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

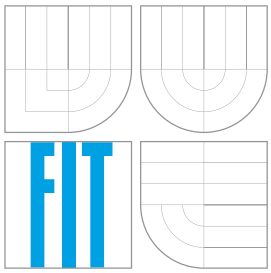
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN PLANER

BRNO 2014

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 2014

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 obrazu z kamery 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, 2014

Mobilní aplikace zpracování obrazu

Prohlášení

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

.....
Jan Planer
21. května 2015

Poděkování

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

© Jan Planer, 2014.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

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

V první části této kapitoly 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 *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 Historie 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), *Windows Mobile*[1] (2000), *Symbian* (1998) a jiné.

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

(a) IBM Simon

(b) Nokia 9000 Communicator

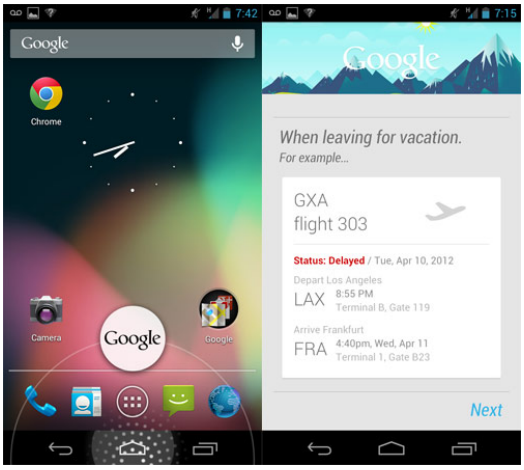
Obrázek 2.1: První chytré telefony. Zdroj: *Wikipedia*

Android

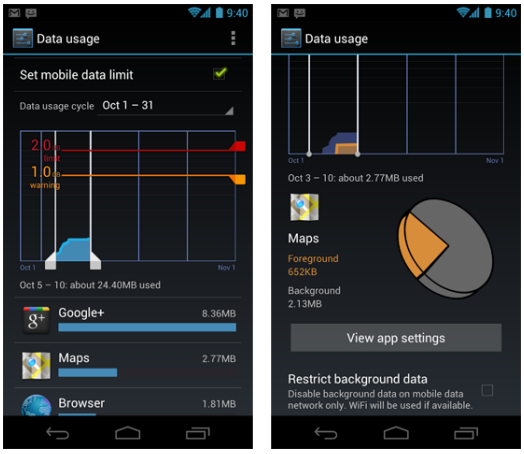
První zmínky o platformě Android se datují do roku 2003, kdy zaměstnanec společnosti Apple Andy Rubin založil malou společnost s názvem Android Inc. Jejím cílem bylo vyvinout systém, který lépe využívá možností mobilního zařízení a lépe bude vyhovovat uživatelským nárokům. v roce 2005 byla Android, Inc. 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. [8]

Dále jsou uvedeny některé významné verze systému: [5]

- **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.

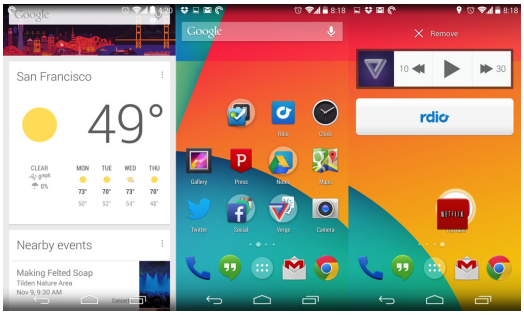


(a) Android Jelly Bean



(b) Android Ice Cream Sandwich

(c) Android Honeycomb



(d) Android Kitkat

Obrázek 2.2: Ukázky jednotlivých verzí systému Android. Zdroj: *Wikipedia*

- **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

Obrázek 2.3: Srovnání vzhledu platforem iOS 6 a iOS 7. [15]

pouze nadále vylepšena integrace sociálních sítí do telefonu, zásadnější změny přinesla v roce 2013 verze 7.0. [15]

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. [15] Srovnání vzhledů obou verzí platformy je vidět na obrázku 2.3.

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. [19]

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) [19]. V budoucnu se spekuluje, že bude platforma Windows Phone zcela sloučena s budoucí verzí desktopového systému Windows 10. [18]

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.

(a) Windows Phone 8.1.

(b) Uživatelské rozhraní systému Windows pro dotykové zařízení.

Obrázek 2.4: Ukázka systému Windows. Zdroj: `microsoft.com`

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. [16]

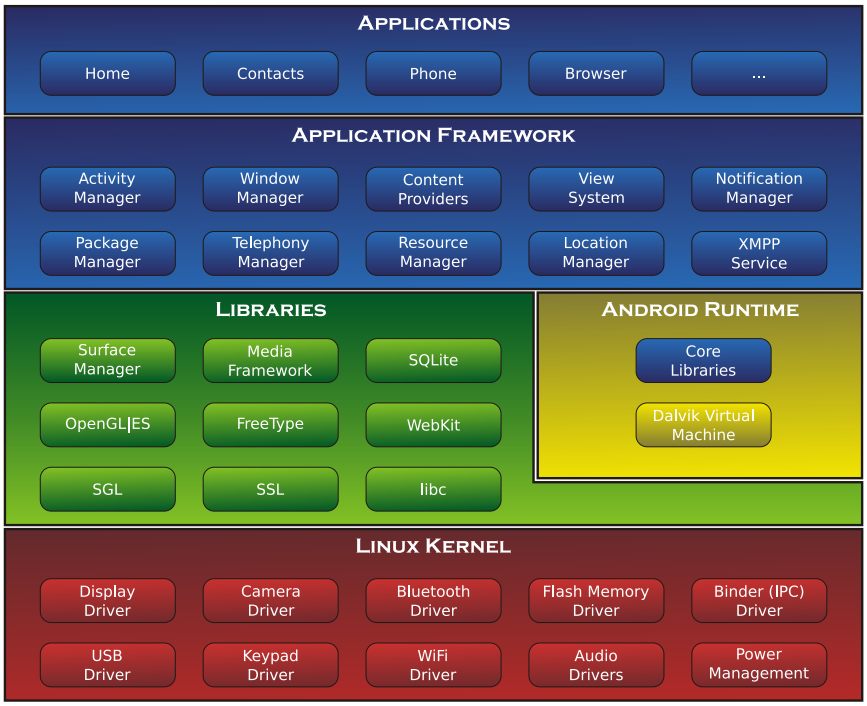
Výsledný kód je mnohem rychlejší, než interpretovaný kód virtuálním strojem. Vzdáleně podobný koncept, který bude diskutovaný v podkapitole 2.2, přináší také novější verze systému Android.

2.2 Architektura systému Android

Architektura celého systému je naznačena na obrázku 2.5. Jak již bylo řečeno v podkapitole 2.1, systém je založený na linuxovém jádře, které odpovídá červené vrstvě v obrázku 2.5. 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: [12]

- **OpenGL ES** – Knihovna určená k vykreslování 3D grafiky.
- **SQLite** – Databázová vrstva.
- **Media Framework** – soubor knihoven určených pro práci s multimédii.



Obrázek 2.5: Architektura platformy Android. Zdroj: *Wikipedia*

- **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. [12]

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 2.5) 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. [20]

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. [11]

2.3 Vývoj Android aplikace

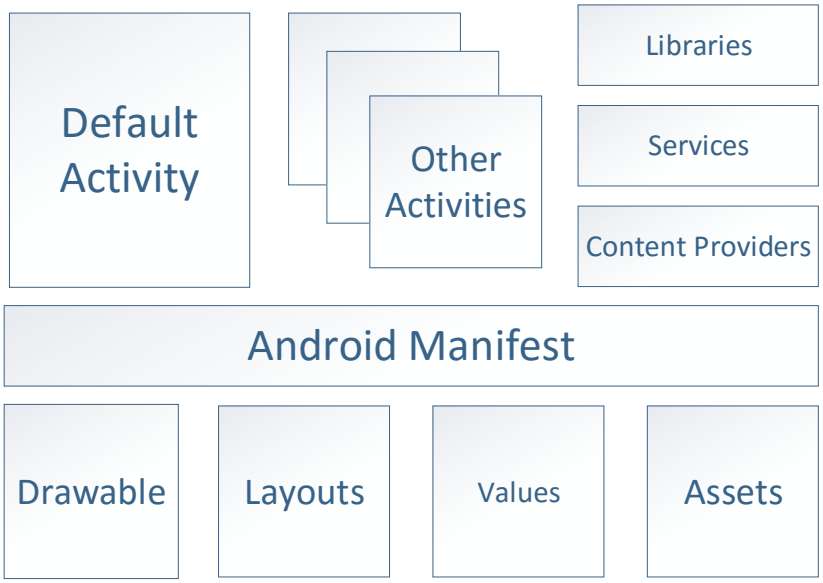
Pro vývoj Android aplikací existuje několik různých vývojových prostředí – například *IntelliJ IDEA*, *Netbeans*, *Eclipse* a jiné. v roce 2013 představila společnost Google vývojové prostředí *Android Studio* (založené na platformě *IntelliJ IDEA*), které se od začátku specializuje na vývoj Android 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.
- **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 2.6 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.



Obrázek 2.6: Logická struktura Android aplikace.

Aktivita

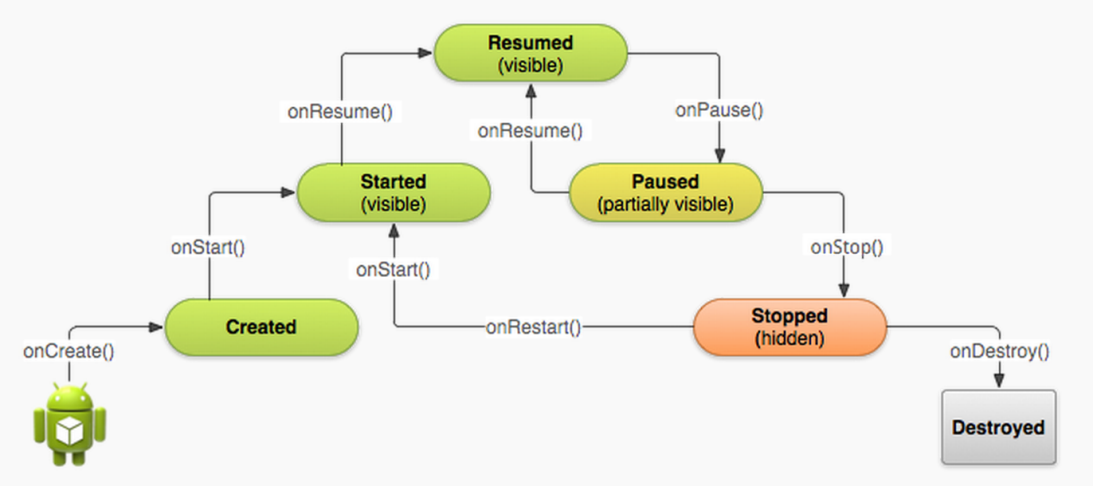
Aktivita (anglicky *Activity*) by měla reprezentovat, jak už její název napovídá, jednoúčelovou obrazovku aplikace. Aplikace samotná se poté může skládat z více těchto aktivit. Po spuštění aplikace je automaticky vytvořena a spuštěna hlavní aktivita a různými uživatelskými akcemi může dojít k přesměrování 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 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 2.7. 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.



Obrázek 2.7: Životní cyklus aktivity. Zdroj: `developer.android.com`

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 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

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

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. [9]

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. [9]

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. [14]

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.

```
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: [13]

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 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

```
// 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: [13]

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 2.8. [13]

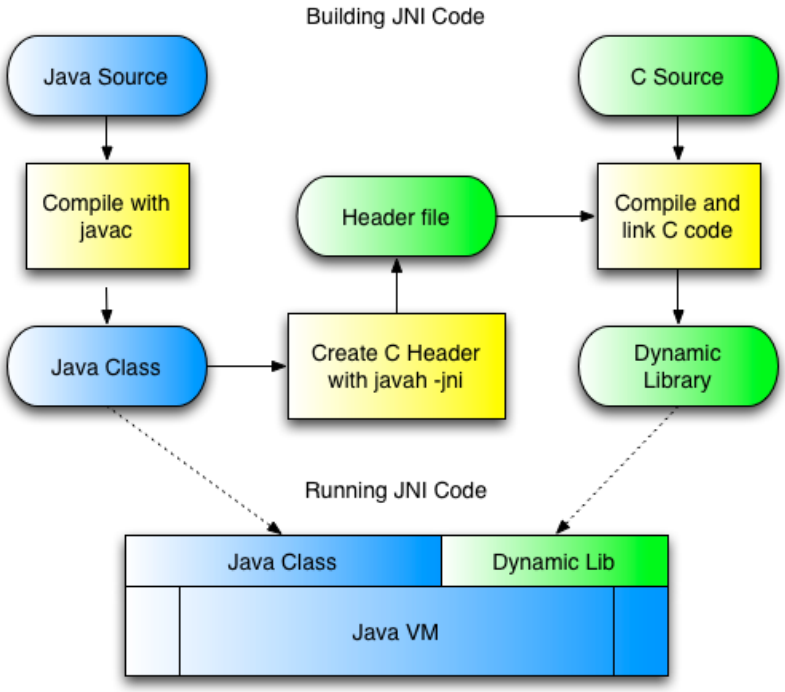
2.5 Tizen

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 2.10. Při vzájemné komunikaci obou zařízení vystupuje smartphone zpravidla jako server a hodinky jako klient. Na straně telefonu musí být nainstalovaná aplikace *Gear Manager*, která slouží ke správě dat v hodinkách. Komunikace mezi oběma stranami probíhá prostřednictvím Bluetooth protokolu. [17]

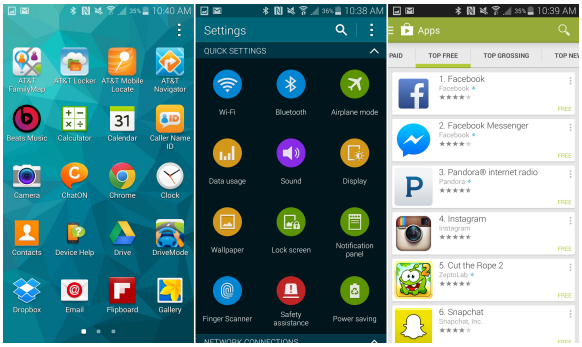
Typy aplikací

V rámci klient-server architektury smartwatch hodinek a hostujícího zařízení lze vytvářet následující typy aplikací: [17]



Obrázek 2.8: Schéma hybridní aplikace. Zdroj: `sbalaji-android.blogspot.cz`

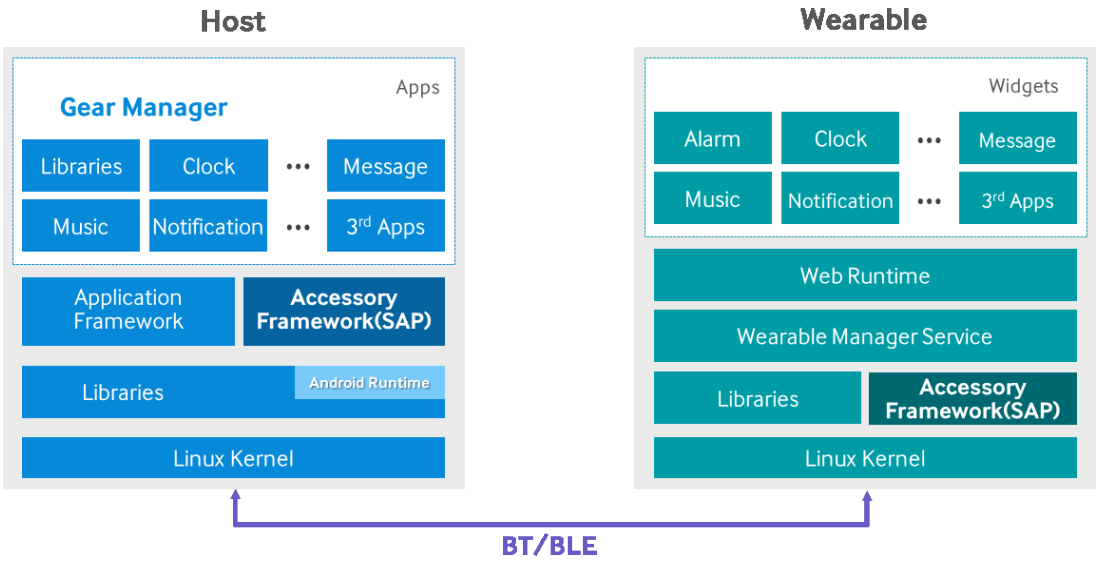
- **Master-Follower aplikace** – Dvojice plnohodnotných aplikací. První běží na hostujícím zařízení, druhá na hodinkách. Tento typ aplikace je vhodný především v případě, kdy obě aplikace mohou fungovat nezávisle na sobě, a při běhu se jedna bez druhé obejde.
- **Integrovaná aplikace** – Jedná se o jedinou 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 ja-



(a) Obrazovka chyrého telefonu se systémem Tizen.

(b) Chytré hodinky se systémem Tizen.

Obrázek 2.9: Ukázka systému Tizen. Zdroj: `samsung.com`



Obrázek 2.10: Klient-server architektura galaxy gear hodinek spárovaných s hostujícím zařízením. Zdroj: [17]

kýmkoliv 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. [17]

2.6 Struktura Tizen aplikace

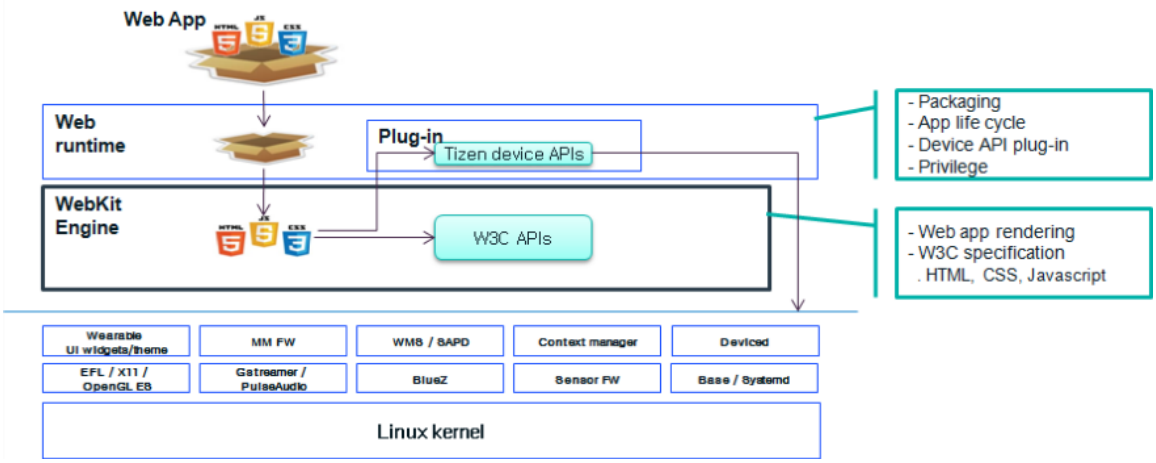
Jak již bylo řečeno, aplikace na hodinky je napsaná v jazyce JavaScript. K definici vzhledu lze použít standardní nástroje jako při vývoji jakékoliv jiné `HTML5` aplikace – tedy kaskádové styly a `HTML5` definice rozložení stránky. Na rozdíl od stolního počítače je však výkon hodinek značně omezený a 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 2.11 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: [7]

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



Obrázek 2.11: Webová aplikace v systému Tizen. Zdroj: [6]

- **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`.

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 hypotetickou aplikaci, ve které bude aplikace vykreslovat animaci analogových hodin, které obsahují hodinovou, minutovou a vteřinovou ručičku. V takové 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: [6]

Jak je vidět v tabulce 2.2, nejlepším řešením 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. Srovnání obou situací je vidět na obrázku 2.12, resp. v tabulce 2.3.

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

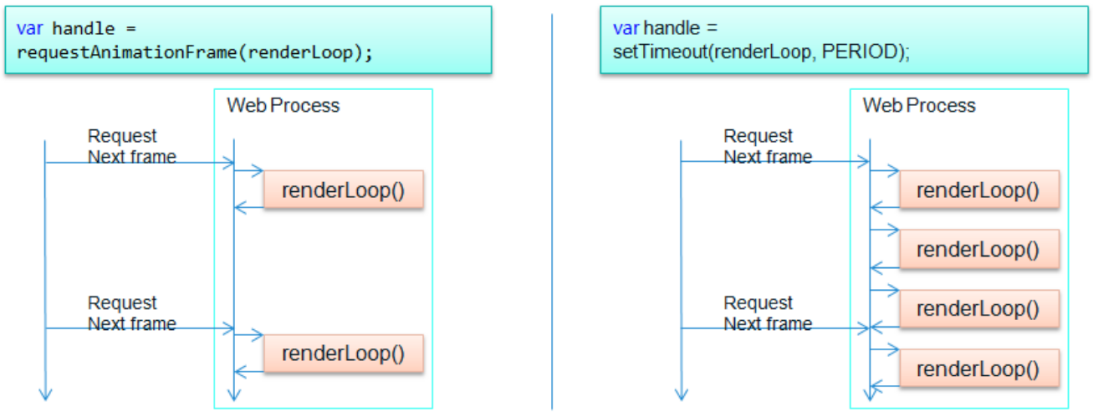
Tabulka 2.3: Srovnání efektivní a naivní vykreslovací smyčky. Zdroj: [6]

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]



Obrázek 2.12: Srovnání využití funkce `requestAnimationFrame` a `setTimeout`. Zdroj: [6]

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 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. [10]

FFmpeg knihovny

Jádrem celého nástroje jsou knihovny sloužící k jednotlivým operacím nad videi či zvukovými záznamy. Níže je uveden jejich stručný seznam: [10]

- **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: [10]

- **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:

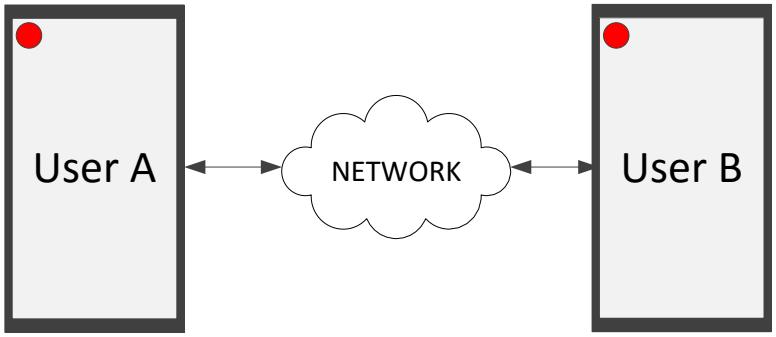
```
$ git clone git@github.com:cine-io/android-ffmpeg-with-rtmp.git
$ cd android-ffmpeg-with-rtmp
$ ./build.sh
```

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.

2.8 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.



Obrázek 2.13: Obecné schéma videohovoru.

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.

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

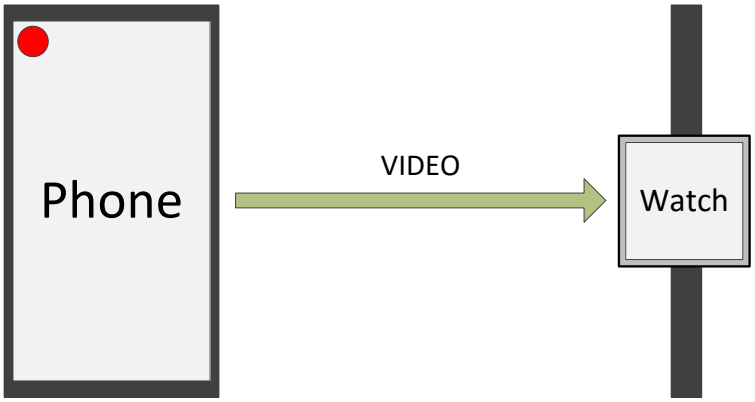
prvního Android zařízení. Mezi tyto aplikace patří například *IP Webcam*, *Home Security IP Cam - Alfred*, *EyeIPCam* a mnoho dalších.

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*.



Obrázek 2.14: Schéma aplikací přenosu obrazu z telefonu na hodinky.

Na obrázku 2.15 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.

vCamera

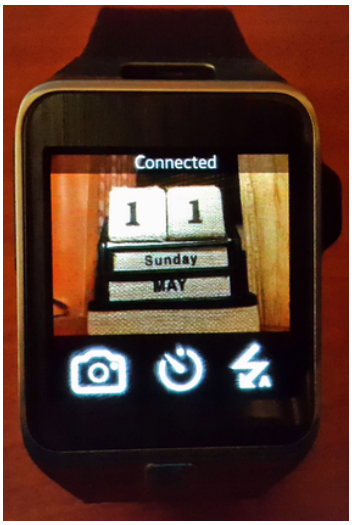
Aplikace umožňuje přenos z fotoaparátu či ze přední kamery mobilu na obrazovku hodinek. Dále umožňuje jednoduché nastavení kamery smartphonu, jako například vypnutí či zapnutí blesku, nastavení krátké prodlevy či focení více snímků zároveň.

vRecord

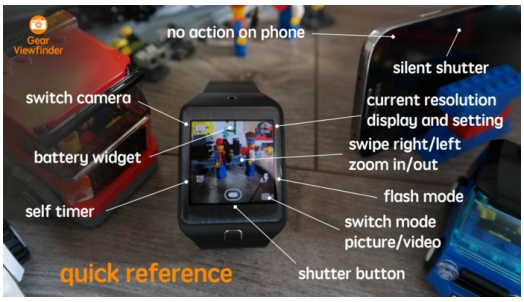
Jedná se o velice podobnou aplikaci, jako v předchozím případě, je však doplněna o rozšiřující funkce, jako například o pořizování videí na vzdáleném zařízení, zaznamenávání zvuku, detekce pohybu, při kterém hodinky upozorní uživatele vibrací apod.

(a) vCamera – úvodní obrazovka

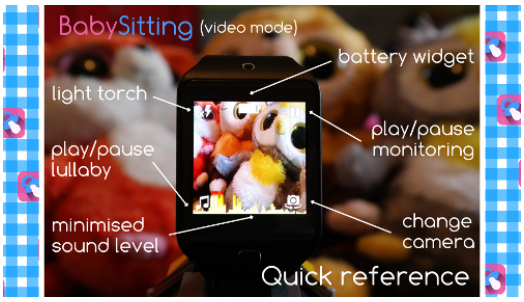
(b) vCamera – přenos obrazu



(c) Gear2cam



(d) Gear ViewFinder



(e) BabySitting

Obrázek 2.15: Ukázky existujících aplikací. Zdroj: `gearapp.challengepost.com`.

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, kteří mohou vzdáleně kontrolovat svého potomka, zatímco spí. 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.

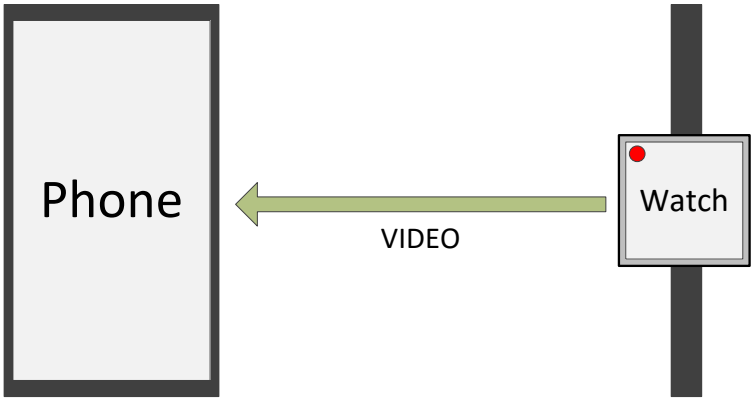
Gear ViewFinder

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.

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.



Obrázek 2.16: Schéma aplikací přenosu obrazu z hodinek na telefon.

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.

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 budou ohodnoceny dosavadní aplikace a jejich použitelnost. Dále budou představeny některé knihovny, které mohou být použity v rámci vývoje aplikace a v závěrečné části kapitoly budou stanoveny cíle, kterých by mělo být v rámci této práce dosaženo.

3.1 Vybrané mobilní aplikace

V kapitole 2.8 byly 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

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

Spydroid-ipcamera

Jednou z takových aplikací je například *Spydroid-ipcamera*, která již byla zmíněna v podkapitole 2.8. 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 pro systém Android

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

Android prostředky

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 2.8 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 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 2.8 byly představeny následující existující aplikace: *vCamera*, *vRecord*, *BabySitting*, *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).

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.

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: Srovnání existujících implementací.

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.

Přenos 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.

3.3 Použitelné technologie

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.

Na straně telefonu se tedy nabízí možnost využít nástroj *FFmpeg* představený v kapitole 2.7, který umožňuje jak měnit parametry přenášeného obraz, tak způsob kódování dat či míru komprese.

Situace je poněkud složitější na straně hodinek, neboť je zde možné využít pouze webové aplikace napsané v JavaScriptu.

Jsmpeg

Jednou z možností, jak dekódovat datový proud, je využití základu knihovny *jsmpeg*, což je JavaScriptová knihovna umožňující s některými omezeními dekódovat `MPEG-1` video formát. Knihovna umožňuje načítat video ze souboru nebo přes *WebSocket* rozhraní.

Výhodou knihovny je především 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.

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 projekt *videoconverter.js*.

Hlavní výhodou využití podobného nástroje by byla 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).

Klíčové snímky H.264 formátu

Další možností přenosu obrazu je zasílání `H.264` videa obsahující pouze klíčové snímky (neboli *I-frame* snímky). V rámci*FFmpeg* nástroje se toho docílí jednoduchým trikem: nastavením `-g` parametru (velikosti skupiny mezi dvěma klíčovými snímky) na hodnotu 0. Tím se zvýší datový tok (v řádu několika stovek procent), ovšem zpracování na klientovi by bylo velmi snadné (a daleko rychlejší, než s využitím *jsmpeg* knihovny).

Prakticky se tento přístup příliš neliší od zasílání jednoho *JPEG* obrázku za druhým. Vhodný by byl především v případě, kdy by zpracování dat v rámci klientské aplikace trvalo příliš dlouho a zároveň by byla přenosová kapacita Bluetooth kanálu nevyužita.

3.4 Specifikace aplikace

V podkapitole 3.3 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 mezi událostí, kterou snímá kamera telefonu a jejím zobrazením 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

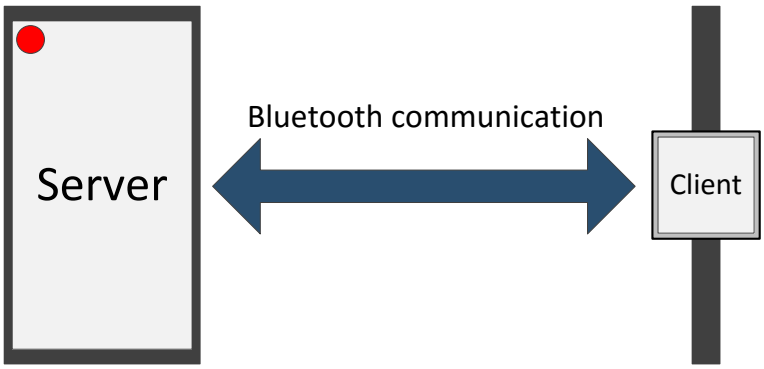
Na straně telefonu se jeví výhodné využít *FFmpeg* nástroj ke kódování jednotlivých snímků pořízených kamerou telefonu. Na straně klienta bude zapotřebí vhodným způsobem jednotlivé snímky dekódovat a zobrazit. Přenos obrazu by měl dosahovat co nejblíže rychlosti 24 snímků za vteřinu při zachování „rozumné“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

Přenos obrazu

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



Obrázek 4.1: Schéma přenosu obrazu.

Další podkapitoly se zabývají návrhem a implementací obou aplikací. Výsledná implementace přenosu obrazu bude na závěr kapitoly otestována a zhodnocena.

4.1 Přenos 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 internetovém prohlížeči.

Jsmpeg

V kapitole 3.3 byla představena JavaScriptová knihovna *jsmpeg*, která umožňuje dekódování obrazu `MPEG-1` videa. Funkčnost a výkonnost knihovny se dá ověřit pomocí skriptu uvedeného v ukázce kódu 4.1.

```
# 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 4.1: Přenos obrazu pomocí nástroje *FFmpeg* a knihovny *jsmpeg*.

V ní je vytvořen jednoduchý server, který pouze přeposílá data přijatá od klienta všem připojeným zařízením. Dále je k serveru v rámci lokální sítě připojeno zařízení s kamerou *USB Camera*, ze které je posílán obraz na server. Poslední zařízení poté ve webovém prohlížeči zobrazuje prostřednictvím serveru video pořízené kamerou.

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 3.3, kde byl popsán způsob vytvoření datového toku obsahující pouze klíčové snímky.

Ukázka kódu 4.2 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).

```
# 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 4.2: Zasílání klíčových snímků H.264 formátu a jejich dekódování

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).

4.2 Omezení technologií a možné způsoby vykreslení obrazu

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.

K maximalizování přenosové rychlosti bude zapotřebí se zaměřit nejprve na nejslabší místo přenosového kanálu. Intuitivně lze usuzovat, že jím bude buď aplikace přímo v hodinkách, nebo přenosový Bluetooth kanál, který nemusí stíhat přenášet potřebné množství dat.

Aplikace v hodinkách

Následující text se bude snažit ukázat teoretický maximální výkon klientské části aplikace v hodinkách. 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.
- **Pomocí <video> elementu** – Další možností je využít již existující implementaci přímo ve webovém jádře a vše tak vlastně ponechat na systému jako takovém. Data přenesená ze smartphonu by pak stačilo vhodným způsobem předat klientovi.
- **Využití <img> elementu** – Zdroj obrázku lze nastavit také jako `Base64` kódovaný `JPEG` obrázek.
- **Akcelerované vykreslení pomocí WebGL** – Poslední možností je dekódovat přenesené rámce obrazu z telefonu a ty poté vykreslit ve `WebGL`.

Přímé kreslení po pixelech

Implementačně nejjednodušším způsobem, jak vykreslit přenesený obraz je, přímé vykreslení přenesených pixelů na <canvas> prvek. V ukázce kódu 4.3 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 4.3: 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 <canvas> prvek. Z toho důvodu tomuto způsobu vykreslení dat nebude věnována další pozornost.

Použití <img> prvku

Rychlost vykreslování pomocí <img> prvku lze jednoduše ověřit například střídavým vykreslováním dvou obrázků:

```
...
var img1 = "/9j/4AAQSkZJRgA ... b34rxyz ... F2/0R/9k=";
var img2 = "/9j/4AAQSkZJRgA ... RGVz3Ph ... j90/6s//Z";

var i = 0;

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

Výpis kódu 4.4: Vykreslení obrazu pomocí <img> elementu.

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.

Využití WebGL

V rámci testování rychlosti vykreslení obrazu pomocí `WebGL` byla vytvořena jednoduchá testovací aplikace (viz ukázka kódu 4.5), 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.

```
// 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 4.5: Vykreslení obrazu pomocí `WebGL`.

Srovnání kreslících metod

Jednotlivé metody jsou přehledně srovnané v tabulce 4.1. Z měření vyplývá, že nejvhodnější metodou by při dostatečné přenosové rychlosti bylo zasílání jednotlivých `JPEG` snímků kódovaných jako `Base64` text.

V případě malé datové propustnosti Bluetooth kanálu by naopak bylo nejvhodnější použít vhodný video formát a dekódované snímky vykreslovat pomocí `WebGL` technologie.

4.3 Návrh systému

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.

	FPS	Poznámky
Přímé kreslení	29	Naivní, v praxi nepoužitelná metoda
Video prvek	dostatečná	Streamování videa není v `HTML5` zatím přímo podporované.[1] Hlavní problém tedy je, jak tato omezení obejít.
Img prvek	79	Velmi jednoduchá a na vykreslování efektivní metoda. `Base64` kódování však zvýší nároky na datový tok
WebGL	44	`WebGL` kreslení by bylo vhodné spojit například s dekódováním nějakého video kodeku přímo na straně hodinek.

[1] Dle `W3C` specifikace je možné v rámci <video> prvku použít libovolný formát videa. Problémem je však zatím chybějící podpora na straně jednotlivých webových prohlížečů.

Tabulka 4.1: Srovnání jednotlivých metod kreslení.

Průběžně je třeba 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.

Návrh přenosového 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.

Na obrázku 4.2 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.

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.

Název	Formát	Popis
Počáteční zpráva	`s:[WxH,[FPS]]`	Zpráva, kterou zasílá klient hned po připojení. Parametry W a H značí rozlišení displeje, FPS maximální přenosovou rychlost
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.

[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.

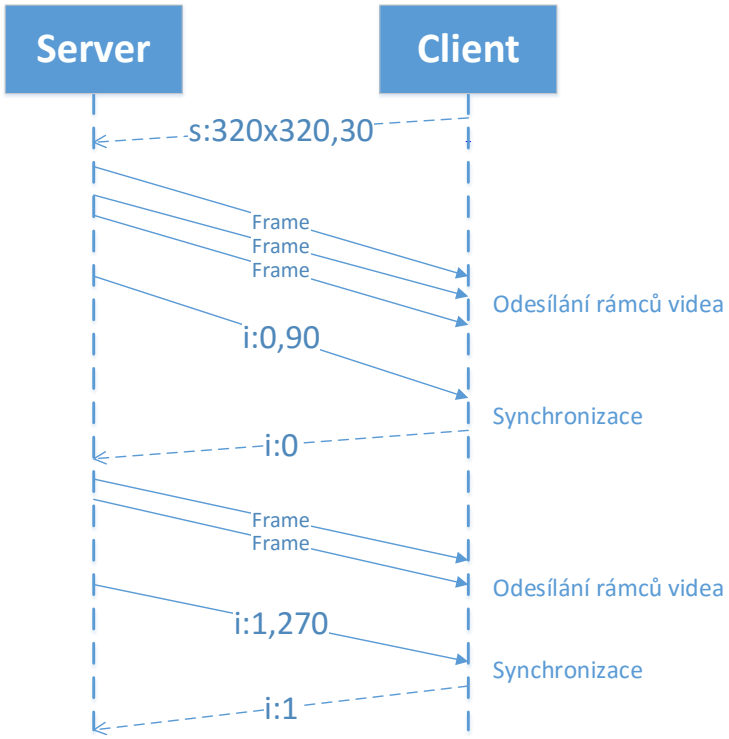
- **Přídavná funkcionalita a uživatelské rozhraní** – Ostatní části aplikace se neliší od běžné webové aplikace a z pohledu návrhu jim tudíž není třeba věnovat zvýšenou pozornost.

Návrh klientské části aplikace je zobrazen na obrázku 4.3. 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 serveru

Server 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. 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 z kamery 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.



Obrázek 4.2: Ukázka komunikačního protokolu mezi klientem a serverem

Schéma serveru je vidět na obrázku 4.4. Systém je navržený především s ohledem na co nejvyšší výkon v rámci dané úlohy. Není problém jej různě modifikovat nebo doplňovat (zde se jako první vybízí vzdálené pořizování fotografií či videí kamerou smartphonu, případně možnost využití přední kamery zařízení).

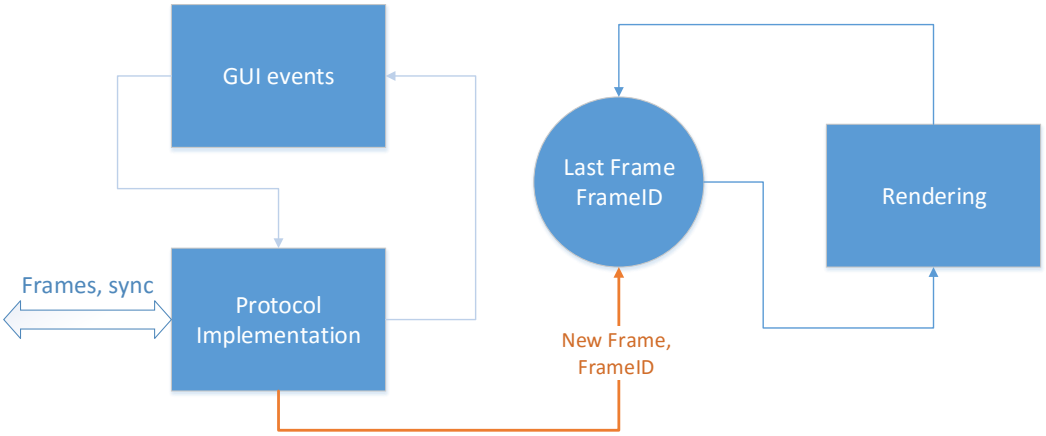
Na druhou stranu by rozsáhlejší změna, jako například požadavek přenášení zvuku, znamenala velký zásah do implementace obou aplikací. Navíc by bylo zapotřebí znovu provést analýzu výkonu obou zařízení v rámci takového nového požadavku, neboť by se jednalo v principu o zcela odlišnou úlohu.

4.4 Implementace

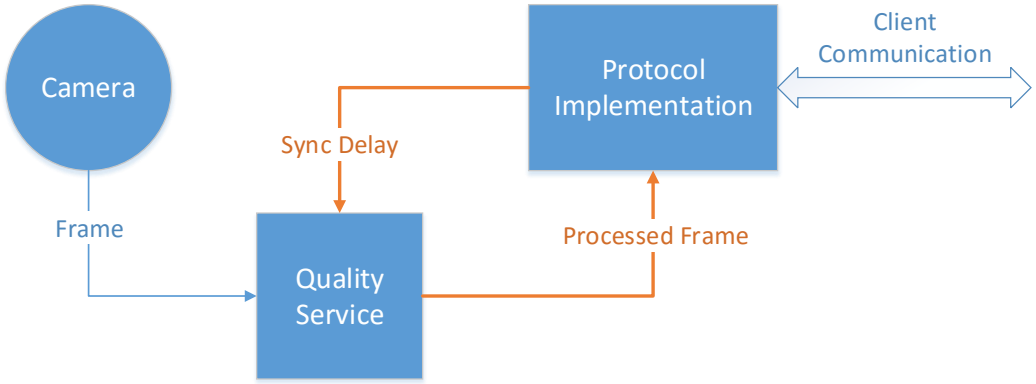
Implementačně nejnáročnější části serverové aplikace je pravděpodobně řízení kvality videa, kterému bude v rámci této podkapitoly věnována zvýšená pozornost.

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.



Obrázek 4.3: Návrh klienta.



Obrázek 4.4: Návrh serveru.

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.

Z pohledu řízení datového toku je vhodné kvalitu videa vyjádřit jediným parametrem. Předpokládejme tedy, že existuje funkce `getParameters(t)`, která obdrží kvalitu videa t nabývající hodnot z intervalu od 0.0 do 1.0 a na jejím základě vybere vhodnou kombinaci výše zmíněných parametrů. Pro jednoduchost si lze představit například lineární interpolaci mezi nejhorší a nejlepší hodnotou daného parametru.

Název parametru	Min	Max	Poznámky
`FPS`	-	24	Viz 3.4
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: Experimentální nastavení parametrů kvality videa.

Modul řídící kvalitu průběžně dostává naměřenou hodnotu aktuálního zpoždění videa a dle toho určuje vhodnou kvalitu videa vyjádřenou například 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.

Při malých hodnotách zpoždění je vhodné kvalitu videa 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 4.5.

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 dle vztahu:

$$Q_{t+1} = Q_t + Q_d(x) \tag{4.1}$$

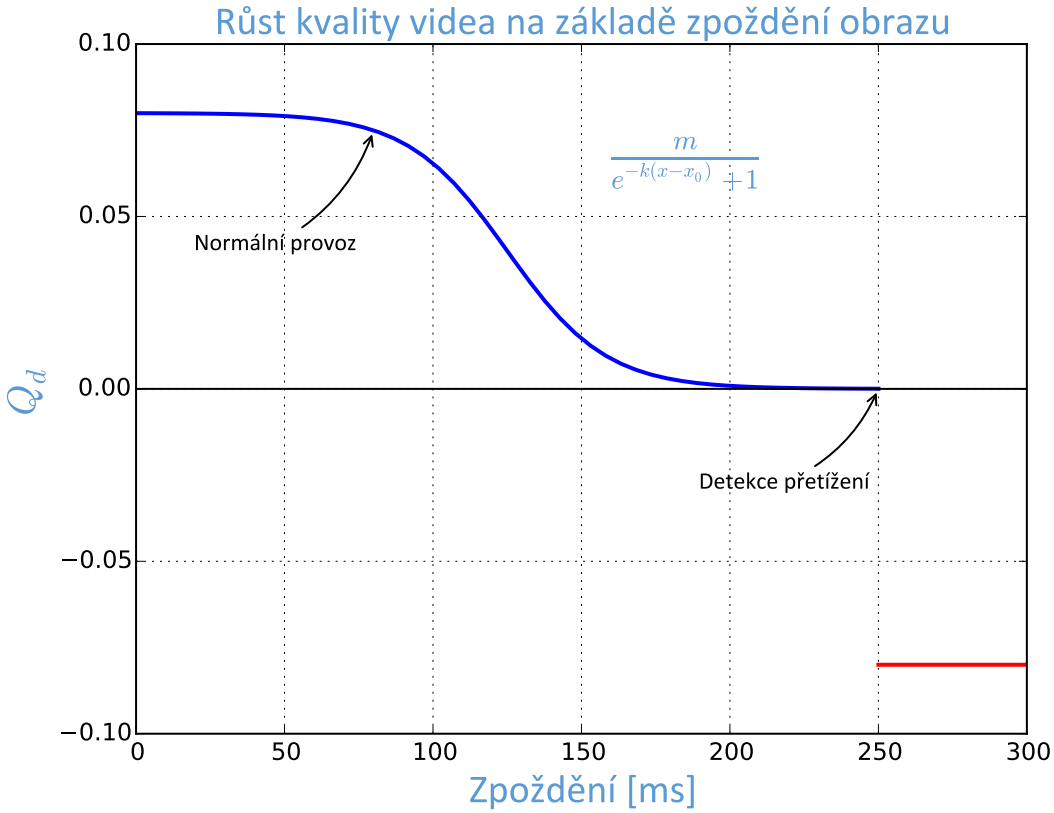
Úprava snímků pořízených kamerou

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.

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

Vytvoření náhledu snímku

Snímek pořízený kamerou může být zmenšen a v případě nesouhlasných poměrů stran oříznut a vycentrován. Způsob zmenšení snímku je ukázán na obrázku 4.6. Originální obrázek je zde reprezentovaný oranžovou barvou. Snímky orientované na výšku jsou zmenšeny dle



Obrázek 4.5: Řízení kvality videa na základě aktuálního zpoždění.

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

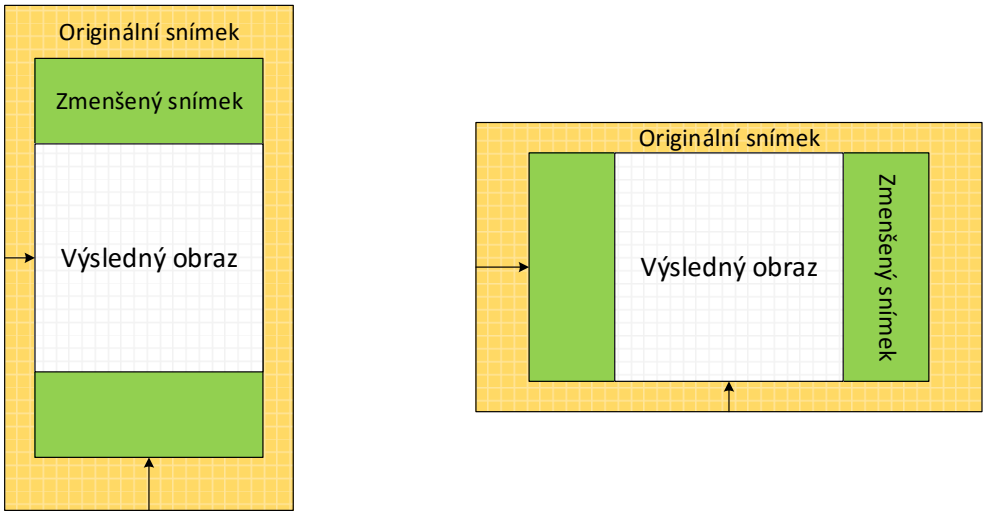
Hlavní nevýhodou takového způsobu zpracování snímků je kromě zvýšené výpočetní náročnosti především převzorkování snímku, díky čemuž může obecně docházet ke zmenšení jeho kvality. Výhodou je poté skutečnost, že uživatel vidí na hodinkách v jednom rozměru celý rozsah kamery telefonu.

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.

4.5 Vliv kvality obrazu na jeho velikost

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).



(a) Obrázek orientovaný na výšku

(b) Obrázek orientovaný na šířku

Obrázek 4.6: Zpracování snímku pořízeného kamerou smartphonu.

Názorné srovnání je uvedeno na obrázku 4.7, 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 4.7 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 100 různých obrázků s klesající mírou komprese:

V obrázku 4.8 jsou zobrazeny některé vybrané vygenerované obrázky. U obrazů s vyso-

(a) Lena 196×196: 65.6 KB

(b) Lena 128×128: 35.4 KB

(c) Lena 64×64: 16.5 KB

Obrázek 4.7: Vliv rozlišení obrazu na jeho velikost.

```
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 4.6: Generování obrázků s rozdílnou mírou komprese.

(a) Lena 100: 224 KB (b) Lena 25: 15.4 KB (c) Lena 1: 5.59 KB

Obrázek 4.8: Vliv míry komprese obrazu na jeho kvalitu.

kým kompresním poměrem dochází k viditelným artefaktům, které však velmi rychle mizí s klesajícím kompresním poměrem.

Velikosti jednotlivých obrazů jsou zaneseny do grafu na obrázku 4.9. 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% minimálního kompresního poměru.

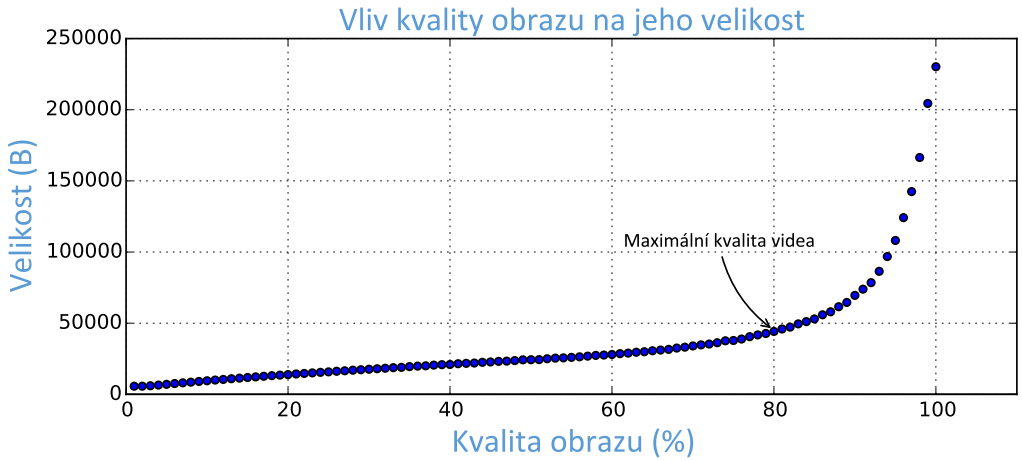
Výběr kvality obrazu

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.

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:

$$PSNR = 10 \log_{10} \frac{255^2}{\frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} (v(i,j) - w(i,j))^2} \tag{4.2}$$

Kde v a w jsou porovnávané obrazy a $M \times N$ jejich rozměry. Pomocí (zkráceného) skriptu 4.7 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



Obrázek 4.9: Graf závislosti velikosti obrazu na jeho kompresním poměru.

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 4.10.

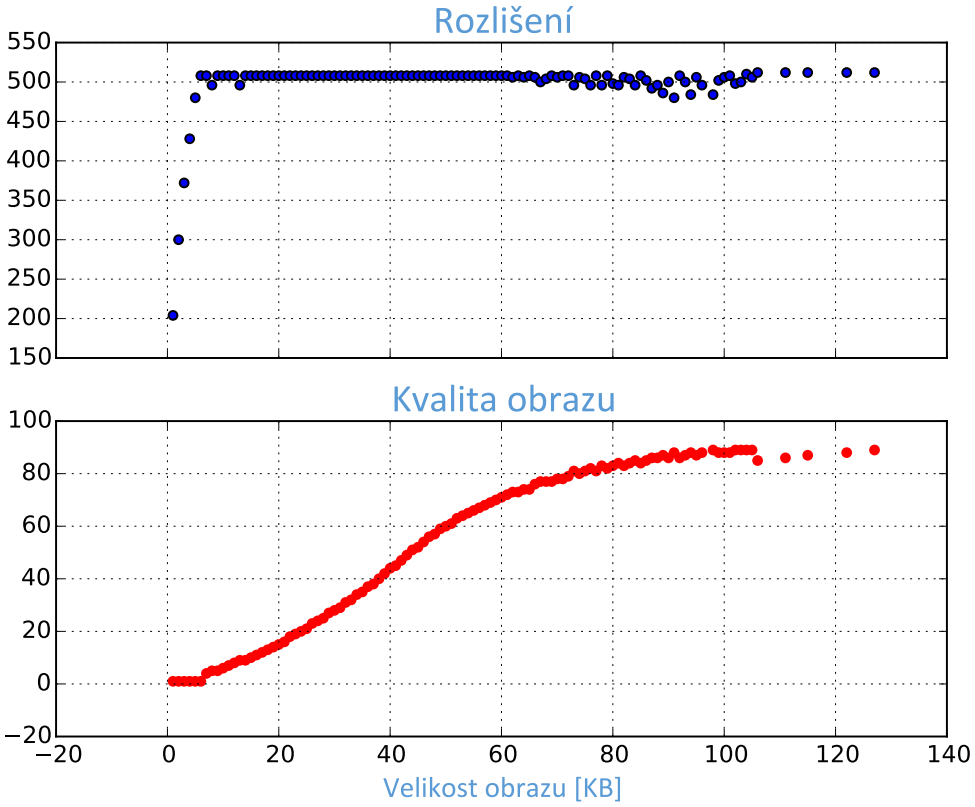
```
import cv2
...
data = initData()
for res,q in allCombinations:
    thumb = cv2.resize(origin, (res, res))
    fileSize = measureSize(thumb, q)
    data[fileSize / 1024].append([thumb, fileSize, q, res])

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

Výpis kódu 4.7: Měření kvality obrázků s různým rozlišením a mírou komprese.

Na obrázku 4.10 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.



Obrázek 4.10: Výběr vhodné kombinace kvality obrazu a jeho rozlišení při dané velikosti obrazu.

4.6 Výsledné nastavení přenosu videa

V předešlé podkapitole bylo ukázáno, jaký vliv má velikost obrazu na jeho kvalitu. Dále byl také v podkapitole 4.3 navržen způsob řízení kvality videa. V rámci takto navrženého systému existuje velké množství parametrů, které lze nejrůznějšími způsoby optimalizovat. Jednotlivé parametry jsou uvedeny níže:

- `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.
- `QUALITY_STEEPNESS` – Parametr určující tvar křivky na obrázku 4.5.

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`	1000 ms
`MAX_DELAY`	140 ms
`MAX_QUALITY_INCREMENT`	0.14
`QUALITY_STEEPNESS`	0.1

Tabulka 4.4: Výsledné experimentální nastavení parametrů.

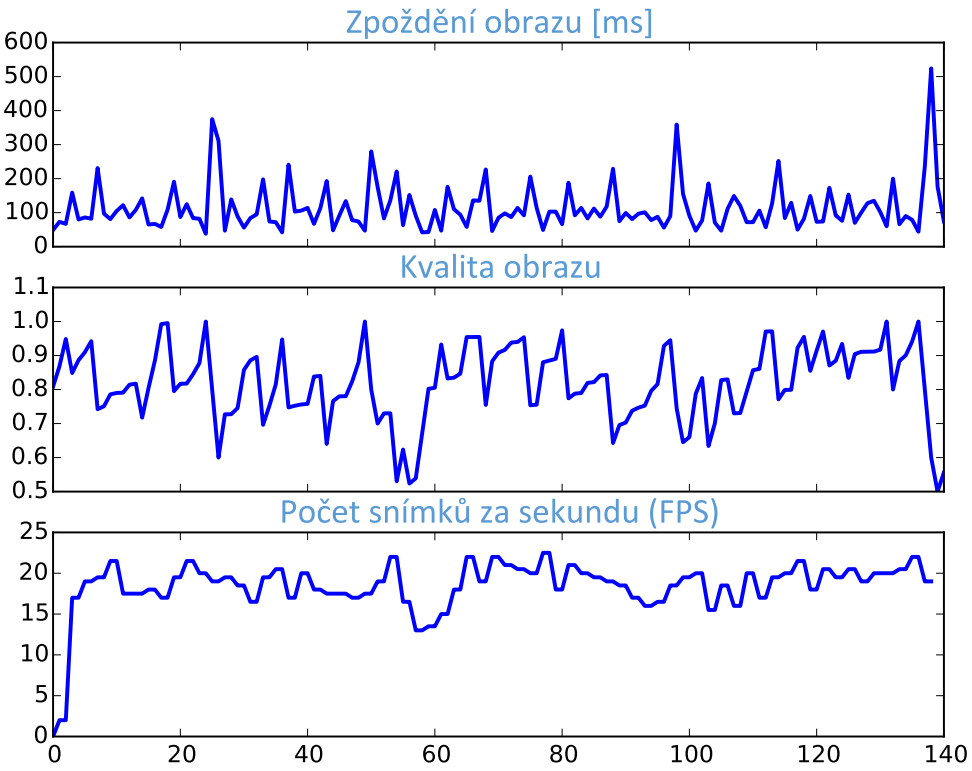
4.7 Měření vybraných parametrů videa

O kvalitě výsledného videa zcela jistě svědčí množství vykreslených snímků za vteřinu, kvalita přenesených snímků a zpoždění videa. Zatímco zpoždění vida 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.

Měření jednotlivých parametrů probíhalo po dobu 140 sekund a to vždy se zasláním synchronizační zprávy (respektive s obdržením odpovědi na danou zprávu). Výsledek měření je znázorněn na grafu v obrázku 4.11.

Jak je na obrázku 4.11 vidět, obnovovací frekvence snímků na displeji hodinek dosahuje v nejlepších případech 22 snímků za vteřinu. Další zvyšování množství přenášení snímků nemá smysl, neboť displej na straně hodinek má obnovovací frekvenci 24 Hz, a tudíž by začalo docházet k častému zahazování některých snímků. Kvalita obrazu se pohybuje někde okolo hodnoty 0.8, což značí v tomto případě obrázky s plným rozlišením displeje hodinek zkomprimované zhruba 50% kompresním poměrem.

Největším problémem jsou náhodně zpožděné synchronizační zprávy, které občas způsobí krátkodobý zbytečný propad kvality videa. Náhodná zpoždění pravděpodobně vznikají při synchronizaci odesílaných či přijímaných zpráv – ať už na straně serveru či na straně klienta. K minimalizaci těchto jevů by bylo zapotřebí vymyslet způsob prioritní obsluhy



Obrázek 4.11: Měření kvality videa při běžném provozu.

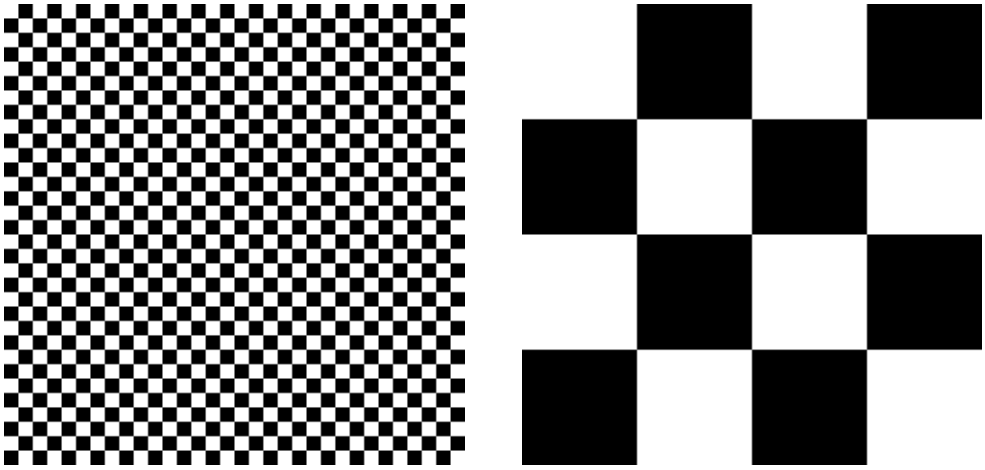
synchronizačních zpráv, což by velmi pravděpodobně byl problém na straně klienta, kde není umožněno nízkoúrovňové programování.

Vliv vnějšího prostředí na kvalitu přenosu

Měření na obrázku 4.11 probíhalo za ideálních podmínek, kdy byla vzdálenost mezi oběma zařízeními minimální a dále byly v testovací místnosti dobré světelné podmínky.

V reálné situaci může být kvalita videa kromě vzdáleností obou zařízení ovlivněna také světelnými podmínkami nebo přenášeným obrazem samotným. Při špatných světelných podmínkách vystavuje kamera telefonu jednotlivé snímky mnohem pomaleji. Rychlé vystavování snímků lze sice vynutit, pak jsou ale snímky videa nedostatečně prosvětlené. Samotný obraz má poté díky `JPEG` kompresnímu algoritmu velký vliv na velikost výsledného `JPEG` snímku. Obzvláště se to projevuje u obrazů s vysokými frekvencemi. Velikost dvou rozdílných snímků se tak pří stejném rozlišení a stejném kompresním poměru může zásadně lišit.

Na obrázku 4.12 je srovnání dvou obrazů se šachovnicovitým vzorem. Liší se pouze ve velikosti jednoho políčka, tím pádem poměr černých a bílých pixelů je v obou případech shodný. Na obrázku vlevo se však vyskytuje mnohem více různých obrazových frekvencí (konkrétně je k zakódování obrazu zapotřebí 550 nenulových `DCT` koeficientů diskrétní kosinové transformace, zatímco na obrázku vpravo jich je zapotřebí pouze 340. Ve scénách, ve kterých se vyskytuje velké množství obrazových frekvencí je díky tomu přenos videa zpo-



(a) Velikost obrazu: 3.13 KB (b) Velikost obrazu: 1.31 KB

Obrázek 4.12: Srovnání účinnosti `JPEG` komprese u obrazů s rozdílným množstvím obrazových frekvencí.

malován, neboť jak dokazuje obrázek 4.12, nároky na datový přenos mohou být v takových případech i násobně vyšší.

Při přenosu obrazu za horších podmínek se také prodlužuje průměrná doba odpovědi na synchronizační zprávy. Při extrémních podmínkách je tedy dobré pracovat s hodnotou parametru `MAX_DELAY` někde v rozmezí 180 až 300 milisekund. V opačném případě aplikace chybně vyhodnocuje zpožďování videa a dochází k velkému poklesu kvality videa, případně až k úplnému zastavení přenosu.

Měření na obrázku 4.13 ukazuje, jak se mění kvalita videa v závislosti na vzdálenosti obou zařízení od sebe. Test byl prováděn na volném prostranství, kde mezi telefonem a hodinkami byla přímá viditelnost. Každý vzorek odpovídá průměru naměřených zobrazených snímků během 16 sekund. Za povšimnutí stojí poměrně nápadité a náhlé poklesy kvality videa, které rozhodně neodpovídají postupnému poklesu kvality tak, jak by člověk intuitivně předpokládal. Nejpatrnější z nich je vidět na vzdálenosti odpovídající 10 metrů.

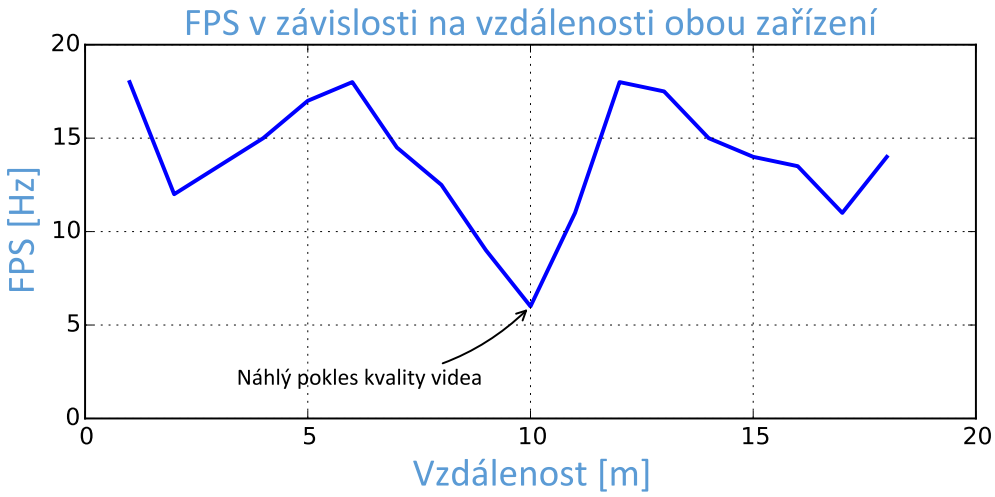
Podobné zakolísání kvality videa vykazuje i přechod do vedlejší místnosti, případně za jinou pevnou překážku. Příčinu takového chování se mi však nepodařilo jednoznačně určit. Pravděpodobně však bude nějakým způsobem souviset s Bluetooth modulem a výkonem vysílaného signálu ať už na straně hodinek nebo na straně telefonu.

4.8 Shrnutí naměřených hodnot

Za dobrých vnějších podmínek (ideálně v dobře osvětlených interiérech, případně v prostorech, kde se nevyskytuje příliš mnoho drobných obrazových elementů) dosahuje přenos rychlosti až 20 snímků za sekundu.

Níže jsou shrnuty případy popsané v předchozích odstavcích, ve kterých je přenos videa pomalejší:

1. **Horší světelné podmínky** – Kamera telefonu začíná při horších světelných podmínkách vystavovat snímky rychlostí zhruba 15 Hz.



Obrázek 4.13: Závislost kvality videa na vzdálenosti obou zařízení.

2. **Velký počet obrazových frekvencí** – Ve scéně, ve které se vyskytuje velký počet obrazových frekvencí (typicky se jedná o exteriéry) roste násobně velikost přenášených snímků a úměrně tomu klesá kvalita výsledného videa.

3. **Dočasné propady kvality** – Při postupném zhoršování vnějších podmínek dochází k dočasnému propadu kvality videa. Tento jev popisuje měření na obrázku 4.13.

Problémy v bodech 1 a 3 jsou dány hardwarem cílových zařízení a pravděpodobně jsou z vývojářského hlediska neřešitelné. V případě posledního problému by nejspíš bylo možné najít kombinaci všech parametrů systému, kde by video působilo za daných podmínek plynuleji, případně by možná šlo snímky před samotnou kompresí vhodně předzpracovat.

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 byla vybrána vhodná metoda k vykreslení přeneseného snímku a té 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.

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 pod 15 Hz. Zasílané snímky mají plné rozlišení displeje hodinek (320×320) pixelů a jsou zkomprimované pomocí `JPEG` formátu zhruba 50% kompresním poměrem.

Aplikaci lze brát jako základ libovolné jiné aplikace pracující na dálku s kamerou jakéhokoliv Android zařízení. Příkladem by mohl být vzdálený hledáček, 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.

V dalším vývoji by bylo možné prozkoumat především oblast moderních kodeků, případně přímo implementovat některý protokol určený pro přenos videa přes Bluetooth kanál. Je ale také docela možné, že taková úloha je nad rámec výkonu výpočetně slabých hodinek. Další rozšíření by se mohlo týkat podpory jiných typů hodinek. V případě jiného 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] Bryan, A.: *Android (Operating System) - Unabridged Guide*. Emereo Pty Limited, 2012, ISBN 978-14-861-9851-1.

[6] 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>

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

[8] 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>

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

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

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

[12] 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.

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

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

[15] 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.

[16] 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>

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

[18] 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>

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

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