videli na Sliki 2.1.

Dodajanje gumba v fragment smo implementirali kot je prikazano na Sliki 6.3. Uporabljen je bil torej razred `android.widget.LinearLayout`, ki vsebuje metodo `addView (...)`. Z omenjeno metodo smo na fragment

dodali gumb za začetek in konec snemanja podatkov, in sicer tako, da metodi kot parameter posredujemo objekt tipa `RecordButton`. Objekti razreda `RecordButton` razširjajo razred `android.widget.Button`, ki je namenjen prikazovanju elementa na katerega lahko uporabnik klikne in s tem povzroči določeno akcijo. Razred `android.widget.Button` torej implementira vse potrebne metode in konstruktorje za delo z gumbi.

```
LinearLayout ll = new LinearLayout(getActivity());
mRecordButton = new RecordButton(getActivity());
ll.addView(mRecordButton,
        new LinearLayout.LayoutParams(
                ViewGroup.LayoutParams.WRAP_CONTENT,
                ViewGroup.LayoutParams.WRAP_CONTENT,
                 weight: 0));

return ll;
```

Slika 6.3: Slika prikazuje kodo za dodajanje gumba v fragment.

6.3 Dodajanje pravic v datoteko *manifest*

Aplikacija v operacijskem sistemu Android mora imeti za uporabo občutljivejših sistemskih funkcij ustrezne pravice. V podpoglavju 4.2.2 smo omenili, da je potrebno dovoljenja za delovanje aplikacije vključiti v datoteko `AndroidManifest.xml`. Aplikacija, ki jo bomo implementirali v okviru diplomske naloge za svoje delovanje zahteva sledeče pravice, ki jih lahko vidimo tudi na Sliki 6.4:

- *ACCESS_NETWORK_STATE*, *CHANGE_NETWORK_STATE* in *INTERNET*, za dostop in spreminjanje informacij o omrežju ter izvajanje omrežnih ukazov.
- *ACCESS_WIFI_STATE* in *CHANGE_WIFI_STATE*, za dostop in spreminjanje informacij o brezžičnih omrežjih.
- *READ_PHONE_STATE*, za dostop do informacij o mobilni napravi.
- *RECORD_AUDIO*, za omogočanje snemanja zvoka iz mikrofona.

- *WRITE_EXTERNAL_STORAGE*, za pisanje na zunanji pomnilnik, ki ga bomo potrebovali za shranjevanje datoteke pri drugem načinu implementacije.

- *MODIFY_AUDIO_SETTINGS*, za spreminjanje zvočnih nastavitev.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="bmo.samplewifi">

    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />

    <uses-feature...>

    <application...>

</manifest>
```

Slika 6.4: Vsebina datoteke `AndroidManifest.xml`.

Brez naštetih dovoljenj nobena izmed različic naše aplikacije ne bi mogla delovati pravilno. Nekatere od naštetih pravic ob prvem zagonu aplikacije zahtevajo dodatno potrditev uporabnika.

6.4 Aplikacija za pošiljanje zvoka v realnem času

Glavni cilj doplomske naloge je implementirati aplikacijo za pošiljanje zvoka v realnem času. Zato bomo v naslednjih podpoglavjih predstavili implementacijo snemanja zvoka iz mikrofona, pošiljanje in prejemanje podatkov preko *socketa* ter predvajanje zvoka.

6.4.1 Snemanje zvoka

Snemanje zvoka iz mikrofona se začne z uporabnikovim klikom na gumb *Start streaming*, ki povzroči klic metode `startMic ()` iz razreda `ChatManager`. Njeno izvorno kodo lahko vidimo na Sliki 6.5.

```
public void startMic () {
    mic = true;
    Thread thread = new Thread((Runnable) () → {
            AudioRecord audioRecorder = new AudioRecord (MediaRecorder.AudioSource.MIC,
                    SAMPLE_RATE, AudioFormat.CHANNEL_IN_MONO, AudioFormat.ENCODING_PCM_16BIT,
                     bufferSizeInBytes: AudioRecord.getMinBufferSize(SAMPLE_RATE,
                            AudioFormat.CHANNEL_IN_MONO, AudioFormat.ENCODING_PCM_16BIT)*10);
            int bytes_read = 0;
            byte[] buf = new byte[BUF_SIZE];
            try {
                audioRecorder.startRecording();
                while(mic) {
                    bytes_read = audioRecorder.read(buf,  offsetInBytes: 0, BUF_SIZE);
                    oStream.write(buf,  off: 0, bytes_read);
                }
                audioRecorder.stop();
                audioRecorder.release();
                mic = false;
                return;
            } catch(IOException e) {
                Log.e(TAG,  msg: "IOException: " + e.toString());
                mic = false;
            }
    });
    thread.start();
}

public void muteMic() {
    mic = false;
}
```

Slika 6.5: Implementacija metod `void startMic ()` in `void muteMic ()`.

Snemanje je implementirano z uporabo knjižnice `android.media.AudioRecord`. Ob kreiranju objekta tipa `AudioRecord` lahko preko parametrov konstruktorju posredujemo podatke o izvoru zvoka, frekvenci vzorčenja zvoka v hercih, formatu zvoka in velikosti medpomnilnika v bajtih. Po uspešni inicializaciji objekta lahko snemanje zvoka začnemo s klicom metode `startRecording ()`. Snemanje zvoka je lahko potratna operacija v smislu, da mora nit, na kateri deluje, ostati ves čas aktivna. To je razlog, da smo za njegovo izvedbo uporabili ločeno nit (angl. *thread*) in ne nit glavne aktivnosti.

Podobno kot pri prvem uporabnikovem kliku na gumb *Start streaming* se ob naslednjem kliku, s katerim naznanimo zaključek snemanja zvoka, kliče metoda `muteMic ()`. Ta nastavi vrednost spremenljivke `mic` na *false* in s tem povzroči prekinitev izvajanja zanke znotraj metode `startMic ()`. Posledično se kličeta metodi `stop ()` in `release ()`, po izvedbi katerih je snemanje zaključeno.

Na Sliki 6.5 lahko znotraj zanke opazimo klic metode `write (...)` iz razreda `ChatManager`, katere pomen bomo pojasnili v naslednjem podpoglavju.

6.4.2 Pošiljanje in prejemanje podatkov preko *socketa*

Pri načinu implementacije aplikacije, ki ga trenutno opisujemo, je potrebno napravi na drugi strani posnete podatke pošiljati sproti. Odločili smo se, da za pošiljanje in prejemanje podatkov preko protokola Wi-Fi Direct uporabimo *socket*. Ta namreč omogoča branje in pisanje podatkov s pomočjo knjižnic `java.io.InputStream` in `java.io.OutputStream`. Za pisanje podatkov v *socket* nam knjižnica `java.io.InputStream` ponuja metodo `write (...)`, ki kot parametre sprejme posnete podatke v obliki tabele bajtov, podatek o odmiku ter podatek o številu bajtov, ki bodo poslani.

Naprava na drugi strani bere podatke znotraj metode `run ()` razreda `ChatManager`. Implementacijo metode v naši aplikaciji lahko vidimo na Sliki 6.6. Na začetku metode so bile inicializirane vse potrebne spremenljivke za branje in pisanje podatkov preko *socketa*. Inicializiran je bil tudi objekt tipa `AudioTrack`, ki je uporabljen za predvajanje zvoka in ga bomo opisali v naslednjem podpoglavju.

Za branje podatkov smo uporabili metodo `int read (...)` razreda `InputStream`, ki sprejme enake argumente kot metoda `write (...)`, s to razliko, da gre v tem primeru za prebrane podatke. Metoda nam vrne število uspešno prebranih podatkov iz *socketa*, ki jih posredujemo objektu tipa `AudioTrack`.

```
private static final int SAMPLE_RATE = 44100; // Hertz
private static final int BUF_SIZE = 8192;

@Override
public void run() {
    try {
        iStream = socket.getInputStream();
        oStream = socket.getOutputStream();
        byte[] buffer = new byte[1024];
        int bytes = 0;

        handler.obtainMessage(MainActivity.MY_HANDLE,  obj: this)
                .sendToTarget();

        AudioTrack track = new AudioTrack(AudioManager.STREAM_MUSIC, SAMPLE_RATE,
                AudioFormat.CHANNEL_OUT_MONO,
                AudioFormat.ENCODING_PCM_16BIT, BUF_SIZE, AudioTrack.MODE_STREAM);
        track.play();
        while (true) {
            bytes = iStream.read(buffer,  off: 0, buffer.length);
            if (bytes == -1) break;
            track.write(buffer,  offsetInBytes: 0, bytes);
        }
        track.stop();
        track.flush();
        track.release();

    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Slika 6.6: Implementacija metode `run ()`.

6.4.3 Predvajanje zvoka

Glede na to, da poskušamo predvajati zvok v realnem času, moramo prebrane podatke iz *socketa* predvajati takoj, torej brez shranjevanja. Zaradi zmožnosti delovanja v t. i. *streaming* načinu smo za predvajanje zvoka uporabili knjižnico `android.media.AudioTrack`. Želen način predvajanja smo določili preko konstruktorja, s katerim določimo tudi frekvenco vzorčenja v hercih, nastavitve prenosnega kanala, format zvoka in velikost medpomnilnika v bajtih. Predvajanja zvoka lahko začnemo s klicem metode `play ()`, same podatke za predvajanje pa objektu tipa `AudioTrack` podamo s pomočjo metode `write ()`. Ko so vsi podatki prebrani, predvajanje zaključimo z za-

porednim klicem metod `stop ()`, `flush ()` in `release ()`. Predvajanje zvoka je implementirano znotraj metode `run ()`, ki je prikazana na Sliki 6.6.

6.5 Aplikacija za pošiljanje zvoka iz datoteke

Drugi način, ki smo ga uporabili za implementacijo aplikacije prav tako omogoča pošiljanje zvoka preko protokola Wi-Fi Direct. Gre za način pošiljanja s pomočjo datoteke. Aplikacija v tem primeru deluje tako, da podatke posnamemo, jih shranimo v datoteko in jih po končanem snemanju preko *socketa* pošljemo drugi napravi. V nadaljnjih podpoglavjih bomo predstavili tudi slednji način implementacije.

```
private void startRecording() {
    mRecorder = new MediaRecorder();
    mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
    mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    mRecorder.setOutputFile(mFileName);
    mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);

    try {
        mRecorder.prepare();
    } catch (IOException e) {
        Log.e(LOG_TAG, msg: "prepare() failed");
    }

    mRecorder.start();
}

private void stopRecording() {
    mRecorder.stop();
    mRecorder.release();
    mRecorder = null;
}
```

Slika 6.7: Implementacija metod `startRecording ()` in `stopRecording ()`.

6.5.1 Snemanje zvoka in shranjevanje v datoteko

V tem načinu implementacije je na fragment `AudioRecordFragment` dodan gumb *Start recording*, ki je objekt tipa `RecordButton`. Ob kliku na gumb za začetek snemanja se kliče metoda `startRecording ()`, ki implementira zajemanje zvoka iz mikrofona in sprotno shranjevanje v datoteko. Kot vidimo na

Sliki 6.7 je snemanje zvoka v tem načinu implementirano s pomočjo knjižnice `android.media.MediaRecorder`. Po kreiranem objektu tipa `MediaRecorder` najprej nastavimo izvor podatkov na mikrofon in nato določimo format izhodne datoteke. V naslednjem koraku določimo datoteko, v katero naj objekt shranjuje podatke (pred tem smo v spremenljivko `mFileName` shranili absolutno pot novo nastale datoteke), na koncu pa izberemo še tip kodiranja. S klicom metod `prepare ()` in `start ()` se začne snemanje in shranjevanje podatkov v datoteko.

Ob uporabnikovem kliku na gumb za konec snemanja najprej kličemo metodo `stopRecording ()`, ki s pomočjo metod vidnih na Sliki 6.7 zaključi snemanje iz mikrofona. V nadaljevanju sledita pretvorba datoteke v tabelo bajtov in pošiljanje, ki ju bomo predstavili v naslednjem podpoglavju.

6.5.2 Pošiljanje in prejemanje datoteke

Pri tem načinu implementacije moramo datoteko pred pošiljanjem pretvoriti v tabelo bajtov, šele nato jo lahko posredujemo razredu `ChatManager`, ki podatke zapiše v *socket*. Prevorba datoteke je implementirana s pomočjo metode `convert (String fName)`, ki preko parametra pridobi ime datoteke, vrne pa ustrezno generirano tabelo bajtov. Izvorno kodo metode za pretvorbo datoteke lahko preberemo na Sliki 6.8.

Postopek implementacje je sledeč. Najprej na podlagi imena datoteke, ki jo pretvarjamo, kreiramo nov objekt tipa `InputStream`, s pomočjo katerega bomo brali podatke iz datoteke. Kreiramo tudi tabelo bajtov ustrezne velikosti, nato pa kličemo pomožno metodo `toByteArray (InputStream in)`, ki nam vrne želeno tabelo. Za kreiranje tabele je v slednji metodi uporabljen objekt tipa `ByteArrayOutputStream`, kateremu posredujemo prebrane podatke iz datoteke. Na koncu s klicem metode `toByteArray ()` kreiramo želeno tabelo.

Ko je tabela bajtov uspešno kreirana, jo preko metode `write (byte[] buffer)` posredujemo razredu `ChatManager`. Ta s pomočjo `DataOutputStream` v *socket* najprej zapiše dolžino podatkov s klicem metode `writeLong`

```
public byte[] convert(String fName) {
    byte[] soundBytes = null;
    try {
        InputStream inputStream = getActivity().getContentResolver().
                openInputStream(Uri.fromFile(new File(fName)));
        soundBytes = new byte[inputStream.available()];

        soundBytes = toByteArray(inputStream);
    }catch(Exception e){
        e.printStackTrace();
    }
    return soundBytes;

}

public byte[] toByteArray(InputStream in) throws IOException {
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    int read = 0;
    byte[] buffer = new byte[1024];
    while (read != -1) {
        read = in.read(buffer);
        if (read != -1)
            out.write(buffer, off: 0,read);
    }
    out.close();
    return out.toByteArray();
}
```

Slika 6.8: Implementacija metod `convert (String fName)` in `toByteArray (InputStream in)`.

`(len)`, kjer je `len` spremenljivka tipa `long`, ki hrani dolžino podatkov, za tem pa še podatke same. Implementacijo metode za pisanje v *socket* vidimo na Sliki 6.9.

```
public void write(byte[] buffer) {
    try {
        long len = buffer.length;
        dos.writeLong(len);
        dos.write(buffer);
    } catch (IOException e) {
        Log.e(TAG, msg: "Exception during write", e);
    }
}
```

Slika 6.9: Implementacija metode `write ()`.

Prejemanje datoteke je implementirano znotraj metode `run ()` v razredu `ChatManager`, podobno kot pri prvi različici aplikacije. Implementacijo metode v tej različici vidimo na Sliki 6.10. V tem primeru je uporabljen razred

```
@Override
public void run() {
    try {
        dis = new DataInputStream(socket.getInputStream());
        dos = new DataOutputStream(socket.getOutputStream());
        byte[] buffer = new byte[8192];
        int bytes = 0;
        long length = 0;
        long lenTmp = 0;
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        handler.obtainMessage(MainActivity.MY_HANDLE,  obj: this)
                .sendToTarget();

        while (true) {
            if (length <= 0 ) length = dis.readLong();
            if (length > 0 ) {
                bytes = dis.read(buffer);
                if (bytes == -1) break;
                lenTmp += bytes;
                out.write(buffer,  off: 0, bytes);
                if (lenTmp == length) {
                    byte[] finalArr = out.toByteArray();
                    handler.obtainMessage(MainActivity.MESSAGE_READ,
                            finalArr.length,  arg2: -1, finalArr).sendToTarget();
                    out.reset();
                    length = 0;
                    lenTmp = 0;
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Slika 6.10: Implementacija metode `run ()`.

`DataInputStream`, saj ta omogoča branje različnega tipa podatkov iz *socketa*. Metoda `readLong ()` nam namreč omogoča, da najprej preberemo podatek o številu bajtov, ki jih pricakujemo (velikost datoteke), nato pa s pomočjo metode `read (byte[] buffer)` preberemo še podatke datoteke. Ko se število prebranih bajtov ujema z velikostjo datoteke, s pomočjo metode `toByteArray ()` razreda `ByteArrayOutputStream` kreiramo tabelo in jo posredujemo fragmentu `AudioRecordFragment`. Ta iz tabele ponovno tvori datoteko in jo nato predvaja. Podatke razredu `AudioRecordFragment` posredujemo z uporabo metod `obtainMessage (...)` in `sendToTarget ()`, ki podatke posredujeta glavni aktivnosti. Znotraj glavne aktivnosti je implementirana metoda `handleMessage (Message msg)`, ki nato podatke po-

sreduje fragmentu `AudioRecordFragment`. V naslednjem poglavju bomo podrobneje pogledali pretvorbo podatkov nazaj v datoteko in predvajanje le-te.

6.5.3 Shranjevanje prejetih podatkov in predvajanje posnetka

Ob klicu metode `onPlay (byte[] buff)` iz glavne aktivnosti v fragmentu `AudioRecordFragment` najprej iz prejetih podatkov generiramo datoteko s pomočjo metode `buildAudioFileFromReceivedBytes (byte[] bytes)`. Ta nam vrne ime novo kreirane datoteke, ki ga nato posredujemo metodi `playAudio (String outputFile)`, ki datoteko predvaja. Implementacija omejenih metod je predstavljena na Sliki 6.11.

Pred kreiranjem datoteki najprej določimo ime, ki vsebuje tudi celotno pot datoteke v pomnilniku. Za pisanje podatkov v novo nastalo datoteko smo uporabili razred `FileOutputStream`, saj ta omogoča shranjevanje celotne tabele bajtov v datoteko s pomočjo klica metode `write (byte[] bytes)`.

Po uspešnem kreiranju datoteke, smo v tem načinu implementacije za predvajanje uporabili knjižnico `android.media.MediaPlayer`. Pri uporabi slednje je pred predvajanjem potrebno najprej ustrezno nastaviti izvor podatkov. Odločili smo se, da datoteko predvajamo asinhrono, kar omogočimo s klicem metode `prepareAsync ()`. Asinhrono predvajanje smo izbrali zaradi uporabe ločene niti, kar omogoča nemoteno delovanje glavne niti, na kateri se izvaja fragment. Predvajanje datoteke začnemo s klicem metode `start ()`, zaključimo pa z metodama `reset ()` in `release ()`.

6.6 Scenarij uporabe aplikacije

V tem poglavju bomo predstavili primer uporabe aplikacije v načinu predvajanja podatkov v realnem času. Scenarij uporabe je prikazan na Sliki 6.12 in je sledeč:

1. Ob zagonu aplikacije se zažene glavna aktivnost, ki išče naprave, s

```
public void onPlay(byte[] buff) {
    String newFileName = buildAudioFileFromReceivedBytes(buff);
    playAudio(newFileName);
}

@NonNull
private String buildAudioFileFromReceivedBytes(byte[] bytes) {
    String outputFile=Environment.getExternalStorageDirectory()
            .getAbsolutePath() + "/output.3gp";
    File path = new File(outputFile);
    FileOutputStream fos = null;
    try {
        fos = new FileOutputStream(path);
        fos.write(bytes);
        fos.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return outputFile;
}

public void playAudio (String outputFile){
    MediaPlayer mediaPlayer = new MediaPlayer();
    try {
        mediaPlayer.setDataSource(outputFile);
        mediaPlayer.prepareAsync();
    } catch (IOException e) {
        e.printStackTrace();
    }
    mediaPlayer.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
        @Override
        public void onPrepared(MediaPlayer mp) {
            mp.start();
        }
    });
    mediaPlayer.setOnCompletionListener((mp) -> {
            mp.reset();
            mp.release();
    });
}
```

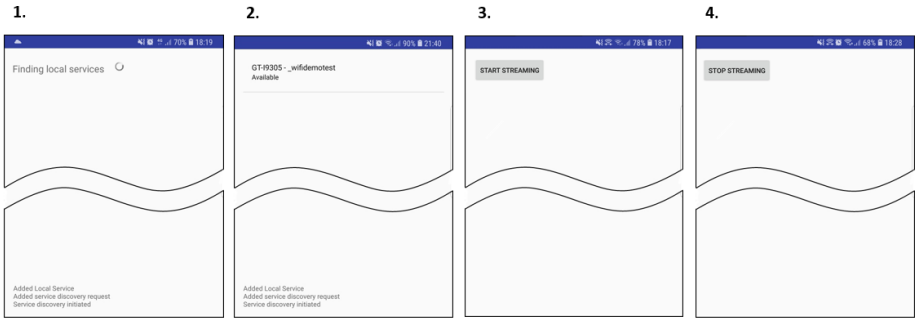
Slika 6.11: Implementacija metod `onPlay (byte[] buff)`, `buildAudioFileFromReceivedBytes (byte[] bytes)` in `playAudio (String outputFile)`.

katerimi se lahko neposredno povežemo.

2. V primeru najdene naprave se ta prikaže na zaslonu in uporabnik lahko s klikom nanjo vzpostavi povezavo.

3. Ko sta napravi povezani, se na zaslonu prikaže gumb za začetek snemanja in oddajanja zvoka.

4. Po začetku snemanja se ime gumba spremeni v *Stop streaming*, uporabnik pa ob kliku nanj prekine snemanje iz mikrofona.



Slika 6.12: Slika prikazuje scenarij uporabe aplikacije.

Naprava na drugi strani začne ob začetku snemanja prve naprave samodejno predvajati prejete podatke. Ko je prejemanje zaključeno, lahko uporabnik druge naprave s klikom na gumb *Start streaming* začne s snemanjem in pošiljanjem zvoka.

6.7 Težave pri implementaciji aplikacije

Spoznali smo delovanje in dva različna načina implementacije naše aplikacije. Sedaj bomo na kratko opisali še težave, ki so se pojavile pri implementaciji posameznega načina.

Ob implementaciji aplikacije, ki omogoča pošiljanje v realnem času, nam je največ težav povzročala izbira Android knjižnic, ki smo jih želeli uporabili. Aplikacijo smo želeli implementirati s pomočjo razredov `AudioStream` in `AudioGroup`. Razred `AudioStream` je različica *RTP streama* (angl. *Real-time Transfer Protocol stream*), ki naj bi z združevanjem v `AudioGroup` omogočala snemanje in predvajanje zvoka na napravi. `AudioGroup` namreč pri delovanju uporablja zanko, ki je sestavljena iz več delov, med katerimi sta tudi branje podatkov iz mikrofona in predvajanje. Težave so se pojavile pri pošiljanju podatkov drugi napravi preko *socketa*, saj posnetih podatkov nismo uspeli poslati do druge naprave, da bi jih ta lahko predvajala.

Pri načinu pošiljanja zvoka s pomočjo datoteke pa nam je največ težav

povzročalo usklajevanje števila poslanih bajtov s številom bajtov, ki jih `ChatManager` posreduje fragmentu `AudioRecordFragment`. Problem se je pojavil pri branju podatkov iz *socketa* s pomočjo metode `read (...)`, ki je predčasno zaznala konec prejetih podatkov. Posledično so bili podatki fragmentu poslani preden so bili vsi prejeti, kar je povzročilo kasnejše motnje pri predvajanju. Težavo smo odpravili s predhodnim pošiljanjem števila bajtov (velikosti datoteke), ki bodo poslani preko *socketa*. Na ta način podatki niso poslani v fragment, dokler se število prebranjih podatkov ne ujema z številom bajtov, ki ji pričakujemo.

Poglavje 7

Sklepne ugotovitve

Na začetku izdelave diplomske naloge smo si zadali cilj, da implementiramo aplikacijo, ki bo prenašala zvok v realnem času. Poleg tega smo želeli implementirati tudi aplikacijo, ki prenaša zvok s pomočjo shranjevanja v datoteko. Namen implementacije na oba načina je bil primerjati zakasnitev in delovanje aplikacij glede na oddaljenost naprav, ki sta povezani preko protokola Wi-Fi Direct.

Za ugotavljanje razlik med zakasnitvami glede na način implementacije, smo se odločili za primerjavo razlik med časom začetka snemanja in časom začetka predvajanja. Kot vidimo v prvi tabeli na naslednji strani, so pri načinu pošiljanja v realnem času zakasnitve v vseh primerih manjše od zakasnitev pri drugi različici aplikacije. Takšne rezultate smo tudi pričakovali, saj je pri prvem načinu implementirano pošiljanje v realnem času, zato se tudi predvajanje začne prej kot pri pošiljanju datoteke. Naslednja hipoteza, ki smo jo postavili pred opravljanjem meritev je bila, da bodo razlike pri prvem načinu implementacije razmeroma konstantne in izkazalo se je, da se predvajanje pri tej različici vedno začne približno sekundo po začetku snemanja.

Na podlagi drugega stolpca tabele lahko razberemo, da so pri aplikaciji za pošiljanje zvoka s pomočjo datoteke, razlike v časih občutno večje in bolj nekonstantne kot pri prvem načinu implementacije. Takšne rezultate smo predvideli že pred začetkom izdelave aplikacij, saj se pri drugem načinu

implementacije predvajanje začne šele po zaključku snemanja. Začetek predvajanja je torej odvisen od dolžine posnetka, ki ga pošiljamo, kar posledično privede do večje zakasnitve pri daljših posnetkih.

Meritev št.	**Zakasnitev v realnem času** [ms]	**Zakasnitev pri pošiljanju posnetka**[ms]
1	1939	5397
2	1679	2310
3	1657	1961
4	1767	13290
5	1589	7544

Kot lahko razberemo iz zgornje tabele, je zakasnitev pri prvem načinu implementacije aplikacije v povprečju dolga približno eno sekundo. Ta čas je sorazmerno velik, zato smo se odločili, da zakasnitev pri prvem načinu izmerimo še pri zmanjšani velikosti medpomnilnika. Velikost medpomnilnika na pošiljateljevi strani smo spremenili iz 1600 bajtov na 1024 bajtov. Poleg tega smo zmanjšali tudi medpomnilnik na prejemnikovi strani, in sicer iz 8192 bajtov na 1024 bajtov. Izboljšane rezultate vidimo v spodnji tabeli.

Meritev št.	**Zakasnitev v realnem času** [ms]
1	186
2	349
3	207
4	179
5	386

Več kot očitno je sprememba velikosti medpomnilnika močno izboljšala čas zakasnitve pri aplikaciji za pošiljanje zvoka v realnem času. Zakasnitev je sedaj v povprečju velika približno 200 ms in je zadovoljiva za nemoteno delovanje naše aplikacije.

Po opravljanju meritev za ugotavljanje zakasnitev smo preverili še delovanje aplikacije glede na oddaljenost naprav. V teoriji naj bi protokol Wi-Fi

Direct omogočal delovanje med napravama oddaljenima do 100 metrov na prostem in do 30 metrov v zaprtih prostorih. Po opravljenem testu delovanja naše aplikacije glede na oddaljenost smo ugotovili, da napravi lahko brez ovir komunicirata na razdalji okoli 30 metrov, medtem ko je v primeru fizičnih ovir (npr. zid), ta delovala pri razdalji največ 10 metrov. Tako kot pri večini standardov, so tudi v našem primeru teoretične omejitve nekoliko višje od tistih, ki jih lahko v praksi dosežemo. Razlog za to je zagotovo testiranje protokola v tako rekoč idealnih pogojih, posledica česar so tudi nekoliko daljše razdalje pri katerih naj bi ta deloval. Nekoliko slabše rezultate od pričakovanih si zagotovo lahko razlagamo tudi z uporabo starejše naprave pri testiranju.

7.1 Možne nadgradnje

Kot smo že v prejšnjem podpoglavju omenili, je zakasnitev pri prvem načinu implementacije aplikacije v povprečju dolga približno 200 ms. Človeško uho zakasnitev zazna šele pri času večjem od 200 ms, zato lahko rečemo da čas zakasnitve pri naši aplikaciji ni velik. Bi pa lahko ne glede na sorazmerno majhen čas zakasnitve pri predvajanju, tega z nadaljnjim razvojem aplikacije še nekoliko zmanjšali.

Eden od predlogov za zmanjšanje zakasnitve bi bila zagotovo uporaba razreda `DatagramSocket`, ki deluje preko protokola UDP. Na ta način bi čas zakasnitve zagotovo še nekoliko izboljšali, saj smo pri naši aplikaciji uporabili *socket*, ki deluje preko protokola TCP. Bi pa seveda v primeru uporabe `DatagramSocketa` nekoliko izgubili na kvaliteti zvoka, saj protokol UDP ne zahteva potrditev prejemanja paketov.

Možna nadgradnja bi bila zagotovo tudi sprememba uporabniškega vmesnika. Namesto trenutnega gumba z napisom *Start streaming*, bi lahko s pomočjo grafičnega oblikovanja izdelali gumb, ki bi omogočal začetek snemanja. Prav tako bi lahko ob začetku oddajanja na eni napravi, uporabniku na drugi strani onemogočili klik na gumb za začetek snemanja. V trenu-

tnem načinu implementacije bi namreč lahko oba uporabnika hkrati pričela s snemanjem in oddajanjem zvoka, kar bi privedlo do prekinitve delovanja aplikacije.

Poglavje 8

Zaključek

V sklopu diplomske naloge smo razvili mobilno aplikacijo za operacijski sistem Android, ki uporabnikom omogoča zvočno komunikacijo. Podatki se pošiljajo preko protokola Wi-Fi Direct, zato smo v diplomskem delu na kratko predstavili osnove omenjenega protokola. Na kratko smo opisali tudi operacijski sistem Android, največji del pa smo posvetili predstavitvi implementacije obeh različic naše aplikacije ter ugotovitvam, do katerih smo prišli.

Postopek implementacije in programiranja je bil sledeč: najprej smo na vzorčni aplikaciji za pošiljanje tekstovnih sporočil ustrezno spremenili uporabniški vmesnik. Nato smo aplikaciji dodali možnost pošiljanja zvoka s pomočjo shranjevanja v datoteko. Po uspešnem delovanju v tem načinu pa smo razvili še aplikacijo za pošiljanje podatkov v realnem času.

Glede na cilje, ki smo si jih zadali pred izdelavo diplomske naloge lahko rečemo, da smo jih uspešno realizirali. Uspešno smo implementirali obe različici aplikacije in na koncu naredili primerjavo želenih elementov. Trenutno implementirani apilkaciji pa bi seveda lahko še nadgradili v skladu s predlogi, opisanimi v prejšnjem poglavju.

Literatura

[1] Wi-Fi Alliance. Wi-fi certified wi-fi direct. *White paper*, 2010. Dosegljivo: `https://www.broadcom.cn/wp-content/uploads/2013/10/Wi-Fi-Direct-White-Paper.pdf`. [Dostopano: 19. 8. 2018].

[2] Android intent and intent filters. Dosegljivo: `https://developer.android.com/guide/components/intents-filters`. [Dostopano: 15. 8. 2018].

[3] Android (operating system). Dosegljivo: `https://en.wikipedia.org/wiki/Android_(operating_system)`. [Dostopano: 14. 8. 2018].

[4] Android page. Dosegljivo: `https://developer.android.com/`. [Dostopano: 19. 8. 2018].

[5] Android studio. Dosegljivo: `https://developer.android.com/studio/`. [Dostopano: 19. 8. 2018].

[6] Android studio intro. Dosegljivo: `https://developer.android.com/studio/intro/`. [Dostopano: 19. 8. 2018].

[7] Android studio (wikipedia). Dosegljivo: `https://en.wikipedia.org/wiki/Android_Studio`. [Dostopano: 19. 8. 2018].

[8] Michiel Appelman, Jeffrey Bosma, and Gerrie Veerman. Viber communication security. *System and network of engineering, university of Amsterdam, Netherlands*, 2011.

[9] Salman A Baset and Henning Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. *arXiv preprint cs/0412017*, 2004.

[10] Android Developers. What is android. Dosegljivo: `http://www.academia.edu/download/30551848/andoid--tech.pdf`, 2011. [Dostopano: 15. 8. 2018].

[11] Face time. Dosegljivo: `https://en.wikipedia.org/wiki/FaceTime`. [Dostopano: 7. 8. 2018].

[12] Nisarg Gandhewar and Rahila Sheikh. Google android: An emerging software platform for mobile devices. *International Journal on Computer Science and Engineering*, 1(1):12–17, 2010.

[13] How skype works. Dosegljivo: `https://www.digitaltrends.com/web/how-does-skype-work`, 2013. [Dostopano: 7. 8. 2018].

[14] Miracast. Dosegljivo: `https://en.wikipedia.org/wiki/Miracast`. [Dostopano: 11. 8. 2018].

[15] Osnove android aplikacij). Dosegljivo: `https://developer.android.com/guide/components/fundamentals`. [Dostopano: 15. 8. 2018].

[16] Pragati Paliwal and Shikha Rajoriya. Android operating system. *GADL J. Recent Innov. Eng. Technol*, 1:1–4, 2016.

[17] Skype. Dosegljivo: `https://en.wikipedia.org/wiki/Skype`. [Dostopano: 7. 8. 2018].

[18] Skype protocol. Dosegljivo: `https://en.wikipedia.org/wiki/Skype_protocol`. [Dostopano: 7. 8. 2018].

[19] `http://lh6.ggpht.com/_X6JnoL0U4BY/S4TyokObWhI/AAAAAAAAURk/R50jJGXV1a4/s1600-h/tmp673_thumb13.jpg`. [Dostopano: 7. 8. 2018].

[20] Store and forward. Dosegljivo: https://en.wikipedia.org/wiki/Store_and_forward. [Dostopano: 7. 8. 2018].

[21] Viber. Dosegljivo: https://en.wikipedia.org/wiki/Viber. [Dostopano: 7. 8. 2018].

[22] Voice over ip. Dosegljivo: https://en.wikipedia.org/wiki/Voice_over_IP. [Dostopano: 7. 8. 2018].

[23] Whatsapp messenger. Dosegljivo: https://en.wikipedia.org/wiki/WhatsApp. [Dostopano: 7. 8. 2018].

[24] Device to device communications with wi-fi direct: overview and experimentation. Dosegljivo: http://www.it.uc3m.es/pablo/papers/pdf/2012_camps_commag_wifidirect.pdf. [Dostopano: 7. 8. 2018].

[25] Wi-fi direct. Dosegljivo: https://en.wikipedia.org/wiki/Wi-Fi_Direct. [Dostopano: 11. 8. 2018].

[26] https://png.icons8.com/metro/1600/wi-fi-direct.png. [Dostopano: 11. 8. 2018].