VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

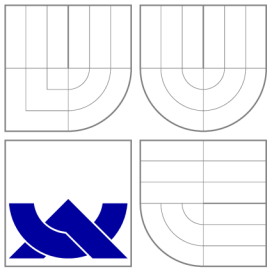
MOBILNÍ APLIKACE ZPRACOVÁNÍ OBRAZU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

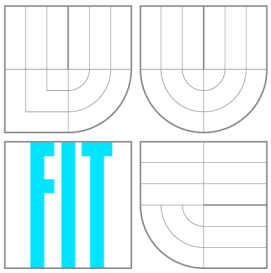
AUTOR PRÁCE
AUTHOR

Bc. JAN PLANER

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE ZPRACOVÁNÍ OBRAZU

MOBILE APPLICATION OF IMAGE PROCESSING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN PLANER

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Dr. Ing. PAVEL ZEMČÍK

BRNO 2015

Abstrakt

Práce se zabývá přenosem obrazu pomocí Bluetooth technologie mezi telefonem a chytrými hodinkami a to především s ohledem na omezené hardwarové prostředky hodinek. Součástí práce je návrh a implementace aplikace typu klient-server, která realizuje přenos a zpracování obrazu z Android telefonu na displej hodinek, které používají systém Tizen.

Abstract

This thesis addresses video streaming from smartphone to smartwatch. The main goal of the thesis is to find a way how to handle the limited hardware features of the watch and make video streaming as fast and smooth as possible. A part of this work is focused on implementation of client-server application, which streams video from Android device to Tizen smartwatch over Bluetooth channel.

Klíčová slova

Streaming videa na mobilním zařízení, Android, Tizen, FFmpeg

Keywords

Video streaming, Android, Tizen, FFmpeg

Citace

Jan Planer: Mobilní aplikace zpracování obrazu, diplomová práce, Brno, FIT VUT v Brně, 2015

Mobilní aplikace zpracování obrazu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Prof. Dr. Ing. Pavla Zemčíka.

.....
Jan Planer
1. srpna 2015

Poděkování

Rád bych poděkoval svému vedoucímu Prof. Dr. Ing. Pavlu Zemčíkovi za odborné vedení a podněty, které mi při řešení této práce poskytl.

Obsah

Kapitola 1

Úvod

V současnosti používá stále více lidí ke každodenním činnostem chytrý telefon (neboli smartphone), do kterého lze instalovat aplikace třetích stran. S rozvojem technologií se v poslední době na trhu objevují i další typy přenosných zařízení s vlastním operačním systémem. Důraz je v této oblasti kladen především na nositelnou elektroniku. Objevují se tak například chytré hodinky (smartwatch), chytré brýle, různé fitness náramky a spolu s nimi i nové typy aplikací.

Řada z nich využívá kombinace chytrého telefonu a nositelného zařízení ke zvýšení komfortu užívání aplikace samotné. Například chytré hodinky mohou uživatele upozornit na nepřečtený email či přijatou SMS zprávu, aniž by musel kontrolovat telefon. V budoucnu si lze představit i daleko složitější scénáře. Kupříkladu chytrý fitness náramek by mohl vyhodnotit blížící se srdeční chorobu a odeslat prostřednictvím telefonu SMS zprávu s potřebnými daty lékaři daného uživatele.

Oblast mobilních aplikací je důležitá především z důvodu prudce se zvyšujícího významu trhu se smartphony a jinou nositelnou elektronikou. Zatímco za rok 2009 bylo celosvětově prodáno přibližně 170 mil. mobilních zařízení, v roce 2013 již stoupl tento počet na více než 1 mld. prodaných kusů. Naopak prodeje stolních počítačů v poslední době spíše stagnují a tak se dá předpokládat i do budoucna rostoucí význam tohoto trhu.

Díky mobilním zařízením nosí v Evropě více než polovina lidí v kapse výpočetní zařízení principiálně podobné klasickému počítači. Sledování vývoje napříč různými mobilními platformami a příbuznými odvětvími mě vždy bavilo. Zároveň mě zajímají i oblasti počítačového vidění a grafiky, a proto jsem si vybral právě toto téma diplomové práce, ve kterém mohu zkombinovat všechny zmíněné oblasti zájmu.

Cílem této práce je vytvořit aplikaci typu klient-server, která umožňuje efektivní přenos obrazu mezi chytrým telefonem a smartwatch hodinkami. Díky takové aplikaci například uživatel nemusí využívat funkce samospouště ve fotoaparátu mobilu, ale může pořídit fotografii jednoduše pomocí hodinek. Na hodinky je přitom přenášen náhled právě pořizované fotografie, tudíž si může být jistý, že bude pořízena tak, jak původně zamýšlel. Důležitá je především rychlost přenosu tak, aby uživatel mohl nejlépe v reálném čase sledovat obraz pořízený kamerou spárovaného zařízení.

V následující kapitole bude provedeno srovnání nejběžnějších mobilních platforem. Dále zde budou představeny blíže platformy Android a Tizen, které jsou z pohledu této práce klíčové, spolu s dostupnými knihovnami a vývojářskými nástroji použitelných ve výsledné aplikaci. Dále bude také představeno několik existujících řešení na vybraných platformách. V kapitole 3 budou nejprve zhodnocena existující řešení a dále budou představeny některé použitelné technologie pro vývoj výsledné aplikace. Na konci kapitoly budou stanoveny

požadavky na výslednou aplikaci. Kapitola 4 se zabývá vlastní implementací aplikace. Ze začátku bude vytvořena řada testů, na základě kterých bude vybrán nejvhodnější způsob přenosu a zobrazení videa. Dále bude následovat popis návrhu, implementace, vyhodnocení a otestování aplikace. V kapitole 5 budou shrnuty dosažené výsledky a nastíněn možný další budoucí vývoj.

Kapitola 2

Přenos obrazu na mobilních platformách

Téma této práce se zabývá zpracováním a přenosem obrazu na mobilních platformách Android telefonu a Tizen hodinek. V první části této kapitoly tedy budou představeny nejvýznamnější mobilní platformy (Android, iOS, Windows Phone) a dále také platforma Tizen, která je rozšířeným systémem na poli chytrých hodinek. Větší pozornost bude věnována právě systémům Android a Tizen, které jsou klíčové z hlediska řešení práce. U těchto platforem bude dále popsána také jejich architektura a životní cyklus aplikace běžící na daném operačním systému.

Druhá část kapitoly bude věnována přenosu obrazu na mobilních platformách. Kromě výčtu některých existujících řešení zde bude řeč také o `MPEG` formátu videa a dále také o *FFmpeg* knihovně a jejích alternativách.

Komplexní popis všech vlastností zmíněných operačních systémů a technologií umožňujících přenos obrazu mezi dvěma zařízeními by byl nad rámec rozsahu této práce. Z toho důvodu jsou v této kapitole vybrány pouze některé relevantní vlastnosti vybraných systémů či knihoven.

2.1 Přehled významných mobilních systémů

K pochopení současného vývoje a nových trendů na poli mobilních platforem je třeba nahlédnout rámcově na historický vývoj tohoto odvětví. První mobilní telefon vůbec zkonstruoval v roce 1984 po několikaletém vývoji doktor Martin Cooper. Telefon vážil zhruba 2 Kg a vzhledově připomínal spíše velkou vysílačku, než zařízení, jaká známe dnes.

Za první chytrý telefon se považuje zařízení *IBM Simon*, jehož prototyp byl představen v roce 1992. Telefon běžel nad operačním systémem *ROM-DOS* a kromě volání nebo posílání SMS zpráv umožňoval také komunikovat prostřednictvím emailu či faxu. Dále také obsahoval řadu aplikací, jako například osobní kalendář, seznam kontaktů, kalkulátor, poznámky a jiné.

V roce 1996 představila společnost Nokia smartphone *Nokia 9000 Communicator*, který podobně jako *IBM Simon* k běhu využíval operační systém *ROM-DOS*. Telefon se stal velmi rychle populárním a odstartoval tak rychlý vývoj v oblasti chytrých telefonů. V této době taky vzniká řada historicky významných mobilních operačních systému, které již mají v dnešní době minoritní podíl na trhu. Mezi nimi jsou například *BlackBerry OS* (1999),

(a) Android Honeycomb

(b) Android Lollipop

Obrázek 2.1: Ukázka systému Android. Zdroj: *Wikipedia*

Windows Mobile[1] (2000), *Symbian* (1998) a jiné.

Android

První zmínky o platformě Android se datují do roku 2003, kdy zaměstnanec společnosti Apple Andy Rubin založil malou společnost s názvem Android Inc. Jejím cílem bylo vyvinout systém, který lépe využívá možností mobilního zařízení a lépe bude vyhovovat uživatelským nárokům. v roce 2005 byla Android, Inc. odkoupena společností Google a o tři roky později byla vydána první verze systému Android, která byla založena na linuxovém jádře. Android 1.0 umožňoval uživatelům stahovat aplikace z Android Marketu a používat řadu vestavěných služeb [9].

Dále jsou uvedeny některé významné verze systému [6]:

- **Cupcake** – Verze z roku 2009. Přinesla řadu vylepšení uživatelského rozhraní (dále UI) a podporu pro přehrávání `MPEG-4` a `3GP` videí.
- **Donut (2009)** – Přidáno vyhledávání založené na mluveném slovu a také syntéza textu na zvuk.
- **Froyo (2010)** – Vylepšena podpora JavaScriptu v internetovém prohlížeči, přidána podpora pro instalování aplikací na externí paměť telefonu, podpora *Adobe Flash*, podpora obrazovek s vysokým počtem bodů na palec (až 320 DPI).
- **Gingerbread (2010)** – Nativní podpora protokolů `SIP VoIP`, `NFC`, nativní podpora doplňujících senzorů (např. gyroskop), podpora obrazovek s velmi vysokým rozlišením, vylepšen výkon systému.
- **Honeycomb (2011)** – Podpora vícejádrových procesorů, vylepšeno UI tak, aby lépe využívalo větší obrazovku tabletů.
- **Ice Cream Sandwich (2011)** – Vylepšení uživatelského rozhraní, přidána zamykací obrazovka.
- **Jelly bean (2012)** – Přidán systém notifikací, vylepšení UI společně s podporou pro tvorbu složitějších prvků, podpora `OpenGL ES 3.0`, podpora `BLE` protokolu.

[1] Jedná se o odlišný systém, než je jeho moderní nástupce Windows Phone

Obrázek 2.2: Srovnání vzhledu platforem iOS 6 a iOS 7 [16]

.

- **KitKat (2013)** – Vylepšení výkonu systému, vylepšení využití paměti systému tak, aby mohl běžet na více různých zařízeních.
- **Lollipop (2014)** – Výrazné změny v UI (*Material design*), podpora UI pro nositelnou elektroniku (Android Wear) a chytré televizory (Android TV).

Apple iOS

Operační systém Apple iOS (dále jen zkráceně iOS) je další systém založený na unixovém jádře. Na rozdíl od Androidu se však jedná o uzavřenou platformu vyvíjenou společností Apple Inc. První verze systému (tehdy ještě pojmenovaného jako iPhone OS) byla představena v roce 2008.

Následující verze přinesla obchod *App Store*, pomocí nějž lze do telefonu instalovat aplikace třetích stran. Další významná verze systému byl iOS 4.0, ve kterém kromě nových funkcí byla přidána podpora paralelního běhu více aplikací (tzv. *multitasking*). Verze 5.0 představená v roce 2011 obsahovala vylepšený systém notifikací, podporu pro cloudové služby společnosti Apple a také nativní integraci sociální sítě Twitter. Dále byla v této verzi také představena virtuální asistentka Siri, která na základě hlasových příkazů pomáhá uživatelům např. vyhledávat informace na internetu. Zatímco ve verzi 6.0 byla v podstatě pouze nadále vylepšena integrace sociálních sítí do telefonu, zásadnější změny přinesla v roce 2013 verze 7.0 [16].

Podobně jako v pozdější verzi Androidu byl představen *Material Design*, o něco dříve představil Apple tzv. *Flat Design*, čili vzhled uživatelských prvků, který je velice jednoduchý, bez zbytečných stínů, gradientů apod. V současnosti je aktuální verze systému 8.0 [16]. Srovnání vzhledů obou verzí platformy je vidět na obrázku 2.2.

(a) Windows Phone 8.1.

(b) Uživatelské rozhraní systému Windows pro dotykové zařízení.

Obrázek 2.3: Ukázka systému Windows. Zdroj: `microsoft.com`

Windows Phone

Windows Phone je poměrně mladý systém představený společností Microsoft v roce 2010. Na první pohled zaujme poměrně netradičním vzhledem ve formě různých dlaždic, které mají velice jednoduchý vzhled. Podobně jako každá jiná mobilní platforma disponuje i Windows Phone vlastním internetovým prohlížečem (obdobou Internet Exploreru), integrací sociálních sítí do nativních služeb telefonu, či vlastním obchodem s názvem *Windows Store* s aplikacemi třetích stran [20].

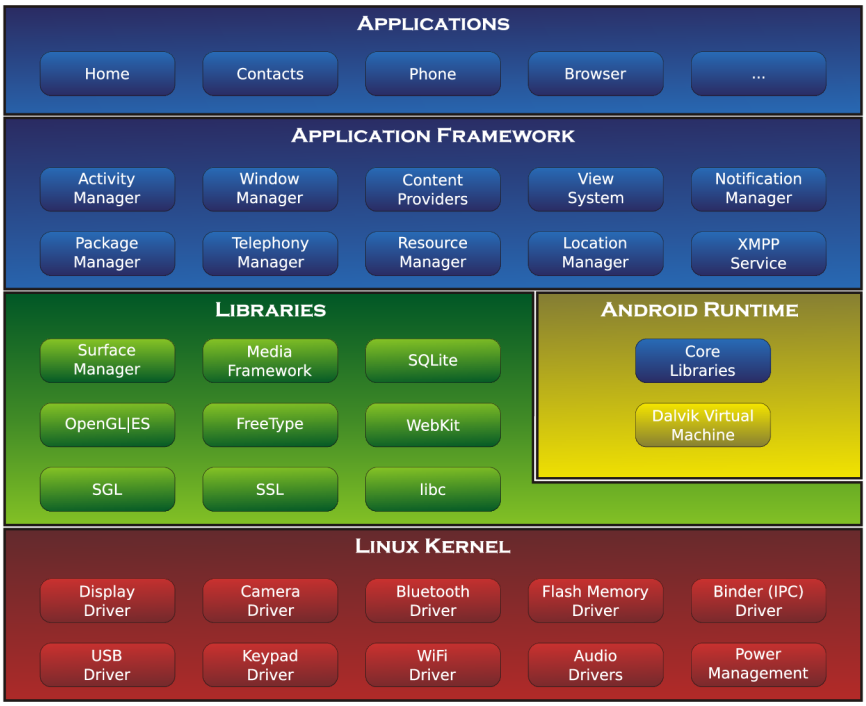
Stejně jako v případě platformy Apple iOS je i na Windows Phone přítomna virtuální asistentka pojmenovaná Cortana. Windows Phone se vyskytuje ve třech zpětně nekompatibilních verzí – Windows Phone 7.0, 8.0 a 8.1 (neuvažujeme-li různé méně důležité aktualizace) [20]. V budoucnu se spekuluje, že bude platforma Windows Phone zcela sloučena s budoucí verzí desktopového systému Windows 10 [19].

Na rozdíl od většiny běžně používaných mobilních operačních systému neběží Windows Phone nad unixovým jádrem, nýbrž nad softwarovou vrstvou zvanou *Windows Runtime* (nebo zkráceně také jen *WinRT*). Prostřednictvím *Windows Runtime* může aplikace komunikovat s jádrem operačního systému a využívat jeho služby.

Aplikace pro systém Windows Phone lze psát v několika různých programovacích jazycích. Mezi nejčastější patří C#, VB.NET či JavaScript, nativní aplikace pak lze psát v C++/CX. Zajímavým konceptem je způsob překládání aplikací napsaných v `.NET` jazycích (nejčastěji se používá jazyk C#). Místo překládání aplikace za běhu pomocí virtuálního stroje (neboli pomocí JIT překladače) je zdrojový kód přeložen nejprve pomocí běžného C# překladače do nízkoúrovňového CIL jazyka.

Takto vzniká kód, který lze zabalit do aplikačního balíčku a odeslat do obchodu *Windows Store*. Dále je aplikace přeložena do MDIL kódu (z angl. *Machine Dependent Intermidiate Language*) za pomocí cloudových služeb. MDIL kód je pak na cílovém zařízení velice jednoduše převeden do nativního kódu a spuštěn na procesoru daného zařízení [17].

Výsledný kód je mnohem rychlejší, než interpretovaný kód virtuálním strojem. Vzdáleně podobný koncept, který bude diskutovaný v podkapitole 2.2, přináší také novější verze systému Android.



Obrázek 2.4: Architektura platformy Android. Zdroj: *Wikipedia*

2.2 Architektura systému Android

Architektura celého systému je naznačena na obrázku 2.4. Jak již bylo řečeno v podkapitole 2.1, systém je založený na linuxovém jádře, které odpovídá červené vrstvě v obrázku 2.4. Jádro tvoří abstraktní vrstvu mezi hardwarem zařízení a zbytkem systému. Ze systému Linux je v Androidu převzatou velké množství vlastností, jako např. správa paměti, procesů, síťových prostředků, ovladačů hardwaru apod.

Nad linuxovým jádrem je mezivrstva různých knihoven (vrstva s názvem *Libraries*), které jsou napsány v jazycích C nebo C++. Tyto knihovny jsou poté zpřístupněny vývojáři prostřednictvím vrstvy *Application Framework*. Níže jsou uvedeny některé příklady knihoven [13]:

- **OpenGL ES** – Knihovna určená k vykreslování 3D grafiky.
- **SQLite** – Databázová vrstva.
- **Media Framework** – soubor knihoven určených pro práci s multimédii.
- **Libc** – Standardní knihovna jazyka C upravená pro vestavěné systémy.

Další vrstvou je již zmíněný aplikační rámec (*Application Framework*). Jedná se o část systému, která je v praxi pro vývojáře nejdůležitější. Umožňuje mu přistupovat k nejrůznějším službám systému, pomocí kterých lze například komunikovat s jednotlivými částmi hardwaru, spouštět jiné aplikace na pozadí, přistupovat k prvkům uživatelského rozhraní apod [13].

Některé služby aplikačního rámce:

- **View System** – Prvky pro tvorbu uživatelského rozhraní, jako jsou např. seznamy, textové pole, tlačítka a jiné.
- **Activity Manager** – řídí životní cyklus aplikací a poskytuje orientaci v zásobníku s aplikacemi.
- **Notification Manager** – Umožňuje aplikacím přidávat vlastní upozornění ve stavovém řádku.
- **Content Providers** – Umožňuje pracovat s obsahem jiných aplikací, jako např. Kontakty v telefonu, kalendář apod.

Android Runtime

V popisu architektury systém (obrázek 2.4) byla záměrně vynechaná část *Android Runtime*, které bude věnovaná celá tato část podkapitoly.

Aplikace pro systém Android jsou vyvíjeny v jazyce Java. Běžný program napsaný v tomto jazyce je nejprve přeložený do byte kódu a poté je byte kód spuštěn ve virtuálním stroji *Java Virtual Machine* (dále jen JVM). Z licenčních důvodů však nebylo možné JVM v systému Android použít. Proto byl do verze Android 4.3 v systému používán virtuální stroj *Dalvik Virtual Machine* (dále jen DVM), který je vyvíjen společností Google. DVM je oproti JVM optimalizován pro mobilní zařízení a liší se také podporou některých Java knihoven. Překlad aplikace napsané pro Android 4.3 a starší poté probíhá tak, že je Java kód přeložen stejným překladačem jako v případě Java aplikace do byte kódu a ten je poté pomocí Dalvik kompilátoru přeložen do byte kódu kompatibilního s DVM [21].

Od verze Android 4.4 je situace poněkud jiná. Z důvodu zpětné kompatibility je v systému i nadále přítomen původní DVM. Nové aplikace jsou však po nainstalování na systém přeloženy do byte kódu Dalviku a poté je tento byte kód přeložen do nativního kódu, který může být spuštěn přímo na procesoru daného zařízení. Tím dochází k výraznému zrychlení celého systému a také k šetření baterie zařízení [12].

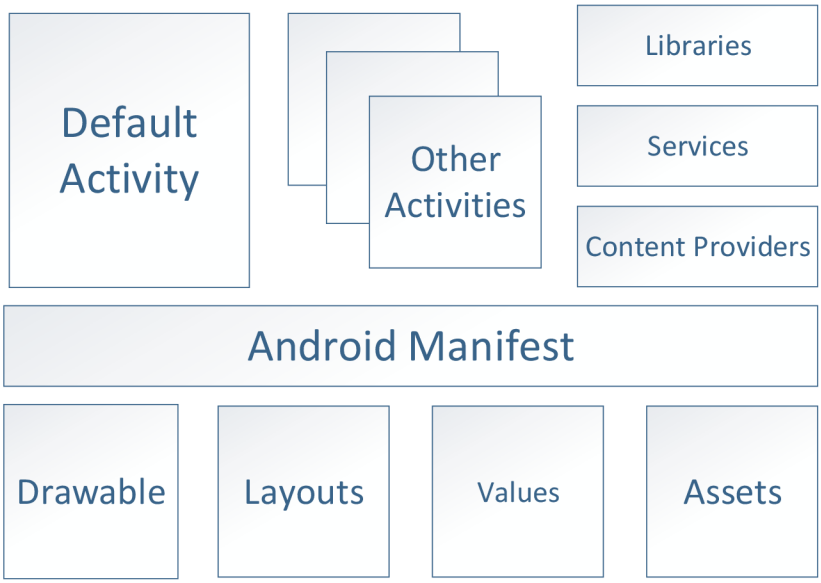
2.3 Vývoj Android aplikace

Pro vývoj Android aplikací existuje několik různých vývojových prostředí – například *IntelliJ IDEA*, *Netbeans*, *Eclipse* a jiné. v roce 2013 představila společnost Google vývojové prostředí *Android Studio* (založené na platformě *IntelliJ IDEA*), které se od začátku specializuje na vývoj Android aplikací. V prosinci roku 2014 byla oficiálně vydaná stabilní verze. Součástí *Android Studia* je i emulátor mobilního zařízení, tudíž se vývojem aplikací může zabývat i vývojář, který nevlastní mobilní telefon.

Struktura Android aplikace

Android aplikace je podobně jako Java aplikace organizovaná do adresářů a podadresářů s předem danou strukturou. Na kořenové úrovni můžeme nalézt (mimo jiné) tyto položky [4]:

- **AndroidManifest.xml** – XML soubor popisující aplikaci samotnou a systémové služby, které aplikace využívá. Příkladem takové služby může být například fotoaparát zařízení či přístup ke kontaktům uloženým v telefonu.



Obrázek 2.5: Logická struktura Android aplikace.

- **bin/** – Cílová složka, kde je uložena aplikace, jakmile je přeložena.
- **libs/** – Obsahuje JAR (Java Archive) knihovny třetích stran.
- **res/** – Obsahuje veškeré prvky uživatelského rozhraní, obrázky, ikony apod.
- **src/** – Zdrojové soubory samotné aplikace.
- **assets/** – Obsah této složky je přibalen do výsledné aplikace. Mohou zde tedy být v podstatě libovolné statické soubory související s aplikací.
- **gen/** – Obsahuje další zdrojové soubory vygenerované vývojovým prostředím.

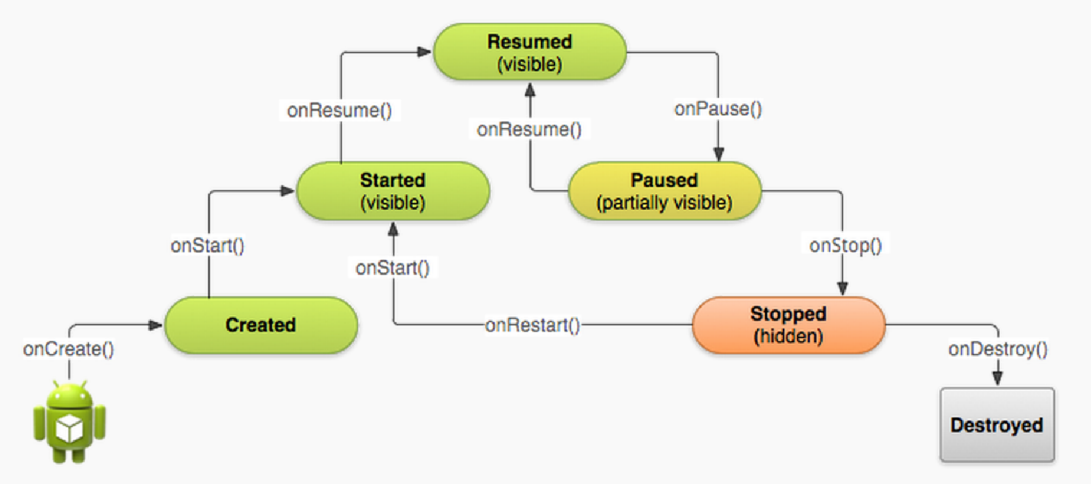
Na obrázku 2.5 je vidět logická struktura Android aplikace. Každá aplikace se skládá z jedné nebo více aktivit, prvků uživatelského rozhraní, může obsahovat přídavné knihovny, využívat nejrůznějších systémových služeb apod.

Aktivita

Aktivita (anglicky *Activity*) by měla reprezentovat, jak už její název napovídá, jednoúčelovou obrazovku aplikace. Aplikace samotná se poté může skládat z více těchto aktivit. Po spuštění aplikace je automaticky vytvořena a spuštěna hlavní aktivita a různými uživatelskými akcemi může dojít k přesměrování na další aktivity.

Životní cyklus aktivity

Systém Android je navržen pro mobilní zařízení s potenciálně nízkou kapacitou RAM paměti. V případě, že by paměť najednou začala docházet, bude muset systém tuto situaci vyřešit – a to tak, že ukončí některé běžící aplikace. Vývojář tedy musí počítat s tím, že



Obrázek 2.6: Životní cyklus aktivity. Zdroj: `developer.android.com`

jeho aplikace může být kdykoliv ukončena. Z toho důvodu se každá aktivita aplikace řídí svým životním cyklem, v rámci kterého může vývojář odchytit důležité události [4].

Celý životní cyklus aktivity je popsán na obrázku 2.6. Aktivita se tedy může nacházet ve třech různých stavech [4]:

- **Běžící** – Aktivita je v popředí a dostává informace o uživatelském vstupu. V tomto stavu jsou všechny aktivity, se kterými uživatel pracuje.
- **Pozastavená** – Pozastavené aktivity bývají částečně překryté jinou aktivitou, případně zcela překryté průhlednou aktivitou. Jsou tedy pořád vidět, nicméně již nedostávají informace o žádném uživatelském vstupu.
- **Zastavená** – Zastavená aktivita již není vůbec vidět, systém ji však ještě neodstranil z paměti.
- **Odstraněná** – Aktivita buď nebyla ještě spuštěna (např. po restartu telefonu) nebo byla systémem ukončena z důvodu nedostatku paměti.

Dále lze v rámci životního cyklu aplikace obsloužit různé důležité události, ke kterým dochází v momentě, kdy se mění stav aktivity. V rámci těchto událostí je typicky zapotřebí alokovat či uvolňovat systémové zdroje, ukládat si stav aktivity pro budoucí uložení apod [4].

- **onCreate** – Tato událost nastává buď v případě, že aplikace je poprvé spuštěna (například po restartu systému) nebo poté co někdy dříve byla systémem ukončena. Typicky je zde inicializované uživatelské rozhraní.
- **onDestroy** – Volané v momentě, kdy je aplikace ukončena.[2] V rámci této události bývají většinou uvolněné zdroje vytvořené v `onCreate` události.
- **onStart, onRestart, onStop** – V momentě, kdy se aplikace přenese do popředí, je volaná událost `onStart`. To nastává buď ve chvíli, kde je aplikace spuštěna, nebo

[2]Pokud systému kriticky dochází paměť, může být volání `onDestroy` vynechané (například v případě přicházejícího telefonního hovoru).

poté co byla zastavena a po nějaké době znovu spuštěna. Při zastavení aktivity je volána událost `onStop`. Při jejím následovném spuštění jsou volány popořadě události `onRestart` a `onStart`.

- **onResume** – Tato událost je spuštěna ihned poté, co byla aktivita spuštěna (ať už při prvním spuštění nebo po restartu), případně poté, co bylo zavřen takzvaný pop-up. dialog nad aktivitou (vyskakující modální dialog). V obsluze této události jsou většinou aktualizované prvky uživatelského rozhraní, které mohly být změněny od posledního zobrazení.
- **onPause** – Párová událost k události `onResume`, která je volaná před událostí `onStop`. Zde by měl programátor obecně uvolnit zdroje alokované v události `onResume`.

Intent

V řadě případů je zapotřebí zaslat zprávu (v technické dokumentaci systému označenou jako *Intent*) napříč jednotlivými systémovými komponenty. Typickými příkladem můžou být například zprávy o změně hardwaru (vložení SD karty), příjem různých dat (např. SMS zprávy) či události spojené s aplikacemi (např. může uživatele zajímat, že daná aplikace byla spuštěna z hlavního menu zařízení). Aplikace může poté tyto systémové zprávy odchytávat a vhodně na ně reagovat nebo také může vytvářet nové zprávy, na které mohou reagovat jiné aplikace. Implementačně se jedná o velice jednoduchý objekt, který může uchovávat libovolná data.

V tabulce 2.1 jsou shrnuty základní atributy každé takové zprávy.

Atribut	**Popis**
Action	Řetězec jednoznačně popisující danou akci. Např. `android.intent.action.DIAL`.
Category	Popisuje, kde a jak může být Intent použit. Například může být použit pouze z hlavního menu telefonu či z webového prohlížeče.
Component	Může obsahovat název třídy, pro kterou je Intent určený.
Data	Datová část intentu. Obecně může obsahovat libovolná data.
Extras	Extra data, která mohou být uložena do Intentu.
Type	Specifikuje `MIME` typ dat uložených v Intentu. Příkladem typu dat může být například `text/plain či vnd.android.cursor.item/email_v2`.

Tabulka 2.1: Struktura Android Intentu [10]

.

Content Provider

V některých případech chceme, aby aplikace umožňovala jiným aplikacím (případně jiným aktivitám) sdílení obsahu. Právě k tomu účelu slouží tzv. poskytovatel obsahu (angl. *Content Provider*). Některé tyto komponenty jsou již v systému předdefinované a umožňují například přístup k multimédiím či osobním kontaktům uloženým v telefonu. Každý *Content*

Provider svým chováním připomíná klasickou databázi s metodami pro získávání, aktualizování, mazání či vkládání záznamů [10].

2.4 Android NDK

Doposud byla řeč hlavně o jazyku Java, který je díky svými objektovými vlastnostmi preferovaný při vývoji širokého spektra aplikací (nejen) na platformě Android. Někdy však může být výhodnější využít nativní jazyk C/C++ a to zejména v těchto případech:

- Výsledná aplikace je výpočetně náročná.
- Uživatel chce využít existující knihovnu, která již je napsaná v jazyce C/C++.
- Je zapotřebí nízkoúrovňový přístup k některým částem systému a neexistuje k tomu dané Java Api.
- Výsledný kód má být přenositelný.

K těmto účelům slouží *Android development kit* (dále jen NDK), který obsahuje knihovny a hlavičkové soubory nezbytné k provázání nativního kódu a zbytku aplikace napsaného v jazyce Java. Dále obsahuje také sadu nástrojů sloužící k překladu C/C++ kódu, dokumentaci, ukázkové aplikace apod [15].

V nejnovější verzi NDK (*Android NDK, revision 10d*) jsou podporované následující instrukční sady:

- `ARMv5TE` včetně `THUMB-1` instrukcí
- `ARMv7-A` včetně `THUMB-2`, `VFPv3-D16` a volitelnou poporou pro `NEON/VFPv3-D32` instrukce
- **x86** instrukce
- `MIPS` instrukce

Více podrobností o jednotlivých instrukčních sadách lze nalézt v technické dokumentaci – viz [1].

S pomocí Android NDK lze vytvořit jak čistě nativní aplikaci, tak hybridní aplikaci, v rámci které je možné kombinovat nativní kód s Java kódem. Typickým scénářem pro hybridní typ aplikace může být například situace, kdy se programátor rozhodne v již existující Java aplikaci využít nějakou C/C++ knihovnu. V následujících odstavcích bude uvažován pouze hybridní typ aplikací.

Java Native Interface

K provázání obou částí aplikace slouží tzv. *Java Native Interface* (dále jen JNI). Díky němu lze v kódu interpretovaném virtuálním strojem volat nativní metody a naopak.

V ukázce kódu 2.1 je primitivní Java aplikace, která po spuštění zavolá nativní metodu `sayHello`. Za zmínku stojí klíčové slovo `native` v definici metody `sayHello`. Díky

```java
public class HelloJNI {
    static {
        System.loadLibrary("hello");
    }
    private native void sayHello();

    public static void main(String[] args) {
        new HelloJNI().sayHello();
   }
}
```

Výpis kódu 2.1: Java aplikace volající nativní metodu. Zdroj: [14]

```c
// Soubor HelloJNI.h
#include <jni.h>

// ...
JNIEXPORT void JNICALL Java_HelloJNI_sayHello(JNIEnv *, jobject);
// ...

// Soubor HelloJNI.c
#include <jni.h>
#include <stdio.h>
#include "HelloJNI.h"

JNIEXPORT void JNICALL Java_HelloJNI_sayHello(JNIEnv *env, jobject thisObj)
{
    printf("Hello World!\n");
}
```

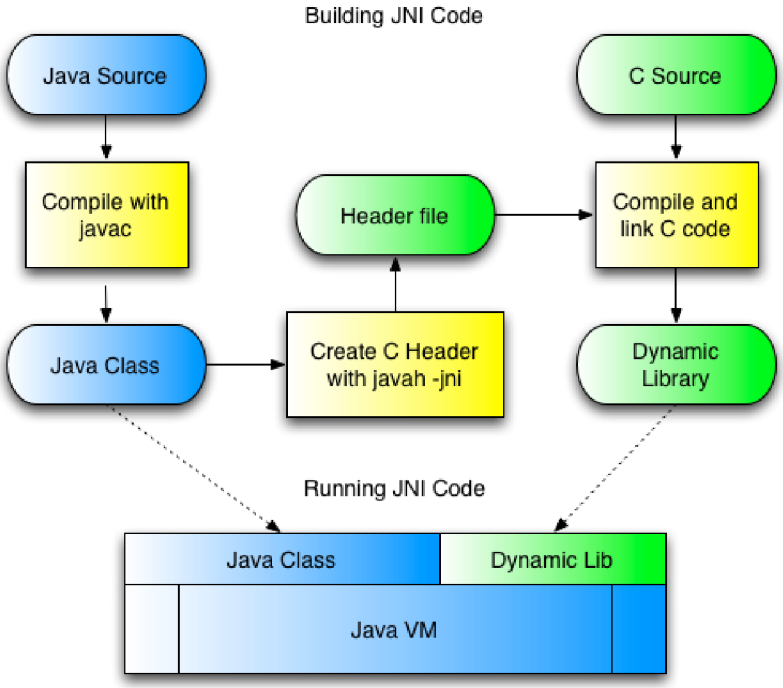
Výpis kódu 2.2: Nativní metoda volaná Java aplikací. Zdroj: [14]

tomu interpret Java byte kódu pozná, že definici dané metody má hledat jinde – konkrétně ve sdílené knihovně `hello`, která je načtena při spuštění aplikace pomocí příkazu `System.loadLibrary`.

Implementace metody `sayHello` je uvedena v ukázce kódu 2.2. Z pohledu jazyka C se tedy jedná o standardní kód, který lze přeložit do sdílené knihovny. Název nativní metody musí dodržovat určitou jmennou konvenci, díky které je název `sayHello` uvozen řetězcem `Java_HelloJNI_`. Ten nezaměnitelně popisuje třídu, ve které je deklarován Java protějšek dané nativní metody. Dále tato metoda obdrží dva parametry: `env` a `thisObj`.

Parametr `env` reprezentuje samotné JNI. Obsahuje všechny možné potřebné metody v interakci s virtuálním strojem, které slouží například ke konverzi nativních datových typů, vytváření nových objektů, vyvolávání výjimek apod. V podstatě lze voláním těchto metod docílit čehokoliv, co lze naprogramovat v jazyce Java.

Parametr `thisObj` ukazuje na Java objekt, kterému přísluší volaná metoda. V tomto případě by to tedy byla instance třídy `HelloJNI`. Způsob přeložení obou částí do funkčního celku je patrný z obrázku 2.7 [14].

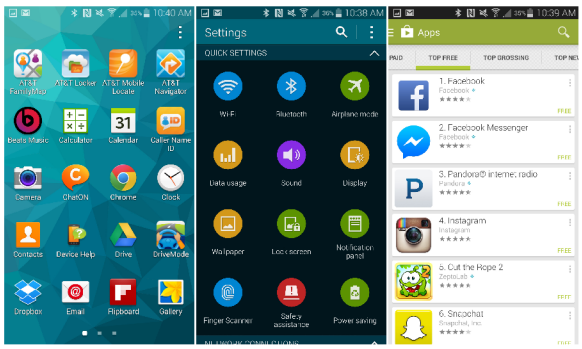


Obrázek 2.7: Schéma hybridní aplikace. Zdroj: `sbalaji-android.blogspot.cz`

2.5 Tizen

Tizen je otevřený operační systém založený na linuxovém jádru. Zaměřuje se na všechny možné vestavěné systémy včetně chytrých telefonů, hodinek, fotoaparátů, televizorů, systémů v automobilech a podobně. Zajímavostí je, že na rozdíl od platformy Android jsou aplikace pro Tizen vyvíjeny za pomocí standardních webových technologií (JavaScript, `HTML5`, `CSS`) a poté spuštěny prostřednictvím webového jádra, které přímo komunikuje s operačním systémem. Nativní aplikace psané v jazyce C++ je možné vyvíjet až od verze Tizen 2.0.

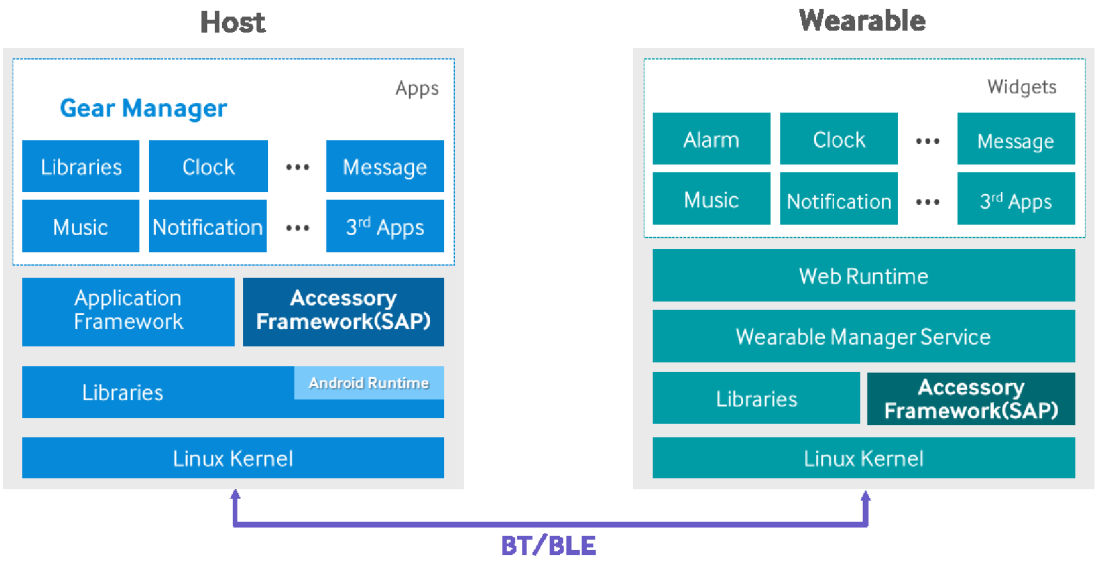
Následující text bude zaměřen především na verzi systému použitou v *Samsung Galaxy Gear Watch* hodinkách. Úvodem je třeba říci, že hodinky jsou vždy spárované se smartphonem (nebo tabletem) – viz obrázek 2.9. Při vzájemné komunikaci obou zařízení vystupuje smartphone zpravidla jako server a hodinky jako klient. Na straně telefonu musí být na-



(a) Obrazovka chyrého telefonu se systémem Tizen.

(b) Chytré hodinky se systémem Tizen.

Obrázek 2.8: Ukázka systému Tizen. Zdroj: `samsung.com`



Obrázek 2.9: Klient-server architektura galaxy gear hodinek spárovaných s hostujícím zařízením. Zdroj: [18]

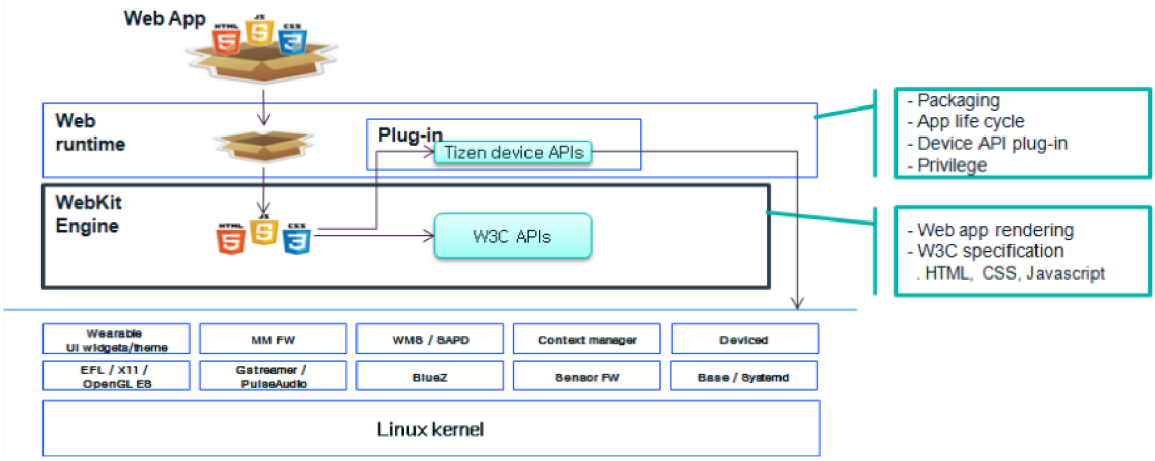
instalovaná aplikace *Gear Manager*, která slouží ke správě dat v hodinkách. Komunikace mezi oběma stranami probíhá prostřednictvím Bluetooth protokolu [18].

Typy aplikací

V rámci klient-server architektury smartwatch hodinek a hostujícího zařízení lze vytvářet následující typy aplikací [18]:

- **Master-Follower aplikace** – Dvojice plnohodnotných aplikací. První běží na hostujícím zařízení, druhá na hodinkách. Tento typ aplikace je vhodný především v případě, kdy obě aplikace mohou fungovat nezávisle na sobě, a při běhu se jedna bez druhé obejde.
- **Integrovaná aplikace** – Jedná se o jedinou aplikaci pro hostující zařízení, která má v sobě zabalenou klient aplikaci pro hodinky v podobě aplikačního balíčku. Aplikace pro hodinky je nainstalovaná automaticky spolu s hostující aplikací. Tento koncept je vhodný především v případech, kdy klientská část aplikace se neobejde bez hostující.
- **Standalone** – Samostatná aplikace pro hodinky. Jedná se zpravidla o jednoduché aplikace, jako jsou například hodiny či stopky, které ke svému běhu nepotřebují jakýmkoliv způsobem komunikovat s telefonem.

Hostující aplikace bývá zpravidla Android aplikace běžící na telefonu či tabletu. Naopak aplikace v hodinkách se spíše podobá webové aplikaci vytvořené v `HTML5` kódu s definovaným vzhledem pomocí kaskádových (`CSS`) stylů. Kód vlastní aplikace je napsán v JavaScriptu [18].



Obrázek 2.10: Webová aplikace v systému Tizen. Zdroj: [7]

2.6 Struktura Tizen aplikace

Jak již bylo řečeno, aplikace na hodinky je napsaná v jazyce JavaScript. K definici vzhledu lze použít standardní nástroje jako při vývoji jakékoliv jiné `HTML5` aplikace – tedy kaskádové styly a `HTML5` definice rozložení stránky. Na rozdíl od stolního počítače je však výkon hodinek značně omezený a interpret JavaScriptu je z principu o něco pomalejší, než nativní kód. Při vývoji výpočetně náročnější aplikace může být poté klíčové vhodně využít ty vlastnosti webového jádra, které jsou hardwarově akcelerované.

Na obrázku 2.10 je vidět, jakým způsobem aplikace komunikuje s linuxovým jádrem prostřednictvím webového jádra. Situace se tedy moc nemění od vykreslení obyčejné webové stránky.

O běh aplikace se stará *Web Runtime*, což je softwarová vrstva, která umožňuje aplikaci běžet mimo webový prohlížeč. Prostřednictvím *Web Runtime* lze komunikovat se systémem. Programátor má tak možnost přistoupit k souborovému systému, systémovému času, k některým hardwarovým komponentám, jako je například kamera či různé senzory apod. O vykreslení stránky se stará webové jádro *WebKit*, což je jádro používané například v prohlížečích Safari nebo Google Chrome[3].

V rámci Tizen Web aplikace je možné využívat většinu vlastností moderních `HTML5/CSS3` vlastností. Níže jsou uvedeny některé příklady [8]:

- **HTML5 prvky** – Například <header>, <mark>, <footer>, <nav> a jiné.
- **Pokročilé CSS3 vlastnosti** – Rotace a translace objektů, animace objektů, stíny, gradienty apod.
- **Nové formulářové prvky** – Email, url, vstup pro vyhledávání, číslo…
- **HTML5 Canvas**
- **WebGL** – Nízkoúrovňové vykreslování 3D grafiky. Podporované API je založené na verzi `OpenGL ES 2.0`.

[3]Prohlížeč Google Chrome využívá webové jádro *Blink*, které z *WebKitu* vychází.

V některých případech může výběr vhodných vlastností webového prohlížeče značně ovlivnit výkon výsledné aplikace a s tím i spojenou spotřebu baterie. V následujících odstavcích jsou shrnuty některé tipy, pomocí kterých lze ušetřit cenných hardwarových prostředků.

Srovnání vykreslování pomocí <Canvas> prvku a CSS vlastností

Mějme aplikaci, ve které bude vykreslována animace analogových hodin, které obsahují hodinovou, minutovou a vteřinovou ručičku. Detailní popis aplikace lze nalézt v [7].

V této situaci je možné zvolit tři různé strategie: Vykreslovat každou ručičku zvlášť na jiný <canvas> prvek, použít jeden sdílený <canvas> prvek, nebo všechny ručičky vykreslovat pomocí pokročilých `CSS3` vlastností (Každá ručička bude tedy reprezentovaná jedním `DOM` elementem, na který jsou aplikované příslušné kaskádové styly). Ve všech případech budou hodiny vykreslovány s maximální frekvencí LCD displeje a cílem bude navrhnout aplikaci tak, aby byla co nejšetrnější z pohledu spotřeby baterie.

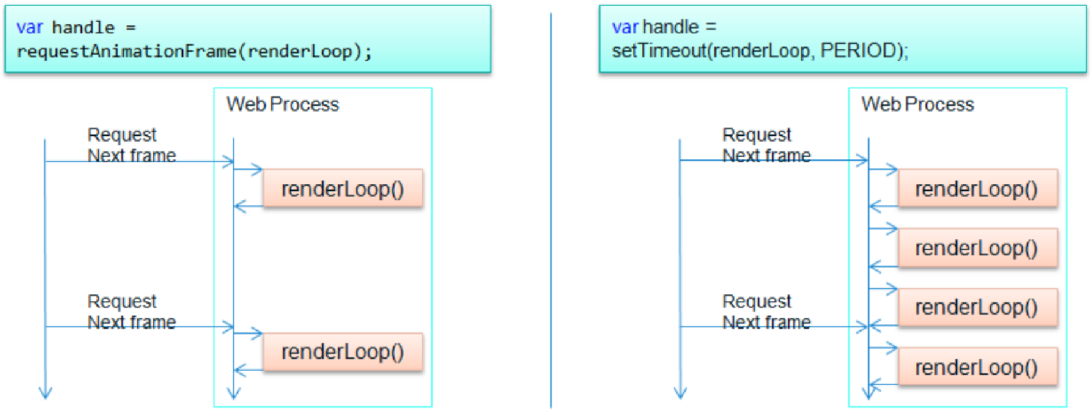
	CSS (3 objekty)	**1 canvas prvek**	**3 canvas prvky**
Výkon (mW)	193.8	224.2	243.2

Tabulka 2.2: Srovnání různých metod vykreslování objektů. Zdroj: [7]

Jak je vidět v tabulce 2.2, řešením vyžadující nejnižší příkon bylo v tomto případě použití pokročilých `CSS3` vlastností. Prvek <canvas> by naopak bylo pravděpodobně výhodnější použít v případě, kdy by bylo vykreslováno velké množství relativně jednoduchých objektů.

Optimalizace vykreslovací smyčky

Dalším způsobem, jak vylepšit výkon aplikace je vhodně navrhnout vykreslovací smyčku. Naivním řešením je použít JavaScriptovou funkci `setTimeout`, které je předáno zpětné volání (angl. *callback*), které je volané pravidelně s danou periodou. K překreslování obrazovky však obecně dochází s jinou periodou, a tak se může stát, že mezi dvěma volání překreslení obrazovky dojde k více (zbytečným) voláním kreslící funkce.



Obrázek 2.11: Srovnání využití funkce `requestAnimationFrame` a `setTimeout`. Zdroj: [7]

Srovnání obou situací je vidět na obrázku 2.11, resp. v tabulce 2.3. Podobně jako v předchozím případě je i zde srovnán potřebný příkon pro běh aplikace.

	requestAnimationFrame	**setInterval**
Výkon (mW)	285	323

Tabulka 2.3: Srovnání efektivní a naivní vykreslovací smyčky. Zdroj: [7]

WebGL

Pro malé aplikace, ve kterých se vykresluje pouze několik málo objektů je zpravidla kreslení pomocí <canvas> prvku dostačující. S narůstajícím množstvím grafických primitiv je však vhodnější zvolit vykreslování pomocí `WebGL`. Následující tabulka srovnává výkon <canvas> prvku s `WebGL` na referenčním mobilním zařízení se systémem Tizen [3]:

Počet objektů	10	30	100	250
Canvas 2D (FPS)	60	45	12	5
WebGL (FPS)	60	60	60	60

Tabulka 2.4: Srovnání výkonu canvas prvku a `WebGL`. Zdroj: [3]

Z tabulky 2.4 je patrné, že při použití většího počtu prvků je kreslení bez hardwarové akcelerace téměř nepoužitelné a vývojář by měl zvolit `WebGL` technologii.

2.7 MPEG formát videa

Kompresí video i audio záznamů se mimo jiné zabývá expertní skupina `MPEG` (*Moving Picture Experts Group*). Prvním formátem multimediálních dat vydaným touto skupinou se stal dle standardu `ISO/IEC 11 172` formát `MPEG-1` [5].

MPEG-1

`MPEG-1` se skládá ze tří důležitých částí [5]:

- **MPEG-1 system** – specifikuje logickou strukturu formátu a dále také metody použité k uložení audia, videa a dalších dat do standardního binárního datového proudu.
- **MPEG-1 video** – definuje kódování videa.
- **MPEG-1 audio** – definuje kódování zvuku.

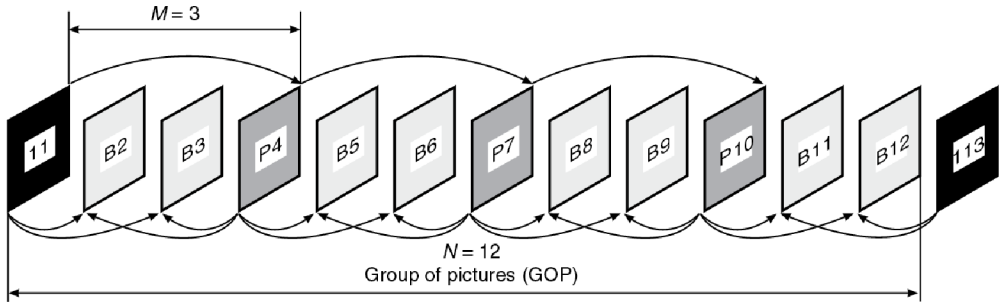
Ze zmíněných částí je z pohledu této práce nejdůležitější především způsob kódování videa, který vychází z `JPEG` formátu, avšak snaží se snížit redundanci dat ve snímcích videa, které jsou časově blízko u sebe.

Kodér na základě předchozích snímků technikou kompenzace pohybu predikuje následující snímek. Chyba predikce může být poté zakódována do výsledného datového proudu, a tím je na základě kvality predikce sníženo množství potřebných dat k zakódování videa.

`MPEG-1` video formát rozlišuje následující typy snímků [5]:

- **I-Frame** – Klíčový snímek zakódovaný pomocí `JPEG` formátu.
- **P-Frame** – Predikovaný snímek kódovaný na základě předchozího klíčového snímku s využitím techniky kompenzace pohybu.
- **B-Frame** – Snímek interpolovaný pomocí předchozího a následujícího `I` nebo `P` snímku.

Podle požadavků na výslednou kvalitu lze experimentovat s množstvím P/B-snímků mezi dvěma I-snímky. Ve výsledném datovém proudu se dokonce mohou nacházet pouze I-snímky nebo mohou zcela chybět B-snímky. U řady kodeků se mezi dvěma I-snímky nachází 3 P-snímky a celkem 8 B-snímků [5]. Schématicky je toto uspořádání snímku naznačeno na obrázku 2.12.



Obrázek 2.12: Příklad uspořádání jednotlivých typů snímků v `MPEG-1 video` formátu. Zdroj: [5]

Kompenzace pohybu

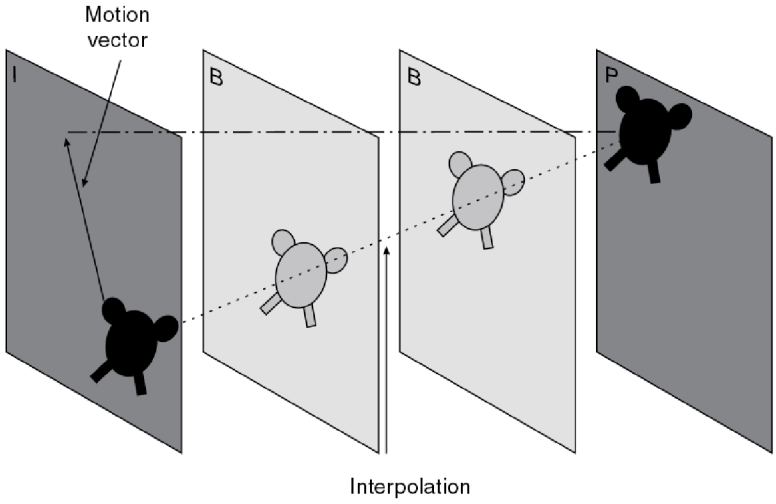
Kvalita odhadu predikovaných snímků má zásadní vliv na výslednou kvalitu videa, případně na jeho velikost. `MPEG` kódování rozdělí snímek na bloky pixelů o velikosti 16×16 pixelů a poté se snaží nalézt korelaci odpovídajících bloků v sousedních snímcích. Výsledkem je tzv. *Motion vector*, který udává právě směr pohybu jednotlivých bloků ve videu [5]. Ukázku odhadu pohybu lze vidět na obrázku 2.13.

MPEG-2

`MPEG-2` je rozšíření formátu `MPEG-1`, jehož hlavním účelem je umožnit přenášet video pro účely digitálního vysílání. Definuje čtyři různé úrovně rozlišení videa (od 360×288 až po `HDTV` rozlišení 1920×1152).

Dále také definuje řadu profilů videa dle účelu použití. Některé z nich jsou [5]:

- *Simple profile* – Snižuje nároky na kodér a dekodér videa za cenu zvýšeného objemu potřebných dat. V tomto profilu nejsou používány B-snímky.
- *Main profile* – Snaha o kompromis mezi náročností na dekódování a množství přenesených dat.
- *High profile* – Formát pro `HDTV` vysílání



Obrázek 2.13: Kompenzace pohybu ve formátu `MPEG`. Zdroj: [5]

Dále `MPEG-2` obsahuje některá další vylepšení, která umožňují snížit nároky na objem dat za cenu složitějšího zpracování videa. Za zmínku stojí například možnost zasílání prokládaných videí [5].

MPEG-4

S ohledem na `HDTV` vysílání definuje `MPEG-4` pokročilé kódování videa, které umožňuje snížit objem množství přenášených dat až o 50%. Úspora dat spočívá především v sofistikovanějším způsobu odhadu pohybu objektů ve scéně a dále také použitím *Integer* transformace 4×4 s proměnnými bloky od 16×16 do 4×4 pixelů místo diskrétní kosinové transformace [5].

2.8 FFmpeg a příbuzné knihovny pro práci s videem

FFmpeg je multiplatformní balík knihoven a nástrojů sloužících k záznamu a přenosu videa či audia. Dále také umožňuje nejrůznější operace nad videi včetně převodu mezi různými formáty, změny velikosti, změny míry komprese apod. Mezi podporované kodeky patří například `MPEG-4`, `H.264`, `AAC`, `FLAC` a mnoho dalších. Celý projekt je dostupný ve formě zdrojových kódů pod licencí *GNU Lesser General Public License (LGPL)* verze 2.1 [11].

FFmpeg knihovny

Jádrem celého nástroje jsou knihovny sloužící k jednotlivým operacím nad videi či zvukovými záznamy. Níže je uveden jejich stručný seznam [11]:

- **Libavutil** – Obsahuje užitečné nástroje a datové typy sdílené s ostatními knihovnami. Mezi ně patří například generátory náhodných čísel, podpůrné matematické funkce, kryptografické funkce a další.
- **Libswscale** – Obsahuje efektivní implementace změn formátu či velikosti obrazu.
- **Libswresample** – Knihovna určená především k převzorkování či změně formátu audio záznamů.

- **Libavcodec** – Jedna z nejvýznamnějších částí balíku *FFmpeg*. Umožňuje kódování či dekódování nejrůznějších audio či video formátů.
- **Libavformat** – Umožňuje práci s multimediálními kontejnery.
- **Libavdevice** – Implementuje komunikaci s nejrůznějšími vstupními či výstupními zařízeními. Mezi podporovanými zařízeními nechybí například *Video4Linux2*, *VfW*, *DShow* či *ALSA*.
- **Libavfilter** – Knihovna, která umožňuje nejrůznější filtrování audia nebo videa.

FFmpeg nástroje

Součástí *FFmpeg* balíku je i řada nástrojů, které mohou v řadech případů pomoci řešit některé jednodušší nebo často se opakující úlohy. Mezi ně patří například [11]:

- **ffmpeg** – Nástroj umožňující rychlé zpracování audia či videa. Ve své podstatě umožňuje využívat všechny důležité funkce jednotlivých *FFmpeg* knihoven. Mezi nejpoužívanější funkce tohoto nástroje patří konverze formátů audia nebo videa či modifikace nejrůznějších parametrů.
- **ffplay** – Jednoduchý přenositelný media přehrávač využívající *FFmpeg* knihovny a *SDL* knihovnu.
- **ffprobe** – Nástroj, který automaticky zjistí a vypíše informace o daném multimediálním souboru či toku.
- **ffserver** – Audio či video server pro přenos videa. Pomocí něj je možné například data načtená pomocí *ffmpeg* nástroje přeposílat po síti dalším klientům.

Všechny výše zmíněné nástroje fungují jako standardní programy příkazové řádky. Níže je uvedeno několik málo příkladů využití jednotlivých nástrojů:

- Změna formátu videa:
  ```
  ffmpeg -i input.mp4 output.avi
  ```
- Zjištění připojených zařízení umožňujících záznam videa:
  ```
  ffmpeg -y -f vfwcap -i list
  ```
- Záznam videa z kamery:
  ```
  ffmpeg -y -f vfwcap -r 25 -i 0 out.mp4
  ```
- Streamování videa:
  ```
  ffserver -f doc/ffserver.conf &
  ffmpeg -i input.mp4 http://localhost:8090/feed1.ffm
  ```

Kromě samotného *FFmpeg* balíku existují také různé projekty, které se snaží přenést funkcionalitu *FFmpegu* na systém Android. Níže jsou představeny některé z nich.

Aplikace FFmpeg 4 Android

FFmpeg 4 Android je aplikace, která umožňuje vykonávat příkazy nástroje *ffmpeg* prostřednictvím jednoduchého uživatelského rozhraní. Uživatelé si ji mohou stáhnout zdarma přes službu *Google play*. Jakákoliv jiná aplikace poté může s touto aplikací komunikovat prostřednictvím standardních systémových prostředků. Nevýhodou je, že se v tomto případě nejedná o knihovnu, nýbrž o aplikaci třetí strany, kterou cílový uživatel nemusí mít nainstalovanou. V rámci *Google play* obchodu existují také další podobné aplikace, jako například *ffmpeg codec*, *ffmpeg for android* nebo *ffmpeg cmd* [2].

Knihovna jjmpeg

Jjmpeg je knihovna napsaná v jazyce Java, která si dává za cíl být jakousi mezivrstvou mezi Java (android) aplikací a *FFmpeg* knihovnou. Mezi její hlavní nevýhody patří především velmi slabá dokumentace a také fakt, že je projekt od poloviny roku 2010 neudržovaný.

Android-ffmpeg-with-rtmp

Jedná se o kolekci skriptů, které umožňují zautomatizování přeložení *FFmpeg* knihovny pro Android s použitím Android NDK. Nakonfigurování celé *FFmpeg* knihovny je s použitím této sady skriptů relativně jednoduché:

```
$ git clone git@github.com:cine-io/android-ffmpeg-with-rtmp.git
$ cd android-ffmpeg-with-rtmp
$ ./build.sh
```

Výpis kódu 2.3: Konfigurace FFmpeg pro platformu Android

Podobné řešení poskytuje také knihovna *AndroidFFmpegLibrary*, která kromě skriptů k přeložení *FFmpeg* knihovny obsahuje také příklady užití a některé předpřipravené funkce pro snazší integraci *FFmpegu* do projektu.

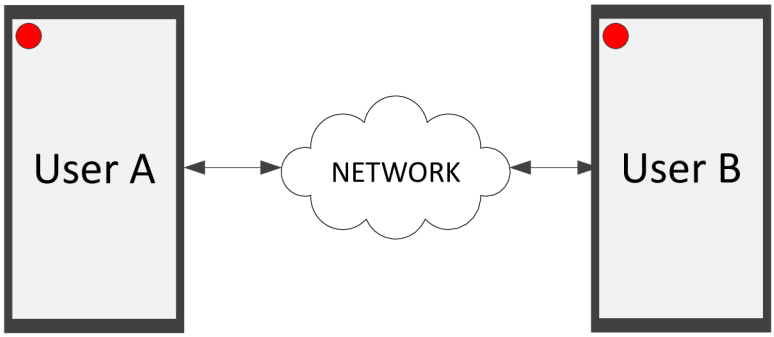
2.9 Mobilní aplikace přenosu obrazu

Přenos obrazu na dnešních telefonech je s narůstajícím výkonem běžných zařízení relativně zvládnutý problém. Pravděpodobně každý, kdo vlastní chytrý telefon, má alespoň nějaké zkušenosti s videohovory, případně s používáním nějaké aplikace třetí strany umožňující přenos obrazu z kamery fotoaparátu.

Skype

Skype je služba (od roku 2011 vlastněna firmou Microsoft), která umožňuje videohovory, zasílání instantních zpráv a dále obsahuje řadu placených služeb, pomocí kterých lze například zasílat SMS zprávy, telefonovat do tradičních telefonních sítí apod.

Komunikace přes *Skype* probíhá šifrovaně a decentralizovaně v rámci *peer-to-peer* architektury. K videohovoru je zapotřebí připojení zařízení o výkonu běžného smartphonu a připojení k internetu o rychlosti alespoň 128/128 kb/s.



Obrázek 2.14: Obecné schéma videohovoru.

Google Hangouts

Google Hangouts je konkurenční služba provozovaná firmou Google. Poskytuje velmi podobnou škálu funkcí, jako *Skype*. Kromě videohovorů až deseti uživatelů současně umožňuje také zasílání zpráv, nebo fotografií. Obsahuje také integraci do sociální sítě *Google+*.

K videohovoru je zapotřebí internetové připojení s minimální rychlostí 300/300 kb/s. Obě aplikace umožňují také hovory s větším počtem účastníků. V takovém případě je však přirozeně zapotřebí násobně rychlejšího internetového připojení.

IP Webcam

Nejrozšířenější kategorií v této oblasti jsou pravděpodobně aplikace nejrůznějších IP kamer. Jedna z možných použití je například situace, kdy uživatel nechá telefon s kamerou doma a pomocí druhého zařízení připojeného k internetu může sledovat prostor natáčený kamerou prvního Android zařízení. Mezi tyto aplikace patří například *IP Webcam*, *Home Security IP Cam - Alfred*, *EyeIPCam* a mnoho dalších.

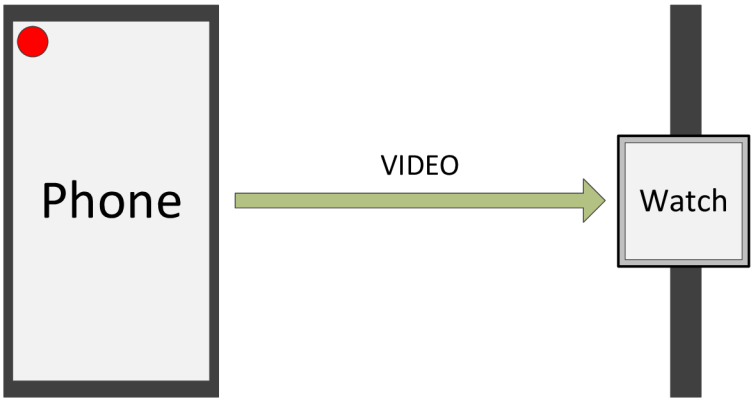
Libstreaming

Pro programátory toužící podobnou aplikaci vytvořit by mohla být vhodnou variantou knihovna *Libstreaming*, která je veřejně dostupná pod licencí *GPL*. V ní je obsažena implementace přenosu videa pomocí `RTP` protokolu. Mezi podporované kodeky této knihovny patří `H.264`, `H.263`, `AMR` a `AAC`. Ukázka knihovny je zahrnuta v rámci aplikace *Spydroid-ipcamera*, kterou je možné volně nainstalovat z *Google play* obchodu.

Přenos obrazu mezi telefonem a hodinkami

V obchodu s aplikacemi pro *Samsung Galaxy Gear* hodinky existuje několik aplikací, které řeší přenos obrazu z telefonu na hodinky. Mezi populární patří například aplikace *vCamera*, *vRecord* (od stejného autora jako vCamera), *BabySitting*, *Gear Viewfinder* nebo *Gear2Cam*.

Na obrázku 2.16 jsou ukázky vybraných aplikací. Všechny zmíněné aplikace nicméně z pohledu přenosu obrazu vypadají velice podobně, liší se tak především v rozšířené funkcionalitě a možnostech nastavení.



Obrázek 2.15: Schéma aplikací přenosu obrazu z telefonu na hodinky.

vCamera

Aplikace umožňuje přenos z fotoaparátu či ze přední kamery mobilu na obrazovku hodinek. Dále umožňuje jednoduché nastavení kamery smartphonu, jako například vypnutí či zapnutí blesku, nastavení krátké prodlevy či focení více snímků zároveň.

vRecord

Jedná se o velice podobnou aplikaci, jako v předchozím případě, je však doplněna o rozšiřující funkce, jako například o pořizování videí na vzdáleném zařízení, zaznamenávání zvuku, detekce pohybu, při kterém hodinky upozorní uživatele vibrací apod.

Gear2Cam

Svými funkcemi a rozsahem je *Gear2Cam* aplikace velice podobná aplikaci *vCamera*. Umožňuje vzdáleně vyfotit fotografii (i po krátké prodlevě), využít přední i zadní kameru fotoaparátu, či nastavit způsob využití blesku

BabySitting

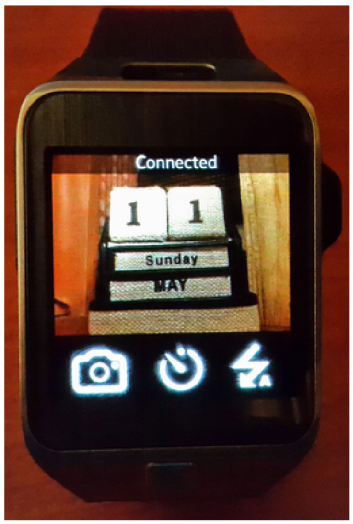
Aplikace *BabySitting* je určena pro rodiče malých dětí, aby mohli na dálku kontrolovat svého potomka. Primárním cílem aplikace je detekovat zvýšenou hladinu hluku v dané místnosti a prostřednictvím hodinek na to upozornit uživatele. Přenos obrazu se zdá být tedy sekundární funkcí této aplikace.

Gear ViewFinder

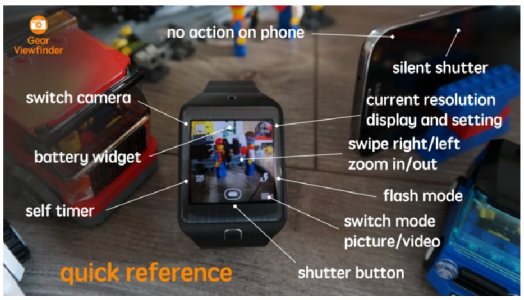
Jak už samotný název napovídá, jedná se o vzdálený hledáček mobilního telefonu. Umožňuje pořizování fotek a videí. Dále obsahuje podobně bohatou škálu nastavení, jako aplikace v předchozích případech.

(a) vCamera – úvodní obrazovka

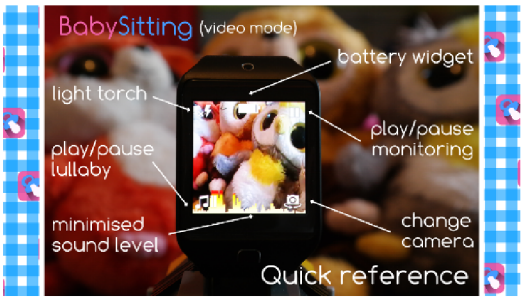
(b) vCamera – přenos obrazu



(c) Gear2cam



(d) Gear ViewFinder



(e) BabySitting

Obrázek 2.16: Ukázky existujících aplikací. Zdroj: `gearapp.challengepost.com`.

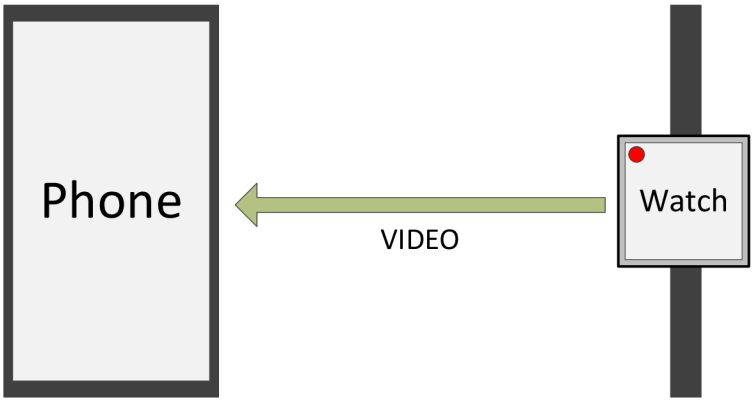
Přenos obrazu z hodinek na telefon

Některé typy chytrých hodinek mají vestavěnou kameru, tudíž lze provádět živý přenos obrazu i v opačném směru – tedy z hodinek na telefon. Pro koncového uživatele se jedná o velmi podobnou aplikaci jako v předchozích případech.

Naopak z pohledu vývojáře je však zapotřebí se v tomto případě vypořádat přesně s opačným problémem, kdy je k dispozici relativně výkonné zařízení (smartphone) k dekódování videa, zatímco kódování videa může být na méně výkonném hardware hodinek problém.

Gear2spy

Jednou z aplikací, která umožňuje živý přenos videa z kamery hodinek, je *Gear2spy*, jejímž autorem je Varun Chatterji. Aplikace umožňuje kromě přenosu videa také samotné video (včetně zvuku) nahrát a poté záznam přenést na telefon, kde může být zpětně přehráno.



Obrázek 2.17: Schéma aplikací přenosu obrazu z hodinek na telefon.

Spyman

Dalším příkladem je aplikace *Spyman*, která umožňuje sekvenční pořizování fotografií kamerou hodinek a jejich následné přenesení na displej telefonu. Aplikace kromě samotného přenášení obrazu umožňuje také nastavit rozlišení jednotlivých snímků.

Kapitola 3

Zhodnocení stavu a plán práce

V první části této kapitoly bude ohodnocena kvalita a použitelnost dosavadních aplikací realizujících přenos obrazu z telefonu na hodinky či z hodinek na telefon. Dále budou představeny některé knihovny, které mohou být použity v rámci vývoje výsledné aplikace a v závěrečné části kapitoly budou stanoveny cíle, kterých by mělo být v rámci této práce dosaženo.

3.1 Tradiční mobilní aplikace přenosu obrazu

Vzdáleným přenosem videa se zabývá celá řada různých typů mobilních aplikací. Video lze přenášen nejen mezi dvěma telefony, ale také mezi telefonem a webkamerou, stolním počítačem případně jiným síťovým zařízením.

Přenos mezi dvěma telefony

Co se týče aplikací přenosu videa mezi dvěma telefony, byly v kapitole 2.9 představeny některé známé aplikace umožňující videohovory i v rámci větších skupin účastníků. Mezi nejrozšířenější patří například aplikace *Skype*, *Google hangouts*, *ooVoo* nebo *viber*. Výkon dnešních smartphonů je pro tento typ aplikací dostatečný, a tudíž ke kvalitnímu videohovoru stačí uživateli dostatečně rychlé internetové připojení.

Ve srovnání s aplikací přenosu videa mezi telefonem a hodinkami se musí tyto aplikace vyrovnat s následujícími problémy:

- **Počet uživatelů** – Počty uživatelů jednotlivých služeb dosahují až několika stovek miliónů. Veškerá infrastruktura spojená s provozem dané služby musí tedy spolehlivě zvládat zpracovávat velké objemy dat.
- **Bezpečnost** – Videohovory jsou přenášeny přes internet. Jelikož se jedná o velice citlivá data, musí systém zajistit, aby se k nim jakýmkoliv způsobem nedostal případný útočník. Šifrování dat či jakýkoliv jiný způsob jejich zabezpečení však může vyžadovat zvýšené nároky na výpočetní výkon jednotlivých zúčastněných zařízení.
- **Propojení s telefonní sítí** – Některé programy umožňují propojení s běžnou telefonní síti. Například aplikace *Skype* umožňuje realizovat hovory s uživateli využívající běžný telefon, ale telefonovat lze také opačným způsobem, kdy je *Skype* uživateli přiděleno speciální číslo, na které lze volat z běžného telefonu.

- **Různá rozšíření** – Aplikaci samotnou mohou také komplikovat různé další požadavky uživatelů. Mimo videohovoru tak lze běžně zasílat instantní textové zprávy, soubory nebo sdílet obrazovku.

Aplikace IP kamer (Spydroid-ipcamera)

Řada mobilních aplikací umožňuje komunikaci s IP kamerami. Dále existuje také řada aplikací, které naopak umožňují proměnit libovolný telefon v IP kameru.

Jednou z takových aplikací je například *Spydroid-ipcamera*, která již byla zmíněna v podkapitole 2.9. Aplikace spustí v daném zařízení `HTTP server`, na který se lze připojit z běžného webového prohlížeče a sledovat obraz pořízený kamerou zařízení.

V případě, že je telefonu přidělena veřejná IP adresa, lze sledovat obraz kamery kdekoliv s připojením k internetu. V opačném případě přenos funguje pouze v rámci lokální počítačové sítě.

Aplikace podporuje kodeky `H.264` nebo `H.263`. Dále umožňuje přenos obrazu o maximálním rozlišení 1280×720 pixelů, rychlostí až dvacet snímku za sekundu.

Nástroje přenosu videa pro systém Android

Pro systém Android existují jak nativní řešení, která umožňují streaming videa, tak různé knihovny třetích stran.

Co se týče standardních prostředků, je k dispozici balíček tříd `android.media`, který obsahuje implementaci kodérů a dekodérů vybraných video formátů. Podporovány jsou běžné video či audio formáty. Použití standardních prostředků je poměrně nízkoúrovňové a s tím se pojí i jistá těžkopádnost jejich použití.

Libstreaming

To se do jisté míry snaží řešit například knihovna *libstreaming*, o které již byla řeč v podkapitole 2.9 v souvislosti s aplikací *Spydroid-ipcamera*, která knihovnu přímo využívá.

Knihovna je v současnosti udržovaná jejím autorem a umožňuje mimo jiné například streaming videa přes `RTP` protokol, dále podporuje `H.264` a `H.263` video formáty a ke kódování zvukového záznamu podporuje například `AAC` formát.

3.2 Přenos obrazu mezi telefonem a hodinkami

V případě moderních telefonů existuje řada aplikací, které umožňují poměrně kvalitní přenos obrazu. Přenos obrazu na hodinky je především z důvodu značně omezeného výkonu hodinek poněkud komplikovanější.

V podkapitole 2.9 byly představeny následující existující aplikace: *vCamera*, *vRecord*, *BabySitting*, *Gear Viewfinder* a *Gear2Cam*. Bohužel žádná ze zmíněných aplikací není otevřeným softwarem a zároveň u žádné z nich ani jejich autoři neuvádí, jakým způsobem je obraz přenášen. Nicméně lze alespoň subjektivně tyto aplikace porovnat a na základě chování se pokusit uhodnout, jakým způsobem je přenášení obrazu prováděno.

V tabulce 3.1 jsou porovnány klíčové vlastnosti jednotlivých aplikací z pohledu přenosu obrazu. Všechny zmíněné aplikace s výjimkou *BabySitting* a *ViewFinder* obsahují ovládací panel, který zabírá zhruba 20% plochy obrazovky, a tudíž k přenosu obrazu jim stačí menší datový tok (otázkou však je, zda toho autoři jejich aplikací využili).

Název aplikace	Kvalita obrazu	Rychlost přenosu	Zpoždění	Cena
Gear2Cam	Průměrná	Nadprůměrná	Malé	Zdarma
vRecord	Nadprůměrná	Velmi podprůměrná	Značné	Zdarma
vCamera	Průměrná	Nadprůměrná	Malé	Zdarma
BabySitting	Průměrná	Průměrná	Přijatelné	40 Kč
Gear ViewFinder	*	*	*	39 Kč

Tabulka 3.1: Subjektivní srovnání existujících implementací.

Aplikace *Gear ViewFinder* v tabulce není uvedena, protože obsahuje veliké množství nastavení parametrů přenosu obrazu (především rozlišení, míra komprese obraz, režim přenosu apod.), tudíž by bylo zapotřebí nejprve najít rozumný kompromis mezi vzájemně protichůdnými parametry aplikace a poté provést srovnání s ostatními implementacemi.

Subjektivně nejhůře bych hodnotil aplikaci *vRecord*, která má největší zpoždění přenosu obrazu a navíc zvládá přenést nejmenší počet snímků za časovou jednotku. Jednoznačně určit nejlepší aplikaci se zdá být velmi obtížné, obzvláště vezmeme-li v úvahu různou kvalitu přeneseného obrazu nebo dokonce procentuální využití plochy displeje hodinek.

Z chování jednotlivých aplikací lze usoudit, že nějakým způsobem jsou přenášeny jednotlivé snímky za sebou přes Bluetooth kanál. Přenos obrazu ve všech aplikacích je nevyhnutelně více či méně zpožděný a v žádné z uvedených aplikací není výsledné zobrazení snímků plynulé. Dále žádná z uvedených aplikací neimplementuje žádný mechanismus řízení kvality videa. Všechny mají přednastavenou určitou kvalitu obrazu (případně umožňují kvalitu obrazu nastavit ručně) a během přenosu obrazu je regulováno pouze množství přenesených snímků za vteřinu, což se negativně projevuje například v situaci, kdy je spojení obou zařízení rušeno vnějšími vlivy a počet přenesených snímků za sekundu tak začne klesat velmi blízko k nule.

3.3 Přenos obrazu z kamery hodinek na telefon

V případě opačného přenosu byly testovány programy *Gear2spy* a *Spyman*. Z pohledu přenosu obrazu vykazovalo chování obou aplikací mnohem horší výsledky, než aplikace přenášející obraz v opačném směru.

Spyman

Aplikace *Spyman* pouze posílá jeden vyfotografovaný snímek za druhým. Poté co je snímek vyfotografovaný, je uložen do permanentní paměti hodinek, odeslán na telefon a poté nejspíš smazán, tudíž se v žádném případě nejedná o efektivní přenos obrazu.

Gear2Spy

Gear2Spy dosahuje o něco lepších výsledků, avšak kvalita videa je ve srovnání s aplikacemi, které přenášejí obraz na displej hodinek, nesrovnatelně nižší co se týče množství přenesených snímků za čas nebo zpoždění videa.

Srovnání s ostatními aplikacemi

Ve srovnání s třídou aplikací, které přenáší obraz ze smartphonu na hodinky, je tedy výsledné video v obou případech podstatně nižší kvality. Důvodem může být například absence JavaScriptového API, které by umožňovalo nízkoúrovňový přístup k jednotlivým rámcům obrazu kamery, což nutí autory aplikací přenášet vyfotografované snímky. V takovém případě by se jednalo o systémové omezení, které by pravděpodobně nešlo jednoduše obejít. Druhou možností je samozřejmě také z pohledu přenosu videa nekvalitní návrh obou aplikací, což však vzhledem k účelu aplikací může být pro koncového uživatele akceptovatelné.

3.4 Použitelné knihovny

Výsledná aplikace by měla být robustní a jednoduše upravitelná. V ideálním případě by mělo být jednoduché změnit způsob kódování přenášených dat, parametry přeneseného obrazu (rozlišení, míra komprese) a to i s ohledem na využití zařízení (hodinek) s větším rozlišením displeje.

Pomoci docílit toho mohou různé volně dostupné existující knihovny. Na straně telefonu se nabízí možnost využít nástroj *FFmpeg* představený v kapitole 2.8, který umožňuje jak měnit parametry přenášeného videa, tak způsob kódování dat či míru komprese.

Situace je poněkud složitější na straně hodinek, neboť je zde možné využít pouze knihovny napsané v JavaScriptu. Tyto nástroje lze rozdělit na dvě skupiny – na klasické javascriptové knihovny a tradiční nástroje zkompilované speciálním překladačem do JavaScriptu.

Jsmpeg

Jsmpeg je javascriptová knihovna umožňující s některými omezeními dekódovat `MPEG-1` video formát. Knihovna umožňuje načítat video ze souboru nebo přes *WebSocket* rozhraní a následně jej zobrazit pomocí `WebGL` technologie.

Výhodou knihovny je její velikost (necelých 75 kB v nekomprimované verzi). Mezi její nevýhody patří především některá omezení:

- **Chybějící podpora B-snímků** – V rámci `MPEG` formátu se mohou vyskytovat snímky, které jsou kódované pomocí předchozího, ale i následujícího snímku (tzv. B-snímky). Knihovna je nepodporuje, avšak její autor uvádí, že většina kodeků stejně tyto snímky nepoužívá.
- **Šířka videa** – Musí být dělitelná dvěma.
- **Podpora jiných formátů** – Knihovna podporuje pouze `MPEG-1` formát, tudíž by v budoucnu bylo velmi obtížné například změnit použitý kodek.
- **Zastaralost formátu** – `MPEG-1` je poměrně starý formát a v současnosti existují modernější alternativy. Na druhou stranu se jedná o poměrně jednoduchý formát a k přenosu obrazu na hodinky může být dostačující.
- **Konstantní kvalita videa** – Základní verze `MPEG` formátu neumožňuje měnit kvalitu videa dle aktuálního stavu přenosového kanálu, což by mohlo způsobovat problémy při kolísání maximální přenosové rychlosti.

Broadway.js

Broadway.js je JavaScriptová knihovna, která umožňuje dekódování `H.264` video formátu. V tomto případě byla využita implementace dekódování `H.264` videa v rámci platformy Android, která byla pomocí nástroje *Emscripten* zkompilovaná do JavaScriptu a poté zoptimalizovaná pomocí překladače *Closure Compiler*.

Videoconverter.js

Další možností je využít *FFmpeg* knihovnu zkompilovanou do JavaScriptu například pomocí překladače *emscripten*. Příkladem takové knihovny je například nástroj *videoconverter.js*. Hlavní jeho výhodou je možnost libovolně měnit parametry přenášeného videa. Nevýhodou je pak především velikost výsledné knihovny (27.5 MB v plné verzi podporující všechny kodeky, která však může být zkomprimovaná na 6.9 MB).

3.5 Specifikace požadavků na přenos

V podkapitole 3.4 byly shrnuty možné technologie pro přenos a následné dekódování obrazu (resp. videa). Kvalitu přenosu lze posuzovat podle několika různých (vzájemně protichůdných) parametrů:

- **Zpoždění** – Za zpoždění je považován čas od vystavení snímku ze zdroje videa (například kamery telefonu) po jeho zobrazení na displeji hodinek. Dílčí zpoždění vzniká v několika částech aplikace: při kódování obrazu, jeho následném dekódování, ale především jeho přenosu přes Bluetooth kanál. Zpoždění obrazu u existujících aplikací dosahuje několika stovek milisekund.
- **Kvalita obrazu** – Dalším parametrem přeneseného obrazu je jeho kvalita (do ní lze zahrnout rozlišení obrazu, případně míru jeho komprese). Je zřejmé, že vyšší kvalita přeneseného obrazu bude nepříznivě ovlivňovat zbývající parametry.
- **Přenosová rychlost** – Posledním parametrem přenášeného videa je počet přenesených snímků za sekundu. Pro přenos plynulého obrazu je zapotřebí přiblížit se snímkovací frekvenci oka, která je zhruba 30 Hz. Ve filmovém průmyslu se poměrně dlouho používala snímkovací frekvence 24 Hz.

V rámci implementace výsledných aplikací bude zapotřebí nejprve vybrat nejvhodnější metodu ke kódování snímků a dále také nejvhodnější způsob k vykreslení přeneseného snímku. V neposlední řadě bude také zapotřebí k efektivnímu přenosu zjistit, jaké množství dat je možné přenést a zpracovat v rámci dané architektury.

Je zřejmé, že teoretické maximální množství přenesených dat může v rámci běhu aplikace značně kolísat. Možné příčiny takových změn můžou být například:

- **Vnější změny prostředí** – Během přenosu se může měnit vzdálenost telefonu od hodinek. Dále se také může měnit prostředí samotné, například tím, že se mezi obě zařízení dostanou překážky, které mohou rušit spojení. V takovém případě se zmenší přenosová rychlost spojení, což může negativně ovlivnit kvalitu videa.
- **Vnitřní změny výkonu** – K různým změnám může docházet i na straně serveru, případně klienta. Na straně hodinek hrozí postupné zahřívání zařízení vlivem hardwarově

náročného zpracování datového toku. Po překročení určité meze systém automaticky omezí výkon aplikace. Tomu se také musí přizpůsobit přenos videa.

K podobným jevům může docházet i v telefonu. Výkon telefonu je sice obecně řádově vyšší, přenos videa je však přenášen na pozadí a paralelně s ním může v telefonu docházet k prioritnějším událostem, které mohou způsobit zpomalení přenášení jednotlivých snímků.

Z těchto důvodů bude zapotřebí kontrolovat a dynamicky měnit kvalitu přeneseného obrazu tak, aby byl co nejefektivněji využit přenosový kanál mezi oběma zařízeními a nedocházelo tedy ať už k jeho přetížení či zbytečnému nevyužívání volného přenosového pásma.

Parametry výsledné aplikace

Na straně telefonu se jeví výhodné využít *FFmpeg* nástroj ke kódování jednotlivých snímků pořízených ze zdroje videa. Na straně klienta bude zapotřebí vhodným způsobem jednotlivé snímky dekódovat a zobrazit. Přenos obrazu by měl dosahovat co nejblíže rychlosti 24 snímků za vteřinu při zachování „rozumné“ kvality obrazu.

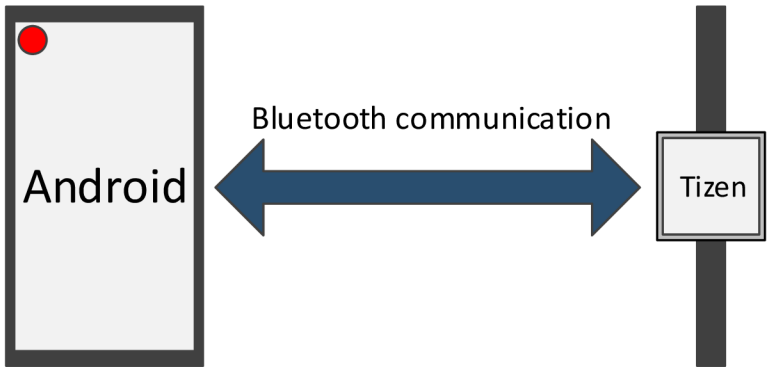
Velmi důležitým parametrem je také zpoždění obrazu. Smyslem aplikace je živý přenos videa, tudíž by zpoždění obrazu mělo být minimalizováno na nejmenší možnou hodnotu. K většímu zpoždění může docházet například v případě, kdy jedna strana posílá data rychleji, než je druhá strana schopna zpracovávat. Data jsou pak zapisována do vyrovnávacích pamětí na obou stranách a zpracována s výrazným zpožděním, které při špatném návrhu aplikace může dosahovat i několika sekund.

Kapitola 4

Aplikace přenosu obrazu

Následující kapitola se zabývá vlastní implementací aplikace typu klient-server, která realizuje přenos obrazu z telefonu na hodinky. Schéma architektury je vidět na obrázku 4.1. V první části kapitoly bude ukázáno několik experimentů, na základě nichž byly poté navrženy vhodné metody k přenášení a vykreslování jednotlivých snímků videa.

První skupina experimentů je díky struktuře Tizen aplikace prováděna mezi dvěma stolními počítači a má za úkol zjistit technologické limity dnešních webových frameworků. Další experimenty, jejichž účelem je pro změnu zjistit hardwarové limity cílového zařízení jsou prováděny přímo na hodinkách.



Obrázek 4.1: Schéma přenosu obrazu.

Na základě výsledků jednotlivých experimentů je poté v dalších kapitolách popsán návrh aplikací a společného komunikačního protokolu. Dále bude také diskutováno, jakým způsobem lze nejvhodněji modifikovat vzájemně protichůdné parametry kvality videa v případě, že je detekováno zahlcení přenosového kanálu. Výsledná implementace přenosu obrazu bude na závěr kapitoly otestována a zhodnocena.

4.1 Experimenty přenosu obrazu v rámci webových aplikací

Možné způsoby přenosu obrazu z mobilu na hodinky mohou nastínit některá existující řešení umožňující v rámci webové aplikace zobrazení videa z kamery v běžném interneto-

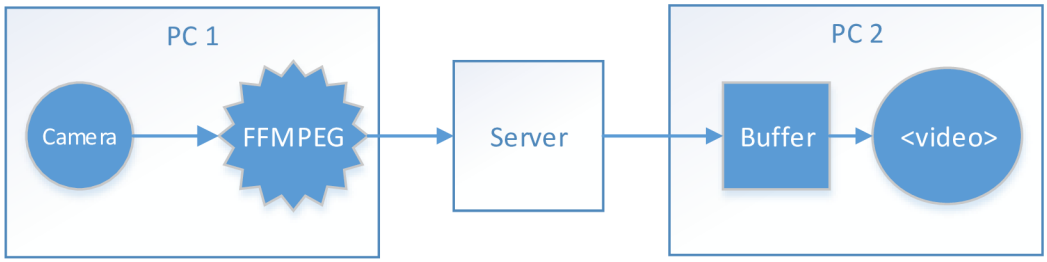
vém prohlížeči. Vzhledem k tomu, že aplikace na straně hodinek běží ve webovém jádru vycházejícím z webového jádra *Webkit*, lze následující experimenty až na některá specifika hodinek provádět přímo v běžném webovém prohlížeči.

Cílem těchto experimentů je určit možné způsoby zasílání a dekódování videa v rámci moderních webových prohlížečů. Tradiční webové aplikace zpravidla využívají nejrůznějších protokolů pro přenos videa (např. `RTP` protokol) nebo umožňují zobrazit video přenesené přes `HTTP` protokol. Streaming videa přes `HTTP` protokol umožňuje například `MPEG-DASH` protokol.

Cílová aplikace však využívá k přenosu dat Bluetooth protokol, kde tradiční síťové infrastruktury nelze využít. Je proto zapotřebí hledat jiné způsoby přenosu a zobrazení videa.

HTML Video prvek

První experiment, jehož schéma je vidět na obrázku 4.2, se týká přenosu videa a jeho následného vykreslení pomocí <video> prvku.



Obrázek 4.2: Schéma experimentu s `<video>` prvkem.

Obraz pořízený kamerou připojenou k prvnímu počítači je zpracován pomocí nástroje *FFMpeg* a následně odeslán na server, který data přeposílá připojeným klientům. Na druhém počítači jsou data ze serveru přijata a uložena do sdílené paměti, odkud je výsledné video načítané webovým jádrem a zobrazeno.

Zásadním problémem je v tomto případě chybějící podpora `HTML5` standardu pro zápis videa a jeho následné načtení ze sdílené paměti[1]. V některých ohledech omezeném webovém jádře na hodinkách podpora chybí [3]. Uvedený způsob přenášení videa je tedy pro účely cílové aplikace nepoužitelný.

Jsmpeg

Další možností je dekódovat přenesené video přímo pomocí interpretu JavaScriptu. V kapitole 3.4 byla představena knihovna *jsmpeg*, která umožňuje dekódování obrazu `MPEG-1` videa. Funkčnost a výkonnost knihovny se dá ověřit pomocí skriptu uvedeného v příloze v ukázce kódu B.1.

[1] Ke dni 31.3.2015 je zveřejněný návrh na potřebné rozšíření specifikace, které již některé moderní webové prohlížeče podporují – viz `http://www.w3.org/TR/media-source/`

Schéma experimentu zůstává stejné, jako v předešlém případě. Opět jsou data pořízena kamerou (zařízení *USB Camera*), zakódována pomocí *FFMpegu* a následně odeslána na klienta, kde jsou však zpracována pomocí knihovny *jsmpeg*.

V rámci takto nakonfigurované sítě není problém díky výkonu stolního počítače vysílat obraz o velikosti 640×480 pixelů rychlostí 30 snímků za sekundu.

Přenos klíčových snímků H.264 formátu

O možnosti přenosu klíčových snímků `H.264` formátu byla již řeč v kapitole 3.4, kde byl popsán způsob vytvoření datového toku obsahující pouze klíčové snímky.

Způsob přenášení je naznačen v příloze v ukázce kódu B.2. Ta předpokládá, že na stranu klienta jsou zasílány jednotlivé klíčové snímky v `base64` kódování. V reálné aplikaci by bylo kódování snímků prováděno již na straně serveru (telefonu).

Zobrazení snímku je ponecháno čistě na webovém frameworku. Dá se předpokládat, že dekódování snímku a následné zobrazení (včetně uzpůsobení velikosti snímku rozměrům obrazovky) bude probíhat hardwarově akcelerovaně. Případně se dá podobně jako v předchozích případech využít k vykreslení snímku `WebGL` prostředí (v takovém případě by bylo vhodnější ponechat snímky nezakódované).

Zasílání jednotlivých snímků

Předchozí metody se snažily na klienta přenést video zakódované v nějakém hojně používaném video formátu. Zásadní nevýhodou takového přístupu je velmi obtížná regulovatelnost datového toku. Například v rámci `MPEG` formátu nelze standardním způsobem zasílat video s libovolně malým množstvím snímků za vteřinu. Dle použitého standardu (`PAL` či `NTSC`) nelze využít frekvenci snímkování nižší než 25 Hz respektive 29.97 Hz.

Naopak zasílání jednotlivých snímků takovými problémy netrpí. K jejich zobrazení lze využít například stejné metody jako v předchozím případě (viz ukázka kódu B.2). Podrobněji budou způsoby zobrazení jednotlivých přenesených snímků popsány v podkapitole 4.3.

4.2 Omezení Tizen aplikací

Jak již bylo řečeno v kapitole 2.5, aplikace na straně hodinek se velice podobá klasické `HTML5` aplikaci interpretované webovým prohlížečem. Prvním problematickým místem přenosu obrazu tedy může být výkon samotného interpretu JavaScriptového kódu umocněný velice omezeným výkonem samotných hodinek. Naopak výkon smartphonu je oproti hodinkám poměrně vysoký, tudíž bude vhodné snažit se provádět náročné výpočty spíše na straně telefonu.

Další omezení celkové přenosové rychlosti může přinášet samotný Bluetooth kanál mezi hodinkami a telefonem. Aplikace by si měla uspokojivě poradit i v případech, kdy je např. mezi oběma zařízeními větší vzdálenost a dochází k rušení přenosového signálu.

Softwarová omezení

Mimo omezení spojených s nevýkonným hardwarem hodinek jsou aplikace mnohdy limitované dostupným programátorským rozhraním určeným ke komunikaci webového jádra s unixovým jádrem. Programátor se tak nemůže dostat přímo k nízkoúrovňovým systémovým komponentám, na druhou stranu je mu však k běžným úlohám poskytnuto vysokoúrovňové rozhraní, které se k jednodušším úlohám využívá mnohem snadněji.

Dalším zdroje různých softwarových omezení může být také vlastní webové jádro, které nemusí podporovat některé nestandardní způsoby využití jednotlivých vlastností, podobně jako tomu bylo v předchozí podkapitole v případě využití <video> prvku.

Omezení daná výrobcem hodinek

Dalším limitujícím omezením pro přenos obrazu je nemožnost zasílat obecná binární data přes Bluetooth kanál. Rozhraní dodávané výrobcem hodinek tak umožňuje zasílat pouze textová data nebo celé soubory s libovolným obsahem.

Přitom prakticky všechny formáty videa, se kterými pracují nástroje zmíněné v kapitole 3.4, pracují převážně s binárně kódovaným multimediálním obsahem.

Z pohledu cílové aplikace je tedy důležité zvážit práci s jednotlivými soubory na úrovni jednotlivých snímků. V případě zasílání textových zpráv je zapotřebí vhodně kódovat binární data videa. Zde se naskýtá možnost využití například `Base64` kódování.

4.3 Způsoby vykreslení snímků na hodinkách

V podkapitole 4.1 bylo vyloučeno použití <video> prvku v klientské části aplikace. Spolu s omezeními popsanými v podkapitole 4.2 bylo ukázáno, že je zapotřebí na klienta zasílat video po jednotlivých snímcích, které je zapotřebí po přijetí dekódovat, zpracovat a zobrazit.

V následujícím textu budou popsány jednotlivé metody dekódování a následné zobrazení snímku. Efektivnější z pohledu hardwarových prostředků budou zřejmě metody, u kterých je dekódování a zobrazení snímku prováděno přímo webovým jádrem. Dá se předpokládat, že v takovém případě bude zpracování snímku probíhat hardwarově akcelerovaně a tedy i rychleji. Zobrazení přeneseného obrazu lze realizovat následujícími způsoby:

- **Přímé vykreslení** – Jestliže by byl výkon hodinek dostačující, stačilo by jednoduše přenesený obraz pixel po pixelu vykreslit na obrazovku a celá aplikace by tím pádem byla poměrně jednoduchá. V dalším odstavci však bude vysvětleno, proč je tento přístup spíše naivní.
- **Využití <img> elementu** – Zdroj obrázku lze nastavit jako `Base64` kódovaný `JPEG` obrázek. Druhou možností je prohlížeči zadat cestu k souboru obsahující obrázek, který má být vykreslen.
- **Akcelerované vykreslení pomocí WebGL** – Poslední možností je dekódovat přenesené rámce obrazu z telefonu a ty poté vykreslit ve `WebGL`.
- **Pomocí <video> elementu** – Pokud by potřebné vlastnosti byly podporované webovým jádrem, jednalo by se pravděpodobně o nejlepší způsob vykreslení videa, ať už z pohledu množství přenesených dat či nároků na samotné vykreslení videa. Možnost využití <video> prvku tak zůstává otázkou pro budoucí verze systému.

Přímé kreslení po pixelech

Implementačně nejjednodušším způsobem, jak vykreslit přenesený obraz, je přímé zobrazení přenesených pixelů na <canvas> prvek. V ukázce kódu 4.1 je jednoduchá funkce, která celý <canvas> prvek velmi neefektivním způsobem vyplní šedou barvou.

Po proběhnutí cyklu 1000 volání této funkce se ukazuje, že vykreslení obrazu tímto způsobem trvá zhruba 34 ms, což by odpovídalo zhruba 29 snímkům za sekundu, aniž by byl jakýmkoliv způsobem zpracován datový proud přenesený z telefonu.

Samotné vykreslení existujícího snímku by přitom mělo být podstatně rychlejší, aby ve zbývajícím čase bylo možné zpracovávat data přenesená prostřednictvím Bluetooth kanálu. Zároveň se tak ukazuje, že výkon klienta není dostatečný k provádění jakýchkoliv operací nad jednotlivými pixely obrazu.

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
...
var canvasData = ctx.getImageData(0, 0, width, height);

function draw() {
    for (var i = 0; i < width * height; i++) {
        var r, g, b = 128;
        // red
        canvasData.data[i * 4 + 0] = r;
        ...
    }
    ctx.putImageData(canvasData, 0, 0);
}
```

Výpis kódu 4.1: Naivní zobrazení přeneseného obrazu na <canvas> prvku.

Vykreslení pomocí <video> prvku

Zpracování videa je na hodinkách hardwarově akcelerované a výsledek se nijak neliší od videa spuštěného na běžném počítači (tedy až na velikost obrazovky). Na stránkách `developer.tizen.org` je ke stažení ukázková aplikace *Camera*, která k zobrazení dat z kamery na hodinkách využívá právě <video> prvek.

Jak již však bylo řečeno v podkapitole 4.1, v současné době nelze <video> prvek bez dalšího rozšíření webového jádra oproti `HTML5` standardu použít.

Použití <img> prvku (`Base64/JPEG`)

Rychlost vykreslování pomocí <img> prvku lze jednoduše ověřit například střídavým zobrazováním dvou `Base64` kódovaných obrázků, podobně jako je tomu v ukázce kódu 4.2.

Rychlost kreslení tímto způsobem odpovídá téměř 80 snímkům za sekundu. Aby byla vyloučena možnost, že příliš rychlé vykreslování je způsobeno nějakou vyrovnávací pamětí uvnitř vykreslovacího jádra prohlížeče, byl vyzkoušen i test, kde byly oba obrázky náhodně poškozovány prostým nahrazením krátké sekvence znaků uvnitř `Base64` kódovaného řetězce.

V tomto případě klesla vykreslovací rychlost na hodnotu přibližně 40 snímků za sekundu. Výrazný pokles výkonu však přisuzuji spíše práci s velmi dlouhými textovými řetězci než zvýšení náročnosti kreslení. Hodnoty textových řetězců jsou totiž v JavaScriptu neměnné (`string` je tzv. *immutable* datový typ), a proto nahrazení podřetězce je zapotřebí provést vytvořením kompletně nového řetězce, do kterého jsou příslušné znaky překopírovány s lineární složitostí.

```
...
var img1 = "/9j/4AAQSkZJRgA ... b34rxyz ... F2/0R/9k=";
var img2 = "/9j/4AAQSkZJRgA ... RGVz3Ph ... j9O/6s//Z";

var i = 0;

function renderImage() {
    var src = "" + (i % 2 == 0 ? img1 : img2);
    $("#img").attr("src", src);
    i++;
}
```

Výpis kódu 4.2: Vykreslení obrazu pomocí <img> elementu.

Použití <img> prvku (práce se soubory)

Vykreslení obrázku uloženého v souboru má oproti `Base64/JPEG` kódovanému obrázku tu výhodu, že je díky binárnímu formátu nutné přenášet méně dat přes Bluetooth kanál. Na druhou stranu je zapotřebí obrázek načítat z permanentní paměti, což může být pomalejší oproti obrázku načítaného z dynamické paměti. V případě `Base64/JPEG` kódovaného obrázku je ale na druhou stranu navíc zapotřebí dekódovat `Base64` řetězec.

Otestovat rychlost kreslení tímto způsobem lze například změřením doby potřebné pro vykreslení velkého množství předpřipravených obrázků.

Je však zapotřebí vypořádat se s následujícími problémy:

- Webový prohlížeč může obrázky načítat asynchronně. Načítání původního obrázku může poté zrušit v momentě, kdy je zjištěno, že se zdroj změnil. V extrémním případě tak může nastat situace, kdy je načten kompletně pouze poslední obrázek testovací sady.

  Je tedy zapotřebí detekovat, že byl obrázek korektně načten a zobrazen, a teprve poté načíst další.

- Je nutné vyloučit vliv vyrovnávací pamětí, do které si prohlížeč ukládá obrázek typicky pro pozdější znovupoužití. V případě přenosu videa by takové chování bylo nežádoucí, neboť by zbytečně zatěžovalo cílové zařízení.

- Test může být i tak zkreslen dalšími vyrovnávacími paměťmi (například cache pamětí pevného disku).

Test rychlosti vykreslování ze souborů je uveden v ukázce kódu 4.3. Jakmile je obrázek načten (událost `Image.onload`), tak lze začít načítat další.

Výsledky experimentu naznačují, že vykreslení obrázku ze souboru je na cílovém zařízení o něco rychlejší oproti vykreslení `Base64/JPEG` kódovanému snímku. Rychlost kreslení odpovídá v tomto případě zhruba 97 snímkům za vteřinu.

Práce s velkým množstvím souborů však přináší řadu dalších problémů, díky kterým je nakonec takový způsob přenosu videa v současné situaci nepoužitelný. Podrobněji bude

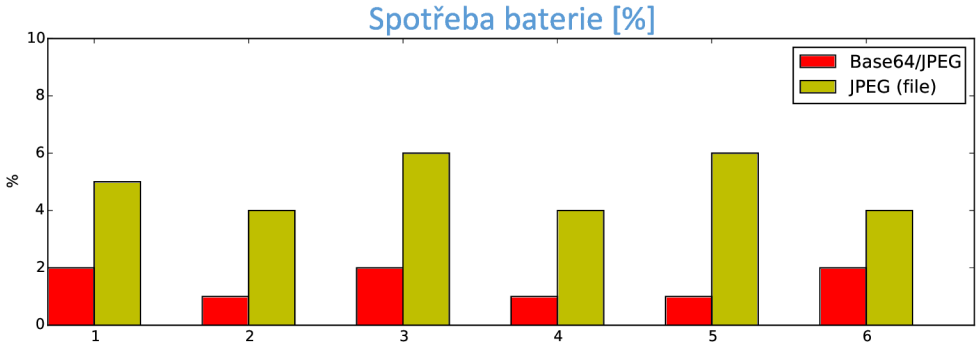
```
<meta http-equiv="cache-control" content="no-cache">

...
var img = 1;
document.getElementById('img').onload = function() {
    if (img == 100) {
        return;
    }
    var imgSource = 'img/Test' + img + '.jpg';
    document.getElementById('img').src = imgSource;
    img++;
};
...
```

Výpis kódu 4.3: Vykreslení obrazu pomocí <img> elementu.

problém se soubory diskutován v podkapitole 4.9. Dále je také při tomto způsobu vykreslení zvýšená náročnost na výpočetní zdroje (respektive na baterii hodinek) oproti obrázku načteného z `Base64/JPEG` kódovaného řetězce uloženého v operační paměti.

Na obrázku 4.3 je výsledek šesti různých experimentů, v rámci kterých byl výše zmíněným způsobem vykreslen obrázek ve 40 000 iteracích. Na první pohled je vidět více než dvojnásobná náročnost na výpočetní zdroje u vykreslení obrázků načtených ze souboru, přestože vykreslovací smyčka je v tomto případě ukončena pouze v nepatrně kratším čase.



Obrázek 4.3: Spotřeba baterie u dvou nejrychlejších metod vykreslení obrázku.

Využití WebGL

V rámci testování rychlosti vykreslení obrazu pomocí `WebGL` byla vytvořena jednoduchá testovací aplikace (viz ukázka kódu v příloze B.3), ve které byl načten obrázek do textury a pomocí fragment shaderu vykreslen. Podobně jako v předchozím případě byl otestován běh několika překreslení stránky za sebou.

Tento způsob kreslení jednotlivých snímků odpovídá zhruba 44 snímkům za sekundu na cílovém zařízení. Pro srovnání výsledek stejného testu provedeného na průměrném stolním počítači by odpovídal zhruba 240 vykreslených snímků za sekundu.

Kromě vyšší rychlosti vykreslení obrazovky umožňuje `WebGL` například velmi jednoduše

měnit velikost obrazu. Z telefonu by tak bylo možné s využitím stejného kódu posílat obraz s nižším rozlišením, který by byl prakticky za stejný čas roztažený přes celou obrazovku a vykreslen.

Srovnání kreslících metod

Jednotlivé metody jsou přehledně srovnané v tabulce 4.1. Z měření vyplývá, že nejrychleji je obrázek vykreslen v případě, kdy je přímo načten ze souboru nebo z `Base64/JPEG` kódovaného textového řetězce a posléze zpracován webovým jádrem.

	FPS	Poznámky
Přímé kreslení	29	Naivní, v praxi nepoužitelná metoda
Video prvek	-	Vido prvek dle předchozích experimentů nelze využít.
Img prvek I	79	Velmi jednoduchá a na vykreslování efektivní metoda. `Base64` kódování však zvýší nároky na datový tok
Img prvek II	97	V případě zasílání jednotlivých `JPEG` souborů se oproti `Base64` kódování sníží datový tok. Práce s velkým množstvím souborů je však problematická z pohledu přenosu dat přes Bluetooth kanál. Dále je také velmi náročná na výpočetní zdroje – viz měření na obrázku 4.3.
WebGL	44	`WebGL` kreslení by bylo vhodné spojit například s dekódováním nějakého video kodeku přímo na straně hodinek.

Tabulka 4.1: Srovnání jednotlivých metod kreslení.

Z omezení systému Tizen popsaných v kapitole 4.2 vyplývá, že (pravděpodobně optimální) vykreslování přeneseného videa pomocí <video> prvku nelze využít.

Nezbývá tedy jiná možnost, než zasílat jednotlivé snímky videa jako `JPEG` komprimované obrázky a na straně hodinek je některým z výše uvedených způsobů vykreslovat. V úvahu připadají možnosti zasílání celých souborů s jednotlivými snímky, případně zasílání `Base64/JPEG` kódovaných snímků.

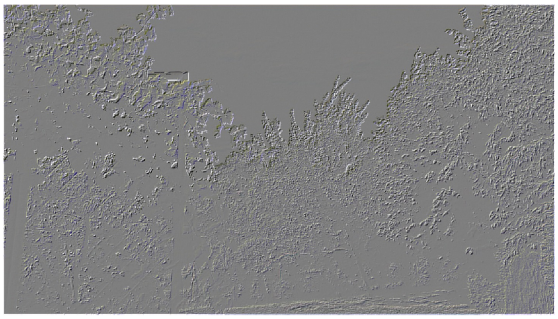
V obou případech lze takto přenesený snímek přímo vykreslit pomocí <img> prvku nebo jej lze načíst do `WebGL` textury a poté akcelerovaně zpracovat. Tento způsob přenosu videa je také mnohem vhodnější z pohledu řízení datového toku přenášeného přes Bluetooth kanál, o kterém bude řeč v podkapitole 4.6.

4.4 Vyhodnocení datového toku

V rámci zasílání jednotlivých snímků videa se lze inspirovat u používaných video formátů a pokusit se alespoň částečně zmenšit časovou redundanci videa, kdy mezi časově blízkými snímky jsou jen velmi malé rozdíly.

Z toho důvode lze (podobně jako v případě `MPEG-1` video formátu) zasílat některé snímky kódované jako rozdíl oproti předchozímu klíčovému snímku. V následujícím textu je za klíčový snímek označen normální snímek získaný ze zdroje videa (kamery), zatímco rozdílový snímek je možné spočítat následujícím způsobem:

(a) Fotografie (9.14 MB).



(b) Rozdílový snímek (935 KB).

Obrázek 4.4: Ukázka rozdílu dvou fotografií pořízených krátce po sobě.

$$d = \frac{f - k}{2} + 127 \tag{4.1}$$

Kde d je výsledný rozdílový snímek, k je poslední klíčový snímek a f je aktuální kódovaný snímek. Dekódování probíhá analogicky:

$$f = 2 \cdot (d - 127) + k \tag{4.2}$$

Rozdílový snímek obsahuje menší množství informací, a tudíž se dá předpokládat, že jeho velikost bude menší než velikost klíčového snímku, což by mohlo způsobit rychlejší přenos videa. Na druhou stranu výpočet rozdílových snímků klade další hardwarové nároky na obě spojená zařízení. Na straně hodinek však lze využít `WebGL` a snímky dekódovat akcelerovaně pomocí jednoduchého *Fragment Shaderu*. Na straně telefonu lze rozdílové snímky počítat akcelerovaně například pomocí *RenderScriptu*, který umožňuje výpočet přesunout na grafický čip telefonu.

Ukázka vypočteného rozdílového snímku je na obrázku 4.4. V tomto případě klesla velikost snímku zhruba na desetinu. V reálné aplikaci by však pravděpodobně byly jednotlivé snímky komprimované podstatně více než fotografie, a tudíž by pokles datového toku nebyl tolik znatelný.

Skupina snímků (GOP)

Podobně jako je tomu v případě `MPEG` formátu, lze i v tomto případě hovořit o snímcích mezi dvěma klíčovými snímky jako o skupině snímků (*Group of pictures – GOP*). V rámci skupiny snímků jsou však všechny rozdílové snímky na rozdíl od `MPEG` formátu kódované pouze vůči poslednímu klíčovému snímku.

Tím se sice zhorší kvalita pozdějších rozdílových snímků, na druhou stranu se však o něco sníží hardwarová náročnost klientské aplikace (konkrétně o jedno kopírování mezivýsledného snímku při výpočtu každého rozdílového snímku). Schématicky je tento rozdíl naznačen na obrázku 4.5.

Velikost skupiny snímků má poměrně výrazný vliv na celkovou velikost videa. Pokud je skupina malá, sestává se výsledné video z velkého množství klíčových snímků. U velké skupiny naopak začíná docházet k velkým rozdílům mezi klíčovým a aktuálním snímkem a v extrémním případě může velikost rozdílového snímku dosahovat velikosti klíčového snímku.

(a) MPEG (bez b-snímků). (b) Zjednodušená varianta.

Obrázek 4.5: Zjednodušené kódování rozdílových snímků.

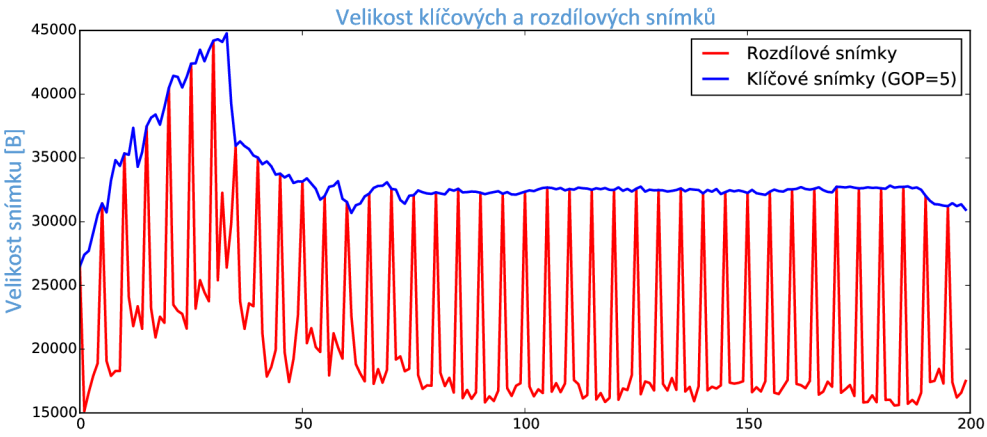
Pro stanovení optimální velikosti skupiny snímků je možné se například inspirovat u existujících `MPEG` kodérů videa, kde se velikost skupiny pohybuje okolo 12 snímků. V případě přenosu videa z telefonu na hodinky je však možné, že počet přenesených snímků za sekundu nebude dosahovat hodnot běžného videa. V takovém případě by bylo nutné velikost skupiny adekvátně upravit.

Zhodnocení metody

Experiment vyhodnocující účinnost této metody lze nalézt v příloze této práce. V rámci něj byl napsán skript v jazyce python s využitím *OpenCV* knihovny, který zpracovával výstup z webkamery s rozlišením 640 × 480 pixelů a vypočítával jednotlivé rozdílové snímky dle rovnice 4.1.

Výsledek experimentu lze vidět na grafu na obrázku 4.6, kde v rámci stejného měření byly porovnávány velikosti snímků zasílaných oběma způsoby. Úspora dat potřebných k přenesení videa se v závislosti na daných podmínkách a konkrétním nastavením velikosti skupiny snímků pohybuje okolo 40%.

Dalšího snížení datového toku by bylo možné dosáhnout sofistikovanějšími způsoby běžně používaných v moderních video kodecích, kde je na základě minulých snímků různě složitě predikován následující snímek, a klientovi je zaslána pouze chyba predikce. Podrobněji byly jednotlivé způsoby kódování `MPEG` videa popsány v kapitole 2.7. Příliš složité kódování videa je však s ohledem na výkon hodinek nepoužitelné. Spotřeba baterie při použití složitějšího kódování videa (které ale umožňuje přenést menší množství dat) bude diskutována v podkapitole 4.9.



Obrázek 4.6: Rozdíl ve velikostech klíčových a rozdílových snímků.

4.5 Návrh aplikací a komunikačního protokolu

Systém tvoří architekturu klient-server, v rámci které jsou jednotlivé snímky přenášené ze serveru na klienta. Datový přenos začíná připojením klienta k serveru spolu se zasláním inicializační zprávy, ve které klient může o sobě sdělit různé užitečné informace, například rozlišení displeje.

Průběžně je zapotřebí také kontrolovat zpoždění přenosu a v případě, že je detekován nějaký problémový stav, je třeba snížit datový tok. Naopak pokud k žádným problémům nedochází, je možné průběžně zvyšovat kvalitu přenášeného videa. Podobných principů se využívá například i v řízení přenosové rychlosti v `TCP` síťovém protokolu.

Klíčové vlastnosti systému

Předem nelze jednoznačně určit, který způsob přenosu snímků (`Base64` kódované snímky nebo soubory s jednotlivými rámci) a následně který způsob vykreslení přeneseného snímku je nejvhodnější.

Mezi některé významné vlastnosti výsledné aplikace tedy patří:

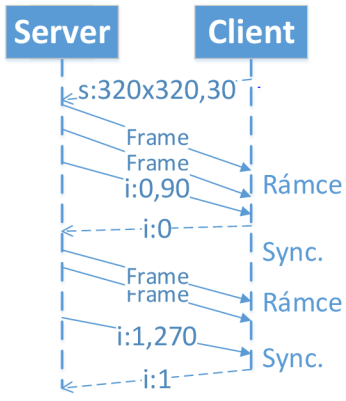
- Možnost přenášet jednotlivé snímky po souborech (případně `Base64/JPEG` kódované snímky) na základě uživatelsky nastavitelné konfigurace.
- Možnost přenášet různě veliké skupiny snímků (speciálním případem je skupina snímků obsahující právě klíčový snímek – poté není zapotřebí kódovat a dekódovat jakékoliv rozdílové snímky).
- Na straně serveru lze jednoduše přidat dodatečné zpracování snímků například ve formě nějakého obrazového filtru.

Návrh komunikačního protokolu

Na rozdíl od síťového provozu jsou zprávy přes Bluetooth kanál přenášeny spolehlivě. Obdobou detekce ztráty paketu v `TCP` protokolu je však v tomto případě situace, kdy klient přestává snímky zpracovávat v reálném čase a začíná docházet ke zpožďování videa.

K tomu účelu by mohla sloužit speciální zpráva, na kterou klient odpoví předem dohodnutým způsobem. Na straně serveru je poté možné měřit dobu odpovědi a podle toho vyhodnotit aktuální zahlcení přenosového kanálu.

V tabulce 4.2 jsou stručně popsány jednotlivé zprávy, které mohou být zasílány mezi klientem a serverem. Měnitelné parametry jsou psány velkými písmeny. Nepovinná část zprávy je poté uzavřena v hranatých závorkách.



Obrázek 4.7: Komunikační protokol

Na obrázku 4.7 je zobrazen příklad komunikace mezi klientem a serverem. Konkrétně je v tomto případě zaslána počáteční zpráva, ve které klient deklaruje, že jeho velikost obrazovky je 320×320 pixelů a nechce dostávat více než 30 snímků za sekundu. Dále jsou v rámci připojení přeneseny dvě synchronizační zprávy a mezi nimi dvě dávky rámců videa.

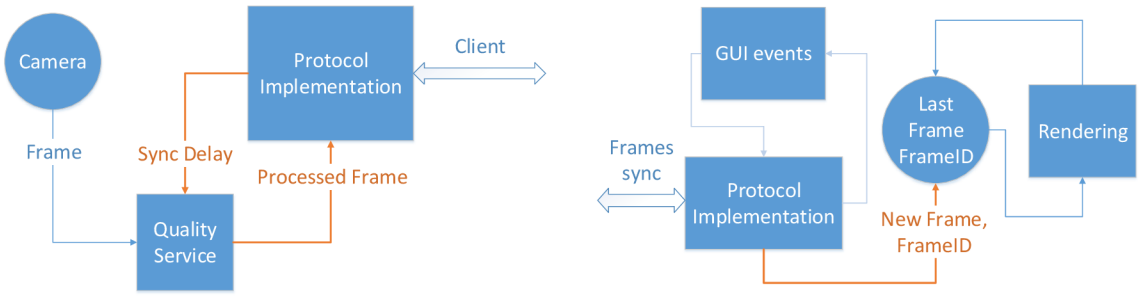
Název	Formát	Popis
Počáteční zpráva	`s:[WxH[,F[,G]]`	Zpráva, kterou zasílá klient hned po připojení. Parametry W a H značí rozlišení displeje, F maximální přenosovou rychlost a G velikost skupiny snímků.
Rámec videa	`/9g...`[1]	`Base64` kódovaný `JPEG` obrázek posílaný ze serveru na klienta.
Synchronizační zpráva	`i:C[,O]`	Zpráva, kterou jednou za čas pošle server klientovi. Parametr C je postupně se inkrementující čítač. Spolu s ním lze volitelně posílat také orientaci zařízení (parametr O). Podle něj je pak možné na klientovi se vyrovnat i s pootočeným obrazem videa.
Synchronizační odpověď	`i:C`	Klient jednoduše odpoví tak, že parametr C odešle jako odpověď zpátky na server.
Pořízení snímku	`p:D`	Příkaz k pořízení snímku kamerou telefonu. Parametr D určuje prodlevu před vyfotografováním obrazu.
Změna nastavení	`x:K,V`	Příkaz ke změně nastavení způsobu přenášení obrazu. Parametry K a V jsou dvojice klíč-hodnota jednoznačně určující novou hodnotu daného nastavení.
Náhled fotografie	`d:`	Po zaslání příkazu odešle telefon na hodinky naposledy pořízenou fotografii.
Zoom	`z:D`	Slouží k přiblížení nebo oddálení kamery. Parametr D určuje relativní změnu přiblížení, která je limitována možnostmi kamery daného zařízení.

[1] V rámci `JPEG` formátu je specifikováno, že soubor musí začínat binární sekvencí hexadecimálních čísel `FF DB`, což odpovídá v `Base64` kódování sekvenci znaků `/9g=`. Díky tomu lze jednoduše rozlišit obrázky od ostatních příkazů.

Tabulka 4.2: Popis jednotlivých zpráv protokolu.

Jak již bylo řečeno, pomocí příkazu `x:K,V` lze za běhu měnit některé parametry přenosu. Mezi ně patří například:

- `PHOTO_DELAY` – Umožňuje nastavit výchozí prodlevu (v sekundách) při pořizování fotografie.
- `TRANSFER` – Nastavuje, zda se snímky přenáší jako `Base64/JPEG` text – konstanta (`TRANSFER_BASE64` nebo po jednotlivých souborech (`TRANSFER_FILES`).
- `GOP` – Umožňuje nastavit novou hodnotu skupiny snímků.
- `FILTER` – Nastavuje index aktuální scény.



(a) Návrh serveru.

(b) Návrh klienta.

Obrázek 4.8: Návrh klient-server architektury.

Návrh aplikace na hodinkách

Aplikace na hodinkách by se dala rozdělit do tří nezávislých částí:

- **Implementace komunikačního protokolu** – Aby byla data zpracována co nejrychleji, je implementace protokolu oddělena od vykreslování snímků. To umožní v případě zahlcení kanálu data rychle přečíst a nepotřebné snímky jednoduše zahodit.
- **Vykreslovací smyčka** – O optimalizaci vykreslovací smyčky byla již řeč v podkapitole 2.6. Nový snímek je tedy vykreslen v rámci události `requestAnimationFrame`, a to pouze za předpokladu, že se liší od posledně přijatého.
- **Přídavná funkcionalita a uživatelské rozhraní** – Ostatní části aplikace se neliší od běžné webové aplikace a z pohledu návrhu jim tudíž není třeba věnovat zvýšenou pozornost.

Návrh klientské části aplikace je zobrazen na obrázku 4.8b. Část aplikace implementující protokol přijímá příkazy vyvolané uživatelskými akcemi a na základě nich komunikuje se serverem. Přijaté snímky poté předává vykreslovací smyčce, která je zpracovává.

Návrh uživatelského rozhraní

Uživatelské rozhraní aplikace na hodinkách tvoří úvodní obrazovka, která je virtuálně rozdělena na čtvrtiny, kde kliknutí (resp. dotyk prstem) příslušné čtvrtiny vyvolá jednu z následujících akcí:

1. Vzdálené pořízení snímku a jeho následný přenos na hodinky. Po přenosu snímku je fotografie zobrazena na displeji hodinek. Uživatel má také možnost nastavit krátkou prodlevu před pořízením fotografie.
2. Přepínání mezi různými scénami podporovanými ze strany telefonu. Příklady scény mohou být například černobílé snímky.
3. Zobrazení / schování uživatelského prvku, který zobrazuje aktuální počet přenesených snímků za sekundu.
4. Přechod na stránku s veškerým nastavením.

Graficky je uživatelské rozhraní naznačeno na obrázku 4.9.

(a) Úvodní stránka (b) Stránka s nastavením (c) Detail nastavení

Obrázek 4.9: Návrh uživatelského rozhraní klientské aplikace.

Návrh serveru

Server, jehož schéma je vidět na obrázku 4.8a, je implementován v rámci služby běžící na pozadí. Stejně jako klientskou část aplikace lze i serverovou rozdělit na několik logických bloků:

- **Propojení s kamerou** – Část aplikace starající se o příjem dat z kamery zařízení (obecně lze mít ale i jiné typy zdroje videa). V rámci systému Android existují předpřipravené knihovny k připojení ke kameře a následné získávání jednotlivých snímků. Tyto snímky bude zapotřebí vhodným způsobem zpracovat a připravit k odeslání klientovi.
- **Implementace protokolu** – Podobně jako na straně klienta je i na straně serveru zapotřebí implementovat logiku přenášení dat. Kromě komunikace s klientem a získávání snímku ze zdroje videa je zapotřebí také vyřešit řízení kvality přenášeného obrazu.
- **Řízení kvality videa** – Klíčová komponenta, která určuje optimální kvalitu snímku na základě zpoždění odezvy na synchronizační zprávu zaslanou klientovi. Schopnost predikovat nejvyšší možnou kvalitu následujícího snímku, aniž by došlo k zahlcení Bluetooth kanálu v podstatě přímo ovlivňuje vizuální kvalitu výsledného videa.

Ve schématu jsou pro jednoduchost opomenuty části aplikace implementující rozšířenou funkcionalitu, jako například pořizování fotografií, implementace různých scén a jiné.

4.6 Implementace

Implementačně nejnáročnější části serverové aplikace je pravděpodobně řízení kvality videa, kterému bude v rámci této podkapitoly věnována zvýšená pozornost. Dále bude v této podkapitole popsán také způsob zpracování snímků telefonu předtím, než jsou odeslány klientovi.

Řízení kvality přenosu

V případě, že by kvalita přenosu nebyla řízena vůbec, mohou v zásadě nastat dvě různé situace. Pravděpodobnější z nich je, že bude telefon zapisovat na klienta příliš mnoho dat,

které nebude možné v reálném čase zpracovávat. Data se tedy začnou postupně hromadit ve vyrovnávacích pamětech a video se začne postupně zpožďovat. Zpoždění videa se postupně kumuluje, až dosáhne hodnot, kdy se aplikace stane nepoužitelnou. Dokonce může také dojít k přehlcení vyrovnávací paměti a pádu aplikace.

Ve druhém případě je přenosové pásmo nevyužité a video nedosahuje takové kvality, jaké by mohlo. Cílem řízení kvality obrazu by tedy měla být minimalizace zpoždění videa a zároveň nalezení co nejlepší kombinace následujících parametrů: přenosové rychlosti (dále jen `FPS`, z angl. *Frames Per Second*, čili počet snímků za sekundu), rozlišení obrazu a míry komprese.

Hranice jednotlivých parametrů lze experimentálně nastavit například dle tabulky 4.3. Experimentovat však lze i s různými jinými alternativami. Kombinace nejlepších parametrů v tabulce 4.3 tak určuje kvalitu videa, kterou již dále nemá smysl zvyšovat. S kombinací nejhorších parametrů budou jednou za čas odeslány silně podvzorkované nekvalitní snímky. Detailnější analýze optimálnímu výběru jednotlivých parametrů videa bude věnována podkapitola 4.7.

Parametr	**Min**	**Max**	**Poznámky**
`FPS`	-	24	Viz 3.5
Rozlišení obrazu	160×160	320×320	Hodnota nastavena dle klienta
Míra komprese	100	0	Udává komprimační poměr `JPEG` formátu výsledného snímku.
Zpoždění obrazu	0 ms	250 ms	Zpoždění nad maximální mez indikuje zahlcení kanálu. Na základě toho by mělo dojít ke snížení datového toku snížením ostatních parametrů videa.

Tabulka 4.3: Příklad nastavení parametrů kvality videa.

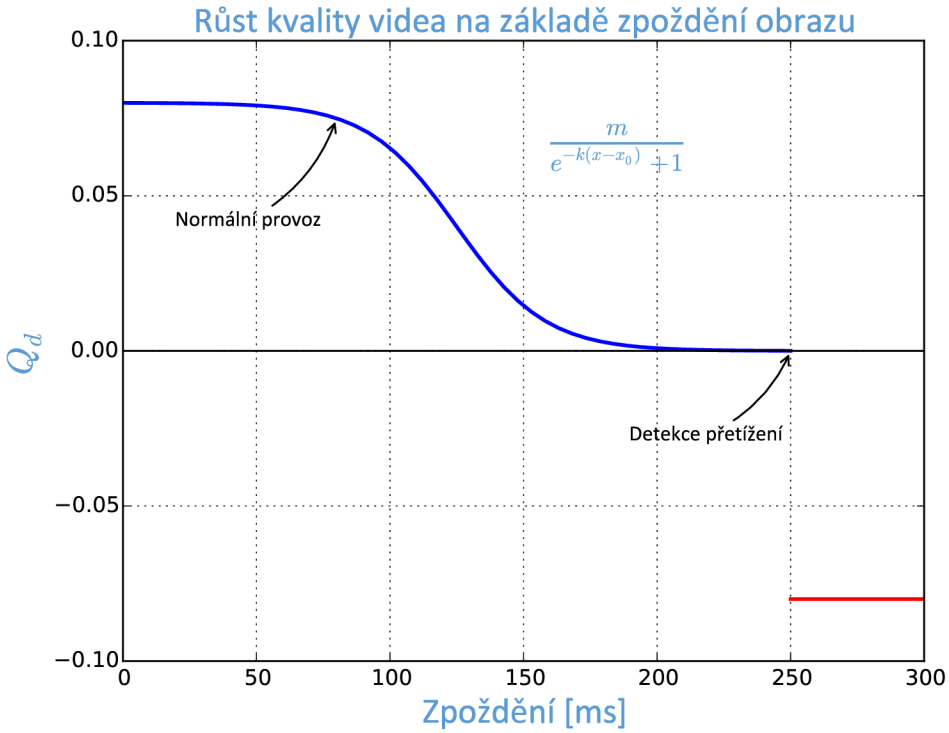
Řízení datového toku provádí speciální modul, který průběžně dostává naměřenou hodnotu aktuálního zpoždění videa a dle toho určuje vhodnou kvalitu videa vyjádřenou parametrem Q. Pokud je odezva klienta rychlá, může kvalitu videa zvyšovat. Jakmile překročí experimentálně zvolenou mez, mělo by dojít k prudkému poklesu kvality tak, aby se problém zpožďování videa co nejrychleji vyřešil. Podobného principu je využito také v `TCP` síťovém protokolu při řízení síťového toku.

Parametr Q nabývá hodnot z intervalu od 0.0 do 1.0. Na základě jeho velikosti jsou poté pro další snímky nastaveny jednotlivé parametry videa (kvalita snímků, frekvence snímkování, případně rozlišení). Nejjednodušším způsobem může být například lineární interpolace mezi nejhorší a nejlepší hodnotou daného parametru videa na základě kvality Q.

Při malých hodnotách zpoždění je vhodné kvalitu videa Q zvyšovat rychleji. Naopak pokud se zpoždění začíná blížit limitnímu, měla by kvalita zůstávat nezměněna. Takové chování lze popsat například pomocí upravené exportní funkce, kterou můžeme pojmenovat $Q_d(x)$, jejíž průběh je zobrazen na obrázku 4.10.

Parametr m značí maximální růst kvality videa, která nabývá hodnot z intervalu od 0 do 1, v rámci jedné odpovědi na synchronizační zprávu. Parametr k poté určuje „strmost“ křivky a x_0 je polovina maximálního zpoždění (v tomto případě tedy 125 ms).

Po každé odpovědi na synchronizační zprávu je poté kvalita videa jednoduše upravena



Obrázek 4.10: Řízení kvality videa na základě aktuálního zpoždění.

dle vztahu:

$$Q_{t+1} = Q_t + Q_d(x) \tag{4.3}$$

Úprava snímků zdrojového videa

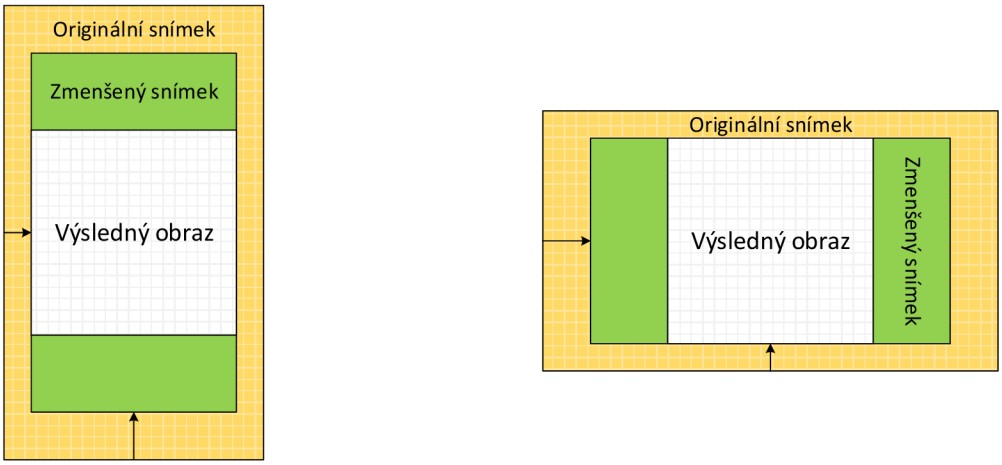
Navržený protokol umožňuje na hodinky zasílat video z libovolného zdroje. Zdrojové video přitom může mít obecně jiné rozlišení, než jaké je rozlišení displeje hodinek. Každý snímek je tedy zapotřebí před odesláním na hodinky zpracovat a poté zakódovat. Běžné využití aplikace předpokládá přenos videa z kamery telefonu na hodinky. V následujícím textu bude tedy místo o obecném zdroji videa řeč především o kameře telefonu.

V systému Android je možné pořizovat snímky pouze několika přednastavených rozlišení, které jsou zpravidla dány výrobcem telefonu, typem kamery apod. Každý snímek je tedy zapotřebí před odesláním na hodinky zpracovat a poté zakódovat.

Nezpracované snímky pořízené kamerou jsou kódovány pomocí `YUV` barevného modelu, který obsahuje jednu jasovou složku a dvě barevné. V případě že nejsou využívané rozdílové snímky a zároveň není zapotřebí snímky upravovat v rámci klientem požadované scény, lze z takového snímku jednoduše vytvořit náhled a ten přímo převést do `JPEG` formátu. V opačném případě je možné převést snímek do `RGB` barevného modelu například pomocí předpřipraveného *RenderScript* modulu `renderscript.ScriptIntrinsicYuvToRGB`.

Vytvoření náhledu snímku

Na základě rozlišení displeje klienta, které je zaslané hned na začátku přenosu, je zapotřebí vybrat vhodné rozlišení snímků. Zde se nabízí například možnost vybrat takové rozměry, u nichž se celkový počet pixelů co nejvíce blíží počtu pixelů displeje klienta. Dále je možné vybrat co nejmenší rozlišení kamery tak, aby ani jeden rozměr nebyl menší než odpovídající rozměr displeje klienta.



(a) Obrázek orientovaný na výšku

(b) Obrázek orientovaný na šířku

Obrázek 4.11: Zpracování snímku pořízeného kamerou smartphonu.

Snímek pořízený kamerou může být zmenšen a v případě nesouhlasných poměrů stran oříznut a vycentrován. Způsob zmenšení snímku je ukázán na obrázku 4.11. Originální obrázek je zde reprezentovaný oranžovou barvou. Snímky orientované na výšku jsou zmenšeny dle poměru šířek požadovaného obrázku a originálního. Pokud je snímek orientovaný na šířku, je zmenšen dle poměru výšek požadovaného obrázku a originálního.

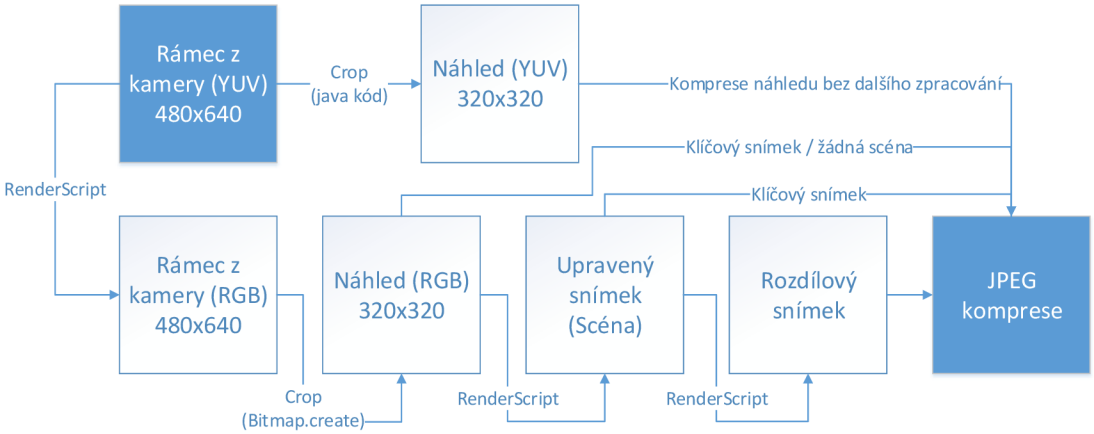
Zmenšený obrázek (vyznačený zeleně na obrázku 4.11) je poté oříznut podle středu, čímž vzniká snímek výsledné velikosti (vyznačený bílou barvou). Ten je dále převeden do `JPEG` formátu s požadovaným komprimačním poměrem, zakódován pomocí `Base64` kódování a následně odeslán klientovi.

Hlavní nevýhodou takového způsobu zpracování snímků je kromě zvýšené výpočetní náročnosti především převzorkování snímku, díky čemuž může obecně docházet ke zmenšení jeho kvality. Výhodou je poté skutečnost, že uživatel vidí na hodinkách v jednom rozměru celý rozsah kamery telefonu.

Vyříznutí snímku

Druhou možností je prostý výřez obrázku tak, aby výřez odpovídal rozlišení displeje hodinek. Výhodou tohoto řešení je obecně vyšší kvalita výsledného obrazu a nižší výpočetní nároky na telefon.

Detailní schéma zpracování snímku včetně výpočtu rozdílových snímků a případné modifikace snímku v závislosti na zvolené scéně je na obrázku 4.12. Rozměry snímků v tomto schématu se mohou lišit v závislosti na hardwarové konfiguraci konkrétních zařízení.



Obrázek 4.12: Schéma zpracování snímku pořízeného kamerou.

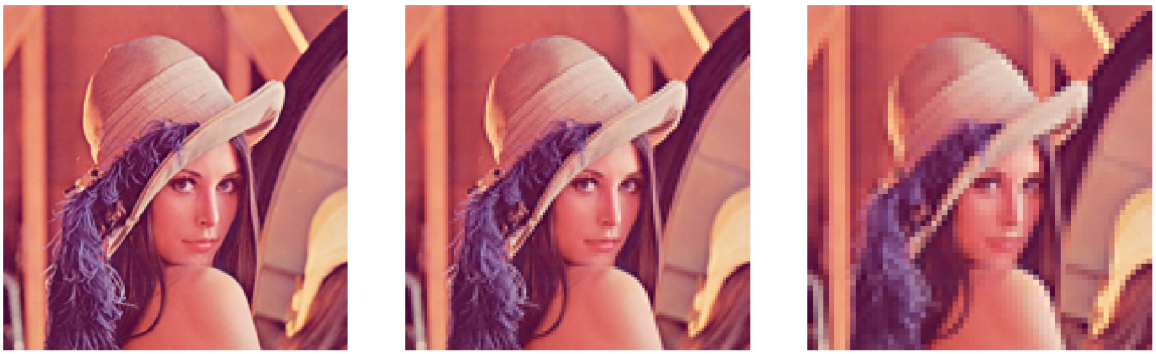
4.7 Optimální výběr parametrů videa

K nalezení vhodné rovnováhy mezi jednotlivými parametry videa je vhodné si uvědomit vliv kvality obrazu na jeho velikost (a tedy i na čas potřebný ke zpracování každého snímku).

Co se týče rozlišení obrazu, je názorné srovnání uvedeno na obrázku 4.13, kde byl originální obrázek velikosti 512 × 512 pixelů v různých mírách podvzorkován. Všechny obrázky poté byly převedeny do JPEG formátu a vykresleny na 1/3 stránky.

Z obrázku 4.13 je patrné, že i obrázek výrazně podvzorkovaný může ve videu vypadat přijatelně. Při převedení problému na velikost displeje hodinek se zdá být tedy rozumné experimentovat s obrázky mezi velikostmi zhruba od 160×160 do 320×320 pixelů. Závislost velikosti obrazu čistě na jeho rozlišení je zhruba kvadratická.

Podobný vliv na velikost výsledného obrazu má také míra `JPEG` komprese, kterou umí některé knihovny pro práci s obrazem nastavit. Pomocí *OpenCV* knihovny pro jazyk Python lze vygenerovat mnoho různých obrázků s klesající mírou komprese (viz ukázka kódu v příloze B.4).



(a) Lena 196 × 196: 65.6 KB (b) Lena 128 × 128: 35.4 KB (c) Lena 64 × 64: 16.5 KB

Obrázek 4.13: Vliv rozlišení obrazu na jeho velikost.

Na obrázku 4.14 jsou zobrazeny některé vybrané vygenerované obrázky. U obrazů s vysokým kompresním poměrem dochází k viditelným artefaktům, které však velmi rychle mizí s klesajícím kompresním poměrem.

Velikosti jednotlivých obrazů jsou zaneseny do grafu na obrázku 4.15. Z něj je patrné, že ze začátku roste kvalita obrazu spolu s klesajícím kompresním poměrem zhruba lineárně. Od určité kvality (zhruba okolo hodnoty 80) však lineární závislost přestává platit a velikost obrazu začíná prudce růst.

Z pohledu přenosu videa se tedy jeví, že nemá příliš smysl využívat velmi kvalitních obrázků (veliký nárůst kvality obrazu spolu s potřebným výkonem na jeho zpracování nestojí za velmi malý vizuální nárůst kvality obrazu). Pro přenos se zdá být vhodné vybírat kvalitu obrázku někde mezi 0% a 80% maximální možné kvality.

Výběr kvality obrazu

Při výběru obrazu požadované velikosti zůstává otázka, zda je vhodnější vybrat obrázek s nižším (vyšším) rozlišení nebo raději upravit kompresní poměr obrázku při zachování stejného rozlišení. Takové rozhodování by bylo možné provádět za běhu u každého snímku zvlášť. To by však bylo nad rámec hardwarových možností telefonu, a proto jsou parametry všech snímků ve videu vybrány stejným způsobem na základě následujícího experimentu.

Přestože měření kvality obrazu je velmi spjato se subjektivním vnímáním rozdílů mezi jednotlivými snímky, lze jeho kvalitu alespoň přibližně určit například pomocí poměru signálu k šumu:

$$PSNR = 10 \log_{10} \frac{255^2}{\frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} (v(i,j) - w(i,j))^2} \tag{4.4}$$

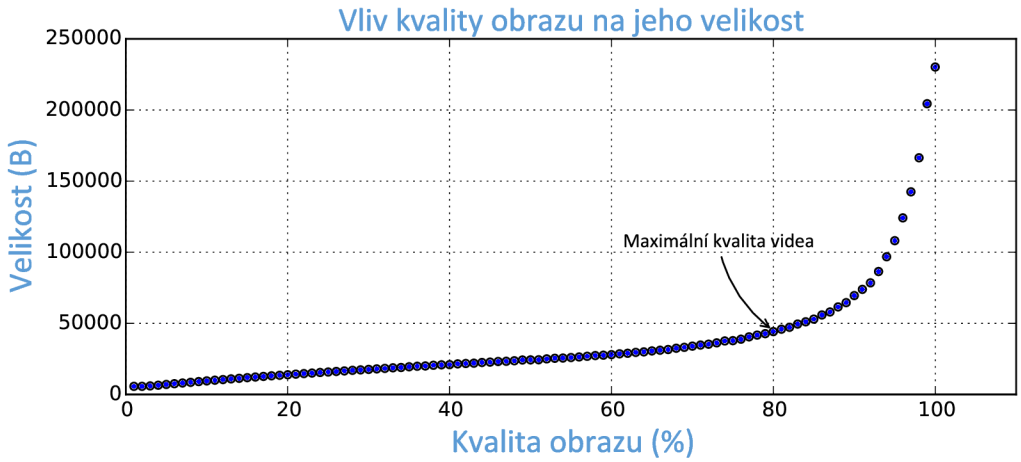
Kde v a w jsou porovnávané obrazy a $M \times N$ jejich rozměry. Pomocí (zkráceného) skriptu 4.4 lze vygenerovat sadu obrázků, které se liší mírou komprese a rozlišením. Tyto obrázky jsou poté setříděny do skupin obrázků se stejnou velikostí (po zaokrouhlení na KB). Každý obrázek v rámci skupiny je poté převzorkován na velikost originálního obrázku. Rozlišení a míra komprese obrazu s maximálním poměrem signálu k šumu jsou zaneseny do grafu na obrázku 4.16.

(a) Lena 100: 224 KB

(b) Lena 25: 15.4 KB

(c) Lena 1: 5.59 KB

Obrázek 4.14: Vliv míry komprese obrazu na jeho kvalitu.



Obrázek 4.15: Graf závislosti velikosti obrazu na jeho kompresním poměru.

```
import cv2
...
data = initData()
for res,q in allCombinations:
    thumb = cv2.resize(origin, (res, res))
    fileSize = measureSize(thumb, q)
    data[fileSize / 1024].append([thumb, fileSize, q, res])

for k in data:
    res,q = findBestAccordingToPSNR(data[k])
    ...
```

Výpis kódu 4.4: Měření kvality obrázků s různým rozlišením a mírou komprese.

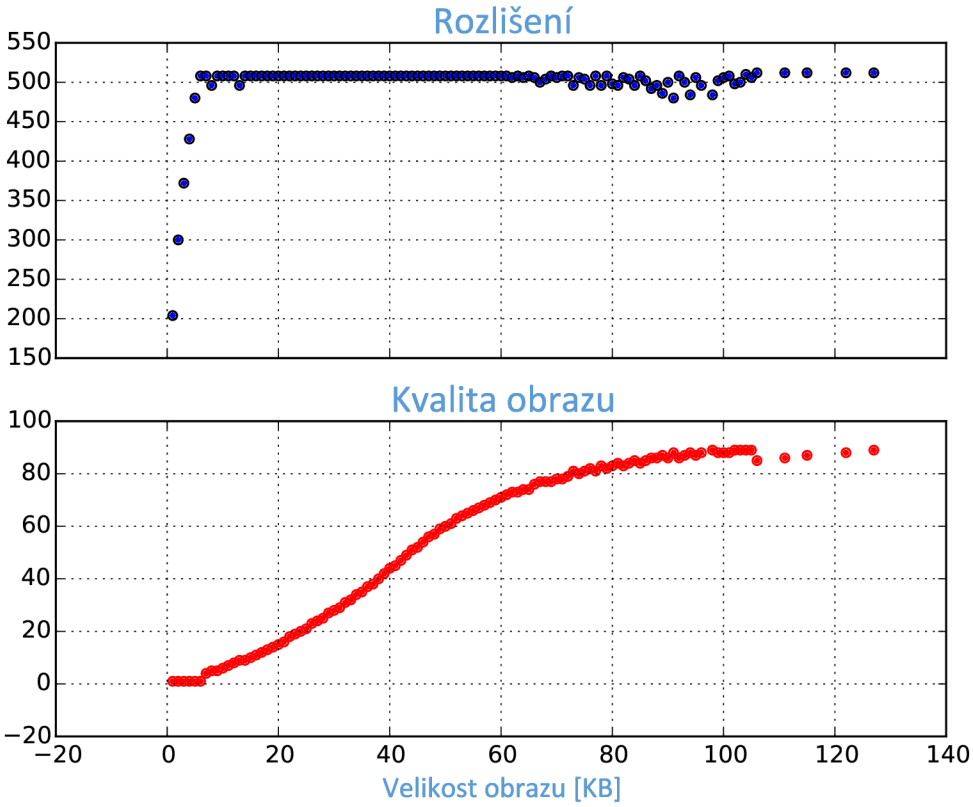
Na obrázku 4.16 jsou vidět nejlepší vybrané kombinace rozlišení a kompresního poměru při dané velikosti obrazu. Ukazuje se, že pro většinu velikostí se optimální rozlišení blíží maximálnímu. Na druhou stranu kompresní poměr obrazu během celého průběhu postupně roste.

U velmi malých obrázků je zřejmé, že rozlišení i při velmi vysokém kompresním poměru musí klesat. Naopak u obrázků s velmi nízkým kompresním poměrem začíná takto navržený algoritmus selhávat, neboť skupiny obrázků s danou velikostí začínají být velmi malé, což může výsledky značně zkreslovat.

Z pohledu přenosu videa z předešlého měření jednoznačně plyne závěr, že rozlišení obrazu není zapotřebí dynamicky měnit. V případě velmi pomalého přenosu lze absenci extrémně malých snímků kompenzovat sníženým počtem přenesených snímků za sekundu.

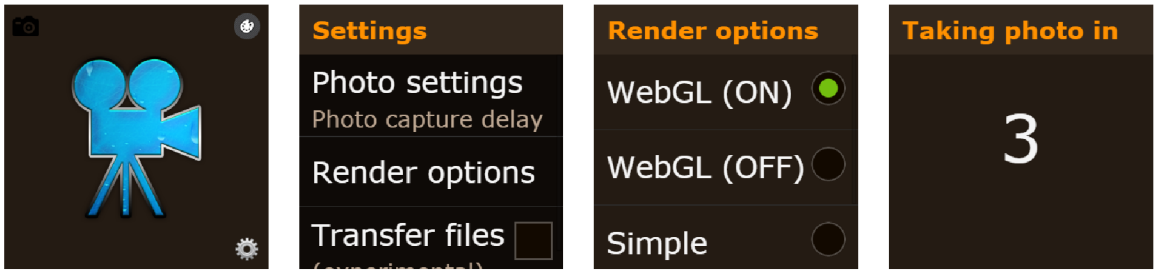
4.8 Výsledné nastavení přenosu videa

V předešlé podkapitole bylo ukázáno, jaký vliv má velikost obrazu na jeho kvalitu. Dále byl také v podkapitole 4.5 navržen způsob řízení kvality videa. V rámci takto navrženého systému existuje velké množství parametrů, které lze nejrůznějšími způsoby optimalizovat. Jednotlivé parametry jsou uvedeny níže:



Obrázek 4.16: Výběr vhodné kombinace kvality obrazu a jeho rozlišení při dané velikosti obrazu.

- `FPS` – Počet přenesených snímků za sekundu. Spolu s parametrem `IMAGE_QUALITY` se jedná o jediné parametry, které se mění dynamicky v rámci přenosu.
- `IMAGE_RESOLUTION` – Rozlišení přenášeného obrazu. Jak již bylo řečeno v předešlé podkapitole, je tento parametr nastaven před začátkem přenosu dle velikosti displeje klienta.
- `IMAGE_QUALITY` – Kompresní poměr přenášeného obrazu.
- `SYNC_PERIOD` – Perioda zasílání synchronizačních zpráv. Kratší perioda zasílání synchronizačních zpráv znamená vyšší režii řízení přenosu a s tím i související pokles kvality videa. Při nastavení příliš dlouhé periody naopak může dojít k situaci, kdy není zavčas podchycen problém při přenášení obrazu. Než dojde ke snížení kvality videa je Bluetooth kanál zahlcen a dochází ke značnému zpoždění videa.
- `MAX_DELAY` – Maximální akceptovatelné zpoždění. Při větším zpoždění dochází k nárazovému snížení kvality videa.
- `MAX_QUALITY_INCREMENT` – Přírůstek kvality při nulovém zpoždění. Příliš vysoké hodnoty způsobují prudký nárůst kvality při nižším zpoždění následovaný častějším dočasným přehlcením kanálu. Naopak při nižších hodnotách je průměrné zpoždění nižší, což na druhou stranu nepříznivě ovlivní kvalitu obrazu.



(a) Úvodní obrazovka (b) Přehled nastavení (c) Detail nastavení (d) Odpočet

Obrázek 4.17: Snímky některých obrazovek výsledné aplikace.

- `QUALITY_STEEPNESS` – Parametr určující tvar křivky na obrázku 4.10.

Určit optimální kombinaci veškerých zmíněných parametrů by bylo značně komplikované. Navíc by výsledek pravděpodobně záležel na dalších vnějších podmínkách, jako jsou například osvětlení místnosti, vzdálenost hodinek od smartphonu a jiné.

Jednotlivé parametry tudíž byly nastaveny experimentálně dle tabulky 4.4. Oproti odhadnutým parametrům v tabulce 4.3 byla tedy především výrazně zkrácena doba akceptovatelného zpoždění obrazu.

Název parametru	Hodnota
`SYNC_PERIOD`	1400 ms
`MAX_DELAY`	275 ms
`MAX_QUALITY_INCREMENT`	0.14
`QUALITY_STEEPNESS`	0.1

Tabulka 4.4: Výsledné experimentální nastavení parametrů.

4.9 Vyhodnocení přenosu

Výsledná aplikace je na straně telefonu implementovaná jako služba běžící na pozadí. Na straně hodinek se aplikace sestává z několika různých obrazovek, jejichž náhledy jsou na obrázku 4.17.

Implementované řešení umožňuje přenášet jednotlivé rámce videa (ať už se jedná o klíčové či rozdílové snímky) jako jednotlivé soubory či jako `Base64/JPEG` kódovaný text.

Během testování se ukázalo, že architektura poskytnutá výrobcem hodinek neumožňuje zasílat dostatečné kvantum souborů na to, aby se dalo hovořit o přenosu videa. Přenos jednotlivých souborů začne v takovém případě končit chybou a bezchybně se přenese pouze nepatrné procento souborů.

I přesto je ale možnost přenosu souborů v programu zachována pro možné budoucí využití v případě, že by novější verze systému pro hodinky umožňovala efektivnější přenos velkého množství malých souborů.

(a) Scéna 1 (3.96 MB).

(b) Scéna 2 (9.14 MB).

Obrázek 4.18: Náhledy scén, ve kterých bylo prováděno měření rychlosti přenosu videa.

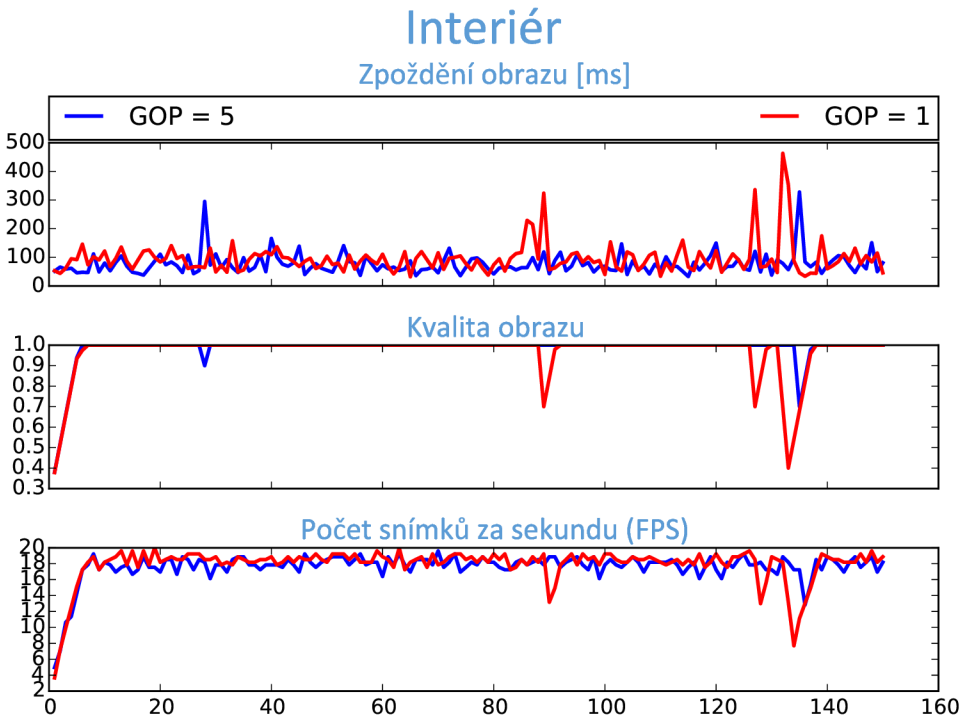
Kvalita přeneseného videa

O kvalitě výsledného videa zcela jistě svědčí množství vykreslených snímků za vteřinu, kvalita přenesených snímků a zpoždění videa. Zatímco zpoždění videa a jeho kvalitu lze měřit na straně serveru, počet skutečně vykreslených snímků je nutné měřit na straně klienta. Hlavním důvodem je především možnost zahození některých snímků v případě, že jsou jednotlivé snímky zasílány příliš rychle a klient stihne v rámci jedné periody překreslení obrazovky zpracovat více než jeden snímek.

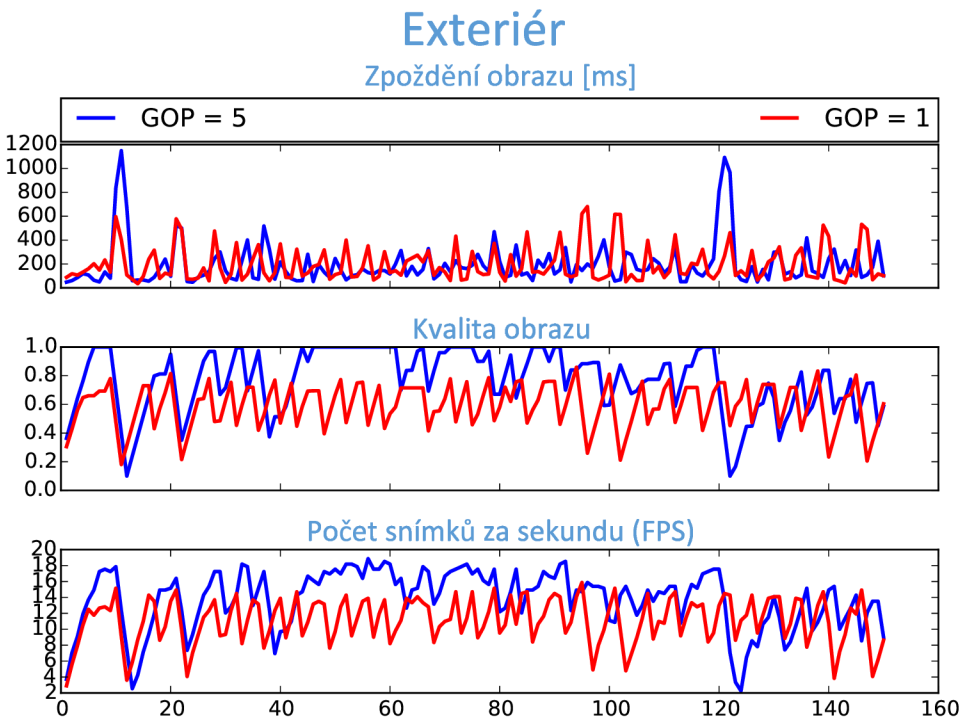
Pro měření kvality přeneseného videa byly zvoleny dvě různé scény – jedna v interiéru a druhá ve venkovním prostředí. V obou scénách se pohybovala pozorovaná osoba tak, aby rozdíly mezi časově blízkými snímky nebyly nulové. Náhledy obou scén je možné vidět na obrázku 4.18. Za povšimnutí stojí především diametrálně odlišná velikost fotografií pořízených v jednotlivých scénách, což velmi ovlivňuje kvalitu výsledného videa.

Měření jednotlivých parametrů probíhalo po dobu zhruba dvou minut a to vždy se zasláním synchronizační zprávy (respektive s obdržením odpovědi na danou zprávu). V obou scénách bylo navíc měření prováděno dvakrát. Jednou s využitím rozdílových snímků (velikost skupiny byla v tomto případě nastavena na hodnotu 5). Druhé měření odpovídá prostému zasílání pouze klíčových snímků a jejich zobrazení bez využití `WebGL` (`GOP = 1`).

Výsledek měření je znázorněn na grafech na obrázcích 4.19 a 4.20. V první scéně jsou výsledky obou metod srovnatelné (obě přenášejí víceméně kvalitní snímky rychlostí okolo 18 Hz s minimálním zpožděním. Ve druhé scéně dosahuje naopak lepších výsledků metoda zasílání rozdílových snímků, která viditelně umožňuje zasílat větší množství kvalitnějších snímků.



Obrázek 4.19: Měření kvality videa při běžném provozu (scéna 1).

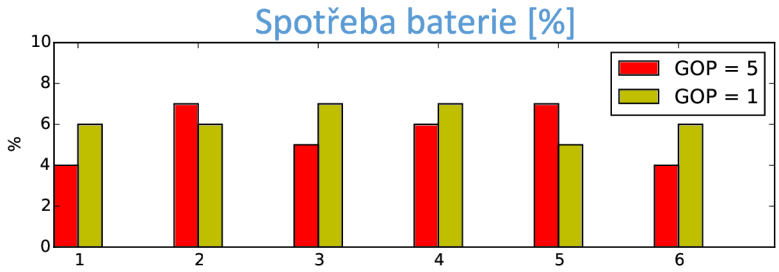


Obrázek 4.20: Měření kvality videa při běžném provozu (scéna 2).

Spotřeba baterie

S kapacitou dnešních baterií použitelných v chytrých hodinkách bývá důležitým parametrem také energetická náročnost aplikace. V případě, že by jedna z metod měla tendenci spotřebovávat velké množství baterie, je otázkou, zda by koncový uživatel neocenil spíše šetrnost k baterii, byť by daná metoda mohla dosahovat o něco lepších výsledků.

Z toho důvodu byla provedena série šesti měření spotřeby baterie v rámci pětiminutových přenosů videa. Z výsledků měření, které jsou na obrázku 4.21, plyne, že obě metody jsou pro koncového uživatele takřka srovnatelné (pouze nepatrně lepší co se týče spotřeby se zdá být metoda zasílání rozdílových snímků).



Obrázek 4.21: Spotřeba baterie hodinek při pětiminutovým přenosu videa.

Využití na pozadí

Díky tomu, že je serverová část implementovaná jako služba běžící na pozadí, lze aplikaci spustit i bez jakékoliv interakce s telefonem. Aplikace tudíž může běžet i v režimu, kdy je displej telefonu vypnutý. To však s sebou nese jistá omezení z důvodu šetření baterie telefonu. Výpočet rozdílových snímků tak trvá podstatně déle a telefon přestává zvládat v reálném čase zpracovat dostatečně velké množství snímků. V tabulce 4.5 jsou shrnuty výsledky podobného měření, jako je na obrázku 4.19, avšak se zamknutým displejem.

V případě tohoto měření nedochází k zahlcování přenosového kanálu, neboť průměrná odezva hodinek je poměrně krátká (méně než 150 milisekund). Kvalita přenesených snímků tak v obou případech může stoupat až k maximu, avšak telefon díky sníženému výkonu nezvládá zpracovávat požadované množství snímků, a proto rychlost videa klesá v závislosti na použité metodě pod 15 snímků za vteřinu.

V případě využití kamery telefonu jako zdroje videa by podobných vlastností vykazovalo také měření za špatných světelných podmínek. Tehdy by bylo pro změnu množství přenesených snímků limitováno množstvím snímků, které by za daných podmínek byla schopna vystavit kamera telefonu.

	Zpoždění [ms]	**Kvalita obrazu [%]**	**FPS [Hz]**
Klíčové snímky	108	0.98	15.3
Rozdílové snímky (`WebGL`)	147	0.96	12.2

Tabulka 4.5: Průměrné hodnoty kvality videa při měření s vypnutým displejem

4.10 Shrnutí výsledků

Aplikace umožňuje zasílat data dvěma různými použitelnými způsoby, přičemž první z nich používá pouze klíčové snímky videa. Tento způsob přenosu je vhodný v případě, kdy je telefon zamknutý. Dále by byl také vhodnější v případě, kdy by uživatel vlastnil starší a méně výkonný telefon, který by hardwarově nezvládal dekódovat rámce ze zdroje videa, pořizovat náhledy jednotlivých snímků a poté ještě vypočítávat rozdílové snímky.

V ostatních případech je lepší zasílat některé snímky kódované jako rozdíl oproti klíčovým snímkům. Nejvíce se rozdíl projeví v případě, kdy přenášený obraz obsahuje velké množství obrazových elementů, a tudíž `JPEG` komprese není natolik účinná. Snížený objem přenášených dat se dále pozitivně projeví v případě horších podmínek Bluetooth spojení, například když je signál rušen vnějším prostředím. Přehledné shrnutí jednotlivý metod lze nalézt v tabulce 4.6.

	Klíčové snímky	**Rozdílové snímky**
Jednoduchá scéna	Ano	Ano
Složitá scéna (obsahuje mnoho obrazových elementů)	Ne	Ano
Zamknutý telefon	Ano	Ne
Nekvalitní Bluetooth spojení	Ne	Ano
Pomalý zdroj videa	-	-

Tabulka 4.6: Vhodné využití jednotlivých metod za daných podmínek.

Kapitola 5

Závěr

Cílem práce bylo vytvoření aplikace typu klient-server realizující přenos obrazu z Android zařízení na chytré hodinky s operačním systémem Tizen. Aplikace byla úspěšně navržena, implementována a posléze také otestována a vyhodnocena.

Pro návrh a realizaci bylo nejen nutné seznámit se s platformami Android a Tizen, ale také s webovým jádrem *Webkit*, v rámci něhož jsou spouštěny jednotlivé Tizen aplikace. Pro stanovení požadavků na výslednou aplikaci bylo zapotřebí se nejprve seznámit s existujícími řešeními a posléze provést řadu testů, které měly určit limitní parametry přenášeného videa. Na základě této analýzy byly vybrány vhodné metody k vykreslení přeneseného snímku a těm byl podřízen formát dat posílaných z telefonu na hodinky. Dále byl vytvořen mechanismus dynamického adaptování kvality videa tak, aby byl co nejefektivněji využit přenosový kanál. K tomu účelu byl navrhnut jednoduchý přenosový protokol, který umožňuje serveru přizpůsobovat kvalitu snímků aktuálnímu stavu spojení zařízení, díky čemuž nedochází k přehlcení přenosového kanálu, na druhou stranu je ale využita celá jeho kapacita.

Video je na straně telefonu v závislosti na použité metodě přenosu zpracováno a poté posíláno buď po jednotlivých klíčových snímcích, nebo po skupinách, ve kterých je určité množství snímků kódovaných jako rozdíl od posledního klíčového snímku. V obou metodách přenosu se ukázalo nejvhodnější z důvodu softwarových a hardwarových omezení hodinek jednotlivé snímky posílat jako `Base64/JPEG` kódovaný text na klienta, kde je poté snímek akcelerovaně dekódován a zobrazen. Mimo vytvoření náhledu snímku ze zdroje videa a případného výpočtu rozdílového snímku umožňuje aplikace v závislosti na zvolené scéně snímky před odesláním dodatečně zpracovat (například použitím libovolného obrazového filtru, převodem snímku do stupňů šedi apod.).

Výsledné video je viditelně kvalitnější, než u kterékoliv jiné testované aplikace. Při ideálním Bluetooth spojení obou zařízení dosahuje přenos snímků rychlosti okolo 20 Hz při zanedbatelném zpoždění obrazu. V reálných situacích, kdy nejsou zajištěny ideální světelné podmínky nebo je Bluetooth signál z jakéhokoliv důvod rušen, klesá rychlost přenosu snímků v závislosti na zvolené metodě pod 15 Hz.

Aplikaci lze využít jako základ jiné aplikace pracující na dálku s libovolným zdrojem videa. Příkladem využití může být například aplikace vzdáleného hledáčku, pomocí kterého lze pořizovat fotografie či videa. Přestože je klientská aplikace cílena na platformu Tizen, bylo by jednoduché ji rozšířit pro potřeby libovolného zařízení s moderním webovým prohlížečem.

Práce mi umožnila dozvědět se mnoho nového o systémech Android a Tizen. Dále jsem měl také možnost se blíže seznámit s interní implementací webových frameworků a lépe

pochopit fungování hardwarové akcelerace klíčových komponent webového jádra.

Další vývoj by se mohl týkat podpory jiných typů hodinek. V případě odlišného operačního systému by však nejspíš bylo nutné aplikaci upravit, neboť je velmi optimalizovaná z pohledu webového jádra na straně klienta a v případě jiného typu aplikace by přenos obrazu nemusel být příliš efektivní. Dalšího vylepšování aplikace by také bylo možné dosáhnout optimalizací velkého množství parametrů přenosu videa, které byly v rámci práce nastaveny experimentálně.

Literatura

[1] Android NDK. [online], [cit. 2015-03-24].
URL https://developer.android.com/tools/sdk/ndk/

[2] FFmpeg4Android. [online], [cit. 2015-03-27].
URL http://ffmpeg4android.netcompss.com/

[3] Tizen Web App Programming. [online], [cit. 2015-03-25].
URL https://developer.tizen.org/dev-guide/2.2.0/

[4] Allen, G.: *Beginning Android 4*. Apress: Distributed by Springer Science Business Media, c2012, ISBN 14-302-3984-0.

[5] Benoit, H.: *Digital Television: Satellite, Cable, Terrestrial, IPTV, Mobile TV in the DVB Framework*. Burlington, Mass.: Focal Press, 2008, ISBN 978-024-0520-810.

[6] Bryan, A.: *Android (Operating System) - Unabridged Guide*. Emereo Pty Limited, 2012, ISBN 978-14-861-9851-1.

[7] Chang, J.: Tizen Webkit For Wearable Devices. In *Tizen Developer Conference*, San Francisco, 2014.
URL http://download.tizen.org/misc/media/conference2014/slides/tdc2014-tizen-webkit.pdf

[8] by Cheng Luo: *Tizen programming*. Chichester: John Wiley, 2014, ISBN 978-111-8809-266.

[9] Elgin, B.: A Google Buys Android for Its Mobile Arsenal. [online], [cit. 2015-01-04].
URL http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal

[10] Frank Ableson, C. K. C. E. O., Robi Sen: *Android in Action*. Manning Publications Co., 2011, ISBN 978-1617290503.

[11] Frantisek, K.: *FFmpeg basics*. Lexington, KY: [Createspace], 2012, ISBN 978-147-9327-836.

[12] Frumusanu, A.: A Closer Look at Android RunTime (ART) in Android L. [online], [cit. 2015-01-07].
URL http://anandtech.com/show/8231

[13] Kaur, P.; Sharma, S.: Google Android a mobile platform: A review. In *Engineering and Computational Sciences (RAECS), 2014 Recent Advances in*, IEEE, 2014, s. 1–5.

[14] Liang, S.: *The Java Native interface*. Reading, Mass.: Addison-Wesley, c1999, ISBN 02-013-2577-2.

[15] Liu, F.: *Android native development kit cookbook*. Birmingham: Packt Publishing, 2013, ISBN 978-1849691505.

[16] Paul Deitel, A. D., Harvey M. Deitel: *iOS 8 for Programmers: An App-Driven Approach with Swift*. Deitel Developer Series, Pearson Education, 2014, ISBN 978-01-339-6541-4.

[17] Prakash, D.: Compile in the Cloud with WP8. [online], 2013, [cit. 2015-03-27].
URL http://blogs.msdn.com/b/msgulfcommunity/archive/2013/03/16/compile-in-the-cloud-with-wp8.aspx

[18] Samsung Electronics Co.: *Samsung Gear Application Programming Guide*. 2014.

[19] Samuel Gibbs, A.: The future of Windows Phone? At the low end, says Alcatel. [online], [cit. 2015-01-06].
URL http://www.theguardian.com/technology/2014/oct/23/the-future-of-windows-phone-at-the-low-end-says-alcatel

[20] Vaughan, D.: *Windows Phone 8*. Indianapolis: Sams Publishing, 2013, ISBN 978-0672336898.

[21] Vávrů Jiří, U. M.: *Programujeme pro Android*. Grada, 2013, ISBN 978-80-247-4863-4.

Příloha A

Obsah CD

- `\src\Experiments\ImageSpeedTestWebGL` – Test rychlosti vykreslování dekódovaného rámce videa pomocí WebGL
- `\src\Experiments\ImageSpeedTest` – Test rychlosti vykreslování dekódovaného rámce videa na <canvas> prvek po jednotlivých pixelech.
- `\src\Experiments\HTMLTestBase64Image` – Test rychlosti vykreslení `Base64/JPEG` kódovaného obrázku, včetně náhodného poškozování obrázku tak, aby se vyloučila možnost znovupoužití obrázku ve vyrovnávacích pamětech webového jádra.
- `\src\Experiments\HTMLTestJpegImage` – Test rychlosti vykreslení obrázku uloženého v souboru na permanentní paměti cílového zařízení.
- `\src\Experiments\FFMPEG_VIDEO_TEST` – (Neúspěšný) experiment posílání dat <video> prvku. Data jsou posílána sítí a pomocí *WebSocket API* přijímána. Detailněji je experiment rozepsán v kapitole 4.1
- `\tex\jpegTest\jpeg.py` – Experiment, který zkoumá degradaci obrazu při jeho měnícím se rozlišení nebo míry `JPEG` komprese. Cílem experimentu je určit, za jakých okolností je pro zachování dané kvality videa měnit rozlišení nebo míru komprese.
- `\tex\jpeg\jpeg.py` – Experiment zkoumající (ne)linearitu růstu velikosti obrazu s klesající `JPEG` kompresí.
- `\src\Experiments\HTMLTestDifferentialImage` – Experiment s rozdílovými snímky (jejich pořizování pomocí *OpenCV* knihovny a následné zobrazení pomocí `WebGL` technologie).
- `\src\Thesis\Thesis` – Android aplikace (server)
- `\src\Thesis\ThesisConsumer` – Tizen aplikace (klient)

Příloha B

Vybrané ukázky kódů

Využití jsmpeg knihovny

```
# server (IP adress: 192.168.1.1)
$ node stream-server.js 123456

# streaming device:
$ ffmpeg -s 320x320 -f dshow -i video="USB Camera" -f mpeg1video -b:v 800k \
-r 30 http://192.168.1.1:8082/123456/320/320/

# client: (IP adress: 192.168.1.2, HTML ommited)
var client = new WebSocket("192.168.1.1:8082");
var player = new jsmpeg(client, {canvas:canvas});
```

Výpis kódu B.1: Přenos obrazu pomocí nástroje *FFmpeg* a knihovny *jsmpeg*.

Přenos videa pomocí klíčových snímků H.264 formátu

```
# server:
$ ffmpeg -s 320x320 -f dshow -i video="USB Camera" -f h264 -vcodec libx264 \
-r 30 -s 320x320 -g 0 -b:v 800k \
http://192.168.1.102:8082/123456/320/240/

# client:
function onKeyFrameReceived(data) {
    $("#target").css("background", "transparent
        url("+data+") top left / 100% 100% no-repeat");
}
```

Výpis kódu B.2: Zasílání klíčových snímků H.264 formátu a jejich dekódování

Akcelerované vykreslení snímku

```
// vertex shader, OpenGL initialization etc.. omitted for clarification
// fragment shader
void main() {
   gl_FragColor = texture2D(image, texCoord);
}
function renderImage(image) {
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
    gl.drawArrays(gl.TRIANGLES, 0, 6);
    ...
}
```

Výpis kódu B.3: Vykreslení obrazu pomocí `WebGL`.

Generování obrázků s rozdílnou mírou komprese

```
import cv2
img = cv2.imread("origin.tif")
for i in range(1,101):
    cv2.imwrite(str(i) + ".jpg", img, [int(cv2.IMWRITE_JPEG_QUALITY), i])
```

Výpis kódu B.4: Generování obrázků s rozdílnou mírou komprese.