Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

Nejc Novak

SPROTNI PRENOS VIDEO VSEBINE MED MOBILNIMI PLATFORMAMI Z OPERACIJSKIM SISTEMOM ANDROID

Diplomsko delo

Maribor, avgust 2016

SPROTNI PRENOS VIDEO VSEBINE MED MOBILNIMI PLATFORMAMI Z OPERACIJSKIM SISTEMOM ANDROID

Diplomsko delo

Študent:	Nejc Novak
Študijski program:	Visokošolski študijski program Računalništvo in informacijske tehnologije
Mentor:	izr. prof. dr. Aleš Holobar, dipl. inž. rač. in inf.
Lektorica:	Margit Berlič Ferlinc s. p., prof. angleščine in slovenščine.

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

Številka: E1066791
Datum in kraj: 13. 04. 2016, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 44/2015)
izdajam

SKLEP O DIPLOMSKEM DELU

1. **Nejcu Novaku,** študentu visokošolskega strokovnega študijskega programa RAČUNALNIŠTVO IN INFORMACIJSKE TEHNOLOGIJE, se dovoljuje izdelati diplomsko delo.

2. **MENTOR:** **izr. prof. dr. Aleš Holobar**

3. **Naslov diplomskega dela:**
SPROTNI PRENOS VIDEO VSEBINE MED MOBILNIMI PLATFORMAMI Z OPERACIJSKIM SISTEMOM ANDROID

4. **Naslov diplomskega dela v angleškem jeziku:**
REAL-TIME VIDEO TRANSMISSION BETWEEN MOBILE PLATFORMS WITH ANDROID OPERATING SYSTEM

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije do 30. 09. 2016 v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

UNIVERZA V MARIBORU FAKULTETA ZA ELEKTROTEHNIKO, RAČUNALNIŠTVO IN INFORMATIKO 8

Dekan:
red. prof. dr. Borut Žalik

Obvestiti:
- kandidata,
- mentorja,
- odložiti v arhiv.

ZAHVALA

Zahvaljujem se mentorju, izr. prof. Alešu Holobarju, za svetovanje, potrpežljivost in vodenje do celotnega zaključka diplomske naloge. Predvsem bi se mu zahvalil za usmerjanje pri izbiri teme, saj mi je bila izdelava diplomske naloge v veliko veselje.

Zahvaljujem se svoji družini in dekletu, ki me je podpiralo v času študija.

Hvala vsem sošolcem in sodelavcem za vse lepe trenutke.

Sprotni prenos video vsebine med mobilnimi platformami z operacijskim sistemom Android

Ključne besede: Android, optimizacija, video, kamera, sprotni prenos.

UDK: 621.397.7-026.26(043.2)

Povzetek

V diplomskem delu je bila izdelana aplikacija za sprotni prenos videa med dvema mobilnima napravama z vmesniki Bluetooth, WiFi ali WiFi Direct. Aplikacija deluje na operacijskem sistemu Android ter omogoča hitro in varno povezovanje med vmesniki v načinu strežnik-odjemalec. Prilagaja se specifikacijam kamere na snemalni mobilni napravi in lastnostim zaslona na mobilni napravi za predvajanje. Narejene so bile optimizacije, testiranja in analize prenosa video vsebin. Najkvalitetnejši in najhitrejši prenosi video posnetkov so prikazani v rezultatih testiranj.

Real-time video transmission between mobile platforms with Android operating system

Keywords: Android, optimization, video, camera, real-time transmission.

UDK: 621.397.7-026.26(043.2)

Abstract

In this thesis, an application was created for real-time video transmission between mobile platforms using wireless technologies Bluetooth, Wi-Fi and Wi-Fi Direct. The application runs on Android operating system with fast and safe connection between the server and the client. An adjustment in video transmission is made according to the cameras specifications on the recording mobile device and according to the screen properties on the displaying mobile device. Optimization and analysis of video transmission are reported. Finally, the implemented solutions supporting the fastest transmission and the highest video quality are discussed in details.

KAZALO

KAZALO SLIK

KAZALO TABEL

SEZNAM UPORABLJENIH SIMBOLOV IN KRATIC

SNS – slika na sekundo (angl. *frame per second*)

NTSC – nacionalni odbor za televizijski sistem (angl. *National Television System Comitee*)

RGBA – rdeča, zelena, modra in alfa kanal (angl. *red green blue alpha*)

RFCOMM – radijsko frekvenčna komunikacija (angl. *radio frequency communication*)

UUID – univerzalni unikaten identifikator (angl. *universally unique identifier*)

LAN – lokalno omrežje (angl. *local area network*)

IP – internetni protokol (angl. *internet protocol*)

TCP – protokol za nadzor prenosa (angl. *transmission control protocol*)

UDP – nepovezovalni protokol za prenos (angl. *user datagram protocol*)

WPA – zaščiten dostop WiFi (angl. *wifi protected access*)

UI – uporabniški vmesnik (angl. *user interface*)

API – aplikacijski programski vmesnik (angl. *application programming interface*)

JNI – javanski domorodni vmesnik (angl. *Java native interface*)

NDK – razvojno orodje, ki podpira povezave med javansko in domorodno kodo. (angl. *native development kit*)

SDK – programsko razvojno orodje (angl. *software development kit*)

1 UVOD

V zadnjih nekaj letih so pametni telefoni postali vsakodneven pripomoček človeka. Podpirajo toliko različnih tehnologij, da redkokomu ne bi bile uporabne. Dve izmed teh tehnologij sta zajem in predvajanje video posnetkov. Tehnologiji sta zelo grafično in procesorsko intenzivni, saj morata obdelati milijone podatkov na sekundo. V naslednjih nekaj odstavkih smo opisali pomembna dejstva in pojme, ki so uporabljeni v diplomski nalogi.

Slike na sekundo (angl. *frame per second*) so merilo kvalitete videa, ki je zajet s kamero. Kinematografski filmi uporabljajo zapis s 24 slikami na sekundo (SNS)[1], medtem ko televizija s standardom NTSC uporablja zapis s 29,97 SNS [2], [20]. Internet se že kar nekaj let osredotoča na video posnetke s 60 SNS, ker lahko običajni osebni računalniki brez težav predvajajo tako kvalitetne video posnetke. V diplomski nalogi smo se osredotočili na posnetke z do 30 SNS, saj uporabljene mobilne naprave ne podpirajo predvajanja video posnetkov nad 30 SNS [20].

Na velikost slike vpliva še barvna globina (angl. *color depth*), ki nam pove, koliko različnih barv lahko predstavimo na zaslonu računalnika oziroma mobilne naprave. Naprave, uporabljene v diplomski nalogi, podpirajo 32-bitne[2] zapise barv v formatu RGBA[3] in lahko predstavijo 4.294.967.296 barvnih kombinacij [1].

Ločljivost zapisa spada med bolj pomembna merila kvalitete videa. Slikovni okvirji so sestavljeni iz pikslov[4]. Večja ločljivost pomeni bolj ostro sliko, a hkrati tudi večjo velikost slike v bitih in zahtevnejšo obdelavo. Ločljivost zelo vpliva na hitrost delovanja aplikacije, saj pri videu obdelujemo veliko slikovnih okvirjev na sekundo in jih pošiljamo preko brezžične povezave na drugo mobilno napravo. Na hitrost pošiljanja posameznega slikovnega okvirja vplivajo velikost slike, stiskanje slike in protokol mrežne komunikacije.

[1] Število slik, ki jih kamera naredi v sekundi.
[2] Osnovna in najmanjša enota informacije v računalništvu.
[3] Barvni prostor.
[4] Slikovni element digitalnih slik, ki predstavlja vsako točko slike v mreži.

V naslednjem odstavku podajamo analizo vpliva ločljivosti na velikost slike. Velikost slike izračunamo po enačbi 1.1.

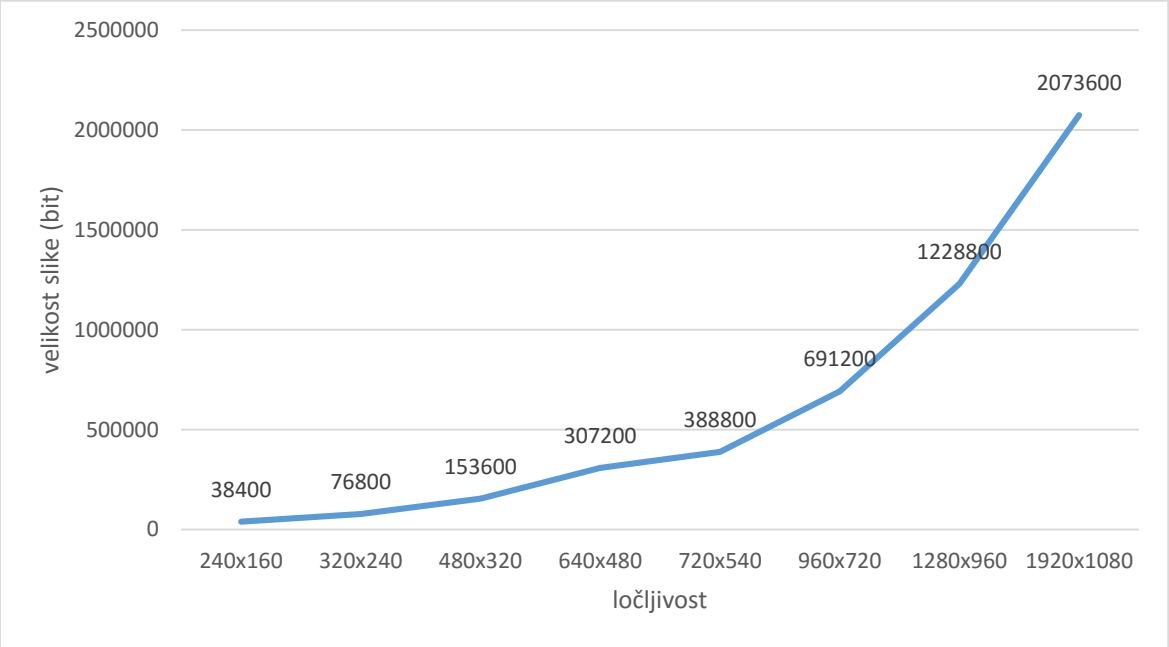
$$velikost\_slike = širina\_slike * višina\_slike * barvna\_globina \quad (1.1)$$

Za izračun velikosti videa moramo dodati še SNS in čas.

$$velikost\_videa = širina\_slike * višina\_slike * barvna\_globina * SNS * s, \quad (1.2),$$

kjer je $širina\_slike$ širina slike v pikslih, $višina\_slike$ višina slike v pikslih, $barvna\_globina$ barvna globina v bitih, s ena sekunda in SNS število okvirjev na sekundo. Velikosti slike in videa sta izraženi v bitih.

V naslednjih primerjavah velikosti slik smo predpostavili 4-zložno barvno globino. Primerjavo velikosti slik različnih ločljivosti prikazuje Slika 1.1.



Slika 1.1: Primerjava velikosti slike različnih ločljivosti.

Ko primerjamo ločljivosti 640×480 in 1280×960, opazimo, da je ločljivost 1280×960 povzročila štirikrat večjo velikost slike kot ločljivost 640×480. Sprotni prenos videa pri

visoki ločljivosti torej ne bo možen v počasnejših komunikacijskih tehnologijah, saj bo prepočasen.

V diplomski nalogi smo se osredotočili na sistem snemanja, prenosa in sprotnega predvajanja video posnetkov v živo. Implementirana rešitev je uporabna na področjih varnostnih kamer, pri opazovanju živali od daleč ali kot kamera na kvadkopterjih. Vsakdo si ne more privoščiti visoko kvalitetne kamere, zato želimo vzpostaviti komunikacijo med dvema pametnima mobilnima napravama, kjer bo možen sprotni prenos videa iz ene naprave v drugo in to brez dodatnih telekomunikacijskih stroškov, ki jih za običajne video klice zaračunavajo ponudniki telekomunikacijskih storitev.

V trgovini Google Play smo poiskali podobne aplikacije našim zahtevam. Poudarimo, da smo našli zelo malo aplikacij za sprotni prenos videa med dvema mobilnima napravama. Večino aplikacij je sprotni prenos videa prenašalo s strežnika s fiksnim naslovom IP na mobilno napravo ali pa z mobilne naprave na lokalni naslov IP, kjer smo ga lahko predvajali v brskalniku. Za sprotni prenos videa med dvema mobilnima napravama smo našli nekaj aplikacij, ki so se najbolj približale našim zahtevam. Te aplikacije so bile Camera Remote [27], Live Stream [28], Spy Camera Monitor [31] in Droid Camera Stream [29], [30]. Aplikacije se medsebojno razlikujejo v funkcionalnostih, zato bomo na kratko opisali vsako posebej.

Aplikacija Live Stream [28] je enostavna za uporabo, podpira le prenos preko komunikacijskega vmesnika WiFi. Izbiramo lahko le ločljivost snemanja kamere in kvaliteto stiskanja JPEG. Na zaslonu izpisuje število prikazanih slik na sekundo, vendar ne moremo določiti točnega prikaza slik na sekundo. Če se snemanje prekine, zahteva od nas ponovni zagon aplikacije, da lahko spet vzpostavimo sprotni prenos videa.

Aplikacija Droid Camera Stream [29], [30] je sestavljena iz aplikacij Viewer in Host. Host predstavlja mobilno napravo, ki zajema sprotni prenos videa, Viewer pa napravo za sprejemanje sprotnega prenosa. Podpira le ločljivosti do 640×480 pikslov in maksimalno le 16 slik na sekundo. Uporabniški vmesnik je nepregleden, saj ne vključuje celotne zajete slike s kamere.

Aplikacija Spy Camera Monitor [31] ima lepo oblikovan uporabniški vmesnik. Za sprotni prenos videa uporablja vmesnik WiFi Direct, a ne omogoča izbire ločljivosti ali kvalitete videa. Privzeta ločljivost ni znana, verjetno pa je omejena na 320×240 pikslov.

Aplikacija Camera Remote [27] je najnaprednejša izmed zgoraj naštetih aplikacij. Za sprotni prenos uporablja komunikacijske vmesnike bluetooth, WiFi in WiFi direct. Omogoča pošiljanje sprotnega prenosa na strežnik, kjer ga lahko opazujemo v spletnem brskalniku. Podpira izbiro ločljivosti kamere, ločljivosti zaslona, kvaliteto stiskanja in prikaza števila slik na sekundo. Lahko izberemo učinke, kot so barvni filtri, sivine, približevanje in oddaljevanje. Vse te funkcionalnosti lahko spreminjamo z mobilno napravo, ki sprejema sprotni prenos. Sprotni prenos slike v uporabniškem vmesniku zakrivajo gumbi, ki jih ne moremo odstraniti. V primeru, da se ločljivosti zaslona uporabljenih mobilnih naprav razlikujejo, se v mobilni napravi z večjo ločljivostjo uporabniški vmesnik prikaže v točni ločljivosti druge naprave. Gumbi za nastavitve so na sliki videa. Hitrost prikazovanja s ločljivostjo 640×480 pikslov je bila približno 6–8 slik na sekundo.

Cilj diplomske naloge je bila vzpostavitev sprotnega prenosa videa preko aplikacije z operacijskim sistemom Android s tremi komunikacijskimi tehnologijami: Bluetooth, WiFi in WiFi-Direct. Želeli smo izdelati aplikacijo, ki bo prikazovala najkvalitetnejši video pri najvišji hitrosti, kjer bo rezultat prikazovanja boljši kot v zgoraj omenjenih aplikacijah. Ob tem mora biti uporabniški vmesnik enostaven in razumljiv za vsakega uporabnika. Na koncu smo prikazali ugotovitve in rezultate tehnologij z različnimi načini optimizacije.

Aplikacijo smo sprogramirali v razvojnem okolju android studio in v programskem jeziku java. Pomagali smo si s knjigo [20] in internetnimi viri [25]. Za testiranje aplikacije smo uporabili mobilne naprave Huawei G620S-L01, Asus Zenpad 8.0 in HTC Desire 620G.

V naslednjih poglavjih predstavljamo celoten postopek izdelave aplikacije, implementacijo modulov, optimizacijo kode, testiranje in analizo rezultatov.

2 APLIKACIJA ZA KOMUNIKACIJO MED MOBILNIMI NAPRAVAMI

Aplikacija podpira tri komunikacijske tehnologije: Bluetooth, Wi-Fi in Wi-Fi direct.

Za vsako tehnologijo je v prvem oknu uporabniškega vmesnika ustvarjen gumb. Gumb nas popelje v modul nastavitev za izbrano tehnologijo. Nastavitve vmesnika si določimo sami, da lahko povežemo med seboj dve mobilni napravi operacijskega sistema android. Uporabniški vmesnik nastavitev se razlikuje glede na zahteve posamezne komunikacijske tehnologije. Število nastavitev smo precej omejili, da smo lahko hitreje in enostavneje povezali napravi, saj preveč nastavitev zmede novejše uporabnike. Za primere nejasnosti smo dodali pomoč za novejše uporabnike, ki pove, kako uporabljati uporabniški vmesnik v trenutni izbrani tehnologiji.

Povezave med napravami so temeljile na sistemu strežnik–odjemalec (angl. *server–client*). Strežnik je prejemal podatke, odjemalec pa pošiljal.

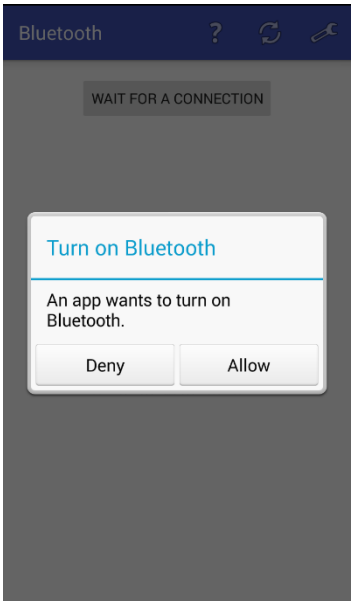
2.1 Komunikacijski vmesnik bluetooth

Bluetooth je brezžična komunikacijska tehnologija za pošiljanje podatkov na kratke razdalje. Izumljen je bil leta 1994 in od takrat so izšle štiri verzije vmesnika. Naprave, uporabljene v diplomski nalogi, podpirajo verzijo bluetooth 4.0, zato smo se osredotočili le na to verzijo.

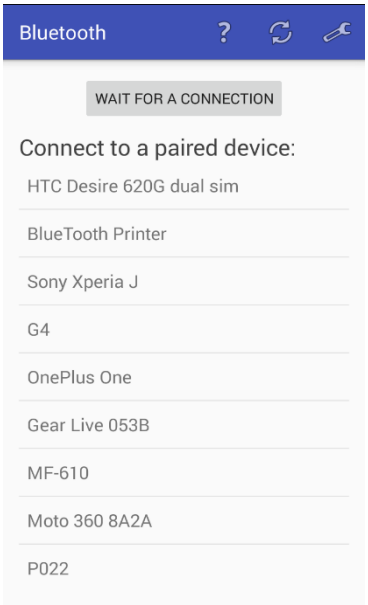
Bluetooth verzije 4.0 podpira hitrosti prenosa do 25 Mbit/s in domet do 60 metrov, vendar v praksi te številke zelo nihajo [3]. Operacijski sistem android za komunikacijo med napravami podpira le protokol RFCOMM[5], saj je podprt in kompatibilen na številnih operacijskih sistemih [4].

[5] Protokol bluetooth, ki uporablja radijsko frekvenco za pošiljanje podatkov med napravami.

V naslednjem poglavju bomo opisali uporabniški vmesnik[6] aplikacije in implementacijo vmesnika bluetooth [21], [23]. Želeli smo doseči, da bo aplikacija enostavna in robustna in bo tudi nov uporabnik zlahka razumel njeno delovanje. Ob izbiri vmesnika bluetooth nas naprava opozori, da moramo najprej omogočiti vmesnik bluetooth (Slika 2.1.1). V primeru zavrnitve te zahteve se aplikacija vrne v prvotno okno izbire komunikacijske tehnologije. V primeru odobritve vklopa vmesnika bluetooth se pokaže uporabniški vmesnik, ki vsebuje gumb za čakanje povezave odjemalca in seznam seznanjenih naprav[7] (angl. *paired devices*). Iz seznama je potrebno izbrati, na kateri strežnik se odjemalec želi povezati (Slika 2.1.2). Za uspešno povezavo moramo torej s strežnikom čakati odjemalca in z odjemalcem izbrati strežnik s seznama seznanjenih naprav.



Slika 2.1.1: Pojavno okno z opozorilom o vklopu vmesnika bluetooth.

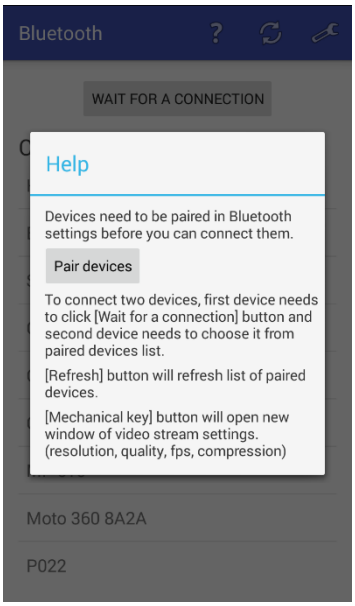


Slika 2.1.2: Seznam seznanjenih naprav.

V primeru, ko želene naprave ni na seznamu seznanjenih naprav, lahko uporabimo pomoč, ki nas vodi do modula za dodajanje naprave. V pomoči smo opisali osnovne korake, s katerimi lahko povežemo dve napravi z vmesnikom bluetooth (Slika 2.1.3). Dodali smo gumb, ki odpre osnovne nastavitve vmesnika bluetooth na mobilni napravi in seznani dve

[6] Videz aplikacije, da vsak uporabnik razume delovanje.
[7] Naprave, ki so se vsaj enkrat povezale z vmesnikom bluetooth.

napravi z varnostno kodo. Ob kliku na gumb »osveži« se osveži seznam seznanjenih naprav.



Slika 2.1.3: Pomoč pri vzpostavitvi povezave preko vmesnika bluetooth.

V primeru uspešne povezave med napravama se strežniku odpre okno za sprotni video prenos od odjemalca. Odjemalec pa začne pridobivati sliko iz kamere in jo pošilja strežniku.

V nadaljevanju opisujemo implementacijo vmesnika bluetooth. Predstavili smo le glavne dele programske kode, ki so najbolj pomembni. Protokol RFCOMM zahteva identifikacijo signalov z uporabo niza UUID. Niz UUID ima 128-bitno vrednost in deluje kot identifikator za varno pošiljanje podatkov z vmesnikom bluetooth. Brez niza UUID bi lahko kdorkoli prestregel prenos podatkov [5]. Za vzpostavitev povezave smo morali imeti enak niz UUID na strežniku in odjemalcu (Slika 2.1.4).

```
UUID uuid = UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");
```

Slika 2.1.4: UUID, uporabljen v aplikaciji.

Funkcija na Sliki 2.1.5, ki je na strežniku sprejela povezavo, sprejme vmesnik bluetooth, ki ga je aplikacija sama pridobila od mobilne naprave. Z vmesnikom prične iskati signale z danim nizom UUID. Neskončna zanka nam omogoča, da čakamo odjemalca, da se poveže

na vtičnik (angl. *socket*). Vtičnik smo shranili v globalni razred[8], do katerega smo dostopali kasneje pri sprotnemu prenosu videa.

```
private void acceptClientBT(BluetoothAdapter adapter)
{
    try {
        BluetoothServerSocket bss  = adapter.listenUsingRfcommWithServiceRecord("Camera Live Stream",uuid);
        BluetoothSocket socket = null;
        while(true)
        {
            socket = bss.accept();
            if(socket != null) {
                SocketClass.setBluetoothSocket(socket);
                break;
            }
        }
    }
    catch (IOException e) {Log.e("Error", e+"");}
}
```

Slika 2.1.5: Čakanje strežnika na odjemalca.

Odjemalec s pomočjo funkcije na Sliki 2.1.6 vzpostavi povezavo s strežnikom. Napravo bluetooth smo pridobili preko parametra[9] funkcije. Preko vtičnika poskusi funkcija vzpostaviti povezavo s strežnikom. Ob uspešni vzpostavitvi povezave se shrani uporabljeni vtičnik v globalni razred.

```
private void connectToServerBT(BluetoothDevice device)
{
    try {
        BluetoothSocket socket = device.createRfcommSocketToServiceRecord(uuid);
        socket.connect();
        while (!socket.isConnected()) {}
        SocketClass.setBluetoothSocket(socket);
    }
    catch (IOException e) {Log.e("Error", e+"");}
}
```

Slika 2.1.6: Povezovanje odjemalca.

Sledi pošiljanje podatkov. V našem primeru pošiljamo zajete slike iz kamere odjemalca. Za pisanje podatkov smo uporabili izhodni tok (angl. *output stream*), prikazan na Sliki 2.1.7, za branje podatkov pa vhodni tok (angl. *input stream*), prikazan na Sliki 2.1.8.

[8] Predstavlja skupino funkcij nekega tipa za izpolnjevanje določene naloge.
[9] Vrednost katerega funkcija prenese ob klicu in je lahko uporabljena v njej.

```
OutputStream outputStream = SocketClass.getBluetoothSocket().getOutputStream();
outputStream.write(data);
```

Slika 2.1.7: Pošiljanje podatkov.

```
InputStream inputStream = SocketClass.getBluetoothSocket().getInputStream();
bytes = inputStream.read(buffer);
```

Slika 2.1.8: Branje podatkov.

2.2 Komunikacijski vmesnik WiFi

WiFi je komunikacijska tehnologija na brezžičnem usmerjevalniku LAN[10]. Usmerjevalnik bere podatke in jih pošilja ciljnim naslovom IP. Naprave se medsebojno ločijo s pomočjo naslovov IP [6]. Hitrost v diplomski nalogi uporabljenega usmerjevalnika je bila 54 Mbit/s, vendar je razdalja in pozicija naprav vplivala na dejansko hitrost prenosa.

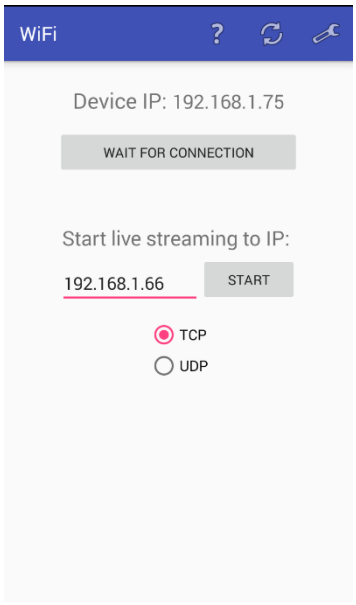
Za primerjanje in prilagajanje prenosa podatkov preko usmerjevalnika smo implementirali komunikacijo preko protokolov TCP in UDP.

TCP je protokol za nadzorovano pošiljanje podatkov, kar pomeni, da protokol poskrbi, da ciljna naprava dejansko prejme podatke. Sprva se vzpostavi povezava med odjemalcem in strežnikom. Hkrati lahko protokol TCP preko omejene pasovne širine pošilja manj podatkov kot protokol UDP, ker pri vsakemu prenosu paketov glava paketa z režijskimi informacijami zasede nekaj zlogov. Glava vsebuje velikost paketa, zaporedno številko in ostale pomembne podatke, ki zagotavljajo uspešen prenos paketa.

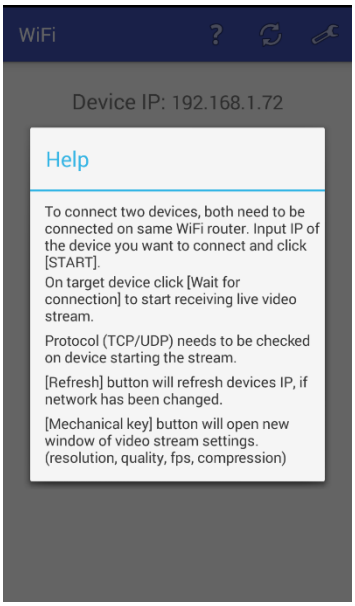
Protokol UDP je nepovezovalni protokol, kar pomeni, da sprva ne vzpostavi povezave med odjemalcem in strežnikom. Protokol UDP pošilja pakete podatkov in ne preverja, ali je ciljna naprava podatke prejela ali ne. Hkrati lahko preko omejene pasovne širine pošlje več podatkov kot protokol TCP, ker vsebuje manj režijskih informacij. Komunikacija je hitrejša, saj ne nadzoruje prenosa paketov [7].

[10] Omrežje, povezano z usmerjevalnikom, kjer lahko naprave, ki so povezane nanj, komunicirajo med seboj.

Pri ustvarjanju uporabniškega vmesnika WiFi smo se poskušali približati uporabniškemu vmesniku bluetooth (Slika 2.2.1). Ob izbiri vmesnika WiFi se naprava poskusi povezati na razpoložljivo omrežje WiFi. Uporabniški vmesnik se prikaže šele, ko naprava od usmerjevalnika pridobi svoj naslov IP. Podan imamo gumb za čakanje sprotnega prenosa videa, ki ob kliku nanj odpre okno in čaka na prikaz slike. Preden lahko odjemalec prične s prenašanjem sprotnega videa, moramo vpisati naslov IP strežnika in klikniti gumb »Start«. Ob uspešnem prepoznanju naprav se v odjemalcu odpre okno za pošiljanje sprotnega videa. V strežniku se čakalno okno zapre in odpre se okno za branje sprotnega videa. Za uporabnike smo dodali pomoč v primeru nejasnosti v aplikaciji (Slika 2.2.2).



Slika 2.2.1: Uporabniški vmesnik WiFi.



Slika 2.2.2: Pomoč pri vmesniku WiFi.

Preden lahko pričnemo pošiljati video, mora odjemalec strežniku sporočiti tip internetnega protokola. Uporabili smo protokol TCP, saj smo želeli, da tako pomemben podatek doseže cilj brez napak. Odjemalec pošlje številko 0, če želi video vsebino pošiljati preko protokola TCP in številko 1, če želi uporabiti protokol UDP. Pošiljanje podatkov z vtičniki (angl. *sockets*) je zahtevalo, da podatke pošiljamo s pomočjo niti (angl. *threads*), ker se pošiljanje podatkov ne sme izvajati v glavni niti (angl. *main thread*) aplikacije. Na Sliki 2.2.3 smo prikazali pošiljanje številke za prepoznavo protokola TCP ter zahteve za vzpostavitev povezave med napravama.

```
new Thread() {
    public void run() {
        try {
            Socket socket = new Socket();
            socket.connect(new InetSocketAddress(SocketClass.getWifiIp(), SocketClass.getWifiPort()));
            SocketClass.setWifiSocket(socket);
            DataOutputStream out = new DataOutputStream(socket.getOutputStream());
            out.writeInt(0);
        } catch(IOException e) {
            Log.e("Error: ", e+"");
        }
    }}.start();
```

Slika 2.2.3: Pošiljanje številke 0, s katero izberemo prenos po protokolu TCP.

Postopek branja smo prikazali na Sliki 2.2.4. Čakali smo na vtičnik odjemalca ter ga shranili v globalni razred za nadaljnjo uporabo. Glede na prebrano prejeto številko izberemo protokol za branje video prenosa (TCP ali UDP).

```
new Thread() {
    public void run() {
        try {
            ServerSocket serverSocket = new ServerSocket(SocketClass.getWifiPort());
            SocketClass.setWifiSocket(serverSocket.accept());
            DataInputStream in = new DataInputStream(SocketClass.getWifiSocket().getInputStream());
            SocketClass.setWifiType(in.readInt());
        } catch(IOException e) {
            Log.e("Error: ", e+"");
        }
    }}.start();
```

Slika 2.2.4: Branje tipa protokola.

Protokol UDP je precej enostavnejši za implementacijo, saj odjemalec nima definiranega rokovanja s strežnikom (angl. *handshaking*), ampak direktno prenaša podatke na podan naslov IP in vrata (Slika 2.2.5). Vendar zato ne preverja uspešnosti prenosa paketov. Eden izmed vzrokov neuspešnosti prenosa paketa je prekinitev povezave ali šibek signal WiFi. V takšnih primerih se podatki izgubijo. Branje podatkov je precej podobno pisanju (Slika 2.2.6).

```
DatagramSocket datagramSocket = new DatagramSocket();
datagramSocket.send(new DatagramPacket(data, data.length, ipAdr, port));
```

Slika 2.2.5: Pisanje podatkov preko protokola UDP.

```
DatagramSocket datagramSocket = new DatagramSocket(port);
DatagramPacket datagramPacket = new DatagramPacket(buffer, buffer.length);
datagramSocket.receive(datagramPacket);
```

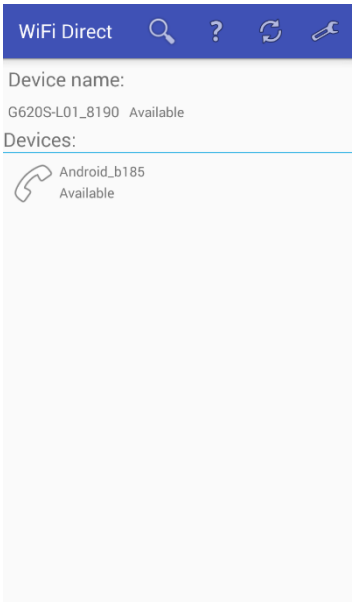
Slika 2.2.6: Branje podatkov preko protokola UDP.

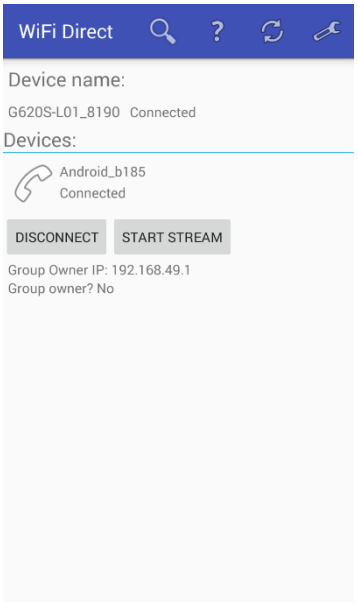
2.3 Komunikacijski vmesnik WiFi direct

WiFi direct je standard, ki ga je razvila industrijska skupina WiFi Alliance proti koncu leta 2010 [8]. Uporablja podobne specifikacije in način povezovanja kot vmesnik WiFi, vendar zanj ne potrebujemo usmerjevalnika. Naprava sama lahko postane usmerjevalnik in tako komunicira vsak z vsakim (angl. *peer to peer*). Uporablja šifriranje[11] WPA2 za zaščito povezave. Dokumentirana hitrost prenosa podatkov je do 250 Mbit/s v dometu do 100 metrov [8]. Za prenos podatkov smo uporabili vtičnike, ki smo jih opisali v poglavju 2.2.

Uporabniški vmesnik vsebuje meni, ki je podoben menijema ostalih dveh komunikacijskih vmesnikov z dodano možnostjo iskanja naprav. Da se napravi uspešno najdeta, morata obe napravi hkrati vklopiti iskanje naprav. Ime prepoznanih naprav smo izpisali v oknu (Slika 2.3.1). Ob izbiri naprave s seznama se nam odprejo gumbi in prikažejo informacije o povezovanju (Slika 2.3.2). Ko se povežemo na izbrano napravo, se nam prikaže naslov IP, na katerem imamo ustvarjen navidezni usmerjevalnik in lahko pričnemo s sprotnim prenosom videa (Slika 2.3.3). Ciljna naprava posluša in čaka na pričetek povezovanja.

[11] Varnostni način skrivanja podatkov, ki jih brez dešifriranja ne moremo prebrati.





Slika 2.3.1: Vmesnik WiFi direct. *Slika 2.3.2: Informacije o napravi.* *Slika 2.3.3: Povezani napravi.*

Implementacija vmesnika WiFI direct je bil dolgotrajen in zapleten postopek, zato primerov programske kode v diplomski nalogi ne podajamo. Namesto tega opisujemo samo pomembne razrede ter postopek vzpostavitve povezave med dvema napravama [22]. Vzpostavitev povezave zahteva veliko procesov, ki tečejo v ozadju.

Razred `WifiP2pManager` je glavni razred, ki definira obstoj vmesnika WiFi direct. Z njim kličemo vse pomembne funkcije.
Razred `Channel` registrira aplikacijo v vmesnik WiFi. Pomembno ga je ustvariti in inicializirati pred kakršnimkoli prenosom podatkov.
Razred `BroadcastReciever` pridobiva stanje vmesnika WiFi direct in se odziva v primeru spremembe povezave ali naprave.
Razred `WifiP2pInfo` vsebuje informacije o ustvarjeni povezavi. Z njim pridobimo IP naslov in vrata s pomočjo metode `ConnectionInfoListener`.
Razred `WifiP2pDevice` predstavlja strežnika ali odjemalca. V seznamu naprav prikažemo vse naprave s pomočjo razreda `PeerListListener`.

Implementacija vmesnika WiFi direct vsebuje mnogo poslušalcev (angl. *listener*) v ozadju, ki čakajo na določene klice. Najprej smo registrirali poslušalce za sledenje spremembam stanja vmesnika WiFi direct na napravi in iskanje naprav. Ustvarili smo primerek razreda

`WifiP2pManager`, ga inicializirali in ga določili razredu `Channel`. Iskanje naprav smo izvedli s poslušalcem `PeerListListener`, ki nam vrne seznam najdenih naprav. Ob zahtevi za povezavo naprav nam je na ciljni napravi razred `ChannelListener` zaznal zahtevo in vzpostavil povezavo WiFi direct med napravama. Takrat smo pridobili podatke s razredom `ConnectionInfoListener`, ki določi strežnika in odjemalca. Izpiše se IP naslov navideznega usmerjevalnika in napravi sta pripravljeni za sprotni prenos videa. Prenos podatkov se izvaja z vtičniki, enako kot pri vmesniku WiFi.

2.4 Primerjava prenosa video vsebin preko različnih komunikacijskih vmesnikov

Uporabniki imamo različne zahteve pri sprotnemu prenosu videa. Želimo lahko visoko kvalitetne slike enkrat na sekundo ali pa slabo kvalitetne slike večkrat na sekundo. Uporabnik si lahko sam določi nastavitve zajemanja slike. Da lahko aplikacija izpolni zahteve, mora uporabnik izbrati pravi komunikacijski vmesnik. Tabela 2.4.1 podaja osnovne lastnosti komunikacijskih vmesnikov. Dokumentirana hitrost in domet sta povzeta iz tehničnih dokumentacij komunikacijskih vmesnikov. Hitrost in domet smo testirali tudi z aplikacijo in tako izmerili dejanske vrednosti. Hitrosti smo testirali z napravo ob napravi in z direktno linijo vidnosti. Linija vidnosti (angl. *line of sight*) je zračna linija med dvema objektoma. Podana velikost paketov predstavlja število zlogov v enem paketu prenosa.

Tabela 2.4.1: Primerjava komunikacijskih vmesnikov.

Vmesniki / Lastnosti	Bluetooth	WiFi		WiFi Direct
		TCP	UDP	
Dok. hitrost	25 Mbit/s	/	/	250 Mbit/s
Dok. domet	60 m	/	/	100 m
Hitrost	1,4 Mbit/s	20 Mbit/s	4 Mbit/s	40 Mbits/s
Domet	30 m	/	/	50 m
Velikost paketov (zlog)	990	1448	65507	povprečno 1444

3 IMPLEMENTACIJA ZAJEMA, PRENOSA IN PREDVAJANJA VIDEO POSNETKA

Zajem videa smo ločili na stisnjeni in nestisnjeni zajem. Velikost podatkov ene same slike je mnogokrat večja v nestisnjenem videu. Vendar pa stiskanje porabi več procesorskega časa. Stiskanje videa ločimo v stiskanje podatkov znotraj posameznega slikovnega okvira (angl. *intra frame*) in medokvirno stiskanje (angl. *inter frame*) [9].

Stiskanje posameznega slikovnega okvira stisne vsako sliko posebej. Tu smo lahko uporabili algoritme za stiskanje slik, kot so JPEG, PNG, WEBP ipd. To stiskanje je počasnejše kot medokvirno stiskanje, vendar je kvaliteta posamezne slike boljša. Procesorsko zahtevno je zlasti za video posnetke z veliko slik na sekundo, velikost datotek pa je lahko tudi do trikrat večja od velikosti datotek medokvirnega stiskanja [10].

Medokvirno stiskanje stisne več slik hkrati, vendar pa je potrebno vnaprej pregledati prihajajoče slike, kar zahteva procesiranje več podatkov hkrati. Deluje na principu stalne slike. Del slike, ki je v več zaporednih slikah enak, si medokvirno stiskanje shrani in posodablja le dele slike, ki se spreminjajo. S tem medokvirno stiskanje prihrani veliko prostora [11].

V diplomski nalogi smo uporabili stiskanje posameznega slikovnega okvira, ker pri sprotnem prenosu videa ne moremo predvidevati naslednje slike za uporabo medokvirnega stiskanja. Če bi vseeno želeli uporabiti medokvirno stiskanje, bi bil prenos podatkov manjši, celoten video prenos pa bi imel večji časovni zamik kot pri stiskanju posameznega slikovnega okvirja.

Stiskanje posameznega slikovnega okvirja in medokvirno stiskanje se povsem razlikujeta v implementaciji, zato bi implementacija obeh načinov stiskanja zahtevala preveč časa. Izbrali smo stiskanje posameznega slikovnega okvirja.

V naslednjih poglavjih opisujemo module za izbiro komunikacijskega vmesnika ter modula za zajem in prikazovanje videa.

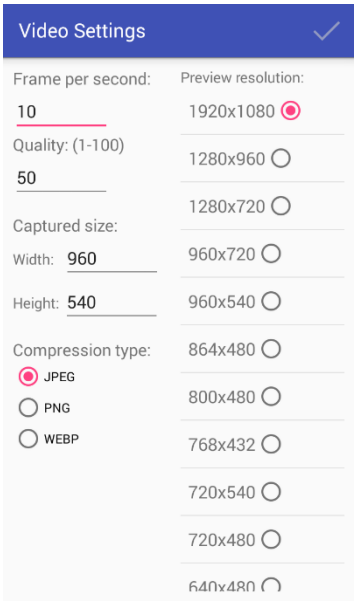
3.1 Modul za izbiro komunikacijskega vmesnika

V modul smo vstavili tri gumbe. Vsak od njih odpre svoj komunikacijski vmesnik. Preden se izbrano okno odpre, se v ozadju izvede vključitev izbrane tehnologije. Ob zapiranju komunikacijskega vmesnika se izbrana tehnologija izključi (Slika 3.1.1).

Slika 3.1.1: Okno za izbiro komunikacijskega vmesnika.

Ob zagonu aplikacije se v ozadju izvede pridobitev informacij o podprtih funkcionalnostih kamere na napravi. Ti podatki so pomembni za uspešno delovanje aplikacije, saj se ob neupoštevanju teh informacij na nekaterih napravah aplikacija ne izvede pravilno in se samodejno zapre. Za nas sta pomembni informaciji o hitrosti kamere (v SNS) in ločljivosti kamere. Dodali smo še funkcionalnosti, s katerimi lahko uporabnik vpiše kvaliteto stiskanja, ločljivost zajemanja slike in tip stiskanja, ki smo jim določili privzete vrednosti. Pri vsakem komunikacijskem vmesniku smo dodali meni. Zadnja možnost v meniju omogoča spremembe nastavitev zajemanja videa (Slika 3.1.2). Želeli smo, da težave nastavitev rešimo pred zajemom videa in se tako izognemo možnim napakam in samodejnim zaprtjem aplikacije. Nastavitve lahko vsebujejo samo številke, kajti aplikacija sama nam ne pusti pisanja črk ali ostalih znakov. Vmesnik lahko zapremo z gumbom nazaj, a se takrat nastavitve ne shranijo. Ob pritisku na kljukico preverimo, če so nastavitve veljavne. V primeru, da so, se nastavitve shranijo, vmesnik se zapre in vrnemo se na

komunikacijski vmesnik. Nastavitve shranimo v globalni razred in jih kasneje preberemo pri zajemu video prenosa.



Slika 3.1.2: Nastavitve zajema videa.

3.2 Modul za zajem videa

V tem modulu smo implementirali zajem videa. Obstaja več načinov pridobivanja slike iz kamere. Te smo raziskali in implementirali v naši aplikaciji. Najprej smo izbrali gradnik `TextureView`, kjer se je izvajal sprotni prenos videa iz kamere. Omenjeni gradnik vsebuje okno za prikazovanje sprotnega prenosa. Najprej smo določili nastavitve kamere glede na prebrane podatke iz globalnega razreda (poglavje 3.1) [24]. Kamero smo pripravili in začeli predogled (angl. *preview*). Sledil je klic metod za zajem posamezne slike.

Za zajem posamezne slike smo implementirali tri metode z imeni `OnSurfaceTextureUpdated`, `OnPreviewFrame` in `CaptureTimedFrame`, ki smo jih ustvarili sami. Program android studio je od nas zahteval, da smo zajem slike izvajali v glavni niti, ki je povezana s strojno opremo mobilne naprave, medtem ko stranske niti niso [18].

Metoda `OnSurfaceTexureUpdated` je povezana z gradnikom `TextureView`. Metoda se izvede ob vsaki spremembi slike na gradniku. V parametrih ne prenese nobenih podatkov, sliko pa lahko pridobimo le z branjem površine.

Metoda `OnPreviewFrame` je povratni klic kamere. Vsaka zajeta slika s kamere kliče to metodo. Deluje podobno kot metoda `OnSurfaceTextureUpdated`, vendar tu s parametri pridobimo podatke, ki predstavljajo sliko.

Ustvarili smo tudi svojo metodo `CaptureTimedFrame`. Tu smo lahko določili točen SNS, saj smo sami vstavili časovni zamik pridobivanja slike. Problem je nastal pri implementaciji metode, ker se morata oba procesa prikazovanja slike na zaslonu mobilne naprave in pridobivanje slike iz površine izvajati v glavni niti. Metoda se izvaja v zanki posebne niti in občasno, odvisno od spremenljivke SNS, izvede metodo v glavni niti za zajem slike (Slika 3.2.1). V našem primeru smo zakasnili zajem slike glede na vrednost spremenljivke `fps`, ki je določala število slik na sekundo.

```
private class sendImageCallback extends Thread
{
    @Override
    public void run() {
        Runnable CaptureTimedFrame = new Runnable() {
            @Override
            public void run() {
                //Zajem slike
            }
        };

        while (true) {
            try{sleep(fps);} catch (InterruptedException e) {}
            runOnUiThread(CaptureTimedFrame);
        }
    }
}
```

Slika 3.2.1: Funkcija `CaptureTimedFrame`.

Zajem posamezne slike smo omogočili v formatih bitmap in YUV. Bitmap je osnovni format slike pri programiranju. Slika je sestavljena iz polja bitov v barvnem prostoru ARGB_8888 [12]. YUV420 je barvni prostor formata NV21, ki je podprt v večini

mobilnih kamer [20]. Pri izberi formatov smo uporabili parametre, izbrane v nastavitvah video zajema, kot so velikost zajema slike, kvaliteta slike in tip stiskanja.

Sliko v formatu bitmap lahko pridobimo na dva načina, in sicer z branjem prikazane slike na zaslonu in pretvorbo barvnega prostora YUV420 v barvni prostor ARGB_8888. Branje prikazane slike na zaslonu pridobi trenutno sliko, ki je prikazana na gradniku `ImageView`, jo spremeni v format bitmap, stisne in pošlje na izhodni tok (Slika 3.2.2).

```
public void onSurfaceTextureUpdated(SurfaceTexture surface) {
    mTextureView.getBitmap(capWidth,capHeight).compress(compressionFormat, quality,baos);
    mClientThread.data = baos.toByteArray();
    mClientThread.write();
    baos.reset();
}
```

Slika 3.2.2: Branje površine s sliko.

Branje slike s pretvorbo barvnega prostora YUV420 v barvni prostor ARGB_8888 za pretvorbo uporabi podatke slike, ki smo jih prejeli preko parametrov metode `OnPreviewFrame`. Podatke smo pretvorili iz barvnega prostora YUV420 v polje formata RGBA in jih uporabili za ustvarjanje slike v formatu bitmap. Za pretvorbo smo uporabili funkcijo `decodeYUV` [13]. Sliko v formatu bitmap smo stisnili z algoritmom, ki ga je uporabnik izbral v oknu nastavitev in poslali na izhodni tok (Slika 3.2.3).

```
@Override
public void onPreviewFrame(byte[] data, Camera camera) {
    final int[] rgb = decodeYUV(data, capWidth, capHeight);
    Bitmap bmp = Bitmap.createBitmap(rgb, capWidth, capHeight, Bitmap.Config.ARGB_8888);
    bmp.compress(compressionFormat, quality, baos);
    mClientThread.data = baos.toByteArray();
    mClientThread.write();
    baos.reset();
}
```

Slika 3.2.3: Branje s pretvorbo.

Sliko v barvnem prostora YUV420 smo pridobili na podoben način kot sliko v formatu bitmap. Vendar se je pri ustvarjanju slike pretvorba barvnega prostora naredila avtomatsko. Pri ustvarjanju razreda `YuvImage` smo porabili podatke, ki smo jih pridobili preko

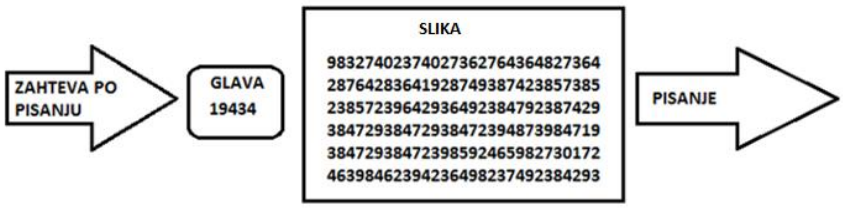
parametrov metode `OnPreviewFrame` ter določili višino in širino želene zajete slike (Slika 3.2.4). Slika je pri pridobitvi obrnjena za 90°.

```
@Override
public void onPreviewFrame(byte[] data, Camera camera) {
    YuvImage image = new YuvImage(data, ImageFormat.NV21, previewWidth, previewHeight, null);
    image.compressToJpeg(new Rect(0, 0, previewWidth, previewHeight), quality, baos);
    mClientThread.data = baos.toByteArray();
    mClientThread.write();
    baos.reset();
}
```

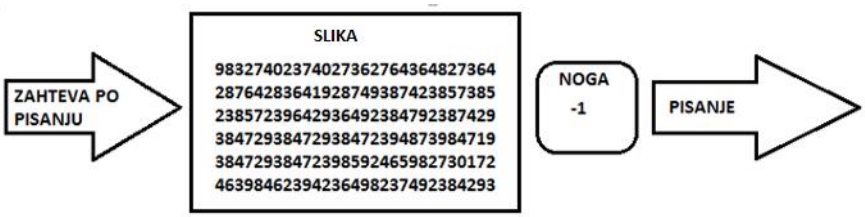
Slika 3.2.4: Zajem slike v formatu YUV.

V primeru, ko uporabljamo komunikacijsko tehnologijo s hitrim prenosom podatkov, lahko preskočimo stiskanje slike. Podatke, prejete preko parametrov metode `OnPreviewFrame` pošljemo direktno na strežnik. V tem primeru se morajo pretvorbe barvnega prostora za prikaz slike izvesti na strežniku.

Ker strežnik ne ve, koliko podatkov sestavlja eno sliko, smo morali zraven podatkov slike poslati še glavo (angl. *header*) ali nogo (angl. *footer*). Glava predstavlja številko, ki opredeljuje velikost posamezne slike (Slika 3.2.5). Noga pa služi kot ločilo različnih slik in je sestavljena iz enega samega zloga (Slika 3.2.6).



Slika 3.2.5: Pisanje z glavo.



Slika 3.2.6: Pisanje z nogo.

Za prepoznavanje slike na strežniku smo potrebovali le glavo ali nogo. Glava pošilja več zlogov, vendar pove strežniku točno, koliko zlogov mora prebrati. Noga pošilja samo en zlog, vendar strežnik ob njem ne ve točno, koliko zlogov mora prebrati.
Program android studio je od nas zahteval, da se pošiljanje podatkov preko vtičnikov ne izvaja v glavni niti, ampak v stranski. Razlog za to je, da le glavna nit posodablja uporabniški vmesnik, prenos podatkov pa se izvaja v neskončni zanki, kjer pa ni možno posodabljanje uporabniškega vmesnika. Vtičnik smo morali inicializirati ob zagonu niti in zatem ustvariti neskončno zanko, ki je čakala na zahtevo po pisanju. V neskončni zanki smo lahko uporabili neposredno določanje podatkov slik ali čakalno vrsto (angl. *queue*) podatkov slik. Implementacija pošiljanja podatkov je bila v obeh primerih podobna, razlikovala pa se je le v povpraševanju po podatkih in pridobivanju podatkov.

Neposredno določanje podatkov slik je pridobilo podatke pri zajemu slik in jih poslalo na strežnik. Če smo med pošiljanjem pridobili nov paket podatkov slike, smo te izgubili in niso bili poslani (Slika 3.2.7).

Čakalna vrsta podatkov slik je postavila v vrsto vsak paket podatkov, ki smo ga pridobili pri zajemu slik. Pred pošiljanjem smo preverjali, ali imamo kakšno sliko v čakalni vrsti. Vzeli smo podatke slike iz čakalne vrste in jih poslali na strežnik. Prikazana je bila vsaka slika s kamere, zato so lahko nastali časovni zamiki pri prikazovanju. Ta rešitev je namenjena hitrim napravam, saj lahko na počasnejših mobilnih napravah nastopijo zelo veliki časovni zamiki, na koncu pa nastopi izjema pomanjkanja pomnilnika (Slika 3.2.8).

```
@Override
public void run() {
    try {
        dos = new DataOutputStream(socket.getOutputStream());
        while(true) {
            if(data != null)
            {
                //Pisanje podatkov
                dos.writeInt(data.length);
                dos.write(data);
                data = null;
            }
            try{sleep(1000/fps);}catch (InterruptedException e){}
        }
    }catch (IOException e){Log.e(TAG, "Error: "+e );}
}
```

Slika 3.2.7: Polje zlogov.

```
@Override
public void run() {

    try {
        DOS = new DataOutputStream(mmSocket.getOutputStream());
        while(true) {
            if(imageListData.size() > 0)
            {
                DOS.writeInt(imageListData.peek().length);
                DOS.write(imageListData.poll());
            }
            try{sleep(1000/fps);}catch (InterruptedException e){}
        }
    }catch (IOException e){Log.e(TAG, "Error: "+e );}
}
```

Slika 3.2.8: Čakalna vrsta slik.

3.3 Modul za sprotno prikazovanje video posnetka na strežniku

V modulu za sprotno prikazovanje smo implementirali sprejemanje podatkov odjemalca, pretvorbe podatkov v obliko za prikazovanje in prikaz posamezne slike. Ta modul je precej krajši od ostalih modulov, saj smo vse pretvorbe barvnega prostora in stiskanje slike izvedli v modulu za zajem. V primeru pošiljanja podatkov brez stiskanja smo pretvorbe barvnega prostora slike izvedli v tem modulu.

Podobno kot pri pisanju prej opisanih modulov smo stranske niti potrebovali tudi pri branju podatkov. Slike smo prikazovali na gradniku `ImageView`. Pri pisanju podatkov smo ustvarili glavo ali nogo, da smo pri branju lahko pridobili celotno sliko. Branje se je izvajalo v neskončni zanki in se je razlikovalo glede na to, ali smo pošiljali podatke z glavo ali z nogo.

Pri sprejemu podatkov z glavo smo najprej prebrali število, ki je predstavljalo velikost prihajajoče slike. Rezervirali smo si pomnilnik, ki je vseboval polje zlogov v velikosti prebrane številke. Prebrali smo vse podatke, jih shranili v polje zlogov in jih prenesli v glavno nit (Slika 3.3.1).

```
public void run() {
    byte[] buffer;
    int length;
    while (true) {
        try {
            length = dataInputStream.readInt();
            buffer = new byte[length];
            dataInputStream.readFully(buffer);
            mImageLoadThread.setData(buffer);
            mImageLoadThread.run();
        } catch (IOException e) {
            Log.e("Error on read data: ", ""+ e);
            break;
        }
    }
}
```

Slika 3.3.1: Branje podatkov z glavo.

Pri branju z nogo smo najprej ustvarili polje zlogov v velikosti maksimalnega števila zlogov, ki jih lahko pošlje izbrani komunikacijski vmesnik v enem paketu. V Tabeli 2.4.1 smo prikazali maksimalne velikosti paketov posameznega komunikacijskega vmesnika.

Začeli smo brati pakete in jih združevali s pomočjo razreda `ByteArrayOutputStream`. Ko smo prebrali paket podatkov s samo enim zlogom, je to pomenilo konec podatkov za eno sliko. Prebrane podatke smo prenesli v glavno nit (Slika 3.3.2).

```
public void run() {
    byte[] buffer = new byte[990];
    int bytes;
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    while (true) {
        try {
            bytes = inputStream.read(buffer);
            if(bytes != 1)
                baos.write(buffer, 0,bytes);
            else
            {
                mImageLoadThread.setData(baos.toByteArray());
                mImageLoadThread.run();
                baos.reset();
            }
        } catch (IOException e) {
            Log.e("Error on read data: ", ""+ e);
            break;
        }
    }
}
```

Slika 3.3.2: Branje podatkov z nogo.

V glavni niti smo podatke najprej pretvorili v format bitmap in jih prikazali v gradniku `ImageView`, ki jih je prikazal na zaslonu mobilne naprave (Slika 3.3.3).

```
private class imageLoadThread extends Thread
{
    byte[] data;
    public void setData(byte[] data) { this.data = data; }
    public void run() {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                imageView.setImageBitmap(BitmapFactory.decodeByteArray(data, 0, data.length));
            }
        });
    }
}
```

Slika 3.3.3: Prikaz posamezne slike.

4 OPTIMIZACIJA PRENOSA VIDEO VSEBIN

Optimizacija je postopek iskanja optimalnejših rešitev v programski kodi. Cilj je izvedba manjšega števila procesov, zmanjšanje števila procesorsko intenzivnih procesov, manjša uporaba pomnilnika in sprotno čiščenje pomnilnika. V diplomski nalogi je bila optimizacija aplikacije ena izmed pomembnejših nalog, saj je izmenjava video vsebin procesorsko zelo intenzivna.

Osredotočili smo se na optimizacijo procesov za sprotni prenos videa, saj smo želeli doseči hitrost prenosa 30 slik na sekundo. Vsaka ponovitev zanke je predstavljala en slikovni okvir, ki smo ga prenašali od odjemalca do strežnika. Izognili smo se optimizaciji uporabniškega vmesnika, povezovanju naprav in neponavljajočih izvedb programske kode v ozadju.

Optimizacijo smo končali, ko ni bilo več možnih rešitev, ki bi še kakorkoli izboljšale aplikacijo. Iskanje rešitev smo omejili na iskanje po internetu ter raziskovanje in testiranje različnih optimizacij programskih kod. Cilj optimizacije je bil sprotni prenos videa s največjo možno ločljivostjo, kvaliteto in številom slik na sekundo.

4.1 Optimizacija javanske programske kode

Optimizacija programske kode je osnovni postopek iskanja rešitev znotraj programskega jezika. Sami smo optimizirali programsko kodo ob programiranju. Osredotočili smo se na neskončne zanke, kjer je bila pomembna najhitrejša možna izvedba zank.

Vse inicializacije spremenljivk, razen alokacije slikovnega okvirja, smo izvedli zunaj zank. Spremenljivke, uporabljene v nitih, smo ustvarili globalno za hiter dostop po celotni aplikaciji. Na sliki 4.1.1 smo prikazali primer neoptimizirane kode, ki predstavlja sprejem in prikaz posamezne slike ob vsaki ponovitvi zanke.

```
while (true) {
    try {
        DataInputStream dataInputStream = new DataInputStream(SocketClass.getWifiSocket().getInputStream());
        int length = dataInputStream.readInt();
        data = new byte[length];
        dataInputStream.readFully(data);
        runOnUiThread(setImage);
        Runnable setImage = new Runnable() {
            @Override
            public void run() {
                imageView.setImageBitmap( BitmapFactory.decodeByteArray(data, 0, data.length));
            }
        };
        runOnUiThread(setImage);
    } catch (IOException e) {
        Log.e("Error on read data: ", ""+ e);
        break;
    }
```

Slika 4.1.1: Primer neoptimizirane kode.

Na Sliki 4.1.2 pa smo prikazali optimizirano kodo s Slike 4.1.1.

```
Runnable setImage = new Runnable() {
    @Override
    public void run() {
        imageView.setImageBitmap( BitmapFactory.decodeByteArray(data, 0, data.length));
    }
};
int length;
while (true) {
    try {
        length = dataInputStream.readInt();
        data = new byte[length];
        dataInputStream.readFully(data);
        runOnUiThread(setImage);
    } catch (IOException e) {
        Log.e("Error on read data: ", ""+ e);
        break;
    }
}
```

Slika 4.1.2: Primer optimizirane kode.

Poudarimo, da je bila alokacija pomnilniškega prostora znotraj zanke nujna, saj rezerviramo točno določeno velikost sprejetega slikovnega okvirja. Zaradi stiskanja slik se velikost posamezne zajete slike razlikuje. Ob vsaki ponovni alokaciji pomnilnik sprosti pomnilniški prostor s sliko, ki smo jo prikazali na zaslonu. V primeru, ko bi alokacijo pomnilnika izvedli zunaj zanke, bi morali predvidevati zgornjo mejo velikosti prebranih slikovnih okvirjev. Ker ne poznamo točne velikosti, bi morali rezervirati več pomnilnika, da bi lahko poskrbeli za prikaz vseh slik. Vse ostale inicializacije so se izvedle zunaj zanke. Tako smo pohitrili izvedbo zank za nekaj milisekund. Namesto da smo ob vsaki

ponovitvi zanke ustvarjali metodo `setImage`, smo jo ustvarili pred zanko in jo po branju slike le klicali.

Pri pošiljanju podatkov smo izbrali pošiljanje z glavo, saj je prejemanje podatkov z nogo zahtevalo združevanje več manjših paketov podatkov, kar je v programskem jeziku java zelo počasno.

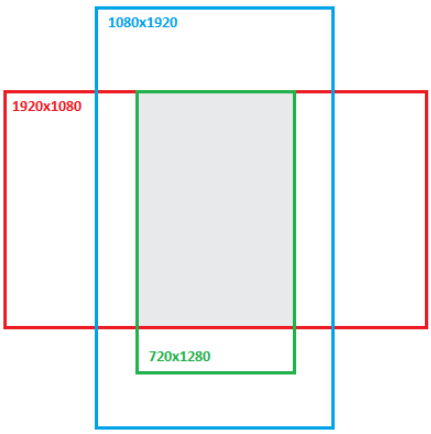
4.2 Optimizacija zajema, prenosa in sprotnega prikaza videa

Operacijski sistem android nam je optimizacijo aplikacije še otežil. Privzeti format kamere je NV21 z barvnim prostorom YUV420. Da smo lahko prikazali sliko, pa smo potrebovali sliko v barvnem prostoru ARGB_8888, rotirano za 90°. Operacijski sistem android že vsebuje metode za pretvorbo slike iz barvnega prostora YUV420 v barvni prostor ARGB_8888, vendar so bile za naše zahteve precej počasne.

Najprej smo morali ugotoviti, kateri postopek je trajal največ časa, zato smo za vsako metodo testirali čas izvedbe. Kot smo pričakovali, je največ časa potrebovala metoda za stiskanje, pretvorbo in rotiranje formata bitmap. Metoda je sprejela parametre, kot so ločljivost in kvaliteta slike, ne pa slike s kamere. Sliko s kamere smo s formatom bitmap lahko ustvarili le s pretvorbo podatkov slike v številsko polje pikslov, kjer je bila tudi pretvorba barvnega prostora YUV420 v barvni prostor ARGB_8888. Zato smo brali sliko z zaslona mobilne naprave in ta izvedba je bila dosti hitrejša, da smo z ločljivostjo snemanja kamere 1920×1080 in ob pridobivanju slike z ločljivostjo 720×1280 in kvaliteto 50 % je pa metoda na mobilni napravi Huawei potrebovala povprečno 240 ms, kar nam je omogočalo le prenos pri hitrosti 4,1 SNS. Kvaliteta slike določa stopnjo stiskanja z izgubami v formatu bitmap. Ker smo pridobivali sliko z zaslona mobilne naprave, je bila posamezna slika že pravilno orientirana.

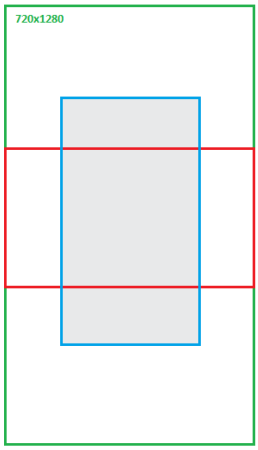
Razred `YuvImage` je z istimi lastnostmi pri pretvorbi barvnega prostora in stiskanju slike porabil le 75 ms, kar nam je omogočalo hitrost prenosa 13,3 SNS, vendar brez rotacije za 90°. Posamezno rotiranje slik je bilo prepočasno, saj smo morali ponovno dostopati do

posameznih pikslov. Hitreje smo rotirali sliko med pretvorbo barvnega prostora. Tako deluje metoda stiskanja in pretvorbe barvnega prostora pri formatu bitmap. Na strežniku smo rotirali celoten gradnik, vendar je bila slika prikazana v dimenzijah zajete slike. Na Sliki 4.2.1 smo prikazali primer problema transformacije slike z ločljivostjo 1920×1080. Rdeči kvadrat predstavlja sprejete dimenzije slike, pridobljene z odjemalca. Modri kvadrat prikazuje rezultat rotiranja slike, kot jo želimo prikazati na zaslonu. Zeleni kvadrat predstavlja velikost zaslona mobilne naprave. Prikaz posamezne slike, ki smo jo prikazali na strežniku, je obarvan sivo.



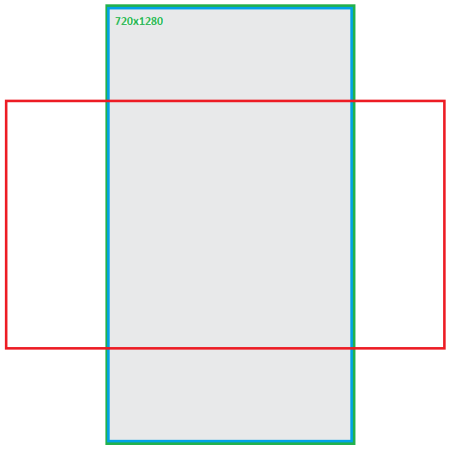
Slika 4.2.1: Prikaz slike v razredu `YuvImage`.

V zgoraj navedenem primeru smo prikazali le del slike in pri tem zgubili velik del informacij. Sliko smo zato zmanjšali glede na ločljivost zaslona mobilne naprave. Ker smo sprejeli sliko v ležečem položaju, je gradnik prikazal sliko v dimenzijah sprejete slike (Slika 4.2.2).



Slika 4.2.2: Prikaz pomanjšane slike v razredu `YuvImage`.

Robove rdečega kvadrata, ki prikazuje sprejeto sliko pred rotacijo, smo postavili izven vidnega polja zaslona mobilne naprave tako, da je imela rotirana slika, prikazana z modrim kvadratom, na koncu približno velikost zaslona mobilne naprave (Slika 4.2.3). S tem smo se izognili rotiranju slike, kjer bi morali dostopati do posameznih pikslov. Tako smo sprejeli in obdelali celotno sliko brez izgube informacij.



Slika 4.2.3: Rezanje sprejete, slike v razredu `YuvImage`.

Sprotni prenos videa brez stiskanja slik je bil najmanj zahteven postopek na strani odjemalca, saj smo pošiljali podatke direktno na strežnik, in sicer brez pretvorbe barvnega prostora. Ker so bile slike vedno iste velikosti in smo vedeli točne hitrosti komunikacijskih vmesnikov, smo lahko izračunali, koliko časa bi potrebovali za prenos posamezne slike

med mobilnima napravama. V poglavju 1 smo že prikazali kako izračunati bitno velikosti slike.

Primerjali smo vmesnika bluetooth in WiFi direct, saj lahko iz Tabele 2.4.1 razberemo, da ima vmesnik bluetooth najnižjo hitrost prenosa, medtem ko ima WiFi direct najvišjo hitrost prenosa. Velikost slik v bitih smo pridobili z množenjem njihove višine in širine ter barvne globine, ki je v našem primeru določena z barvnim prostorom ARGB_8888 in znaša 32 bitov [26].

Vmesnik bluetooth je lahko pošiljal 183.500 B/s. Če smo želeli doseči hitrost 30 SNS, so morale biti slike velike največ 6116 zlogov. Brez stiskanja smo lahko pošiljali slike z ločljivostjo največ 39×39 pikslov.

Vmesnik WiFi direct je lahko pošiljal 5.242.880 B/s. Če smo želeli doseči hitrost 30 SNS, so morale biti slike velike največ 174.762 zlogov. Brez stiskanja smo lahko pošiljali slike z ločljivostjo največ 209×209 pikslov.

Poudarimo, da pri komunikacijskih vmesnikih prenos ni dovolj hiter za sprotni prenos videa v večjih ločljivostih brez stiskanja. Zato smo se posvetili optimizaciji stiskanja slik.

Po internetnih virih smo raziskovali hitrejše načine stiskanja slik in pretvorbe barvnega prostora iz YUV420 v barvni prostor ARGB_8888. Ugotovili smo, da programska oprema android studio ne podpira več dodatnih načinov stiskanja slik kot s formatom bitmap in razredom `YuvImage`. Stiskanje bitov pa smo testirali še z razredom `Deflater`, ki uporablja metodo stiskanja `deflate`[12] [14]. Rezultati stiskanja so bili prepočasni in slike so bile prevelike velikosti za naše zahteve.

Da bi dosegli želene hitrosti prenosa, bi morali stiskanje in pretvorbo barvnih formatov izvesti izven operacijskega sistema android, in sicer s pomočjo dodatkov JNI in NDK v programskem jeziku C ali C++. JNI je razvojno okolje za klicanje programske kode v jezikih C ali C++ neposredno iz programskega jezika java. NDK je razvojno orodje za JNI,

[12] Kombinacija stiskanja z LZ77 in Huffmanovim kodiranjem.

s katerim zgradimo in pripravimo že narejeno aplikacijo. Našli smo nekaj virov [15], [16], [17], [18], ki bi nam morda omogočali hitrejše stiskanje in pretvorbo. Ker smo imeli težave z JNI knjižnicami za stiskanje slik, saj so zahtevale verzijo operacijskega sistema android 5.0 Lollipop ali višje, razvojnega okolja JNI in NDK nismo uporabili.

4.3 Zasnova meritev

Sledilo je testiranje aplikacije glede na zahteve različnih uporabnikov in popravljanje napak. Primerjali smo funkcije stiskanja s formatom bitmap in razredom `YuvImage`.

Za zajem slik s kamere smo uporabili mobilno napravo Huawei G620S-L01 z ločljivostjo zaslona 720×1280 pikslov, s štirijedrnim 1,2 GHz procesorjem Qualcomm in 1 GB pomnilnika. Naložen je bil operacijski sistem android verzije KitKat 4.4. Slike smo sprejemali z mobilno napravo HTC Desire 620G, na kateri je tekel operacijski sistem android 4.4.2 KitKat. Opremljena je bila z osemjedrnim 1,7 GHz procesorjem Cortex-A7. Ločljivost zaslona je bila 720×1280 pikslov, vendar smo zaradi zaslonskih tipk za prikaz lahko uporabili le 720×1184 pikslov.

Za testiranje smo z mobilno napravo snemali mirno sceno s številnimi različnimi barvami (Slika 4.3.1). Ob dinamični sceni bi se velikosti slik zaradi stiskanja preveč razlikovale, kar bi povzročalo nepravilnosti pri računanju povprečnih rezultatov.

Slika 4.3.1: Testna slika.

Scena je bila pri obeh funkcijah, torej pri stiskanju s formatom bitmap in stiskanju z razredom `YuvImage` enaka. V tabeli 4.3.1 smo primerjali obe metodi stiskanja pri različnih ločljivostih in kvalitetah slik. Prikazali smo velikosti stisnjenih podatkov posameznega okvirja (v kB) in čas stiskanja v ms. Ločljivost pošiljanja s formatom bitmap je bila odvisna od velikosti zaslona mobilne naprave, zato smo morali v naših testih preskočiti ločljivost 1920×1080 pikslov.

Tabela 4.3.1: Rezultati testiranja.

	JPEG stiskanje, format bitmap				JPEG stiskanje, format `YuvImage`.			
	100 %	75 %	50 %	25 %	100 %	75 %	50 %	25 %
480x640	247 kB 140 ms	45 kB 100 ms	29 kB 95 ms	19 kB 95 ms	250 kB 80 ms	43 kB 45 ms	29 kB 35 ms	19 kB 30 ms
540x960	419 kB 220 ms	70 kB 155 ms	45 kB 150 ms	29 kB 140 ms	415 kB 115 ms	66 kB 65 ms	44 kB 55 ms	28 kB 50 ms
720x1280	783 kB 380 ms	140 kB 280 ms	89 kB 270 ms	59 kB 265 ms	739 kB 190 ms	112 kB 110 ms	72 kB 85 ms	47 kB 80 ms
1080x1950	/ 	/ 	/ 	/ 	1200 kB 340 ms	165 kB 190 ms	110 kB 170 ms	70 kB 160ms

Opazili smo, da je novejša knjižnica `YuvImage` porabila za stiskanje skoraj polovico manj časa kot pa razred `Bitmap`. Eden iz med razlogov je, da pri pretvorbi slike razred `YuvImage` slike ne rotira. Velikost stisnjenih podatkov je tudi dosti manjša pri večjih ločljivostih. Pri nižjih ločljivostih so si rezultati zelo podobni. Po teh rezultatih smo lahko za nadaljnje analiziranje brez premisleka izbrali stiskanje z razredom `YuvImage`.

4.4 Analiza meritev

Ker smo v poglavju 2.4 in 4.2 izračunali hitrost prenosov, smo lahko iz Tabele 4.3.1 izbrali najboljšo kvaliteto slik za določeno komunikacijsko tehnologijo. Opazili smo, da je doseganje 30 SNS nad ločljivostjo 480×640 pikslov nemogoče. Kvaliteta slike pod ločljivostjo 480×640 pa je bila precej nizka.

Vmesnik bluetooth s hitrostjo prenosa 183 500 B/s vsekakor ni bil dovolj hiter za več kot 10 SNS, če smo želeli ločljivost nad 480×640 pikslov pri 25 % kvaliteti stiskanja. Pri 30 SNS bi potrebovali slike velike le 6116 zlogov in pri 20 SNS slike v velikosti le 9175 zlogov. Pri 10 SNS bi morale biti slike velike približno 18.350 zlogov, kar se približuje zahtevam v Tabeli 4.3.1. Čas stiskanja postane nepomemben, saj je čas prenosa daljši. Uporabniki, ki zahtevajo visoke kvaliteto pri hitrosti 1 SNS, lahko prenašajo slike z ločljivostjo 1920×1080 in kvaliteto stiskanja algoritma JPEG, nastavljeno na 75 %.

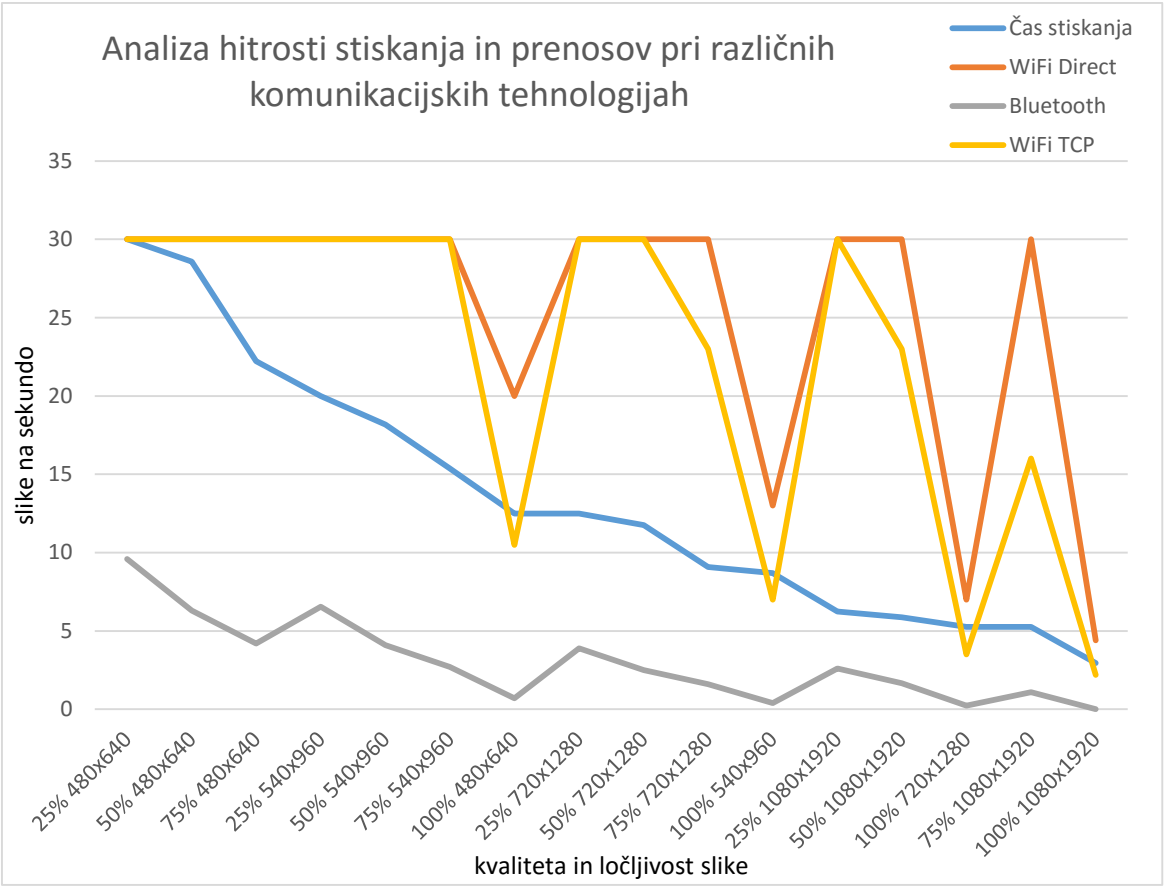
Pri vmesnikih WiFi in WiFi direct je čas stiskanja postal najpomembnejša metrika. Visoko kvalitetne slike smo lahko dovolj hitro pošiljali, vendar pa jih tako hitro nismo mogli stiskati. Z 10 SNS smo lahko zajemali slike z ločljivostjo 720×1280 in s kvaliteto stiskanja med 75 % in 50 % (Slika 4.4.1). Hitrost 20 SNS smo dosegli z ločljivostjo 540×960 in pri kvaliteti stiskanja JPEG, nastavljeni na 25 % (Slika 4.4.2). Za doseganje hitrosti 30 SNS pa smo lahko izbrali le najnižjo ločljivost in kvaliteto v Tabeli 4.3.1, kar je ločljivost 480×640 s kvaliteto stiskanja nastavljeno na 25 % (Slika 4.4.3).

Slika 4.4.1: Lastnosti 50 % 720×1280.

Slika 4.4.2: Lastnosti 25 % 540×960.

Slika 4.4.3: Lastnosti 25 % 480×640.

Z grafom na Sliki 4.4.4 smo prikazali primerjavo časa stiskanja s hitrostjo prenosa vseh komunikacijskih tehnologij. Podatke o stiskanju z razredom `YuvImage` smo vzeli iz Tabele 4.3.1. Navpična os prikazuje hitrost pošiljanja (v SNS), vodoravna os pa kvaliteto slike, ki narašča proti desni. Poudarimo, da je prenos podatkov pri vmesniku bluetooth počasnejši od časa stiskanja in zato predlagamo uporabo drugega komunikacijskega vmesnika. Prenos pri vmesnikih WiFi in WiFi direct je vedno hitrejši kot stiskanje, razen pri zajemanju slike s kvaliteto stiskanja, nastavljeno na 100 %. Priporočamo, da se izogibamo zajemu slik s kvaliteto 100 % in uporabimo največ kvaliteto 80 %.

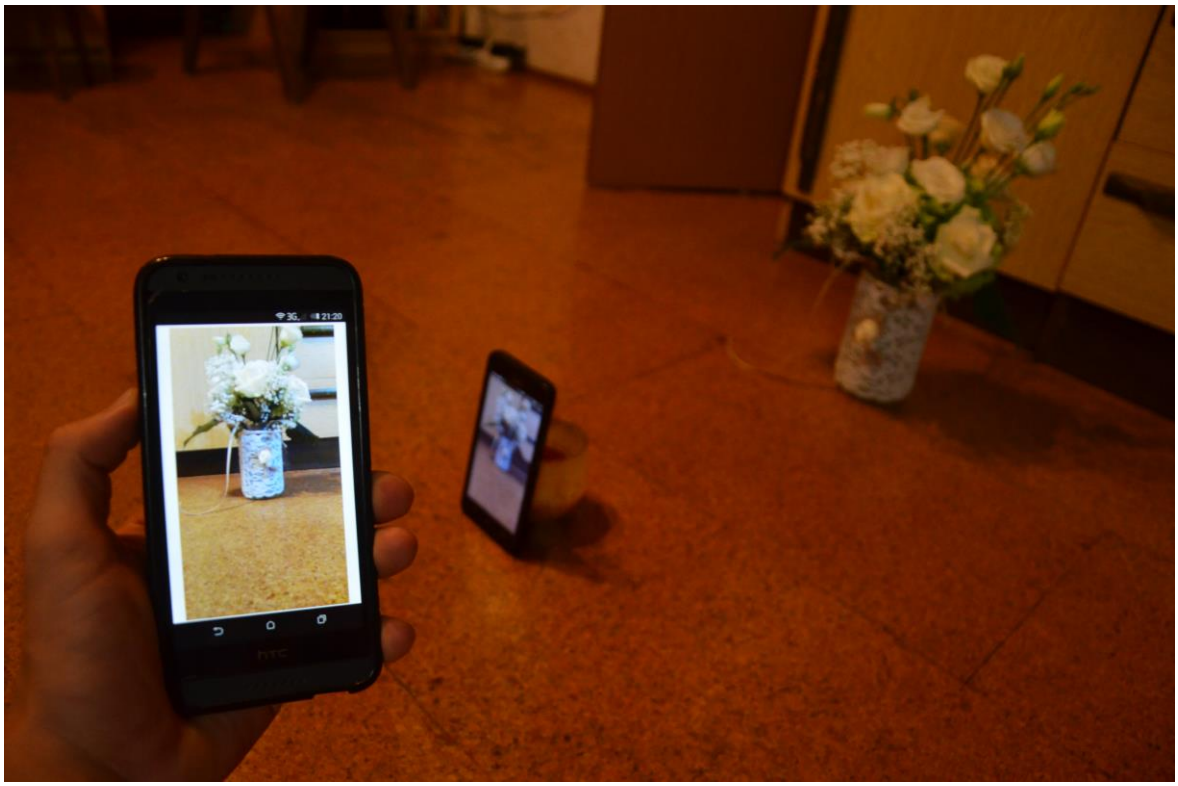


Slika 4.4.4: Analiza hitrosti stiskanja in prenosov pri različnih komunikacijskih tehnologijah.

Sprotni prenosa videa z ločljivostjo 1920×1080 pikslov, kvaliteto stiskanja 50 % smo testirali 10 minut. S komunikacijskim vmesnikom WiFi in WiFi direct je mobilna naprava Huawei, ki je pošiljala prenos slike, porabila 5 % baterije, medtem ko je naprava HTC Desire, ki je sprejemala prenos, porabila le 2 %. Ker uporabljena vmesnika uporabljata za pošiljanje isti razred vtičnikov, so rezultati zato tudi podobni.

S komunikacijskim vmesnikom bluetooth je mobilna naprava Huawei pri pošiljanju podatkov porabila 3 % baterije in naprava HTC Desire pri sprejemanju podatkov porabila 2 % baterije. Ker je ima vmesnik bluetooth vgrajen sistem varčevanja baterije, smo lahko pričakovali večji prihranek baterije na tem vmesniku [3].

Na Sliki 4.4.5 smo prikazali delovanje naše aplikacije. Mobilna naprava, ki jo držimo v roki, sprejema sprotni prenos slike z naprave, ki je postavljena na tleh in pošilja sceno rož.



Slika 4.4.5: Prikaz uporabe aplikacije.

Sprotni prenos videa aplikacije smo primerjali z že obstoječimi aplikacijami, omenjenimi v poglavju 1. V aplikacijah, kjer je bilo možno, smo nastavili zajem kamere na ločljivost 640×480. Prikaz slik na sekundo je v obstoječih aplikacijah bil precej nizek. Iz izkušenj smo lahko ocenili, da so omogočale prikaz videa do največ 10 slik na sekundo. Z našo aplikacijo smo z isto ločljivostjo lahko prikazali 30 slik na sekundo. S tem smo dosegli enega izmed ciljev diplomske naloge.

5 SKLEP

V diplomski nalogi smo preučili delo z video posnetki in se posvetili optimizaciji aplikacije, ki omogoča sprotni prenos videa med dvema mobilnima napravama. Želeli smo prikazati najkvalitetnejši in najhitrejši prenos z ene mobilne naprave na drugo mobilno napravo na operacijskem sistemu android. Ker ima vsak uporabnik različne zahteve pri prenosu video vsebin, smo implementirali komunikacijske vmesnike bluetooth, WiFi in WiFi direct. Aplikacija prepozna specifikacije kamere na mobilni napravi in jih shrani v okno nastavitev. Poskrbeli smo, da aplikacija deluje brezhibno ter da lahko uporabnik spreminja nastavitve med sprotnim prenosom videa.

Aplikacijo lahko uporabljamo kot kamero na daljavo za opazovanje živali, na kvadrokopterjih ali v podobnih scenarijih uporabe, ko si ne moremo privoščiti visoko kvalitetne kamere in nas moti plačevanje dodatnih stroškov teleoperaterju. Cilj diplomske naloge pa ni bila le izdelava aplikacije, temveč tudi analiza rezultatov in študija hitrejših in kvalitetnejših prenosov video posnetkov. Želeli smo doseči hitrosti 30 SNS z ločljivostjo 1920×1080 pikslov, vendar smo takšne hitrosti prenosa dosegli le pri ločljivosti 640×480 pikslov.

Z uporabo novejših brezžičnih komunikacijskih tehnologij, kot sta vmesnika WiFi in WiFi direct, hitrost prenosa ni več problem. Ozko grlo postane hitrost procesiranja na mobilni napravi. Uporabljena mobilna naprava Huawei ni sodila med hitre procesorske naprave, vendar smo s tem dokazali, da bodo rezultati naše diplomske naloge na hitrejših napravah še boljši.

Poudarimo, da operacijski sistem android in programski jezik java nista priporočljiva za obdelavo velike količine podatkov. Sama ne podpirata direktnega združevanja dveh polj, metode napisane s strani programerja pa niso dovolj hitre. Čeprav smo dosegli cilj prikaza sprotnega prenosa videa s 30 slikami na sekundo, nam ni uspelo prikazovanje s istim številom slik na sekundo pri ločljivosti 1920×1080 pikslov. Zato priporočamo uporabo paketov NDK v okolju JNI s programskim jezikom C in C++ za optimizirano procesiranje velike količine podatkov.

SEZNAM UPORABLJENIH VIROV

[1] ComputerHope. Is there a difference between 16-bit, 24-bit, and 32-bit color? Dostopno na: http://www.computerhope.com/issues/ch001557.htm [10. avgust 2016].

[2] Wikipedia. Frame rate. Dostopno na: https://en.wikipedia.org/wiki/Frame_rate [10. avgust 2016].

[3] Wikipedia. Bluetooth. Dostopno na: https://en.wikipedia.org/wiki/Bluetooth [10. avgust 2016].

[4] Bluetooth. RFCOMM. Dostopno na: https://developer.bluetooth.org/TechnologyOverview/pages/rfcomm.aspx [10. avgust 2016].

[5] DataPower Technology, Inc. A Universally Unique IDentifier (UUID) URN Namespace. Dostopno na: https://tools.ietf.org/html/rfc4122 [11. avgust 2016].

[6] GadgetReview. What Does WiFi Stand For and How Does Wifi Work? Dostopno na: http://www.gadgetreview.com/what-is-wifi-what-does-wifi-stand-for-how-does-it-work [13. avgust 2016].

[7] PieterExplainsTech. UDP and TCP: Comparison of Transport Protocols. Dostopno na: https://www.youtube.com/watch?v=Vdc8TCESIg8 [13. avgust 2016].

[8] Tomas Thorn. Wi-Fi Direct: what it is and why you should care. Dostopno na: http://www.techradar.com/news/phone-and-communications/mobile-phones/wi-fi-direct-what-it-is-and-why-you-should-care-1065449 [18. avgust 2016].

[9] Wolfcrow. Intra-frame vs Inter-frame Compression. Dostopno na: http://wolfcrow.com/blog/intra-frame-vs-inter-frame-compression/ [21. avgust 2016].

[10] Wikipedia. Intra frame. Dostopno na: https://en.wikipedia.org/wiki/Intra-frame [21. avgust 2016].

[11] Wikipedia. Inter frame. Dostopno na: https://en.wikipedia.org/wiki/Inter_frame [21. avgust 2016].

[12] Foley J. D. Computer Graphics: Principles and Practice. Boston: Addison-Wesley Professional, 1996. Str. 30–39.

[13] 41 Post. Android: Retrieving the Camera preview as a Pixel Array. Dostopno na: http://www.41post.com/3470/programming/android-retrieving-the-camera-preview-as-a-pixel-array [23. avgust 2016].

[14] Android Developer. Deflater. Dostopno na: https://developer.android.com/reference/java/util/zip/Deflater.html [28. avgust 2016].

[15] Stackoverflow. Converting YUV->RGB(Image processing)->YUV during onPreviewFrame in android? Dostopno na: http://stackoverflow.com/questions/9325861/converting-yuv-rgbimage-processing-yuv-during-onpreviewframe-in-android [28. avgust 2016].

[16] GitHub. PdqImage. Dostopno na: https://github.com/mtnwrw/pdqimage [28. avgust 2016].

[17] GitHub. LZ4-java. Dostopno na: https://github.com/jpountz/lz4-java [28. avgust 2016].

[18] GitHub. Android-GPUimage. Dostopno na: https://github.com/CyberAgent/android-gpuimage [28. avgust 2016].

[19] Phillips B., Stewart C., Hardy B., Marsicano K. Android Programming: The Big Nerd Ranch Guide, 2nd Edition. Atlanta: Big Nerd Ranch, cop, 2015.

[20] Waggover B. Compression for great video and audio: master tips and common sense, 2nd Edition. Amsterdam: Elsevier, Focal Press, cop, 2010. Str. 22, 23, 68, 109, 137–154.

[21] Developer. Android. BluetoothChatService. Dostopno na: https://developer.android.com/samples/BluetoothChat/src/com.example.android.bluetoothchat/BluetoothChatService.html [29. avgust 2016]

[22] Developer. Android. WifiP2pManager. Dostopno na: https://developer.android.com/reference/android/net/wifi/p2p/WifiP2pManager.html [29. avgust 2016]

[23] Developer. Android. Bluetooth. Dostopno na: https://developer.android.com/guide/topics/connectivity/bluetooth.html [29. avgust 2016]

[24] Developer. Android. Camera. Dostopno na: https://developer.android.com/reference/android/hardware/Camera.html [29. avgust 2016]

[25] StackOverflow. Java. Dostopno na: http://stackoverflow.com/questions/tagged/java [29. avgust 2016]

[26] Developer. Android. Bitmap Config. Dostopno na: https://developer.android.com/reference/android/graphics/Bitmap.Config.html [2. september 2016]

[27] Google Play. Camera Remote. Dostopno na: https://play.google.com/store/apps/details?id=com.busywww.cameraremote&hl=sl [5. september 2016]

[28] Google Play. Live Stream. Dostopno na: https://play.google.com/store/apps/details?id=j0y.projects.livestream&hl=sl [5. september 2016]

[29] Google Play. Droid Camera Stream [Viewer]. Dostopno na: https://play.google.com/store/apps/details?id=droid.camera.stream.viewer [5. september 2016]

[30] Google Play. Droid Camera Stream [Host]. Dostopno na: https://play.google.com/store/apps/details?id=droid.camera.stream.host&hl=sl [5. september 2016]

[31] Google Play. Spy Camera Monitor. Dostopno na: https://play.google.com/store/apps/details?id=com.deep.monitor [5. september 2016]

[32] Developer. Android. NetworkOnMainThreadException. Dostopno na: https://developer.android.com/reference/android/os/NetworkOnMainThreadException.html [5. september 2016]

ZA ŠTUDENTE E, ITK, MK, R-IT in TK

Univerza v Mariboru

FERI

Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE ZAKLJUČNEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV

Ime in priimek avtorja-ice: Nejc Novak

Vpisna številka: E1066791

Študijski program: Računalništvo in informacijske tehnologije

Naslov zaključnega dela: Sprotni prenos video vsebine med mobilnimi platformami z operacijskim sistemom Android

Mentor: Aleš Holobar

Somentor:

Podpisani-a Nejc Novak izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Zaključno delo sem izdelal-a sam-a ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija zaključnega dela je istovetna z elektronsko verzijo elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.

Zaključno delo zaradi zagotavljanja konkurenčne prednosti, varstva industrijske lastnine ali tajnosti podatkov naročnika: _____ ne sme biti javno dostopno do _____(datum odloga javne objave ne sme biti daljši kot 3 leta od zagovora dela).

Podpisani izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum zaključka študija, naslov zaključnega dela), na spletnih straneh in v publikacijah UM.

Datum in kraj: Maribor, 7.9.2016 Podpis avtorja-ice: _____

Podpis mentorja: _____
(samo v primeru, če delo ne sme biti javno dostopno)

Podpis odgovorne osebe naročnika in žig:
(samo v primeru, če delo ne sme biti javno dostopno) _____

ZA ŠTUDENTE E, ITK, MK, R-IT in TK

Univerza v Mariboru

Fakulteta za elektrotehniko, računalništvo in informatiko

FERI

IZJAVA O USTREZNOSTI ZAKLJUČNEGA DELA

Podpisani mentor :

Aleš Holobar

(ime in priimek mentorja)

in somentor (eden ali več, če obstajata):

(ime in priimek somentorja)

Izjavljam (-va), da je študent

Ime in priimek: Nejc Novak

Vpisna številka: E1066791

Na programu: Računalništvo in informacijske tehnologije

izdelal zaključno delo z naslovom:

Sprotni prenos.video vsebine med mobilnimi platformami z operacijskim sistemom Android

(naslov zaključnega dela v slovenskem in angleškem jeziku)

Real-time video transmission between mobile platforms with Android operating system

v skladu z odobreno temo zaključnega dela, Navodilih o pripravi zaključnih del in mojimi (najinimi oziroma našimi) navodili.

Preveril (-a, -i) in pregledal (-a, -i) sem (sva, smo) poročilo o plagiatorstvu.

Datum in kraj:
Maribor, 7.9.2016

Podpis mentorja:

Datum in kraj:

Podpis somentorja (če obstaja):

Priloga:

- Poročilo o preverjanju podobnosti z drugimi deli.

ZA ŠTUDENTE E, ITK, MK, R-IT in TK

Univerza v Mariboru

Fakulteta za elektrotehniko, računalništvo in informatiko

FERI

IZJAVA O AVTORSTVU

Spodaj podpisani/-a Nejc Novak

z vpisno številko E1066791

sem avtor/-ica diplomskega dela z naslovom: Sprotni prenos video vsebine med mobilnimi platformami z operacijskim sistemom Android

(naslov diplomskega dela)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

izr. prof. dr. Aleš Holobar, dipl. inž. rač. in inf.

in somentorstvom (naziv, ime in priimek)

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela.
- soglašam z javno objavo elektronske oblike diplomskega dela v DKUM.

V Mariboru, dne 7.9.2016

Podpis avtorja/-ice: