VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZACHYTÁVÁNÍ A KOMPRESE VIDEA NA VESTAVĚNÝCH ZAŘÍZENÍCH

VIDEO CAPTURE AND COMPRESSION ON EMBEDDED DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. OTO DUŠEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. DAVID BAŘINA, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Dušek Oto, Bc.**
Program: Informační technologie Obor: Počítačové a vestavěné systémy
Název: **Zachytávání a komprese videa na vestavěných zařízeních**
Video Capture and Compression on Embedded Devices
Kategorie: Zpracování obrazu
Zadání:

1. Seznamte se s problematikou zachytávání snímků z průmyslových kamer podporující standard GigE Vision. Nastudujte algoritmy kódování videozáznamů, porovnejte jejich vlastnosti a možnosti distribuce komprimovaného streamu.
2. Navrhněte systém, který bude komprimovat snímky z kamery a ty bude dále posílat klientům pomocí vhodného protokolu. Dbejte na jednoduchou rozšiřitelnost o různé typy kamer a kompresních algoritmů.
3. Vyberte vyhovující hardwarovou platformu, navržený systém na ní implementujte.
4. Vyhodnoťte jeho výkonnost. Zaměřte škálovatelnost řešení.

Literatura:

- Dokumentace multimediálního frameworku FFmpeg.
- Dokumentace multimediálního frameworku GStreamer.

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2 a rozpracovaný bod 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Bařina David, Ing., Ph.D.**
Konzultant: Novotný Václav, Ing., YSoft
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 3. června 2020
Datum schválení: 1. listopadu 2019

Abstrakt

Průmyslové kamery se často využívají pro aplikaci strojového učení. Tyto kamery však produkují velký datový tok a při zpracování videa dále od kamery je potřeba ho redukovat. Problém se snaží vyřešit tato práce. Zaobírá se návrhem systému vhodného pro zachytávání videa z průmyslových kamer, jeho komprimaci a další distribuci aplikacím, které ho zpracovávají. Kromě samotného návrhu popisuje i vhodné technologie použitelné pro vývoj takové multimediální aplikace. V případě tohoto systému byly například využity multimediální frameworky FFmpeg a GStreamer, jež značně zjednodušují implementaci jeho nejdůležitějších částí.

Abstract

Industrial cameras are often used in conjunction with the application of machine learning. However, these cameras produce large bitrate and it needs to be reduced when processing video further away from the camera. This thesis tries to solve the problem by design of system suitable for grabbing video from industrial cameras, its compression and distribution to machine learning application. The thesis describes technologies applicable to the development of the multimedia application. For example frameworks FFmpeg and GStreamer were used implementation of the system.

Klíčová slova

zachytávání, komprese, video, průmyslová kamera, vestavěný počítač, FFmpeg, GStreamer, RockPro64, H.264, RTP, RTSP, PylonSDK, Basler AG

Keywords

video acquisition, compression, video, industrial camera, embedded computer, FFmpeg, GStreamer, RockPro64, H.264, RTP, RTSP, PylonSDK, Basler AG

Citace

DUŠEK, Oto. *Zachytávání a komprese videa na vestavěných zařízeních.* Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. David Bařina, Ph.D.

Zachytávání a komprese videa na vestavěných zařízeních

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Davida Bařiny, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....

Oto Dušek

3. června 2020

Obsah

Kapitola 1

Úvod

V dnešní době se často aplikace nasazují do datacenter díky jednodušší správě a dalším benefitům. Dle statistik Eurostatu se jen v Evropské unii od roku 2014 zvedl podíl použití cloudu v podnikové sféře z 19 % na 26 % [4]. Jedním z nasazovaných typů jsou aplikace zpracovávající obraz z kamer strojovým učením. Tyto kamery často produkují velký datový tok, někdy i jednotky gigabitů za sekundu [2]. Při nutnosti posílat video z více kamer do datacenter by byla neúnosně zatěžována síťová infrastruktura, proto se musí video komprimovat. Tím se zmenší datový tok a zajistí se tak snazší a méně náročná další distribuce. Navíc se při ukládání záznamu na disk ušetří místo.

Ve společnosti YSoft Corporation a.s. existuje systém pro automatické testování vestavěných zařízeních s dotykovou obrazovkou (např. tiskáren). Ten se skládá z robotického ramena zakončeného stylusem pro ovládání testovaného zařízení, z kamery zajišťující zpětnou vazbu od testovaného objektu a softwaru implementující celou logiku testování. Systém zpracovává data z kamery a na základě toho reaguje pomocí robotického ramena, případně detekuje zda se zařízení chová podle očekávání. Ovšem použitá kamera produkuje neúnosně velký datový tok pro zasílání na větší vzdálenosti. Proto je cílem této práce navrhnout systém, který umožní posílat data z kamer při nižším datovém toku do datacentra, kde se bude video zpracovávat. Kromě toho umožní připojení ke streamu tzv. operátorům, kteří potřebují zkontrolovat aktuální stav testovaného zařízení pomocí kamery. Systém tedy zachytí snímky z kamery, zmenší jejich velikost a vhodným způsobem video distribuuje dále. Pro komprimaci byl zvolen formát H.264, který je standardem v této oblasti. Navržený systém má být převážně používán pro strojové vidění používané pro rozpoznávání elementů na obrazovkách vestavěných zařízení, toto rozpoznávání je implementováno v jazyce C#. Navíc se zpracování provádí v reálném čase, s ohledem na všechny tyto okolnosti byl systém navrhnut.

Práce je rozdělena na 3 hlavní části. První dvě kapitoly zahrnují rozbor technologií použitelných pro implementaci systému. Kapitola 2 seznamuje se základními pojmy v okruhu multimédií. Popisuje průmyslové kamery a standardy s nimi spojené, algoritmy pro kompresi videa, protokoly pro distribuci videa a vestavěný počítač vhodný pro zpracování videa. V kapitole 3 se rozebírají dva frameworky pro snazší manipulaci a zpracování multimédií. Framework GStreamer je použitý pro zachytávání a komprimaci snímků, zatímco FFmpeg pro přijímání již komprimovaného videa. Kapitola 4 se zaměřuje na samotný návrh a implementaci systému. Poslední část, kapitola 5 pak popisuje způsob a výsledky testování hotového systému.

Kapitola 2

Nahrávání a distribuce videa

Tato kapitola vysvětluje všechny nutné části pro implementaci systému, který zachytává komprimuje a dále distribuuje video. Popisuje tedy často využívané barevné modely, standard pro komunikaci s průmyslovými kamerami přes Ethernet a jednu z kamer podporující tento standard. Dále v sekci 2.3 je rozebrán standard H.264 použitý pro komprimaci, hlavně jeho základy a informace nutné pro jeho konfiguraci. Nakonec vestavěný počítač využitelný pro komprimaci a protokoly pro distribuci videa.

2.1 Význam barevných modelů

Barevné modely [3] zajišťují popis barvy pomocí několika komponent udávající její vlastnosti. Existuje mnoho modelů, každý vhodný pro různé aplikace. Výběr správného typu závisí na požadavcích a použití. Některé modely lépe vystihují to, jak člověk vnímá barvy, jiné jsou vhodně například pro zobrazovací techniku. Při vyjadřování barvy se často využívá dvou pojmů. Jedním z nich je jas, ten udává množství světla vyzařovaného ze zdroje pod určitým směrem. Druhý pojem definuje barevnou složku světla, tzv. barevnost. Ta vyjadřuje vjem světla jako takový při dopadu na sítnici oka člověka.

Příkladem může být RGB model [6], ten využívá 3 komponent pro popis barvy, tj. červené, zelené a modré. Každá komponenta má přiřazené číslo vyjadřující množství složky obsažené ve výsledné barvě. V některých případech se využívá i barevného modelu RGBA, ten navíc obsahuje takzvaný alfa kanál, který definuje průhlednost barvy. Toto se hodí například pro vytváření vodoznaků, či při míchání více obrázků do jednoho. Existuje mnoho formátů jak v tomto modelu vyjádřit barvu pixelů, tj. jak se rozloží složky RGB v paměti. Mezi nejčastěji používané patří RGB24. U něj písmena vyjadřují pořadí komponent v paměti, tedy postupně v paměti náleží červená, zelená a nakonec modrá složka. Číslo 24 určuje počet bitů použitých pro reprezentaci všech barev v pixelu. Pokud není jinak řečeno složky jsou rovnoměrně rozděleny, tedy v tomto případě má každá po 8 bitech. Mezi další používané formáty lze zařadit BGR24, RGBA32 a mnoho dalších.

K dalším rozšířeným modelům patří model YUV ($Y'C_bC_r$) [6], původně využívaný pro vysílání a příjem analogové televize. Později se vytvořil nový model $Y'C_bC_r$ pro digitální vysílání. Ten je principiálně stejný, avšak vůči modelu YUV disponuje jiným měřítkem a rozsahem. Oba modely se skládají ze tří komponent, kde Y vyjadřuje jasnost (angl. luminance) a U (C_r – modrá složka) a V (C_r – červená složka) reprezentuje sytost barvy (angl. chrominance). Díky tomu, že oddělíme od sebe informace o jasu a barvě můžeme jednoduše určit velikost paměti, kterou pro jednotlivé vlastnosti přiřadíme. Lidské oko vnímá daleko

citlivěji změny jasu než změny barvy. Toho můžeme s výhodou využít u formátu YUV a pro komponentu Y (jas) vyhradit více paměti než na sytost barev.

Této operaci se říká tzv. podvzorkovávání. Konkrétně u modelu YUV se úmyslně podvzorkují složky U a V, tedy pro sytost barev vyhradíme méně paměti než na jas. Většinou se u vyjadřování vzorkování využívá notace J:a:b případně u formátu YUV YUVJab, kde J je nejčastěji 4 a reprezentuje délku řádku v bloku (referenční horizontální vzorkování, a i b jsou vztaženy k této hodnotě), a značí počet vzorků (C_r a C_r) v prvním řádků a b počet vzorků (C_r a C_r) v druhém řádku. K často využívaným poměrům patří 4:2:0 (YUV422), sytost barvy se zde podvzorkují horizontálně i vertikálně na polovinu. Existuje mnoho dalších poměrů jako 4:2:2, 4:4:4, 4:1:1.

2.2 Průmyslové kamery

Průmyslové kamery se využívají často pro specifické aplikace s různými požadavky a nároky. Proto se využívá několika standardů, které splňují širší možnosti interakce a další užitečné vlastnosti. Jedním z nich je GigE Vision standard [1] byl vytvořen z důvodu potřeby sjednocení rozhraní pro komunikaci s kamerami a softwarem napříč výrobci. Jak název napovídá, přenos dat probíhá přes Ethernet, což je sada technologií využitelná pro komunikaci v sítích. GigE Vision specifikace definuje aplikační vrstvu v TCP/IP modelu, implementuje zde protokol pro ovládání kamer GVCP (GigE Vision Control Protocol) a pro streamování dat GVSP (GigE Vision Streaming Protocol).

GigE Vision specifikace definuje 3 typy kanálů, pro různé druhy zasílaných dat. Prvním typem je ovládací kanál, pomocí kterého lze posílat příkazy do kamery (zápis a čtení z registru). Pro obrácený směr komunikace slouží kanál zpráv, přes který proudí asynchronní zprávy do aplikace připojené ke kameře. Tímto může kamera informovat o zmáčknutí hardwarové spouště na kameře nebo o jiných událostech vyvolaných v kameře. Poslední typ tzv. stream kanál umožňuje nastavit komunikaci z kamery do zařízení a naopak, ale zároveň kanál může využívat pouze jeden směr. Tento kanál slouží převážně pro zasílání obrazových dat v různých formátech. Jednotlivé kanály tedy můžeme vytvářet nebo rušit pomocí GVCP. Pomocí GVCP lze i různě nastavovat nebo odchytávat události z kamery. Kdežto GVSP protokol je používán pouze při komunikaci přes stream kanál a popisuje, jakým způsobem lze zasílat obrazová data, obrazové informace a jiné volitelné informace.

Kamera Basler acA 1920-40gc

Německá společnost Basler AG[1] se díky širokému portfóliu kamer a jejich doplňků vyšplhala mezi vedoucí výrobce. Dodává kamery pro medicínské využití, sledování dopravy a do dalších průmyslových oborů. Basler AG nabízí jak klasické kamery, tak řádkové, které snímají obraz pouze v jednom řádku. Dále společnost vyvinula i 3D kamery, které jsou schopny snímat i hloubku v obraze. Všechny druhy kamer se vyrábějí s různými typy rozhraní jako USB 3.0, GigE Vision nebo Camera Link (definuje i vlastní fyzickou vrtvu specifickou pro přenos obrazových dat). Jednou z kamer vytvořených společností Basler je model acA 1920-40gc, zobrazený na obrázku 2.1. S cenovou 419 eur se řadí mezi ty levnější modely. Kamera komunikuje skrze Ethernet přes protokol definovaný specifikací GigE Vision. Díky tomu lze s kamerou interagovat standardní cestou a lze k tomu využít i hotové knihovny podporující komunikaci s kamerami napříč výrobci. Další možností je i použití nástrojů přímo

[1] https://www.baslerweb.com/en/

Obrázek 2.1: Kamera acA 1920-40gc od společnosti Basler

od výrobce. Všechny průmyslové kamery od společnosti Basler disponující rozhraním GigE Vision lze ovládat přes knihovnu PylonSDK. Ta umožňuje nastavovat kamery a zachytávat snímky v programovacích jazycích C, C++ i C#.

Kamera, využívající CMOS snímač od Sony IMX249, dokáže barevně nahrávat video s rozlišením až 1920 × 1200 při maximálně 42 snímcích za sekundu. Podporuje vysílání obrazových dat ve formátech Bayer (formát surových dat ze snímače), YUV422 (popsané v sekci 2.1) a v černobílém formátu. Také má globální závěrku (snímá se celý obraz najednou a ne jednotlivě po řádcích) a umožňuje připojit hardwarovou spoušť nebo připojit osvětlení a ovládat ho skrze programové rozhraní kamery.

2.3 Kodeky pro kompresi videa

V minulém století se začala velmi rozvíjet nejdříve analogová a poté i digitální televize [7]. Tím začala revoluce ve vývoji technologií spojených se zpracováním a distribucí videa. Bylo potřeba zavést nové technologie, které budou umožňovat sledovat televizi (a jiné video produkce) s čím dál vyšším rozlišením a lepší kvalitou. Ještě důraznější vliv ovšem na vývoj těchto technologií měl internet, umožnující daleko rozsáhlejší možnosti v oblasti multimédií. Jednou z nejdůležitější potřebou v oblasti distribuce a uchovávání videí se stalo komprimování. To umožňuje při stejných přenosových rychlostech přenést video kvalitnější nebo na disku uchovávat daleko více videozáznamů. To znamená, že při stejných nákladech na pořízenou infrastrukturu lze přenášet a ukládat kvalitnější videa.

Komprese dat je proces, který má za cíl co nevíce redukovat počet bitů nutných pro uložení nebo posílaní daných dat [5]. Toto snížení lze dosáhnout buď snížením datové redundance nebo jejím úplným odstraněním. Tato redundance označuje nadbytečné informace, které lze odstranit ideálně bez vlivu na způsob reprezentace dat. Pokud tedy snížíme redundanci, zmenšíme i velikost potřebného datového toku. Podle způsobu odstraňování redundance se algoritmy dělí na ztrátové a bezztrátové. Pokud zkomprimujeme data bezztrátovou komprimací jsme schopni zrekonstruovat originální data přesně tak jak byly zakódovány. V případě komprese videa či audia by se ovšem nedosahovalo tak dobrých kompresních poměrů, proto se využívá ztrátové komprimace. Ta redukuje takzvanou perceptivní redundanci. Díky tomu, že jsou data určena pro člověka, a ten nemá dokonalé sluchové a zrakové

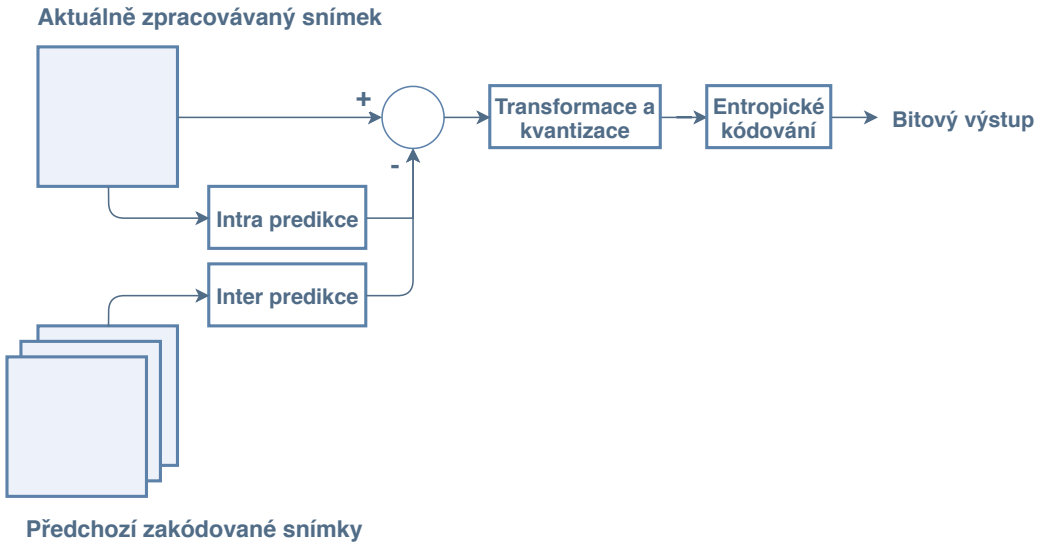
orgány, tak můžeme odstranit z dat takové vjemy, které člověk nezaznamená. U lidí se udává, že slyší zvuky od 20 Hz do 20 kHz, tedy můžeme odstranit zvuky mimo tento rozsah. Po dekomprimaci tedy již nezískáme už zpátky originální data, ale maximálně jim podobné podle nastavení míry komprese. Obdobně bude proces probíhat u videa.

V případě potřeby přenášet video komprimovaných způsobem musíme mít na odesílací straně tzv. enkodér (pokud není video již komprimované), který zpracovává vstupní video stream. Ten ve videu nachází redundance, odstraňuje je a tím zmenší objem přenášených dat. Pokud chceme komprimované video zobrazit na monitoru nebo s ním jinak pracovat potřebujeme tzv. dekodér, který přijímá datový tok a zpátky rekonstruuje originální snímky (ovšem většinou s menší kvalitou). Ty už můžeme zobrazovat např. na monitoru. Často jsou tyto enkódovací a dekódovací jednotky implementovány v hardwaru pro rychlejší zpracování. Softwarová realizace video komprimačních algoritmů bývá často výpočetně náročná a tím pádem i hůře využitelná z důvodu velkých latencí či absence podpory vyšších rozlišení. Obecně se prostředky pro komprimaci dat, ať už jsou implementovány v hardwaru nebo softwaru, nazývají kodeky. Pro vysvětlení co vlastně znamená slovo kodek se musíme podívat na původ tohoto slova v angličtině. Tento název pochází z anglického slova codec, což je zkratka dvou slov enCOder (enkodér) a DECoder (dekodér) nebo COmpression (komprese) a DECompression (dekomprese). Kodek tedy označuje nástroje nebo knihovny, které obsahují jak prostředky pro komprimaci videa tak i pro jeho dekomprimaci.

Před vysvětlením samotného principu komprimace musí být vysvětleno jak vzniká takové video. Video se skládá z po sobě jdoucích jednotlivých snímcích, které jsou rozděleny na stejně velké bloky tzv. pixely. Tento pixel má pak přiřazenou konkrétní barvu a jas. Tímto vzniká 2D pole pixelů popisující obraz pořízený v jednom čase. Toto pole může být naplněno pomocí nějakého senzoru (např. CMOS). Ta obsahuje 2D mřížku na světlo citlivých buněk, které se vystaví světlu po krátký okamžik a tím se promítnou barvy z reálného světa na matici. Video neobsahuje však pouze jeden snímek, ale celou řadu v čase po sobě a nějakým způsobem na sebe navazují. Většinou jsou tyto snímky rozprostřeny rovnoměrně tedy, že jsou pořizovány v pravidelných časových intervalech (běžně desítky snímků za sekundu). Dále v rámci snímků spolu pixely většinou vzájemně souvisí. To znamená, že kolem jednoho pixelu budou pixely stejné nebo podobné barvy. Kromě toho i ve většině případů snímky jdoucí po sobě mají společnou informaci, tedy ve dvou snímkách ve stejné nebo blízké pozici budou mít pixely skoro stejné barvy. Těchto vlastností lze s výhodou využívat při kompresi videa. Například kompenzovat pohyb objektu mezi snímky a tím redukovat datový tok nebo odhadovat barvy pixelu podle jeho sousedů v rámci jednoho snímku.

Ztrátová komprese videa H.264

H.264 [7] je standard popisující způsob dekomprese videa, který vznikl už v roce 2003. Často bývá označován i jako MPEG-4 AVC (Advanced Video Coding), byl totiž vyvinut skupinou MPEG (Moving Picture Experts Group) a VCEG (Video Coding Experts Group). Na obrázku 2.2 je uveden základ principu enkódování snímků tohoto kodeku. Snímek se zpracovává po blocích od velikosti 4×4 pixelů až do maximální velikosti 16×16 pixelů. Nejprve se provede predikce aktuálně zpracovávaného bloku, následně se snímek transformuje a zahodí nadbytečné detaily v obrazu. Nakonec se výsledek zakóduje do vhodné binární podoby, která bude zabírat co nejméně paměti. Dekodér kodeku pracuje přesně inverzně k funkci enkodéru a tak zrekonstruuje originální snímek se sníženou kvalitou. Detailnější popis je uveden v následujícím textu, objasňuje však pouze základní princip fungování.



Obrázek 2.2: Proces enkódování u kodeku H.264

Tento standard využívá dvou typů predikce tzv. intra a inter [7]. Intra znamená, že pro predikci využívá již zakódované pixely ze stejného snímku, naproti tomu při inter se extrapolují hodnoty pixelů na základě předchozích zakódovaných snímků. Důvod proč se využívají pouze zakódované pixely nebo snímky je prostý, dekodér musí mít k dispozici stejné data pro predikci jako enkodér a tím tak mohl zrekonstruovat původní data. To je znázorněno v obrázku 2.2, kde se dekódují snímky z výstupu a ukládají se do paměti pro další použití. Po predikci se vytvoří tzv. reziduum, takže skutečnou hodnotu odečteme od predikce. V podstatě je to chyba predikce. Pokud se tedy podaří predikovat 100% přesně, tak vznikne nulové reziduum. Obecně platí čím menší je reziduum, tím efektivněji ho můžeme zakódovat, protože tím méně nese informace. Tedy s lepší predikcí dosáhneme lepší míry komprese. Pro dosažení optimální míry komprese a výpočetní náročnosti se při predikci snímek rozdělí na stejně velké bloky. Typicky 16 $\times$ 16, ale i menší podle nastavení a míry detailů, které obraz obsahuje. U predikce v rámci jednoho snímku (intra) se prochází obraz po blocích a analyzují se sousední zakódované pixely typicky nad nebo nalevo od bloku. Podle toho odhadne jak by mohly vypadat pixely v aktuálním bloku. Pro odhad napříč snímky se používá inter predikce, která prochází obraz též po blocích a hledá v okolních (zakódovaných) snímcích podobný region. Po něm pátrá v konkrétním snímku pouze do určité vzdálenosti od aktuálního bloku kvůli omezení náročnosti výpočtu. Z jeho pozice určí vektor určující směr pohybu bloku tzv. odhad pohybu. Predikce se tedy odvodí od podoby nalezeného regionu. Při kompenzaci pohybu se ovšem musí spolu s komprimovaným videm zaslat i odhad pohybu, aby dekodér mohl rekonstruovat data.

Dalším krokem je transformace predikovaného snímku. Tato část slouží pro dekorelaci dat. Zjednodušeně řečeno se jedná o převod do frekvenční domény, tedy popíšeme snímek množstvím koeficientů udávajících velikost obsažených frekvencí. Pokud tedy obrázek obsahuje hodně detailů nebo se barvy pixelů často mění, tak snímek obecně obsahuje větší frekvence. Naopak pokud je obrázek monolitický, tak se skládá z nulové frekvence. Po transformaci, která převedla obraz do vhodnějšího formátu, se z obrazu zahodí informace (typicky vysoké frekvence obrazu), to zajistí menší velikost obrazu. Tato operace se nazývá kvantizace. Tou lze dosáhnout vysokou míru komprese za cenu nižší kvality obrazu. Avšak

často se volí takové hodnoty kvantizace, že člověk nepostřehne změnu. Čím vyšší míra kvantizace, tím více ze snímku zahazujeme dat. Díky tomuto zahazování si však enkodér musí uložit již pozměněné snímky, aby je mohl použít pro predikci. Zajistí se tak, že dekodér má k dispozici stejná data, a tak může provést identickou predikci a úspěšně rekonstruovat snímek. U enkodéru tedy na výstup kvantizace aplikujeme inverzní operaci a zrekonstruujeme pozměněný snímek, který může být dále použit pro predikci (viz obrázek 2.2).

Při nastavování parametrů enkódování je důležité znát kontext použití a velikost dostupného přenosového pásma nebo úložiště. Pokud chceme efektivněji komprimovat video musíme očekávat, že čas potřebný ke komprimaci snímku bude vyšší. Naopak když je potřeba živě streamovat video musíme správně vybalancovat parametry tak, abychom zajistili přípustné zpoždění, nepřekročili maximální přenosovou rychlost a přitom zajistili co nejlepší kvalitu obrazu. Jedním z nejdůležitějších parametrů je tzv. GOP (angl. group of pictures) neboli skupina obrázků. Ta definuje posloupnost typů snímků, které po sobě následují.

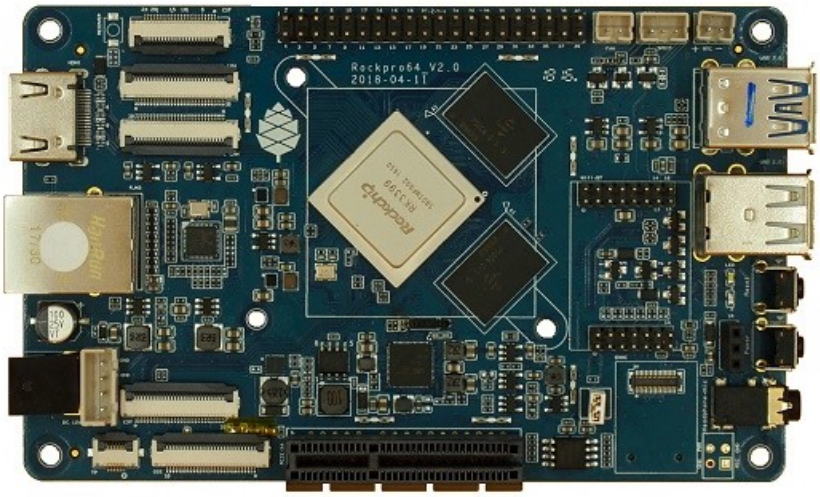
Rozlišují se celkem tři typy snímků podle typu použité predikce: I (intra) snímek, P snímek a B snímek. I snímek označuje, že v snímku byla použitá pouze intra predikce, tedy odhad pouze podle pixelů ve stejném snímku. Takový snímek musí být ve skupině pouze jednou na začátku. P snímek je odvozen od předchozích zakódovaných snímků, které byly pořízeny v čase před aktuálním snímkem. Naproti tomu B snímek používá pro predikci jak snímky v čase předchozím tak budoucím. Tím se může ještě zvýšit latence, protože před vlastním dekódováním B snímků se musí prvně dekódovat v čase předchozí snímky, ale i budoucí ze kterých je snímek odvozen. Všechny zmíněné typy snímků se pak skládají do určité posloupnosti nazývané jako skupina. Každá skupina musí začínat I snímkem, ten nezávisí na okolních, takže máme okamžitě k dispozici celý obraz. Poté následují další typy snímků odvozených mimo jiné z tohoto I snímku. Příklad takové sekvence může být IPBP. Nejprve tedy obsahuje nezávislý I snímek, poté P snímek odvozený od prvního (obecně od předchozího snímku v sekvenci), dále B snímek, který může být predikován od jakéhokoli snímku ve skupině a nakonec P snímek. Taková skupina se pak skládá do neustále se opakující sekvence (IPBPIPBP...) a díky tomu už vznikne kontinuální video. Tím, že video se skládá z opakujících se skupin a I snímek může být obsažen pouze jednou, tak vlastně velikost skupiny určuje za jak dlouho se je schopné video zotavit z chyb. Pokud se tedy ztratí paket obsahující video data potřebná pro odvození dalších snímků musí se počkat na začátek nové skupiny. Obdobně velikost skupiny určuje interval po jaké době je možné zobrazit video streamované ze serveru, protože musíme při připojení počkat na první I snímek. Kromě toho na velikosti zavisí i míra komprese. Pokud máme větší skupinu můžeme dosáhnout větší úspory dat, protože se více využije inter predikce (může se predikovat z více snímků) a také se nemusí tak často posílat celý I snímek. Tak se dokáže velmi významně zvýšit kompresní poměr u videa.

Většina kodeků umožňuje nastavení typů enkódování v závislosti na kvalitě nebo velikosti produkovaného datového toku. První z módů, na který lze enkodér nastavit se nazývá mód konstantního datového toku (CBR – Constant Bitrate). Ten obsahuje zpětnou vazbu měřící bitovou rychlost na výstupu a na základě toho upravuje kvantizační konstanty tak, aby dosáhl požadované velikosti datového toku. Tento přístup je vhodný především pokud máme omezenou šířku pásma. Díky omezení se datový tok v tomto pásmu spíše podaří přenést. Naproti tomu mód proměnného datového toku (VBR – Variable Bitrate) nastaví konstantní kvantizační konstanty a tím zaručuje stejnou kvalitu všech snímků. S tím, že se může významně měnit velikost datového toku a nelze ho s jistotou předvídat. To může být v řadě aplikací problém, protože máme omezenou přenosovou rychlost. Naopak při kom-

primaci videa do souborů je tento přístup efektivnější a všechny scény ve videu budou mít porovnatelnou kvalitu.

2.4 Vestavěný počítač RockPro64

Počítač RockPro64[2] patří do kategorie tzv. SBC počítačů (Single Board Computer). To znamená, že se skládá z jedné desky plošných spojů, na které se nachází všechny potřebné komponenty pro běh. Typicky tyto počítače obsahují minimálně volatilní paměti RAM, nevolatilní paměť, procesor, napájecí obvody, napájecí konektory a jiné konektory pro komunikaci s okolním světem. Jak je vidět na obrázku 2.3, RockPro64 byl osazen celou řadou konektorů jako např. PCIe 4x, Ethernetu, HDMI, USB. RockPro64 patří k těm výkonnějším



Obrázek 2.3: RockPro64 počítač osazený procesorem Rockchip RK3399

SBC počítačům. Nachází se v něm totiž celkem šestijádrový procesor RK3399[3] vyvinutý společností Rockchip. Skládá se z čtyřjádrového ARM Cortex-A53 a dvoujádrového ARM Cortex-A72 procesoru. Tento čip ovšem neobsahuje jen procesor, je to tzv. SOC (System On Chip). To znamená, že nemusíme na desku s procesorem přidávat ještě další komponenty pro základní běh počítače, ale můžeme využít ty integrované přímo na čipu. Díky tomu lze bez dalších nadbytečných součástek k procesoru připojit řadu komunikačních rozhraní. Pro ukládání nevolatilních dat můžeme připojit eMMC paměti nebo SD karty. Pokud potřebujeme připojit zobrazovací jednotku (např. monitor), můžeme s ním komunikovat pomocí HDMI, DisplayPort či pomocí dalších méně rozšířených rozhraních. Kromě toho procesor RK3399 obsahuje USB kontrolér 2.0 i 3.0, gigabitovou síťovou kartu, rozhraní PCIe a spoustu dalších standardních rozhraní. Procesor dále disponuje integrovanou grafickou kartou ARM Mali-T860MP4 využitelnou pro výpočty nebo vykreslování. Dále také rozsáhlými možnostmi zpracovávání videa pomocí různých hardwarově implementovaných dekodérů a enkodérů. Dekódovat dokáže řadu běžných formátů, mezi ně patří MPEG-1, MPEG-2, MPEG-4, H.263, H.264, H.265, VC-1, VP9, VP8, MVC. Pro enkódování videa už není tak

[2] https://www.pine64.org/rockpro64/
[3] http://rockchip.wikidot.com/rk3399

široce vybaven. Komprimovat videa dokáže do tří formátů, a to MVC, H.264 a VP8. Formát MVC (rozšíření kodeku H.264) slouží pro enkódování streamu z více kamer, které snímají jeden objekt z více úhlů. Dokáže tedy komprimovat video daleko efektivněji. Na snímcích bude stejný objekt z různých úhlů, takže na nich bude značné množství redundantních informací, které lze odstranit. Dalším podporovaným formátem je H.264, o něm pojednává předchozí sekce 2.3. Velkou výhodou při využívání těchto hardwarových komponent při vývoji nové multimediální aplikace je existence knihovny MPP (Media Process Platform), ta obsahuje veškeré ovladače pro enkodéry a dekodéry. Dále společnost Rockchip implementovala plugin do frameworku GStreamer (popsán v sekci 3.1) pro jednodušší zpracování multimédií. Tím se značně zkracuje čas potřebný pro vývoj aplikace.

2.5 Protokoly pro distribuci videa

RTP protokol pro přenos videa

RTP (Real-time Transport Protocol) je protokol definován dokumentem RFC, ten popisuje formu posílání multimédií přes síť. Tento dokument vznikl už v roce 1996 v RFC 1889, od roku 2003 byl nahrazen specifikací RFC 3550. Protokol se však postupně aktualizuje pomocí řady rozšíření, které formulují přesný formát pro různé typy přenášených dat a další specifika. RTP protokol se převážně používá v kombinaci s UDP protokolem nezajišťujícím spolehlivost doručení. RTP ovšem může být používán i v kombinaci s jinými, které jsou vhodné pro přenášení požadovaných dat. Tento protokol je jeden z vůbec nejrozšířenějších pro přenos multimédií. Používá se například i pro volání přes internet tzv. VoIP (Voice over Internet Protocol). RTP se ovšem většinou integruje v kombinaci s jinými protokoly zajišťující doprovodné funkce. Standard totiž pouze popisuje formát paketů pro přenášení dat, ale už ne další služby pro navazování spojení, ovládání, zajišťování kvality doručení, či způsob jak zjistit informace o multimediálním toku. Proto se protokol často vyskytuje s RTSP protokolem (viz sekce 2.5), ten zajišťuje a ovládá spojení mezi serverem a klientem, který požaduje přijímat RTP tok. Samotný RTP protokol disponuje pouze jedním typem paketu, který obsahuje multimediální data (tedy nejčastěji video a audio) a hlavičku identifikující data a jiné metadata. Tato hlavička je popsána níže. Popsány jsou pouze nejdůležitější části hlavičky.

1. Typ přenášených dat (PT) – označuje formát dat (např. H.263, JPEG a další)
2. Sekvenční číslo – první číslo ve spojení by mělo být vygenerováno náhodně a další se inkrementují s každým poslaným paketem
3. Časová značka – podobně jako u sekvenčního čísla by měla být první značka vygenerována náhodně, další už však jsou relativně vztaženy k sobě. Například data jednoho snímků rozdělená ve více paketech budou mít stejnou časovou značku. Dva po sobě jdoucí snímky budou mít rozdílnou časovou značku úměrnou podle rozestupu těchto snímků

RTSP protokol pro řízení přenosu videa

RTSP (Real-Time Streamimg Protocol) je síťový protokol určený pro zajištění spojení mezi streamovacím serverem a klientem, poté umožňuje různým způsobem zjišťovat informace o multimediálním toku nebo spojení ovládat (spustit přehrávání či ho pozastavit apod.).

Protokol je popsán v dokumentu RFC 7826. Ten požaduje, aby servery a klienské aplikace podporovali TCP a TCP přes TLS (šifrované spojení), ale můžou komunikovat i přes jiné protokoly. Samotný přenos multimédií po síti není zajištěn pomocí tohoto protokolu, tuto funkci realizují jiné protokoly. Často využívaný je však protokol RTP popsaný v předchozí sekci. Oproti tomuto protokolu je RTSP komunikace obousměrná a posílané zprávy se dělí na požadavek a na odpověď. To tedy znamená, že klient může poslat požadavek a server mu odpoví, zda se operace zdařila a v případě, že nezdařila připojí se důvod nezdaru. Tyto zprávy jsou u RTSP přenášeny v textové formě (podobné protokolu HTTP). Na začátku požadavku je uvedena požadovaná operace (start přehrávání, pauza, apod.), poté URI (Uniform Resource Identifier – identifikátor zdroje, ke kterému je požadován přístup) zdroje, kterého se dotaz týká. Ten se skládá z prefixu rtsp://, IP adresy nebo názvem destinace následovaného identifikátorem požadovaných dat (např. rtsp://server.com/audio.mp3). Následuje název protokolu, jeho verze a hlavička obsahující informace o požadavku (délka těla požadavku, apod.). Na konci zprávy se nalézá tělo požadavku. Forma odpovědi vypadá obdobně, akorát obsahuje informaci o provedení požadavku a kódu chyby.

RTSP protokol je nezávislý na druhu přenášených dat. Proto před začátkem vlastního přenášení dat musí klient zjistit informace o nabízených multimediálních streamech. Server nám tyto data poskytne po poslání požadavku `DESCRIBE`. Konkrétně nám může odpovědět s typem streamu (video, audio, ...), typem kódování dat, o možných protokolech použitých pro přenos vlastních dat a další informace relevantní pro příjem streamu. Tyto informace jsou přenášeny ve formátu SDP (Session Description Protocol). SDP protokol mimo jiné používá i populární přehrávač VLC media player. Po získání informací o multimediálních streamech dostupných na serveru si můžeme pomocí požadavku `SETUP` specifikovat, které streamy chceme přijímat a jakým způsobem. Server nám odpoví jestli požadované možnosti podporuje a případně dospecifikuje některé nastavení, pokud nebylo upřesněno klientem. Tento požadavek však můžeme zaslat i později při již započatém streamování. Server nyní má informace o tom, o které streamy má klient zájem. Klient má tedy možnost začít streamovat pomocí posílaní požadavku `PLAY`, kdy už začnou přicházet multimediální data (např. pomocí RTP protokolu). Pokud server podporuje na daném zdroji změnu pozice ve streamu, můžeme specifikovat od kterého místa chceme posílat data případně do jakého místa (specifikace intervalu). Naopak zastavit posílání multimédií můžeme pomocí požadavku PAUSE. Pro nastavování parametrů serveru lze využít metody `SET_PARAMETER` a pro vyčítání parametrů existuje požadavek `GET_PARAMETER`.

2.6 Konkurenční řešení

Existuje mnoho průmyslových kamer vhodných pro strojové vidění, avšak většina disponuje pouze rozhraním GigE Vision nebo jiným specializovaným rozhraním pro kamery. Tyto kamery většinou nepodporují kompresi videa. To znamená, že z kamery musíme přijímat velký datový tok až 1 Gbit za sekundu (při rozlišení 1920 × 1080 a 30 snímků za sekundu). Pokud bychom chtěli zpracovávat video v počítači (datacentru), který je vzdálený od kamery. Znamenalo by to vysoké nároky a náklady na potřebnou infrastrukturu. Při vyšším množství kamer už by ani toto řešení nebylo přípustné. Následující 2 produkty umožňují komprimovaní videa a jeho distribuci pomocí protokolu RTSP/RTP (viz sekce 2.5). Jedním z nich je kamera Basler[4] bip2-1920-30c, která má obdobné parametry jako kamera popisovaná v sekci 2.2, disponuje také globální uzávěrkou a snímá obraz v rozlišení až 1920 ×

[4] https://www.baslerweb.com/en/

1080 ovšem při nižší snímkovou frekvenci až 30 hertzů. Naproti tomu byla vybavena lepším snímačem od společnosti Sony IMX 174. Ten umožňuje pořizovat kvalitnější snímky bez šumu i při horších světelných podmínkách. Zásadně se tyto kamery však liší v ceně, ta je kolem 1100 eur, což je více než dvojnásobek ceny první kamery. Dále se různí v rozhraní přenášející data. Tato kamera již údajně nepodporuje GigE Vision protokol, ale lze využít standardní technologie pro přenos multimédií jako je RTSP protokol (popsán v sekci 2.5). Velkou výhodou je integrace video enkodéru a s tím i podpora streamování videa v komprimované formě. Tím se dosáhne významné úspory přenášených dat. Kromě H.264 (popsán sekci 2.3) lze posílat video ve formátech MJPEG a MPEG-4.

Další řešení nabízí společnost NorPix[5]. Ta vyvinula software Web Streamer Encoder pro zpracování videa z více průmyslových kamer najednou. Dokáže zachytávat snímky z až 12 kamer, komprimovat je a různým způsobem je distribuovat dále pomocí RTSP nebo HLS (HTTP Live Streaming). Tento software umožňuje zpracovávat video z kamer s podporovaným standardem GigE Vision s rozhraním USB 3.0 a Ethernet. Dále pak kamery se specializovaným rozhraním Camera Link a CoaXPress. Z kamer může obraz komprimovat do formátů H.264 a HEVC (H.265). Jednou nevýhodou systému je absence širší podpory software. Může totiž pracovat pouze s operačním systémem Windows a počítačem vybaveným procesorem od společnosti Intel. Ovšem nabízí možnost akcelerace kompresních algoritmů na grafických kartách od firmy NVIDIA a integrovaných grafických kartách na procesorech Intel (QuickSync GPU). Další nevýhodou je cena, která byla podle poptávky kolem 1100 eur. Cena se ovšem týká pouze jedné licence softwaru, k tomu je ještě potřeba přikoupit kameru. V případě použití kamery acA 1920-40gc by celková cena tohoto řešení byla kolem 1519 eur.

[5] https://www.norpix.com/

Kapitola 3

Knihovny pro zpracování multimédií

Pro urychlení implementace multimediálních aplikací se v řadě případů používají různé frameworky. Ty implementují funkce zjednodušující práci s videem i audiem. Zde jsou popsány 2 frameworky, které jsou jedny z nejpoužívanějších. Popsány jsou však pouze základní aspekty nutné pro pochopení frameworku a realizaci diplomové práce. Knihovna GStreamer byla použita pro implementaci multimediálního serveru. To vyžaduje její hlubší pochopení, proto byl výrazněji rozveden narozdíl od frameworku FFmpeg, který byl pouze využit pro tvorbu klientské knihovny. Její tvorba totiž vyžadovala použít pouze několik funkcí pro připojení k serveru a pro přijetí a dekódování snímků.

3.1 GStreamer multimediální framework

GStreamer[1] je framework pro práci s multimédii založený na konstrukci grafů filtrů. Výhody GStreameru využívají především Linuxové přehrávače videí Banshee, Clementine, Exaile a další. Ale existují i další typy dostupných aplikací například pro streamování videa, provozování video konferencí a podobně. Tento framework je oblíbený hlavně z důvodu jeho modulárnosti, jednoduchého vytváření aplikací pro zpracování multimédií a jeho velikosti. Základ GStreameru má velikost zhruba pár stovek kilobytů v binární formě. Ten v podstatě nic neumí, jen implementuje programové rozhraní pro snadnou stavbu filtrů a dalších funkcí pro práci s multimédii. Avšak framework jde rozšířit o desítky již existujících zásuvných modulů (tzv. pluginů), které například budou umět dekódovat video nebo zobrazit výstup na displeji obrazovky. Díky existenci pluginů však je tak i důležitá otázka způsobu nasazení frameworku s aplikací. GStreamer lze předinstalovat na počítači klasicky instalátorem, staticky slinkovat s aplikací nebo k aplikaci dodat všechny potřebné sdílené knihovny a pluginy. Pluginy se musí uložit do předem dané složky. Při startu aplikace GStreamer totiž prohledává standardní složky a nahrává nalezené pluginy do paměti. Pokud zásuvné moduly chceme uložit do jiných složek, můžeme nastavit proměnou GST_PLUGIN_PATH, která upravuje cesty pro hledání. Naproti tomu při potřebě slinkovat staticky framework s naší aplikací lze využít nástroj Cerbero (speciálně vyvinut GStreamer komunitou). Ten před samotným slinkováním umožňuje zkompilovat GStreamer podle vlastních potřeb s přizpůsobeným nastavením a závislostmi. Tím je možné ušetřit na výsledné velikosti aplikace.

[1] https://gstreamer.freedesktop.org

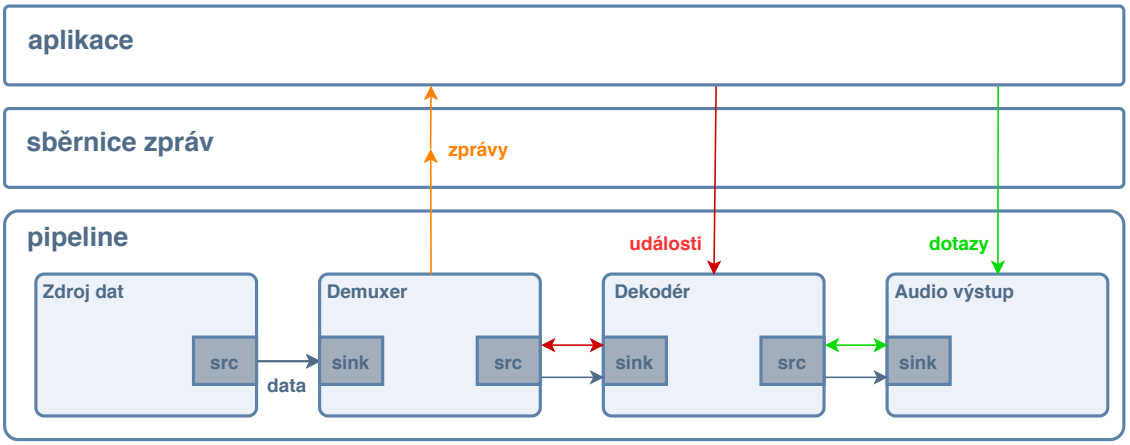
Kromě toho navíc podporuje zabalení výsledné aplikaci vhodně pro všechny používanější Linuxové distribuce.

Jak už bylo řečeno GStreamer staví na grafu filtrů, tento graf se nazývá tzv. pipeline (v překladu potrubí). Kompletně postavená pipeline tedy plní ucelenou funkci jako třeba přehrávač videa. Touto pipeline se data posílají z jednoho konce na druhý s tím, že se může rozdělovat. To znamená, že můžeme mít jeden vstup, ale více výstupů nebo naopak. Data proudí přes fundamentální jednotku GStreameru tzv. elementy. Ty plní určitou implementovanou funkci (např. dekódování videa nebo přehrávání audia). Každý takový element obsahuje minimálně jeden přípojný bod (v terminologii GStreameru nazývaného pad). Pouze přes tyto body můžou multimediální data proudit mezi elementy. Dělí se na dva typy: zdrojové (source) nebo cílové (sink). S tím, že směr komunikace jde vždy od zdrojového k cílovému bodu. Důležitou vlastností padu jsou tzv. schopnosti padu (angl. capabilities). Ty určují formát dat, který jsou schopní přijímat na vstupu nebo odesílat z výstupu. Propojit lze tedy pady jen pokud průnik jejich podporovaných formátů není prázdný.

Podle typů padů se dělí elementy na tři typy. Prvním jsou zdrojové elementy obsahující pouze zdrojové pady, dále cílové s pouze cílovými pady a filtry s oběma druhy padů. Při vytváření pipeliny tedy nejdřív instanciujeme všechny elementy podle použití, přidáme je do pipeliny a poté je propojíme. V okamžiku propojení se začnou všechny elementy domlouvat na kompatibilních formátech. Pokud se úspěšně podaří všechny elementy propojit, můžeme nastavit pipeline do stavu přehrávání (Playing). V tomto okamžiku zdrojové elementy začnou posílat na výstup data, posílat je do filtrů ke zpracování a nakonec je nějakým způsobem zobrazovat, zapisovat do souboru nebo přehrávat budou cílové elementy.

Důležitou součástí každého GStreamer elementu jsou parametry implementované pomocí Glib knihovny, na které framework staví. Ty umožňují přizpůsobovat jejich chování. Každý takový parametr má tak svoje jméno, popis a práva. Tedy zda je možné parametr číst, měnit nebo obojí. Kromě toho definuje i typ omezující hodnoty, které může nabývat (řetězec, číslo s plovoucí desetinnou čárkou, atd.). Tyto vlastnosti lze navíc jednoduše zjistit u každého elementu pomocí nástroje gst-inspect. Ten zajišťuje vypsání všech detailnějších informací o pluginech a jednotlivých elementech. Tedy například jméno, popis, verzi, již zmiňované parametry (jejich typ, povolený rozsah, práva a výchozí hodnota) nebo i jaké data požadují na vstupu, případně jaký typ dat umožňuje produkovat. Výstup z tohoto nástroje je tak nepostradatelný v případě, že konstruujeme vlastní multimediální aplikaci v GStreameru pomocí již hotových elementů. Díky zjištěným informacím tak snadno určíme, zda se dva různé elementy dají propojit, a případně se dočteme, které parametry musíme změnit pro požadovanou funkcionalitu a chování.

Pro zajištění správného fungování frameworku je nutná komunikace mezi elementy. Kromě samotných multimediálních dat, které tečou od začátku do konce pipeliny, se musí elementy mezi sebou dorozumívat pomocí událostí a dotazů. Dotaz (angl. query) slouží pro zjištění informací o streamu jako jeho délky nebo aktuální pozice. Takový dotaz doputuje k zodpovědnému elementu, který danou informací zná například zdroj videa, a odpoví. Pokud je potřeba oznámit nějakou změnu stavu nebo informaci, použije se primitivum nazvané událost (angl. events). Když nastane konec streamu (např. přečetl se celý multimediální soubor), tak se pošle EOS (End Of Stream) event napříč celou pipelinou, aby všechny elementy věděli, že už nemají očekávat další data. Existuje však celá řada dalších událostí. Pro komunikaci mezi elementy a aplikací se používají zprávy (angl. messages). Pomocí nich můžou jednotlivé elementy vystavit zprávy na tzv. sběrnici zpráv. Odsud aplikace zjišťuje různé informace o streamu a elementech, a reaguje na ně. Naopak pokud aplikace potřebuje



Obrázek 3.1: Gstreamer – popis komunikace a příklad pipeliny

poslat dotaz nebo elementy informovat o nějaké události, tak se nemusí používat sběrnice zpráv, ale přímo se kontaktují dané elementy.

Dosud byli popsány pouze základní informace o frameworku, které stačí pro většinu vytvářených aplikací pokud nepožadují specifickou funkcionality. Pokud však požadovaný vstup, výstup nebo filtr není implementován žádným z pluginů, můžeme si danou komponentu doprogramovat. Vytvoření pluginu není tak snadné jako běžné používání frameworku a vyžaduje pokročilejší pochopení fungování GStreameru. Ten staví na knihovně GLib, takže zásadní je nastudování této knihovny. Ta do jazyku C a frameworku dodává možnosti objektového programování, základní typy pro běžné použití, a jiné užitečné funkcionality. K vytvoření nového pluginu lze přistoupit dvěma způsoby, buď ho můžeme začít psát od začátku, pokud jsme dostatečně zkušení, nebo lze použít program make_element z repozitáře gst-template. Stačí tento skript spustit a předat mu argument se jménem nového pluginu. Tím se vygeneruje kostra potřebná pro implementaci pluginu. Komponenta ve výchozím stavu dědí od obecného elementu GstElement, což je generická komponenta implementující pouze základní funkce. Pokud realizujeme obvyklou funkci můžeme využít již z předdefinovaných tříd. Například pro tvorbu elementu zachytávající data z kamery se hodí třída GstPushSrc (dědící od GstBaseSrc a ta zas od GstElement). Ta zajišťuje propagaci dat ze zdroje do pipeliny.

Užitečným nástroj při vývoji GStreamer aplikace je GstShark[2], otevřený projekt vyvíjený komunitou a společností Ridgerun pro analýzu, optimalizaci a odhalování chyb v pipelinách GStreameru. Tento doplněk je nepostradatelnou součástí při vývoji aplikací pro tento framework. Při běhu dokáže sesbírat různé metriky a informace, tyto data zanalyzovat a zobrazit přehledné grafy. Z nich lze například zjistit nejnáročnější element a na základě těchto informací optimalizovat pipelinu. Kromě vykreslení grafů při analýze, umožňuje za běhu kontinuálně tisknout různé informace do konzole, které lze využít pro rychlé zjištění naměřených hodnot. GstShark nabízí celkem 8 analyzátorů vhodných pro ladění pipeline. Obsahuje jeden, který umí vykreslit strukturu výsledné pipeline při běhu a 7 analyzátorů pro měření:

1. latence snímků v pipeline

[2]https://developer.ridgerun.com

2. času potřebného ke zpracování snímku
3. dosahované snímkové frekvence
4. plánování snímků
5. vytíženosti procesoru
6. bitové rychlosti mezi elementy
7. statistik front mezi elementy

3.2 FFmpeg

FFmpeg[3] je jeden z vůbec nejrozšířenějších multimediálních frameworků, který umožňuje pokročilou práci s videem i audiem. Využívá ho například MPlayer nebo přehrávač VLC media player. Skládá se ze sady knihoven určené pro rozmanitou práci s multimédii. Nejdůležitější z nich se používají pro kódování a dekódování multimédií (libavcodec) a pro jejich takzvané multiplexování a demultiplexování z multimediálních kontejnerů (libavformat). Tyto kontejnery sdružují více multimediálních streamu v jednom datovém toku. Poté FFmpeg obsahuje řadu knihoven pro jiné běžné operace s multimédii (libavutil, libavdevice, libavfilter, libswscale, libswresample) a tři konzolové nástroje pro práci z příkazové řádky. Prvním z nich je utilita ffmpeg pro konverzi mezi různými formáty, dále ffprobe pro analyzování multimediálních streamů a nakonec ffplay pro přehrávání videa pomocí knihovny SDL. Mimo jiné nástroj ffplay umožňuje na přehrávaný stream aplikovat graf filtrů podobně jako GStreamer, i když ne tak přehledně a jednoduše. Oproti frameworku GStreamer totiž není tak modulární a funkcionalita není přidávána skrze pluginy. Nové funkce se musí slinkovat při kompilaci tohoto frameworku. Což může být nevýhoda. Musíme při potřebě aktualizace knihovny překompilovat celou knihovnu (např. byla přidána podpora nového formátu). Na rozdíl od Gstreameru, kde by stačil přidat nebo překompilovat jen plugin. V řadě případů to může být naopak výhoda z důvodu jednoduché integrace. Nemusí se totiž nikde hledat žádné pluginy, všechno je slinkované do jedné velké knihovny.

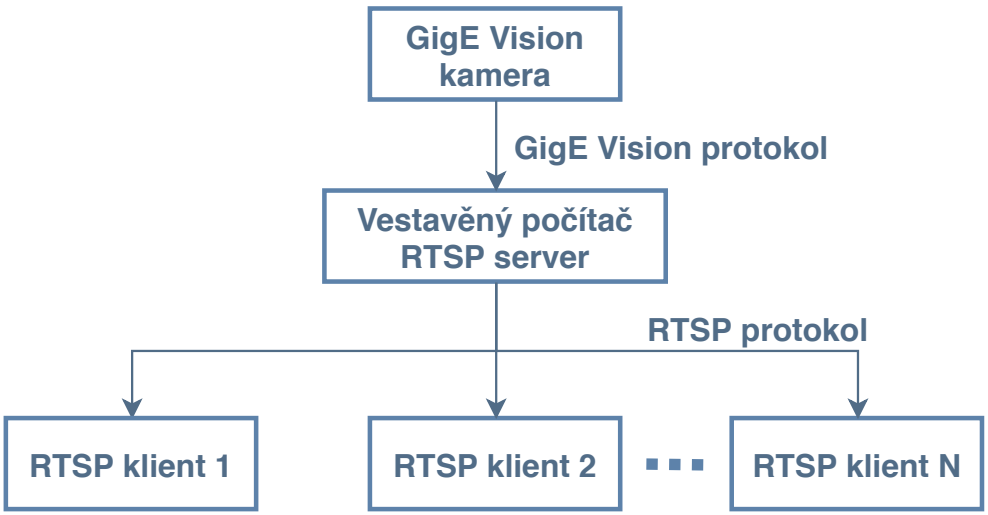
[3] https://www.ffmpeg.org

Kapitola 4

Návrh systému

Další stěžejní částí je návrh vlastního systému pro streamování videa. Hlavním cílem systému bude zredukovat datový tok z kamery a tento tok vhodným způsobem poskytnout uživatelům nebo nadřazenému systému pro další zpracování. Práce se tak zabývá i tvorbou knihovny pro jednoduché přijímání videa ze streamovacího serveru. Dle požadavků by měl systém splňovat několik kritérií pro úspěšně nasazení tohoto softwaru. Pro možnost rychlé reakce nadřazeného systému na informace z kamery je požadována maximální latence půl sekundy, dostatečná kvalita obrazu a možné připojení více klientů pro zajištění přístupu ke kameře operátorům, kteří mohou kontrolovat průběh testování na zařízení, kde je umístěna kamera.

Obrázek 4.1 zobrazuje kompozici navrženého systému se všemi jeho hlavními komponentami. Skládá se z GigE Vision kamery nahrávající video v nekomprimovaném formátu, z té zachytává snímky vestavěný počítač RockPro64, komprimuje je a pomocí protokolu RTSP/RTP ho dále posílá klientům. Následující sekce popisují detailněji návrh jeho jednotlivých částí.



Obrázek 4.1: Blokový diagram navrženého systému

4.1 Kamera a hardware serveru

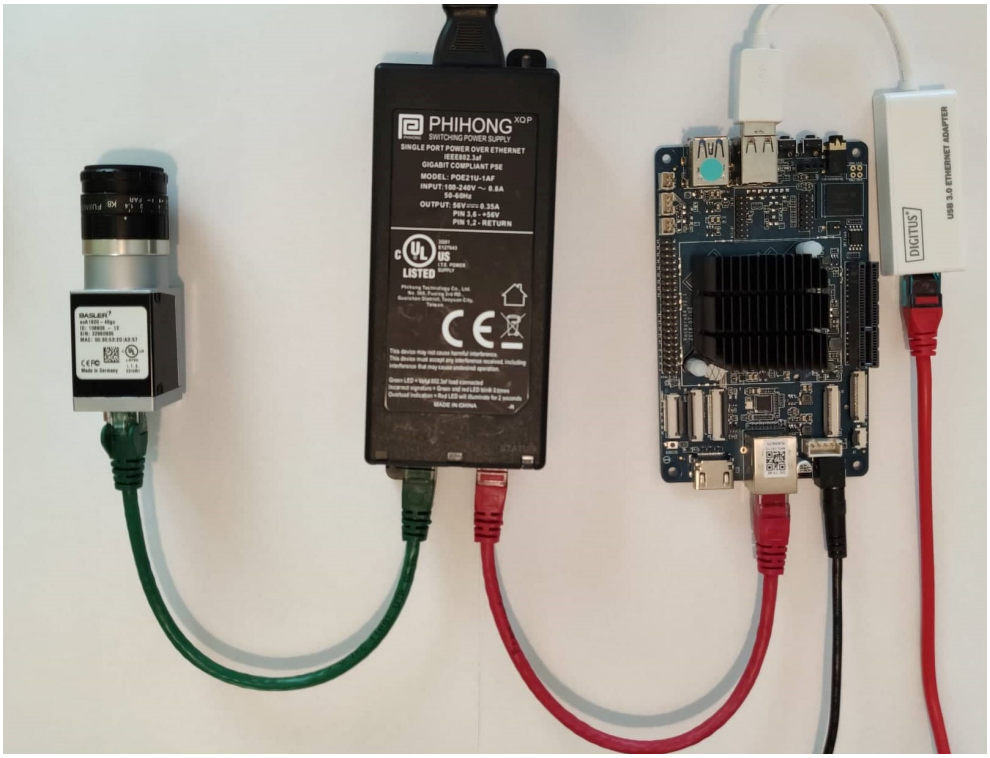
Důležitou komponentou v systému je kamera, která nahrává video. Tu vybral vývojář a expert na umělou inteligenci ve společnosti YSoft Corporation a.s. experimentálně s ohledem na dobrou kvalitu videa pro aplikaci strojového učení. Byla zvolena kamera Basler acA 1920-40gc s rozlišením 1920 × 1200 a snímková frekvencí až 42 Hz. Další parametry a vlastnosti jsou popsány v sekci 2.2. Kamera disponuje rozhraním Ethernet a komunikovat lze s ní pomocí protokolu GigE Vision. Proto server, který s ní bude komunikovat, může používat specializovanou knihovnu dodávanou ke kamerám Basler. Její název je PylonSDK a díky ní lze programově ovládat kameru, tedy ji nastavovat a efektivně zachytávat snímky. Teoreticky by nebylo nutné použít PylonSDK, ale generickou knihovnu, která podporuje obecně komunikaci s GigE Vision kamerami. Díky použití knihovny přímo od výrobce se může, ale využít specifických vlastností kamery a při potenciálních problémech využít podporu společnosti Basler.

Při návrhu průmyslového řešení se řeší především cena a doba výroby používaných komponent. Pro běh serveru byl vybrán vestavěný SBC počítač RockPro64. Ten vyniká svoji relativně nízkou cenou, vysokým výkonem a výrobce navíc garantuje výrobu počítače po několik dalších roků. Kromě toho podporuje komprimaci videa do několika formátů. Byl vybrán jeden z nejpoužívanějších kodeků a to H.264 (formát byl teoreticky rozebrán v sekci 2.3). Počítač dále disponuje gigabitovou síťovou kartou, do které může být připojena kamera. Ta může produkovat relativně velký datový tok, takže by pomalejší síťová karta ani nestačila. Kamera může být propojena se serverem dvěma způsoby pomocí konektoru Ethernet. Jednou možností je připojení kamery a počítače do jednoho síťového přepínače nebo připojením kamery přímo do počítače a ten pak zvlášť do lokální sítě (tento přístup je zobrazen na diagramu systému 4.1). V případě použití druhého ze zmíněného způsobu musí být použita ještě externí karta, protože počítač disponuje pouze jednou síťovou kartou s konektorem. Může být připojena buď do USB 3.0 nebo PCIe 4x konektoru. Tento přístup je vhodný zejména v případě, kdy potřebujeme, aby nikdo jiný neměl přístup ke kameře z okolní sítě. Zároveň to přispěje ke stabilitě zachytávání snímků, protože jedno síťové rozhraní bude vyhrazeno pouze pro kameru. Tím pádem nemůžou ostatní datové toky ovlivňovat zachytávání videa, jako by tomu bylo u připojení kamery a počítače do společného přepínače. Přes síťové rozhraní by tedy bylo posíláno jak video z kamery tak i komprimovaná data určená klientům a ostatní data nesouvisející se streamováním. To by zásadně omezilo počet možných klientů přijímajících video stream při vyšším rozlišení a snímkové frekvenci. Server se tedy skládá z průmyslové kamery, POE napájecího adaptéru, počítače RockPro64 a volitelně doplňující síťové karty. Kromě těchto součástí není potřeba žádných další hardwarových komponent na straně serveru. Pouze je nutné vyřešit situaci, pokud přípojíme kameru přímo do počítače RockPro64. Ta totiž potřebuje nastavit IP adresu pro možnost správné komunikace. To můžeme provést staticky, kdy adresu nakonfigurujeme napevno, nebo dynamicky pomocí DHCP serveru. V případě dynamického přiřazení musíme na počítači nainstalovat a nakonfigurovat vlastní DHCP server. To může znamenat složitější nastavení, ale při dalším využití to má výhodu, že nemusíme nijak ručně konfigurovat kameru. Pro stabilnější a uživatelsky přívětivější řešení je tedy vhodnější využít automatického přiřazení adresy.

Kamera se může napájet dvěma různými způsoby. Přes speciální konektor pomocí rozsahu 12 až 24V napětí nebo přes Ethernet. V případě využití Ethernetu získáme výhodu použití pouze jednoho kabelu. Což může redukovat délku nutné kabeláže, zvláště pak pokud potřebujeme kameru natáhnout dále od zpracovávajícího počítače. Pro napájení přes

Ethernet kamera podporuje standard IEEE 802.3af tzv. POE (Power Over Ethernet), který umožňuje distribuci elektrického proudu pomocí kroucených dvojlinek přes síťový kabel. Díky tomu můžeme použít tzv. POE injektor, pomocí něhož se přes síťový do kamery dovede proud. Jako injektor byl vybrán model POE21U-1AF-R, který je kompatibilní s kamerou. Jeho maximální výkonová zátěž může být až 19,6 W, kamera však dosahuje podle dokumentace špičkových zátěží 3,4 W. Výkon injektoru tedy bohatě dostačuje a kamera tak může být úspěšně napájena.

Obrázek 4.2 obsahuje celkové zapojení multimediálního serveru. Nalevo se nachází kamera Basler, která se napájí pomocí POE adaptéru. Z něho poté síťový kabel dále vede do počítače RockPro64. Do tohoto počítače je pak připojena síť, z které má být server dostupný pomocí UTP kabelu a USB síťové karty značky Digitus DN-3023.



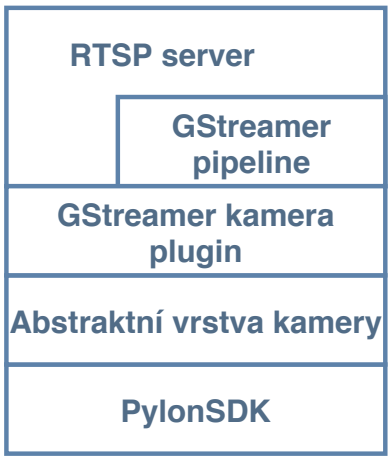
Obrázek 4.2: Fyzické zapojení všech hardwarových prvků serveru

4.2 Software serveru

Aplikace serveru bude mít zaúkol iniciovat komunikaci s kamerou a dále z ní zachytávat snímky, ty zkomprimovat a následně posílat klientům, kteří o ně budou mít zájem. Kromě toho musí umožnovat nastavovat parametry streamu pomocí vhodně zvoleného protokolu. Tato funkcionalita je potřeba především z důvodu ušetření datového pásma nastavením parametrů enkódování nebo i možností nastavením kamery (např. nastavení doby závěrky v případě, že se změní světelné podmínky). Vyjma nastavování musí existovat i způsob pro zjištění modelu a sériového čísla kamery, či jiných informací.

Pro manipulaci s videem byl zvolen framework GStreamer (viz 3.1) a to hlavně z důvodu jeho modulárnosti a multiplatformnosti. Umožňuje snadnou stavbu multimediálních

aplikací a obsahuje řadu už hotových pluginů pro různé účely zpracování videa, takže je ideální volbou pro implementaci serveru. Samotný software serveru se skládá z několika komponent. Na obrázku 4.3 je znázorněné rozložení všech částí serveru a jejich vzájemné vazby. Pro komunikaci se zvolenou kamerou se používá knihovna PylonSDK, ta je využita pro implementaci abstraktní vrstvy (ta je zde pro možnost jednoduché dodělání podpory více kamer). Kvůli modulárnímu návrhu aplikace a značně jednodušší integraci kamery do GStreameru jako zdroje snímků je vytvořen plugin. Ten se skládá z elementu umožňující použití v GStreamer multimediální pipeline, ve které zajistí zdroj snímků z kamery. Pipeline kromě toho obsahuje element pro komprimaci videa a element formující RTP pakety ze streamu. Výstup z této pipeline se poté přeposílá klientům pomocí RTSP/RTP serveru, který implementuje knihovna gst-rtsp-server, ta staví také na frameworku GStreamer. Což umožňuje jednoduchou integraci, protože knihovna nativně očekává výstup z pipeline, který pouze vezme a přepošle klientům. Výsledkem tedy bude samotná konzolová aplikace, která stará o inicializaci kamery, správné nastavení RTSP serveru, vytváření multimediální pipeline apod.



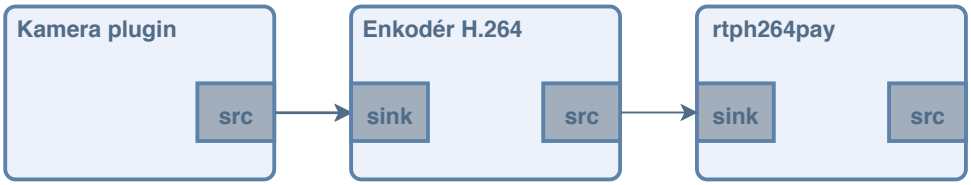
Obrázek 4.3: Struktura serveru

Pro komunikaci s kamerou byla navrhnuta vrstva pro abstrahování ovládání a streamování videa z kamery tzv. Camera abstraction layer (CAL). Ta má za úkol definovat standardní rozhraní používané pro kamery a implementovat ho pro konkrétní typ. V případě použití kamery Basler byla vytvořena pomocí PylonSDK, která zajišťuje komunikaci s libovolnou kamerou od společnosti Basler AG. Ovšem díky této vrstvě je možné jednoduše doplnit podporu pro jiné typy kamer. Při požadavku na použití jiné kamery tak bude stačit implementovat vrstvu pomocí jiné knihovny pro komunikaci s kamerou.

Při realizaci podpory kamer Basler byl zvolen přístup překrývaného zachytávání snímků pro zvýšení výkonnosti při vyčítání videa z kamery. To znamená, že se při inicializaci nastaví kamera a mimo jiné zaalokuje několik bufferů určených pro snímky, ty vloží se do fronty. PylonSDK poté plní postupně všechny volné buffery novými daty ve svém interním vlákně. Následně buffery se snímky vyjme z této fronty implementovaná knihovna, která s nimi může začít pracovat (poskytnout GStreamer pluginu, více viz dále). Pro správnou funkčnost ovšem musíme ještě po každém snímků zaalokovat nový buffer a vložit ho do fronty, aby se nestalo, že se nebude mít k dipozici žádnou paměť pro ukládání snímků. Překrývané zachytávání snímků musí být využito v případě, že chceme dosáhnout maximálního možné

výkonu. Na rozdíl od způsobu, kdy bychom pro každý požadovaný snímek posílali jeden dotaz, totiž kamera pořád produkuje snímky a PylonSDK je pouze zachytává. Odpadá tedy určité časové zpoždění mezi posledním snímkem a novým dotazen na snímek.

Vrstva CAL je dále využita při vytvoření pluginu do frameworku GStreamer obsahující element komunikující s kamerou. Ten má zaúkol při inicializaci správně nakonfigurovat kameru podle zvolených parametrů, následně zachytávat snímky z kamery a propagovat je dále do GStreamer pipeline. Tyto úkony by bylo možné realizovat i bez vytvoření pluginu. Díky pluginu ovšem implementujeme tzv. modulárnější kód, který bude přehlednější, protože plugin jasně ohraničuje požadovanou funkcionalitu. Navíc pokud by vznikl požadavek na jinou aplikaci využívající zachytávání z kamery, můžeme jednoduše znovu použít tento plugin. Stručné informace o tom jak postavit vlastní plugin je popsáno v sekci 3.1. Kromě zachytávání videa musí plugin nakonfigurovat kameru, proto má řadu vlastností (angl. properties), které budou nastavovat chování pluginu a vlastnosti kamery (závěrku apod.). Díky tomu lze přehledně přizpůsobovat výchozí parametry, ale i přijímat požadavky od klienta a podle toho je upravovat. Nastavování parametrů je detailněji řešeno dále v textu.



Obrázek 4.4: Pipeline zpracovávající snímky

Element pro zachytávání videa je dále používán v pipeline GStreameru. Spolu s ním se v pipeline nachází enkodér do formátu H.264, element pro vytváření RTP paketů (v GStreamer frameworku používán element rtph264pay). Video se tedy nejprve zachytí pomocí vytvořeného elementu, enkóduje se a převede na pakety RTP. Podstatné elementy v pipeline znázorňuje obrázek 4.4. Pipeline navíc obsahuje i element queue (fronta). Tyto elementy jak již název napovídá, obsahují frontu snímků. To zajistí možnost oddělení zpracovávání snímků v různých částech pipeline rozdílnými vlákny. V tomto případě byla jedna fronta vložena před enkodér. Ta zajistí, že pro zachytávání snímků z kamery bude vyhrazeno jedno vlákno. Zachytávání z kamery je totiž jedna z nejnáročnějších operací, protože musí zpracovávát až téměř 1 Gbit/s. Kromě elementů dodaných programátorem ovšem knihovna gst-rtsp-server do pipeline zapojí i další pro zajištění distribuce RTP paketů klientům. Například přidává tzv. rtpbin pro zajištění možnosti využívání RTCP protokolu (např. pro synchronizaci času). Nakonec přidává element multiudpsink pro samotné posílání paketů RTP připojeným klientům přes socket.

Při vytváření pipeline je potřeba vybrat z dostupných elementů právě enkodér H.264. V našem případě budeme spouštět server pouze na počítači RockPro64, kde bude zajištěna přítomnost konkrétního elementu. Ovšem pokud budeme chtít server spustit i na jiném hardwaru, tak je vhodné zaručit nalezení toho nejvhodnějšího aktuálně dostupného enkodéru. K nalezení toho správného GStreamer nabízí několik funkcí. Každý plugin má poznačený podporovaný vstupní a výstupní formát a jeho typ. Tedy jestli je to enkodér, zdroj dat nebo jiný typ elementu. Proto stačí vyfiltrovat pouze enkodéry, které podporují YUV422 jako vstup (formát výstupu z kamery), a vybrat ten s největší prioritou. Vyšší priorita typicky znamená, že plugin je vhodnější pro dané použití něž jiné nebo je výkonnější apod. Po nalezení správného enkodéru je potřeba vytvořit i všechny elementy nutné

pro zpracování a distribuci videa. Tedy element pro zachytávání snímků z kamery, enkodér H.264 a element pro vytváření RTP paketů z videa. Pokud máme k dispozici všechny elementy je možné se pustit do konstrukce samotné pipeline. Do ní přidáme všechny vytvořené elementy a ty následně propojíme. Pipeline je tak úspěšně připravena pro použití. Tato pipeline se na serveru ovšem nevytváří před startem vlastního RTSP serveru, ale při obsluze klientů. Přesné chování záleží na nastavení.

O obsluhu klientů se stará RTSP knihovna GStreameru. Ta zajišťuje příjem příkazů od klienta a následné odesílání dat z výstupu výše zmíněné pipeliny. RTSP knihovna umožňuje vytvořit novou pipeline pro každého klienta zvlášť nebo jednu, kterou používají všichni připojení klienti zároveň. Vzhledem k tomu, že je k dispozici jen jedna kamera a typicky jeden enkodér do formátu H.264, musíme nastavit tzv. sdílenou (angl. shared) pipeline. V tomto případě se při připojení prvního klienta vytvoří jedna sdílená pipeline, poté je z ní posílán video stream ostatním později připojeným klientům. Pokud se odpojí poslední klient, nepotřebná pipeline se může zastavit a dealokovat. Díky tomu nejsou spotřebovávány zbytečně výpočetní zdroje. GStreamer RTSP server pro zkonstruování pipeline využívá návrhového vzoru továrna (angl. factory). Je potřeba tedy vytvořit továrnu, která při žádosti instanciuje novou pipeline. Knihovna ve výchozím stavu ovšem nepodporuje vytváření pipeline přímo v kódu, umožňuje pouze definovat pipeline pomocí řetězce (viz dokumentace programu gst-launch). To značně omezuje možnost použití, daleko flexibilnější je vytváření pipeline programově, kde je možné ji dynamicky vytvářet podle dostupných elementů apod. Takže se musí implementovat nová továrna, které nastavíme ukazatel na funkci vracející pokaždé novou zkonstruovanou pipeline. Při žádosti se tedy zavolá daná funkce vracející pipeline, jež RTSP server může využívat.

4.3 Nastavování parametrů streamu

Při používání systému se můžou měnit požadavky například na kvalitu a rozlišení obrazu, či se změní dostupná šířka pásma. Kromě toho může uživatel chtít upravit parametry obrazu zachytávaného z kamery podle aktuálních světelných podmínek nebo způsobu použití. Pokud se například zhorší světelnost v okolí kamery, bylo by vhodné zvětšit dobu uzávěrky kamery. Proto je nutné navrhnout mechanismus pro možnost vzdáleného přizpůsobení parametrů. Ten by ideálně neměl přinášet závislost na další technologii či protokolu a v budoucnu případně dovolil změnu parametrů jen autentizovaným uživatelům s potřebnými právy. RTSP protokol (více v sekci 2.5), který již byl využit v systému pro připojení klientů a ovládání videa, podporuje nastavování vlastností streamu. Možnosti autentizace jsou stejné jako u protokolu HTTP, takže lze např. využít autentizaci pomocí tokenu nebo komunikaci skrze TLS. Obě tyto možnosti jsou implementovány v RTSP knihovně v GStreameru. Díky jednoduchosti RTSP protokolu tak lze snadno vytvořit vlastní parametry streamu, které lze měnit pomocí příkazu SET_PARAMETER a číst pomocí GET_PARAMETER. Každý parametr má jméno a textovou hodnotu. Pro implementaci tak stačí definovat vlastní parametry pro upravení, odchytávat požadavky z klienta a podle toho nastavovat příslušné elementy pipeline, tedy buď H.264 enkodér, element pro zachytávání videa z kamery nebo i další pro dodatečnou úpravu videa. Zde jsou vyjmenovány základní využitelné parametry pro možnost vzdálené úpravy:

1. Rozlišení
2. Snímková frekvence

3. Doba uzávěrky kamery
4. H.264 mód – CBR nebo VBR (viz 2.3)
5. H.264 kvantizační parametry
6. H.264 výstupní bitová rychlost
7. Případně další podle potřeby

Procesor RK3399 obsažený v počítači RockPro64, disponuje již hotovým pluginem do frameworku GStreamer obsahující enkodér do formátu H.264. Ten na pozadí využívá knihovny MPP, která implementuje přístup k hardwarovým periferiím pro zpracování multimédií. Tento element má ovšem nevýhodu v podobě napevno nastavených parametrů pro enkódování. Proto je potřeba plugin upravit pro možnost jejich nastavení. Tyto parametry může být potřeba nastavit podle dostupné šířky pásma a provozu ostatních zařízení na síti. Dále můžou vzniknout požadavky tyto parametry potřeba nastavovat vzhledem k maximální povolené latenci. Podobná potřeba vznikne při vytváření elementu pro zachytávání videa. Ten musí disponovat několika parametry pro uzpůsobení chování kamery. U enkodéru i zdroje videa lze tyto parametry jednoduše implementovat pomocí Glib knihovny, kterou GStreamer využívá (více viz 3.1). Tato knihovna umožňuje zaregistrovat více parametrů různých typů. Následně stačí přidat funkce, které je budou nastavovat či číst (tedy tzv. setter a getter). Tyto parametry lze pak měnit u libovolného elementu pomocí funkce g_object_set nebo číst pomocí g_object_get. Toho lze s výhodou využít u dynamického nastavování parametrů přes RTSP protokol, protože nám stačí znát textový název parametrů a pomocí zmíněných funkcí můžeme změnit nebo číst parametry.

Při příjmu žádosti o změnu parametru, tak z aktuálně vytvořené multimediální pipeline zjistíme element, kterého se upravovaný parametr týká a jednoduše tento parametr nastavíme. Důležité je tak vhodně pojmenovat RTSP parametry. Budou se tak skládat ze dvou částí oddělených podtržítkem. První bude název elementu a druhou název parametru (např. encoder_mode). Díky této skladbě tak bude jednoduché najít příslušný element v pipeline a navíc to zajistí automatickou integraci nových parametrů. V případě, že byly přidány do elementů nové parametry nebo byl přidán nový element do multimediální pipeline. Nemusí se tak v případě nutnosti používat další parametry měnit logika uvnitř RTSP serveru. Pro zajištění kompatibility napříč více použitých typů elementu je pouze potřeba zajistit přemapování názvů parametrů. Server totiž může v budoucnu běžet na více místech s podporou různých H.264 enkodérů. Tyto elementy enkodéru, ale nemusí mít shodně pojmenovány parametry i když vykazují stejnou funkcionalitu. Proto v tomto případě je potřeba přemapovat RTSP parametr na konkrétní název parametru daného elementu. Toto přemapování je umístěné v souboru settings.ini (se standardním ini formátem) v sekci pojmenované podle daného typu elementu (např. Rockchip H.264 enkodér). V ní jsou definovány jednotlivé přemapování ve formátu 'RTSP parametr'='parametr elementu'. Tedy lze snadno zajistit kompatibilitu napříč různých použitých elementů. Kromě toho v tomto souboru lze definovat výchozí hodnoty parametrů, které by měl server při vytvoření pipeline nastavit.

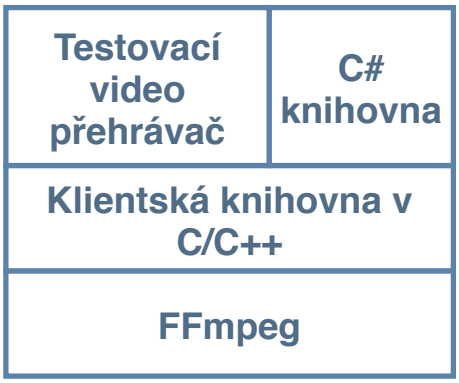
Pro jednoduchost jsme zatím předpokládali, že multimediální pipeline existuje pořád. Jenže jak bylo popsáno, pipeline se zkonstruuje, pouze pokud klient požaduje video stream. Navíc zaniká s posledním odpojeným klientem. Proto musíme nastavené parametry ukládat i jinam. Pokud bychom totiž při RTSP požadavku pouze nastavovali pipeline, nastavenou informaci bychom ztratili. Parametry tak při každém nastavení uložíme do asociativního

pole, které umožňuje rychlejší vyhledávání. A při každém zkonstruování pipeline nastavíme parametry podle uložených hodnot.

4.4 Klientská knihovna

Klient bude využívaný pro příjem videa, musí se tedy umět připojit na server popisovaný výše. To znamená, že pomocí protokolu RTSP se dotáže serveru na informace o dostupném streamu a podle toho inicializuje relevantní prvky jako dekodér videa a spustí streamování. Poté zpracovává pakety, dekóduje jednotlivé snímky a zobrazuje je na displeji nebo dává k dispozici programu využívající klienta. Pro příjem a dekódování videa byl zvolen framework FFmpeg (viz sekce 3.2). Oproti frameworku GStreamer není tak modulární, ale to v tomto případě nevadí. Naopak je velmi robustní a podporuje rozmanité formáty pro příjem multimédií. Nemusí se tak řešit problémy s nacházením potřebných pluginů, všechno je totiž obsaženo přímo v rámci sdílených knihoven FFmpeg.

Při návrhu musí být kladen důraz na jednoduchou integraci klienta do již hotových aplikací, převážně napsaných v jazyce C#. FFmpeg nativně podporuje aplikační rozhraní pouze z jazyka C, případně C++. Proto se práce zabývá jak implementací knihovny v jazyce C, tak i jejím obalením a vytvořením knihovny v C#. Pro vytvoření takového obalu lze využít již dostupné nástroje pro automatické generování tzv. bindingu. To umožňuje použití stejného kódu z obou jazyků bez psaní kódu navíc.

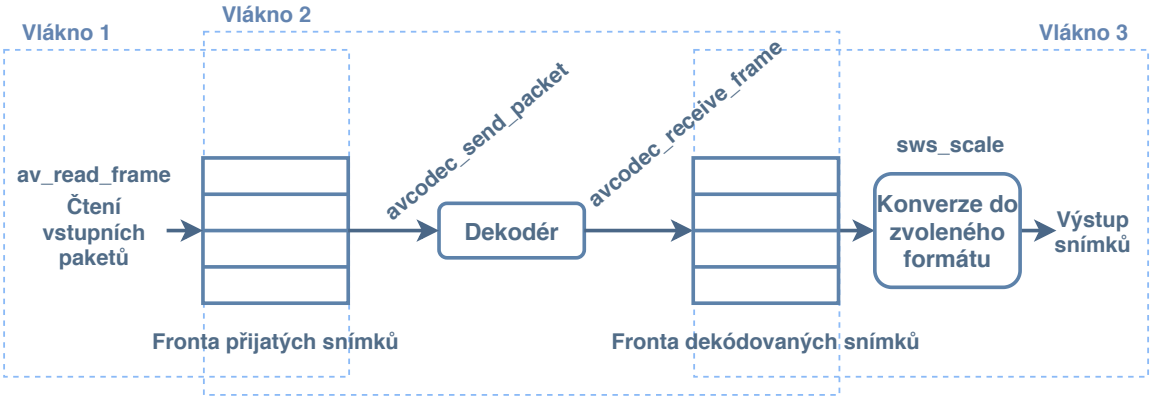


Obrázek 4.5: Struktura klienta přijímajícího snímky ze serveru

Následující text popisuje všechny části klienta a z části i navržený způsob jejich přeložení. Obrázek 4.5 znázorňuje komponenty klienta a jejich vazby. Tedy klientskou knihovnu napsanou v C a C++ spolu s frameworkem FFmpeg. Ta byla poté využita pro stavbu testovacího přehrávače a C# knihovny pro možnost zpracovávání snímků z tohoto jazyka. Nejdůležitější částí je samotná klientská knihovna napsaná v jazyce C/C++ s co nejjednodušším rozhraním pro přehrávání streamu ze serveru RTSP. Umožňuje v podstatě pouze nastavení adresy serveru a formátu videa, ve kterém je potřeba přijímat video. Poté umožňuje zjistit rozlišení streamu, spuštění přehrávání případně jeho zastavení. Pro samotný příjem snímku slouží funkce receive. Ta vrací aktuálně dostupný dekódovaný obraz ve specifikovaném formátu (RGB24 nebo BGR24).

Při startu streamování se nejprve knihovna pomocí FFmpeg funkce avformat_open_input zkusí připojit na zdroj dat (RTSP server) spolu s nastavením parametrů pro příjem dat. Mezi ně patří specifikace velikost vstupních bufferů, maximální zpoždění snímků a další.

V případě úspěchu se podle vlastností streamu najde vhodný dekodér a inicializuje se. Po tomto kroku již můžeme úspěšně přijímat jednotlivé snímky. Obrázek 4.6 popisuje průchod snímků v knihovně. Po inicializaci a spuštění přehrávání se tedy postupně zpracovává video. Práce je rozdělena do celkem 3 vláken (z toho dvě jsou interní vlákna knihovny). Tímto způsobem lze minimalizovat latenci snímků na minimum, než kdyby všechny operace prováděl jedno vlákno. První interní vlákno má za úkol přijímat vstupní pakety pomocí funkce FFmpeg av_read_frame a vkládat je do fronty přijatých paketů. Odtud si je další interní vlákno bere a posílá je do dekodéru H.264. Následně jsou dodávány do fronty dekódovaných snímků. Třetí vlákno reprezentuje program, který používá tuto knihovnu. To znamená, že pouze při žádosti o snímek se vyjme z fronty dekódovaných snímků, převede se do požadovaného formátu a je k dispozici pro zpracování externím programem. Protože systém má zajišťovat přenos a zpracování snímků s co nejmenší latencí, je potřeba implementovat několik důležitých mechanismů. Může nastat případ, že program nestíhá vyčítat dekódované snímky a fronta, jejíž kapacita je úmyslně omezena, se naplní (její velikost určuje maximální tolerovanou latenci). V tomto případě se z fronty musí zahazovat nejstarší snímky a nahrazovat je nově dekódovanými. Obdobně se bude chovat fronta přijatých snímků. Ta se bude přeplňovat v případě, že druhé vlákno nestíhá dekódovat snímky. Zahazování snímků z této fronty ovšem není žádoucí, protože přijdeme o data potřebná pro odvození dalších snímků. Pokud se tak tedy stane, video se obnoví až poté, co dekodér objeví novou skupinu obrázků (GOP – viz 2.3).



Obrázek 4.6: Zpracovávání snímků v klientské knihovně

FFmpeg ve výchozím nastavení umožňuje přehrávat všechny podporované formáty. Avšak pro příjem videa přes RTSP a RTP s formátem H.264 stačí jenom zlomek z nich. Proto je vhodné zkompilovat framework samostatně pro zajištění co nejmenší velikosti knihovny. To ve výsledku znamená, že vývojář nemusí tak dlouho čekat na stahování závislosti a navíc se značně zredukuje velikost aplikace využívající tuto knihovnu. Pro kompilaci a zabalení výsledných artefaktů byl zvolen systém Conan[1]. Pomocí conan skriptu v jazyce Python tak lze automaticky stáhnout zdrojové soubory, přeložit je, zabalit a nahrát výsledný balíček na server. Odkud si ho může kdokoliv stáhnout bez další nutnosti znovu zdlouhavě kompilovat FFmpeg framework. Toto je především výhodné při vývoji, kdy nemusíme pokaždé znovu sestavovat zdrojové kódy frameworku.

[1] https://conan.io/

Pro vytvoření hotového balíčku s frameworkem FFmpeg slouží vytvořený skript conanfile.py (standardní pojmenování Conan skriptu), kde je uvedena adresa archivu se zdrojovými soubory FFmpeg. Odtud si je Conan stáhne a spustí konfigurační skript. Pomocí něj deklarujeme, co všechno se má ve frameworku přeložit a důležité volby specifické pro překladač, architekturu a operační systém. Následně už se pouze soubory zkompilují, nainstalují do dočasného adresáře a zabalí. Takový balíček je už připraven pro nahrání na server. Pokud potřebujeme přeložit soubory pro řadu různých platforem a operačních systému, můžeme využít Conan knihovny, jejíž název je Conan Package Tools (CPT). V tomto případě stačí vytvořit skript build.py a v něm uvedeme všechny konfigurace, pro které chceme, aby byl balíček dostupný. Pokud tento skript spustíme, započne kompilace zdrojových souborů pro všechny uvedené možností a platformy. Alternativou by bylo spouštění skriptu conanfile.py s různými konfiguracemi, CPT však práci s více konfiguracemi zpříjemňuje. FFmpeg je totiž potřeba kompilovat pro Windows a Linux s různými architekturami (x86 a x64), různými konfiguracemi (pro ladění – angl. debug nebo hotové vydání – angl. release). Nakonec se pomocí konzolové aplikace Conan nahrají přeložené balíčky na požadovaný server.

Pro samotný překlad klientské knihovny se využívá nástroje CMake. Ten podporuje vygenerování kompilačních skriptů pro většinu operačních systémů a nástrojů pro přeložení zdrojových souborů jako je Make, GNU Autotools, Ninja a další. V případě této knihovny CMake benefituje z vynikající integrace s balíčkovacím systémem Conan. Ten nabízí CMake skript pro jednoduchý import závislostí z Conan serveru. V tomto případě je potřeba pro použití předem přeloženého a zabaleného frameworku FFmpeg. V případě překládání testovacího video přehrávače ještě pro stažení SDL knihovny pro zobrazování videa. Při spuštění konfigurace projektu pomocí nástroje CMake se tak spustí nejprve Conan, který se podívá do souboru conanfile.txt. Zde jsou uvedeny všechny závislosti projektu a soubory pro import do složky se zkompilovanými artefakty. Takže se zde může deklarovat např. kopírování všech sdílených knihoven frameworku FFmpeg k přeložené knihovně klienta. Po hotovém stažení Conan balíčku s frameworkem FFmpeg začne CMake s kompilací samotného projektu klienta, protože už má všechny potřebné závislosti.

Jak bylo zmíněno knihovna má být využívána pro příjem videa a jeho následné zpracování strojovým učením, které je implementováno v jazyce C#. Vzhledem k tomu, že klientská knihovna je naprogramována v jazycích C++/C, musíme nějakým způsobem zajistit její komfortní používání z prvního zmíněného jazyka. Pro generování vrstvy umožňující přijímání snímku z jazyka C# byl použit nástroj CppSharp zajišťující prozkoumání vstupních hlavičkových souborů a automatické vytvoření C# souborů. Ty na pozadí volají funkce ze sdílených knihoven zkompilovaných C a C++ souborů. Díky použití CppSharp se nemusí ručně psát tyto soubory, stačí spustit skript pro přegenerování při změně rozhraní knihovny. To značně minimalizuje riziko chyb a zjednodušuje úpravu knihovny při přidávání funkcionality.

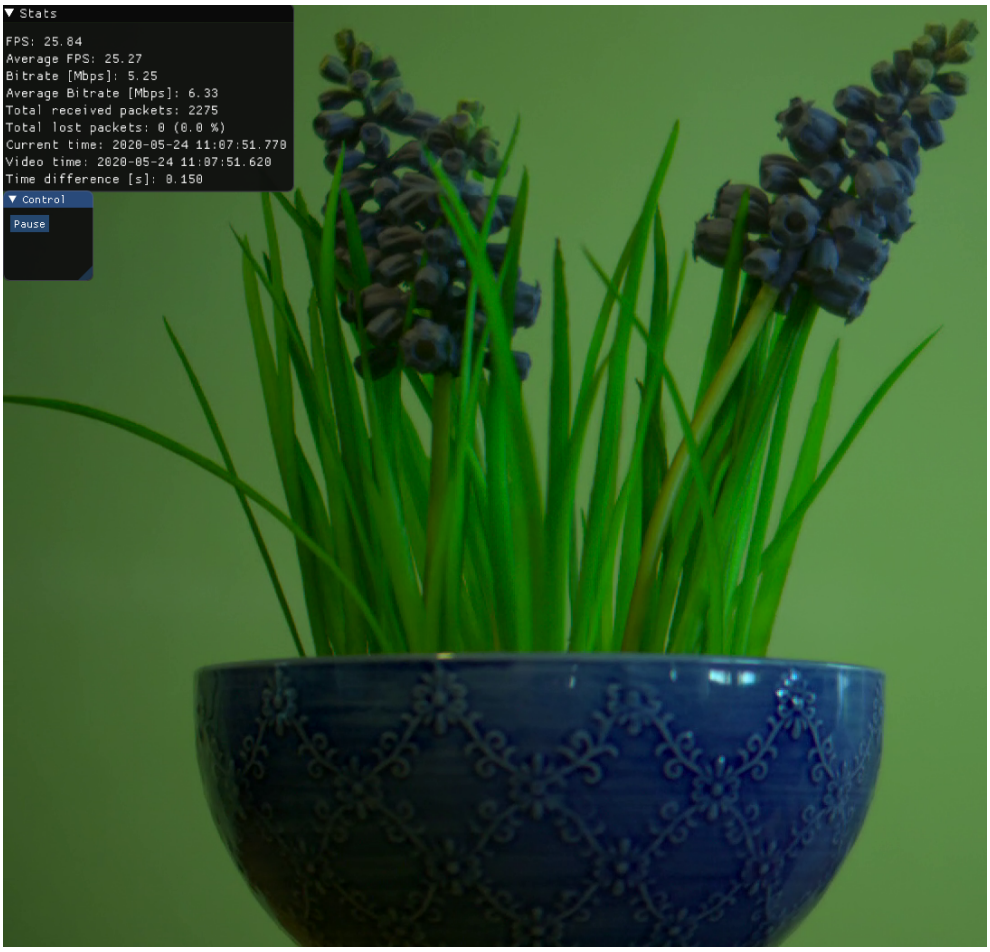
V ekosystému C# se nejčastěji využívá balíčkovacího systému Nuget [2]. Proto je vhodné pro pohodlné využití knihovny vytvořit pomocí nástroje Nuget balíček (v podstatě archiv ve formátu zip se všemi potřebnými soubory), který bude moct být importován do různých externích projektů, které potřebují klientkou knihovnu. V tomto případě není tak jednoduché vytvoření funkčního balíčku, jako když je balíček složen pouze z tzv. managed kódu. Tedy platformně nezávislého kódu napsaného v C# nebo v jiných jazycích podporující .NET. Tato knihovna potřebuje nativní sdílené knihovny a navíc pro každou platformu a architekturu jiné. Klientská knihovna je tedy přeložena pomocí nástroje CMake pro různé

[2] https://www.nuget.org/

konfigurace a výsledné binární soubory jsou přibaleny do balíčku Nuget spolu se zkompilovanými C# soubory.

4.5 Testovací video přehrávač

Součástí projektu klienta je i testovací video přehrávač převážně sloužící pro účely vývoje a ladění. Jeho základ je postaven na frameworku SDL2, který zajišťuje snadnou přenosnost mezi různými operačními systémy. Pro vypsání informací o video streamu se používá knihovna Imgui. Ta předefinovává několik základních stavebních prvků pomocí, kterých lze jednoduše vytvořit grafické rozhraní aplikace (obrázek 4.7). Tato knihovna ovšem potřebuje ke svému běhu jeden z podporovaných vykreslovacích frameworků, proto přehrávač vyžaduje ke svému běhu ještě OpenGL. Pro možnost využití OpenGL musíme dále využít jednu z knihoven, které definují hlavičkové soubory obsahující jeho aplikační rozhraní. V tomto případě bylo využito generátoru Glad. Ten umožňuje podle zvolené verze OpenGL vygenerovat soubory, které lze jednoduše přidat do projektu a využívat tuto knihovnu.



Obrázek 4.7: Grafické rozhraní testovacího přehrávače

Kromě zmíněných frameworků přehrávač využívá navržené klientské knihovny pro přijímání a dekódování obrazu. Aplikace však navíc přidává ještě jedno vlákno pro vykreslování

rozhraní. To zajistí responzivnost aplikace i v případě, že snímky nějakou dobu nebudou přijímány. Klientská knihovna totiž za normálních okolností blokuje program v případě, že je fronta snímků prázdná. V hlavním vlákně aplikace se tedy nejprve připojí na specifikovaný video server a zjistí se parametry streamu. V případě úspěšného připojení se pak nastartuje dodatečné vlákno, do kterého se následně posílají snímky přijaté v hlavním vlákně. V tomto vykreslovacím vlákně se na začátku vytvoří nové okno v operačním systému prostředníctvím knihovny SDL2. V něm se inicializuje kontext OpenGL knihovny a konečně i nastaví Imgui. Kromě toho se musí před samotným zobrazením videa vytvořit OpenGL textura, ve které budou uloženy jednotlivé snímky. Tu je potřeba pro správnou funkcionalitu inicializovat správnými parametry jako formát snímků (RGB formát) a jejich velikost. Poté už stačí do vytvořené textury nahrávat jednotlivé snímky a ty vykreslovat. Kromě přehrávání videa se toto vlákno stará o zobrazení primitivního uživatelského rozhraní vytvořeného pomocí Imgui. To obsahuje jedno malé okno s různými užitečnými textovými informacemi o streamu. Konkrétně zobrazuje:

- Snímkovací frekvenci
- Průměrnou snímkovou frekvenci
- Velikost datového toku v Mbit/s
- Průměrná velikost datového toku v Mbit/s
- Počet přijatých RTP paketů
- Počet ztracených RTP paketů a procenta z celkového počtu paketů
- Aktuální čas v počítači
- Časová značka aktuálně zobrazeného snímku
- Rozdíl času videa a aktuálního času v sekundách

Tyto informace můžou být užitečné pro identifikaci různých chyb jako v případě, že server přiřazuje špatný časový údaj ke snímkům nebo se po síti ztrácejí RTP pakety se snímky. Údaje jsou vykreslovány do Imgui okna, s kterým lze v případě potřeby pohybovat nebo ho sbalit, aby nám nepřekáželo. Pokud by bylo potřeba, tak díky jednoduchosti knihovny lze snadno doimplementovat zobrazování dalších informací. Kromě textových dat, lze tak v budoucnu vykreslovat i například historii snímkové frekvence pomocí grafů. Kromě okna s informacemi aplikace ještě obsahuje panel s tlačítkem pro pozastavení streamu.

V případě, že máme k dispozici pouze příkazovou řádku, aplikace podporuje i ladící mód bez grafického zobrazení. V tomto případě vypisují do konzole všechny výše zmíněné informace o videu. To slouží především pro otestování správného navázání spojení a posílání dat. Textový mód lze vynutit i při startu přepínačem -nodisp.

4.6 Knihovna pro nastavování parametrů serveru

Knihovna pro nastavování či zjišťování parametrů ze serveru je zcela oddělená od zdrojového kódu pro přijímání videa. Je na něm zcela nezávislá. Implementuje totiž RTSP protokol sama a je zbytečné je nějakým způsobem svazovat. Díky tomu je možné jejich nezávislé použití. Můžeme tak nastavovat vlastnosti streamu i z jiného místa, než odkud přijímáme

video bez nutnosti závislosti na knihovně pro přijímání snímků. To zajistí i menší velikost výsledné aplikace.

Protože má být opět knihovna využívaná v rámci aplikace napsané v jazyce C#, je v témže jazyce napsaná i tato knihovna pro co nejjednodušší integraci. Navíc tento jazyk disponuje řadou knihoven, které lze s výhodou využít při implementaci. Jádrem knihovny bude RTSP klient umožňující komunikovat s RTSP serverem přes TCP protokol. Přes něj se budou posílat příkazy pro nastavování nebo vyčítání parametrů serveru. Knihovna tak bude v základu implementovat pouze RTSP metody SET_PARAMETER a GET_PARAMETER. V případě potřeby však bude možné jednoduše dopsat i další metody (např. metodu DESCRIBE pro zjištění informací o video streamu).

Jak bylo řečeno klient se bude připojovat přes protokol TCP, takže pro implementaci klienta můžeme využít třídu TcpClient ze standardní knihovny. Tato třída obsahuje pouze metody pro navázání či ukončení spojení, pro samotnou komunikaci se musí využít NetworkStream. Knihovna klienta využívá jak pro navázání spojení i další komunikaci asynchronní varianty všech metod u tříd TcpClient, NetworkStream a StreamReader (využito pro čtení z NetworkStream objektu po řádcích). Díky tomu se může razantně zvýšit výkon aplikace. Zvláště v případě, pokud bychom chtěli ovládat více serverů najednou. Při správné integraci totiž může překladač C# dobře optimalizovat jednotlivé operace, což může zvýšit efektivitu kódu.

Každý požadavek se posílá ve standardním formátu RTSP (viz sekce 2.5). Tedy nejprve se pošle hlavička s informacemi o délce těla, sekvenčním číslem v rámci připojení začínajícím od nuly, typem RTSP metody, adresy a verze RTSP protokolu. Poté následuje samotné tělo. To v případě nastavování parametrů obsahuje název parametru a jeho hodnotu. Pokud chceme vyčítat hodnotu, tak stačí do těla přidat všechny názvy parametrů, pro které chceme zjistit hodnoty. Server by měl poté odpovědět s názvy úspěšně nastavených parametrů. V případě vyčítání pak s hodnotami požadovaných parametrů. Po poslání příkazu tak musíme počkat na odpověď serveru, abychom zjistili hodnoty nebo zda se parametry podařily nastavit podle očekávání. V případě, že nám RTSP protokol poslal jinou návratovou hodnotu než 400 (což podobně jako u HTTP protokolu značí úspěch), můžeme z odpovědi zjistit z jakého důvodu se daná operace nezdařila (např. jméno parametru nemusí být platné).

Kapitola 5

Testování a vyhodnocování výkonnosti

Při nasazování jakéhokoli softwaru je nutné znát jeho možnosti, spolehlivost a jiné důležité parametry. Existují různé vlastnosti, které nás zajímají u multimediálního streamovacího serveru. Záleží především na druhu a způsobu aplikace. V tomto konkrétním způsobu nasazení pro zpracování obrazu umělou inteligencí v reálném čase se musíme zaměřit na několik základních měřitelných veličin, které se rozeberou v následujících sekcích.

Všechna měření probíhala s nastaveným rozlišením videa na 1920 × 1080 pixelů. Kromě toho byl H.264 enkodér nastaven do módu VBR (viz sekce 2.3) s nastaveným minimálním datovým tokem 5 Mbit/s, cílovým 6,4 Mbit/s a maximálním 7,2 Mbit/s. S těmito parametry se zdálo video mít dostatečnou kvalitu (nebyla téměř postřehnuta změna způsobená komprimací). Dále při všech měřeních byl multimediální server spuštěn na počítači RockPro64 popsaného v sekci 2.4. V případě měření škálovatelnosti se pak připojují RTSP klienti z dvou pracovních stanic. Méně výkonný notebook měl následujícími parametry:

- Model – Notebook HP
- Procesor – Intel Core 2 Duo Processor P8400 2,26 GHz
- Operační paměť – 4 GB
- Operační systém – Windows 10 Home 64-bit
- Síťová karta – USB 3.0 Digitus DN-3023 (notebook podporuje USB 2.0)

Ten byl využit jen pro zatížení multimediálního serveru. Zatímco druhý počítač také zatěžoval, ale navíc na něm byla prováděna různá měření. Tento způsob musel být zvolen z důvodu zlepšení přesnosti. V případě, že byl použit jeden měřící počítač a server zároveň zatěžoval větším počtem klientů, tak se do výsledků měření značně promítl vliv ztráty paketů. Jeden počítač totiž nestačil zpracovávat všechny přijímané pakety. Druhá stanice disponuje následujícími parametry:

- Model – Notebook Dell Latitude E5570
- Procesor – Intel Core i7-6820HQ 2,7 GHz
- Operační paměť – 16 GB

- Operační systém – Windows 10 Pro 64-bit
- Síťová karta – Intel® Ethernet Connection I219-LM

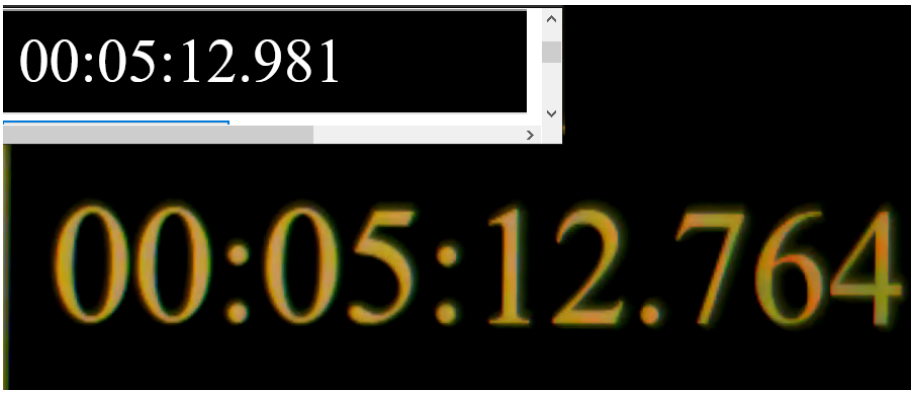
K tomuto počítači byl RockPro64 připojený přes kabel skrze USB 3.0 síťovou kartu do switche TP-Link TL-SG1005D. Do něj se pak připojily také pracovní stanice pro zatěžování serveru a měření. Tento přístup byl vybrán s ohledem na to, že komunikaci nemůžou ovlivňovat jiná okolní zařízení v sítí a tím se vyloučí určitá část chyb při měření. V případě, že byla měřena veličina pouze s připojeným jedním klientem serveru, tak se RockPro64 počítač zapojil přímo do měřícího počítače.

5.1 Měření latence

Jedna z nejdůležitějších veličin, které sledujeme u živých vstupů (např. videokonference), se nazývá latence. Celková latence streamu od kamery k výstupu (divák nebo zpracovávající aplikace) se skládá z několika dominujících částí. Ze zpoždění příjmu snímků z kamery, komprimace videa, doby cesty po daném médiu k cíli, dekódování přijatých snímků a jejich zobrazení. Vyskytují se i různé další složky, ale ty již celkovou latenci neovlivňují v takové míře.

Celková latence daného streamu se dá jednoduše změřit. V případě, že kamera i zobrazovací zařízení přijímající video z multimediálního serveru se nacházejí v jedné lokaci. Na počítači, kde přijímáme video, můžeme spustit stopky ukazující čas s přesností na milisekundy a kameru na tyto stopky namířit. Následně stačí udělat snímek obrazovky, kde se nachází jak čas stopek tak i čas na snímků přijatého z multimediálního serveru. Tyto časy odečteme a máme celkovou latenci. Zmíněný způsob je jeden z nejúčinějších způsobu měření dané veličiny, protože zohledňuje přesně zpoždění, než se video zobrazí divákovi z kamery. Hlavní nevýhoda ovšem spočívá v tom, že spolehlivě nezměříme čas u přechodových jevů (při skokovém zatížení serveru apod.), či pokud nemáme k dispozici zobrazovací zařízení.

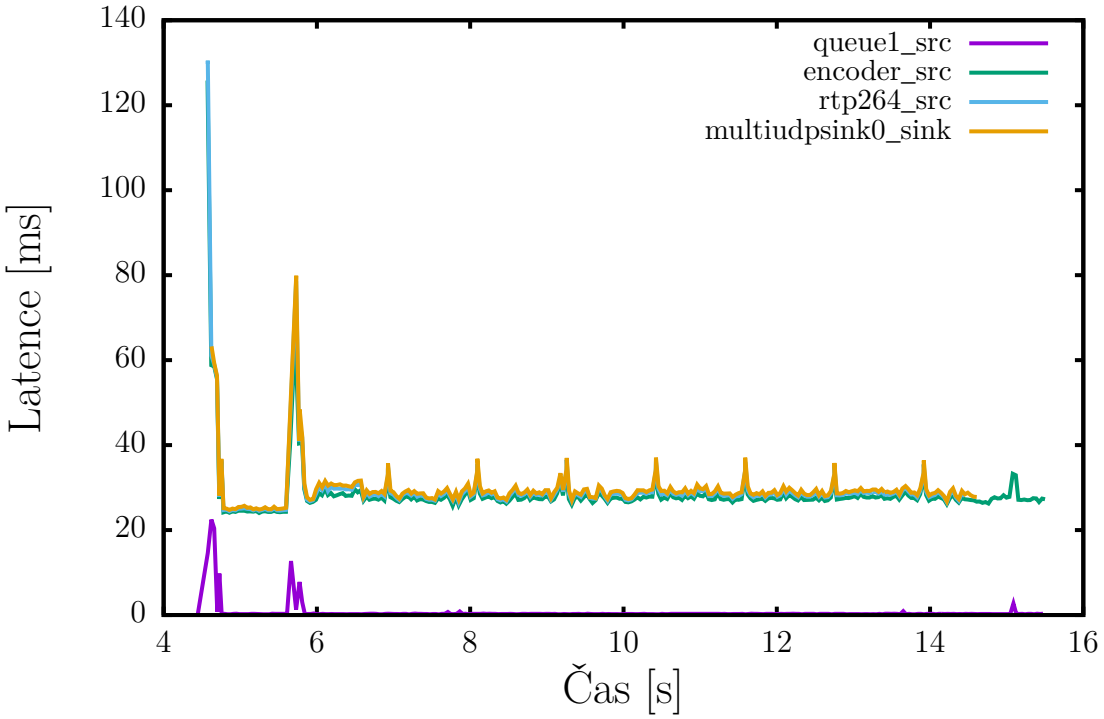
Obrázek 5.1 byl pořízen při měření latence. Vlevo nahoře se nacházejí spuštěné stopky a dole na počítači zobrazující video stream. V tomto videu jsou zachyceny spuštěné stopky, ale zobrazené v testovacím přehrávači přijímající video z kamery ze serveru. Odečtením časů na stopkách $981 - 764$ dostaneme celkovou latenci, která činí 217 milisekund.



Obrázek 5.1: Snímek měření celkové latence streamování

Dalším postupem lze analyzovat latence jednotlivých částí při distribuci z kamery až ke klientovi přijímající video. Ze znalosti jednotlivých časů se lze zaměřit na konkrétní

prvky, které je potřeba optimalizovat a tím výrazněji snížit celkovou latenci. Nejjednodušší v případě tohoto systému je změření časů v multimediální pipeline zpracovávající video z kamery na serveru. Díky nástroji GstShark lze totiž jednoduše změřit latence jednotlivých elementů, jimiž prochází video snímky. Obrázek 5.2 však nezachycuje zpoždění jednotlivých prvků, ale jejich kumulativní latenci. To znamená, že změří čas snímku od vstupu do pipeline až po čas kdy dorazí na výstup jednotlivých elementů. Toto zpoždění je pak vyneseno na následující graf. Z něho je zřejmé, že výrazné zpoždění po prvotním ustálení vytváří enkodér do formátu H.264. Po odečtení z grafu tak dostaneme průměrně 25 milisekund pro průchod celou pipeline.



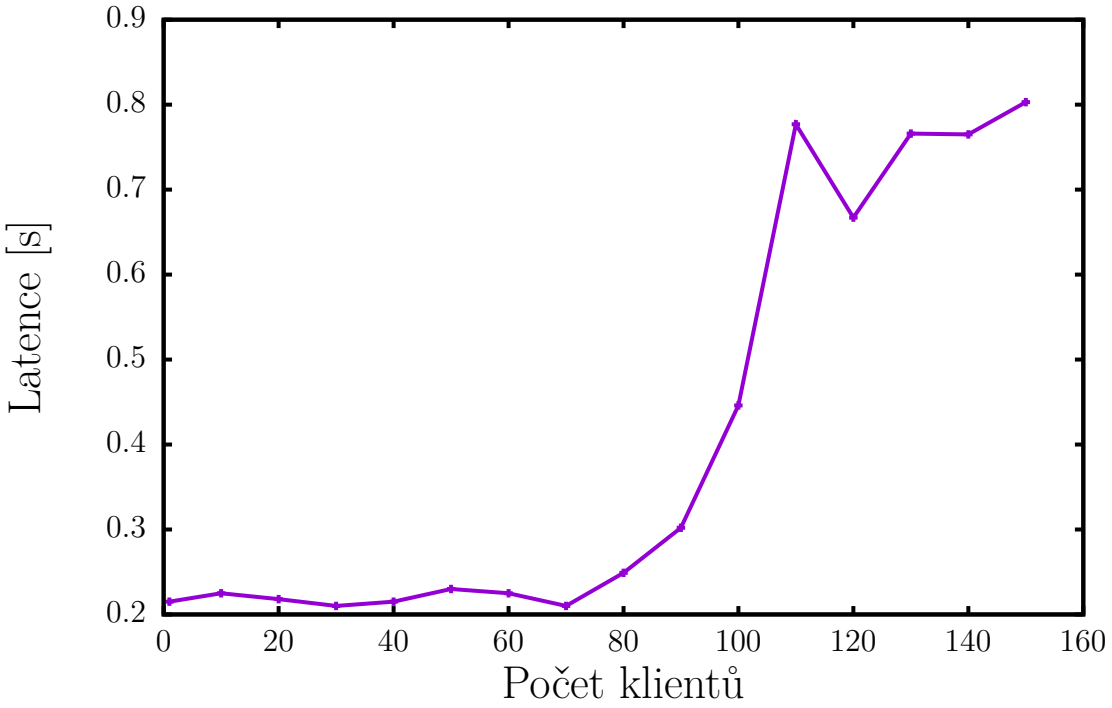
Obrázek 5.2: Graf kumulativní latence jednotlivých elementů v pipeline

Pro další měření bylo nutné nastavit NTP server na měřicí stanici pro synchronizaci času s přesností na jednotky milisekund. S tímto NTP serverem se pak pomocí linuxové utility ntpdate seřídil čas na počítači RockPro64. Tím se zajistí, že časová značka RTP paketů by se měla sjednotit s časem měřícího počítače. Při každém zachycení snímku z kamery se totiž k aktuálně pořízenému snímku přiřadí časová značka. Díky tomu můžeme změřit zpoždění od pořízení až ke zobrazení daného snímku. Testovací přehrávač totiž umí vypsat časový rozdíl mezi aktuálním časem a časovou značkou snímku. Průměrná naměřená hodnota časového rozdílu dosahovala 141 milisekund. Do tohoto zpoždění spadá ovšem i čas průchodu snímku GStreamer pipeline (kolem 25 milisekund). Tedy na odeslání, přijmutí, dekódování snímku zbývá celkem 116 milisekund. Kromě toho lze také odvodit čas nutný pro pořízení a zachycení snímku serverem. Celkové zpoždění od zachycení snímku po zobrazení bylo naměřeno 217 milisekund. Pokud odečteme od této hodnoty předchozí naměřený časový rozdíl (141 milisekund), dostaneme dobu zachycení snímku 76 milisekund.

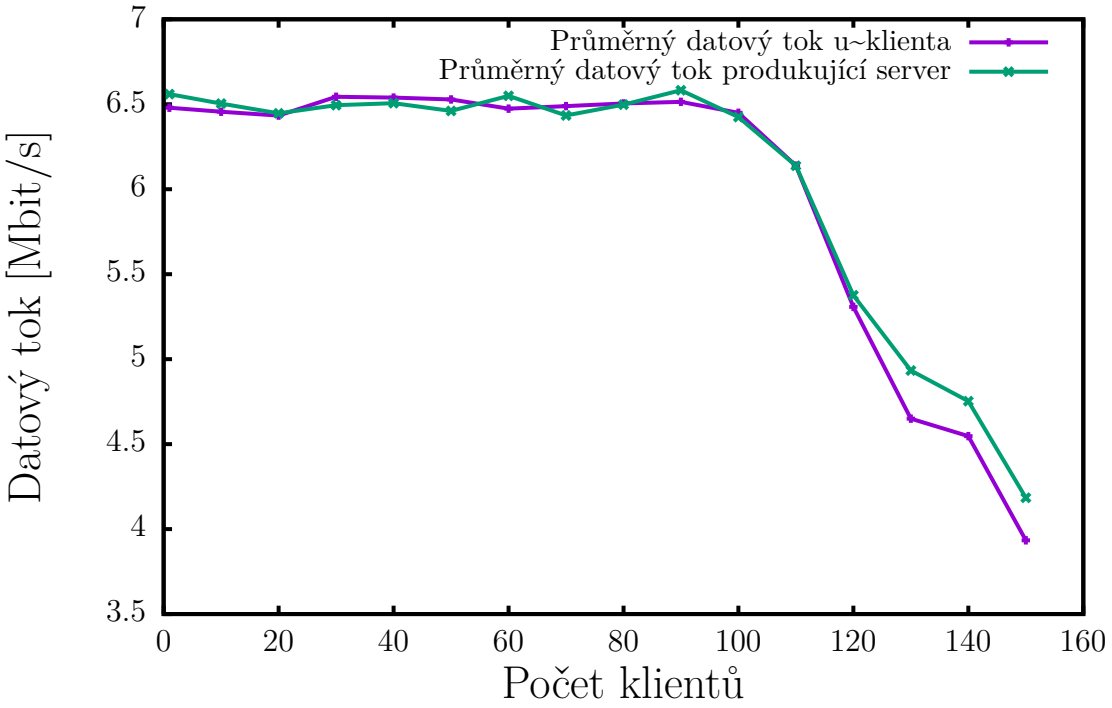
5.2 Měření škálovatelnosti

Další často sledovanou metrikou je škálovatelnost serveru. Ta stručně řečeno definuje kolik klientů server zvládne obsloužit bez výrazného ovlivnění kvality streamovaného videa. V tomto případě ji sledujeme za účelem zjištění kolik tzv. operátorů systému pro automatické testování (více viz kapitola 1) se může připojit na multimediální server zachytávající video z kamery. Tato informace se pak může hodit pro určení, zda je systém pro dané použití dostačující. Kromě toho lze multimediální server upravit takovým způsobem, aby server po připojení určitého maximálního počtu klientů další odmítl. Tedy se omezí celkový počet připojených klientů a tím se zaručí, že kvalita reprodukce neklesne pod určitou mez. Měření škálovatelnosti se může provádět různými způsoby podle toho, které veličiny nás u videa zajímají. V našem případě je kritická latence a snímková frekvence. Primárně se tedy zaměříme na zjištění těchto veličin při daném počtu klientů.

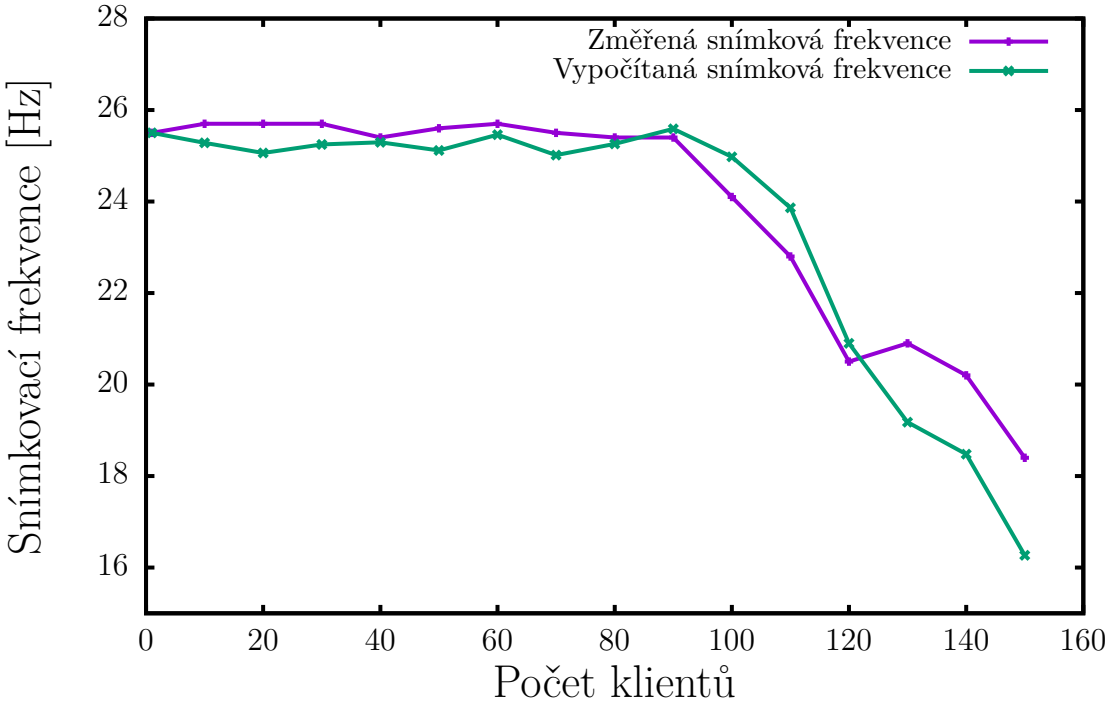
Pro usnadnění a urychlení měření škálovatelnosti byl vytvořen skript v jazyce Powershell od společnosti Microsoft. Ten přes protokol ssh na počítači RockPro64 spustí multimediální server zachytávající video z kamery, na který se vzápětí pomocí nástroje openRTSP připojí určitý počet klientů. Utilita openRTSP byla použita s ohledem na její efektivnost implementace. Na jedné měřící stanici totiž potřebujeme zajištění běhu většího počtu klientů. Po připojení klientů je dále dobré počkat nějaký čas pro ustálení přechodu. Nárazové připojení velkého počtu klientů, by totiž mohlo ovlivnit výsledky měření. Proto bylo měření vždy započato s časovou rezervou 10 sekund. Následně se provedly různé typy měření.



Obrázek 5.3: Závislost počtu připojených klientů na latenci videa změřeného u klienta



Obrázek 5.4: Závislost počtu připojených klientů na odchozím datovém toku ze serveru a také datový tok přepočítaný na vzorkovací frekvenci



Obrázek 5.5: Závislost počtu připojených klientů na příchozím datovém toku změřeného u klienta

Jednou z nejdůležitějších parametrů je latence, kterou jsme již měřili pro jednoho klienta. V tomto měření ovšem se nejdříve připojí daný počet klientů a poté podle postupu popsaného v sekci 5.1 změříme latenci. Tedy pomocí testovacího přehrávače a stopek odečteme zpoždění. Měřit latenci tímto způsobem lze ovšem pouze na velmi omezeném počtu klientů. Připojených klientů totiž bude více, ale zároveň takto nelze jednoduše zajistit změření zpoždění na více než jednom z nich. To může zavést chybu do měření. Pro minimalizaci chyb, ale bylo provedeno několik měření a jejich hodnoty zprůměrovány. V grafu 5.3 jsou zaneseny naměřené hodnoty pro až 150 klientů (pro větší připojených klientů už nebylo možné spolehlivě měřit latenci kvůli kvalitě přijímaného videa v testovacím přehrávači). Z něj je patrné, že do 80 klientů, se zpoždění drží kolem 200 milisekund. Při větším množství připojených klientů ovšem latence začne stoupat a nad 90 klientů se strmě zvětšuje až ke 600 milisekundám. Do méně než 80 klientů je tedy multimediální server použitelný s minimální újmou na zpoždění videa.

Pro ověření stability serveru je potřeba změřit i další metriku. Jednou z často sledovaných metrik bývá snímková frekvence. Ta určuje kolik snímků za sekundu dokáže server poslat všem klientům. Proto abychom vyloučili chyby spojené s měřením hodnot pouze na jednom klientovi, které mohli vzniknou v případě měření latence, nabízí se možnost měření datového toku posílaného ze serveru. Nemusíme totiž znát absolutní hodnotu snímkové frekvence, ale stačí nám její relativní změna s počtem připojených klientů.

Pro změření datového toku na straně serveru byl napsán skript network_speed.sh v jazyce Bash. Ten vyčítá ze souboru /proc/dev/net údaje o množství poslaných bytů na určité síťové rozhraní. Skript umožňuje zprůměrování datového toku za specifikovanou časovou jednotku, takže vyčte údaje na začátku, na konci měření a z těchto hodnot vypočte průměrný datový tok za jednu sekundu. Kromě průměrného datového toku také dokáže periodicky vypisovat aktuálně využívanou šířku pásma na zadaném síťovém rozhraní. Toto dokáží i nástroje typu Glances, tento nástroj ovšem zbytečně nevytěžuje procesor. A navíc lze vypisované hodnoty snáze dále zachytávat a zpracovávat. Skript totiž do terminálu vypisuje jen naměřené hodnoty.

Začátek měření bude probíhat obdobně jako u zjišťování latence. Nejprve pomocí testovacího skriptu spustíme přes ssh na RockPro64 multimediální server, na který se vzápětí připojí daný počet klientů. Pro usnadnění měření byl ovšem testovací skript v tomto případě ještě upraven. Navíc ještě po připojení klientů vzdáleně spustí skript network_speed.sh, který zprůměruje datový tok za 10 sekund. Tento datový tok se pak ještě vydělí počtem klientů, abychom dostali průměrnou šířku pásma na jednoho klienta. Tato hodnota pak byla zanesena do grafu 5.4. Ten ukazuje obdobnou tendenci jako u změřené škálovatelnosti u latence. Do zhruba 100 klientů se datový tok na jednoho klienta drží kolem 6,5 Mbit/s a poté razantně klesá, kdy pro 150 klientů byla naměřena hodnota 4,2 Mbit/s, což odpovídá poklesu snímkové frekvence až o 35 %. Při tomto testování byla dále naměřena průměrná snímková frekvence snímků kolem 25,5 Hz v případě připojeného jednoho klienta. Pokud odečteme procentuální pokles, tak dostaneme výslednou snímkovou frekvenci 16,3 Hz při 150 připojených klientech.

Informace o výstupní snímkové frekvenci je důležitá pro určení meze výkonnosti serveru. Z hlediska určení kvality přijímaného videa je ovšem zásadní měření na straně klienta. Zde lze zase měřit obdobným postupem jako výstupní datový tok serveru. Tedy spustíme server, připojíme daný počet klientů a po ustálení změříme průměrný přijímaný datový tok na klientském počítači. Ten zase přepočítáme na jednoho klienta. Průměrná hodnota toku je pak zanesena do grafu na obrázku 5.4. Pro porovnání s datovým tokem serveru byla vykreslena do stejného grafu jako datový tok směřující ze serveru. Tyto dvě křivky

vykazují téměř stejné chování, avšak od 120 připojených klientů pravděpodobně měřicí stanice výkonnostně nestačí. To se projevuje menším datovým tokem na straně klienta. Tedy stanice nestíhá přijímat veškeré pakety a ostatní zahazuje.

Pro ověření správnosti naměřených hodnot datového toku a odvozené snímkové frekvence bylo provedeno dodatečné měření. Začátek probíhal stejně jako v ostatních případech měření škálovatelnosti. Pouze se po přípojení daného počtu klientů navíc spustil testovací přehrávač, ze kterého se odečetla průměrná snímková frekvence. Na grafu 5.5 je vykreslena jak změřená tak vypočítaná snímková frekvence. Měření ukazuje, že se chyba těmito dvěma způsoby začne ve větší míře projevovat, pokud připojíme více než 120 klientů. To může souviset se ztrátou paketů na straně měřící stanice.

5.3 Zhodnocení měření

Byla provedena různá měření pro zhodnocení kvality navrženého multimediálního serveru. Na základě nich lze určit, zda server splňuje kladené nároky. Navíc také hranice, při kterých je server ještě použitelný za daných požadavků. V případě připojení jednoho klienta se latence pohybovala kolem 217 milisekund od pořízení obrazu až po vykreslení v testovacím prohlížeči. Což se zdá použitelné v případě nasazení tohoto softwaru v rámci nadřazeného systému určeného pro testování vestavěných zařízení zmíněného v úvodu. Systém musí totiž v relativně krátké době zpracovávat informace z kamery a na základě nich reagovat. Tento čas tak bude značně figurovat v celkové délce trvání testování. Subjektivně se však zdá, že toto zpoždění by mohlo dostačovat. Kromě toho byly změřeny jednotlivé složky tohoto zpoždění. Ty se mohou hodit pro budoucí optimalizaci v případě, že by dosažená latence nestačila. Tabulka 5.1 obsahuje naměřené hodnoty pro jednotlivé složky latence. Díky tomu lze zjistit, že nejvýrazněji do velikosti zpoždění zasahuje odesílání, přijímání, dekódování a zobrazení video. V případě budoucí optimalizace se lze tak zaměřit výhradně na tuto část.

Tabulka 5.1: Shrnutí měření jednotlivých složek latence

Zachycení snímku	Průchod GStreamer pipeline	Odeslání, přijmutí, dekódování a zobrazení	Celkem
~76 ms	~25 ms	~116 ms	217 ms

Pro ověření funkčnosti serveru byla změřena i škálovatelnost řešení při připojení více klientů. Pro stanovení limitů, při kterých se multimediální server může využívat bez ztráty kvality, musíme zanalyzovat naměřené hodnoty a určit jejich maximální práh. Latence serveru je použitelná do zhruba 80 připojených klientů, kdy se pořád drží kolem 200 milisekund. Dále už latence neúnosně stoupá.

Tabulka 5.2: Parametry video streamu při daném počtu připojených klientů

	1 klient	80 klientů	150 klientů
Latence	217 ms	249 ms	803 ms
Snímkovací frekvence	25,5 Hz	25,3 Hz	16,3 Hz

Další důležitou změřenou veličinou byla výstupní snímková frekvence, která se drží kolem 25 Hz do 100 klientů, poté strmě klesá. To je pravděpodobně způsobené tím, že server nestíhá posílat pakety přes sockety klientům. GStreamer element multiudpsink totiž využívá pouze jedno vlákno pro posílání paketů. V případě požadavku na obsluhu většího počtu klientů,

by tak bylo zapotřebí použít jiný element pro posílání RTP paketů. Podle naměřených hodnot je tak nejvíce restriktivní latence, takže lze bez výrazného poklesu kvality obsluhovat maximálně 80 klientů najednou. Tento úpadek metrik při větším množství připojených klientů je nejspíše způsoben tím, že knihovna gst-rtsp-server používá pro posílání RTP paketů element multiudpsink. Ten využívá pouze jedno vlákno pro posílání dat do socketů. Při připojených 100 klientech tak musí jediné vlákno posílat do socketů zhruba 650 Mbit/s. To nestíhá a tak se zvětšuje latence a snímková frekvence se naopak zmenšuje. V případě požadavku na lepší škálovatelnost, se tak musí upravit knihovna gst-rtsp-server a případně element multiudpsink pro posílání dat klientům. Tabulka 5.2 shrnuje naměřené hodnoty pro 1, 80 a 150 klientů.

Nakonec lze díky naměřenému průměrnému datovému toku zhodnotit jeho úsporu bez komprimačního zařízení a s ním. Díky tomu můžeme zjistit, jak moc odlehčíme datovým linkám, po kterých proudí snímky videa, při použití komprimačního serveru. Výstupem multimediálního serveru je 6,5 Mbit/s při změřené průměrné snímkové frekvenci 25,5 Hz s jedním připojeným klientem. Dále lze zjistit datový tok produkující kamerou. Ta byla nastavena tak, že produkuje snímky ve formátu YUV422 s 16 bity na pixel. Pokud spočítáme celkový datový tok při rozlišení 1920×1080 a při frekvenci 25,5 Hz, dostaneme 846 Mbit/s. To odpovídá zhruba 130násobné úspoře dat,

Kapitola 6

Závěr

Na základě popsaných technologií byl vytvořen systém pro redukci datového toku pro posílání videa napříč internetovou sítí. Ten se skládá z kamery, serveru a případných klientů. Server s podporou frameworku GStreamer z kamery zachytává video přes protokol GigE Vision a komprimuje ho pomocí kodeku H.264, jedním z dnes široce používaných formátů. Pro další distribuci videa byla zvolena kombinace protokolů RTSP a RTP zajišťujících přenos snímků. Přijímání přes tyto protokoly a dekódování videa zajišťuje framework FFmpeg. Ten využívá implementovaná knihovna, která značně zjednodušuje vytvoření aplikace pro další zpracovávání obrazu. Přestože byla knihovna naprogramována v jazyce C, pomocí vygenerované vrstvy ji lze využívat i v projektech postavených na jazyce C#. Dále systém umožňuje i nastavení parametrů videa na straně serveru. Klientům tak umožňuje například nastavení parametrů kamery podle aktuálních potřeb pomocí specializované knihovny.

Implementovaný systém již byl částečně integrován do aplikace pro automatické testování dotykových zařízení, kde má komprimovat datový tok z kamery, která poskytuje zpětnou vazbu od testovaného objektu. Díky tomu lze zpracovávat video i v datacentrech bez přebytečného zahlcování síťové infrastruktury. Dle naměřených výsledků tak systém při zachování dostatečné kvality obrazu sníží přenášený datový tok 130krát, což kromě úspory šířky pásma přináší i možnost připojení většího počtu klientů k multimediálnímu serveru. Při používání takového systému je potřeba navíc znát i další parametry. Proto byl implementován testovací přehrávač zobrazující různé informace o video streamu. Ten sloužil spolu s několika skripty pro provedení řady měření zkoumající různé metriky. Zásadní v tomto případě bylo změření škálovatelnosti, snímkové frekvence a latence. Po přijetí videa klientem dosahují snímky latence kolem 200 milisekund a průměrné snímkové frekvence 25,5 Hz. Tyto parametry se téměř nemění až do počtu 80 klientů. Měření tak ověřilo, že systém splňuje kladené nároky na dané metriky. Ovšem také se lze díku němu jednodušeji zaměřit na části, které by musely být optimalizovány, pokud by požadované parametry přestaly stačit.

Literatura

[1] *GigE Vision® Specification version 2.0.* Standard. AIA, 2013.

[2] Felber, P. *10GigE: High-speed for your machine vision task.* Baumer Optronic GmbH, 2019.

[3] Ibraheem, N., Hasan, M., Khan, R. Z. a Mishra, P. Understanding Color Models: A Review. *ARPN Journal of Science and Technology.* Leden 2012, roč. 2.

[4] Kaminska, M. a Smihily, M. Cloud computing - statistics on the use by enterprises. *Eurostat.* 2018.

[5] Nelson, M. a Gailly, J.-L. *The Data Compression Book (2nd Ed.).* USA: MIS:Press, 1995. ISBN 1558514341.

[6] Podpora, M., Korbás, G. a Kawala Janik, A. YUV vs RGB – Choosing a Color Space for Human-Machine Interaction. *Annals of Computer Science and Information Systems.* Září 2014, roč. 3.

[7] Richardson, I. E. *The H.264 Advanced Video Compression Standard.* 2nd. Wiley Publishing, 2010. ISBN 0470516925.