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

SERVEROVÉ ŘEŠENÍ PRO KOMUNIKACI DAT MEZI DRONY

SERVER FOR DATA COMMUNICATION BETWEEN DRONES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JIŘÍ HERRGOTT**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. DANIEL BAMBUŠEK**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

144925

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Herrgott Jiří, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Kybernetická bezpečnost
Název:	**Serverové řešení pro komunikaci dat mezi drony**
Kategorie:	Počítačové sítě
Akademický rok:	2022/23

Zadání:

1. Prostudujte moderní postupy tvorby robustních serverů a analyzujte možnosti sdílení letových a multimediálních dat mezi operátory dronů.
2. Vyberte vhodné metody a nástroje a navrhněte server, který umožní sdílení živých letových a obrazových dat mezi drony a jejich operátory.
3. Navrženou aplikaci implementujte.
4. Proveďte experimenty a vyhodnoťte vlastnosti výsledného řešení.
5. Vytvořte video prezentující klíčové vlastnosti výsledného řešení.

Literatura:

- SEDLMAJER Kamil, BAMBUŠEK Daniel a BERAN Vítězslav. *Effective Remote Drone Control Using Augmented Virtuality.* In: Proceedings of the 3rd International Conference on Computer-Human Interaction Research and Applications 2019. Vienna: SciTePress - Science and Technology Publications, 2019, s. 177-182. ISBN 978-989-758-376-6.
- Dále dle pokynů vedoucího.

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:	**Bambušek Daniel, Ing.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	17.5.2023
Datum schválení:	31.10.2022

Abstrakt

Tato diplomová práce se věnuje přenosu letových a multimediálních dat mezi operátory a drony pro lepší plánování a koordinaci misí v reálném čase. Byl vytvořen program, který primárně poskytuje tyto služby. V práci je kladen důraz na robustnost, znovupoužitelnost a rozšiřitelnost, což umožňuje vytvoření modulů například pro detekci nebo 3D rekonstrukci z přenášených dat. V rámci této práce byly navíc vytvořeny moduly pro ukládání letových i multimediálních dat a modul pro detekci vozidel ze sdíleného obrazu. Byly provedeny experimenty pro vyhodnocení odezvy, potřebných výpočetních zdrojů a využité šířky pásma připojení.

Abstract

This master thesis focuses on the transfer of flight and multimedia data between operators and drones for better real-time mission planning and coordination. A program has been developed that primarily provides these services. The work emphasizes robustness, reusability and extensibility, allowing the creation of modules such as detection or 3D reconstruction from the transmitted data. In addition, modules for flight and multimedia data storage and a module for vehicle detection from shared images have been developed in this work. Experiments were performed to evaluate the response time, the required computational resources and the used connection bandwidth.

Klíčová slova

server, dron, operátor, DroCo, RTMP, WebSocket, nahrávání, MP4, FFmpeg, OpenCV, detekce vozidel, Boost, JSON

Keywords

server, drone, operator, DroCo, RTMP, WebSocket, recording, MP4, FFmpeg, OpenCV, vehicle detection, Boost, JSON

Citace

HERRGOTT, Jiří. *Serverové řešení pro komunikaci dat mezi drony.* Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Daniel Bambušek

Serverové řešení pro komunikaci dat mezi drony

Prohlášení

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

.....
Jiří Herrgott
14. května 2023

Poděkování

Chtěl bych poděkovat vedoucímu této diplomové práce, panu Ing. Danielu Bambuškovi, za všechny cenné rady a za odborné vedení.

Obsah

Kapitola 1

Úvod

Diplomová práce se zaměřuje na přenos letových a multimediálních dat mezi operátory dronů. Při složitějších misích, kde je využito více dronů, je užitečné sdílet letová a multimediální data z dronů. Tato data mohou být interpretována ve vizualizační aplikaci, která obsahuje mapové podklady a v reálném čase zobrazuje polohy dronů včetně multimediálních dat, jako např. video přenos z kamery dronu. Vizualizační aplikace tak může být využita pro koordinaci a plánování misí pozemních jednotek s pomocí operačního střediska. Vizualizační aplikace – DroCo [28] – zprostředkovává zobrazení těchto online dat z dronů do offline 3D modelu světa. Při vývoji této aplikace vznikl i jednoduchý server ve formě skriptu, který přenáší letová data broadcastem všem připojeným a zvládá přenášet pouze jeden video kanál a není rozšiřitelný (nelze napojit moduly, které by například prováděly zpracování dat z videa, detekce, 3D rekonstrukce, atd.). Tyto problémy by měla řešit právě moje práce.

Cílem je tedy vytvoření robustního programu běžícího na vzdáleném serveru, který umožňuje sdílení dat mezi operátory. Dále by server měl umožnit data uchovávat, aby bylo možné zpětně nahlédnout na letová i vizuální data. Zároveň by tato diplomová práce měla obsahovat přesně popsané využité protokoly a technologie, aby se detailně definovalo rozhraní pro komunikaci s klienty a rozhraní pro dodatečné moduly. Program byl primárně vytvořen a testován na klientské aplikaci DroCo.

V první kapitole popíši klientskou aplikaci DroCo, ze které vznikla motivace vytvoření této diplomové práce. Vysvětlím aplikaci jako takovou, její grafické rozhraní a poté její síťové rozhraní. Taktéž charakterizuji jednoduchý server, vzniklý v rámci testování této aplikace. Celkově se tedy lze v této kapitole dozvědět, co předcházelo k vytvoření této práce, která de facto se snaží nahradit a výrazně vylepšit tento již vytvořený jednoduchý server. Ve druhé kapitole se zaměřím na technologie vhodné k přenosu a záznamu letových a multimediálních dat. U každé technologie zhodnotím její výhody a nevýhody a porovnám je. Ve třetí kapitole navrhnu architekturu serverového programu tak, aby byl program robustní a zároveň, aby bylo možné program snadno udržovat a rozšiřovat. Dle těchto požadavků si rozdělím návrh řešení na individuální části – moduly. Odrazím se od druhé kapitoly a zvolím vhodné technologie pro řešení tohoto problému a svůj výběr zdůvodním. Čtvrtá kapitola je zaměřena na implementaci řešení, kterou jsem provedl na základě návrhu z předchozí kapitoly. Věnuji se zde implementaci společného rozhraní a následně jednotlivých modulů, které dohromady plní potřeby dle zadání. V poslední páté kapitole provedu experimenty zhotoveného řešení a vyhodnotím výsledky těchto experimentů vzhledem k účelu této diplomové práce.

Kapitola 2

Vizualizační aplikace pro více-dronové mise

V této kapitole představím klientskou aplikaci DroCo[1] [28], ze které vznikla potřeba této práce. Klientská aplikace slouží pro ovládání dronu s možností využití rozšířené virtuální reality. Tato aplikace umožňuje z pohledu třetí osoby nahlédnout na dron ve 3D scéně, která je tvořena z veřejně dostupných letových dat a využívá letová a vizuální data přenášená z tohoto dronu. Letová data slouží pro zajištění správné polohy vzhledem ke 3D scéně a vizuální data se promítnou na plátno před dron na místo, kde by objekty na obrazu měly přibližně odpovídat objektům z 3D scény, jak je možné vidět na obrázku 2.1. Součástí této aplikace je síťové rozhraní, které umožňuje operátorovi nahlížet na ostatní drony a mohl se tak podle nich korigovat. Operátorovi se zobrazuje i vzdálenost jeho dronu od ostatních, aby snadněji mohl předcházet případným kolizím. Aby se klientské aplikace dostaly k požadovaným letovým datům, byl vytvořen primitivní broadcast server, který bude nahrazen výsledkem této diplomové práce. Podobný problém byl řešen v publikaci [16].

K vyzkoušení této aplikace a otestování funkčnosti mého navrženého řešení, byl mně školou zapůjčen dron značky DJI. S pomocí dronu jsem vyzkoušel klientskou aplikaci DroCo a mohl jsem si tak ověřit funkčnost přenosu letových a multimediálních dat. Vyzkoušel jsem i zobrazení ostatních dronů pomocí klienta, který se připojil na server a posílal falešná letová data.

2.1 Grafické rozhraní

Aplikace má poměrně intuitivní rozhraní pro ovládání. Na obrázku 2.2 lze vidět rozhraní včetně modelu dronu z pohledu třetí osoby a 3D mapovou scénu, ve které se nachází vzhledem k aktuálním GPS souřadnicím. Součástí je ovládání kamery a spousta informací, které jsou potřeba pro bezpečnou a koordinovanou manipulaci s dronem. Na tomto obrázku se také vyskytují ostatní drony, jež jsou připojeny na stejný server a vysílají tedy svá letová data. Ostatní drony v zorném poli se zobrazí jako modely dronů s patrným ohraničením a vzdáleností od dronu operátora. Drony mimo zorné pole se znázorní jako šipky směřující k nim. Šipky se nacházejí u okrajů obrazovky společně se vzdáleností k dronu.

Součástí aplikace je nastavení připojení, jak k serveru pro získání letových dat, tak k video streamům pro získání obrazu od jednotlivých dronů. Nastavení se liší dle typu dronu.

[1] Aplikace vyvinutá skupinou Robo@FIT volně dostupná z: http://github.com/robofit/drone_vstool

Obrázek 2.1: Aplikace DroCo včetně plátna s video přenosem[4].

Aplikace podporuje drony využívající technologii ROS[2] a drony značky DJI. K výběru mezi těmito technologiemi slouží přepínač, který lze vidět na obrázku 2.3. Následující možnosti se pak upraví podle této volby. Při využití technologie ROS se musí nastavit adresa, na níž se nachází ROS Bridge[3] server. U dronů značky DJI je potřeba vyplnit v nastavení UDP [23] port, určený pro přenos video streamu a URL adresu serveru. Jelikož ROS Bridge i DroCo server využívají technologie WebSocket [19], URL musí být zadána ve formátu *ws://host[:port]*, jak lze vidět na obrázku 2.3.

2.2 Síťové rozhraní aplikace

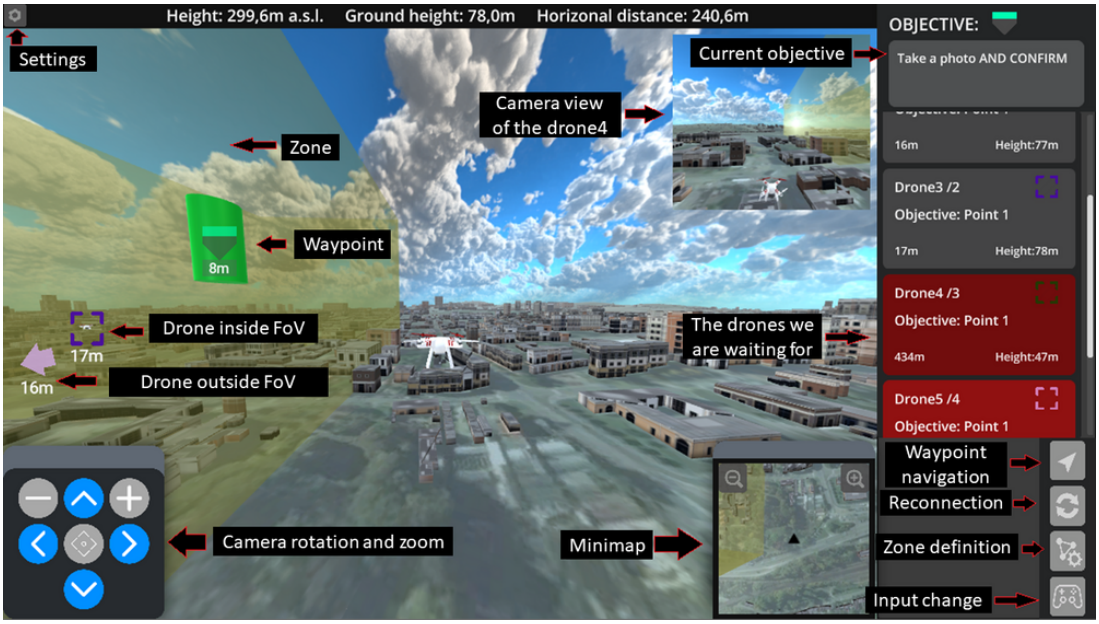
Síťové rozhraní aplikace se skládá ze dvou částí. První je přenos letových dat a druhou je přenos vizuálních dat. Pro přenos letových dat aplikace využívá protokolu WebSocket [19] a rozšířeného serializačního formátu JSON [2]. Aplikace se po zadání URL adresy serveru pokusí k němu ihned připojit. Po navázání spojení udržuje a přijímá zprávy. Jedinou zprávou je zpráva obsahující letová data dronu. Příklad formátu letových dat je možné vidět v bloku 1.

Z bloku 1 ukázky JSON zprávy lze vyčíst, že součástí letových dat je kromě GPS souřadnic v podobě zeměpisné šířky a délky, nadmořské výšky, také informace o rotaci dronu v prostoru v podobě Eulerových úhlů. Dále obsahuje úhlovou hodnotu kompasu a identifikační řetězec, který lze nastavit operátorem dronu. Aplikace tato data využívá pro zobrazení dronů ve 3D mapové scéně.

[2]Robot Operating System je open-source knihovna s nástroji, které slouží pro vytváření robotických aplikací. Knihovna je dostupná z: http://ros.org/

[3]ROS Bridge je protokol pro komunikaci se zařízeními používající ROS technologii. Specifikace protokolu včetně serverové implementace se nachází na: https://github.com/RobotWebTools/rosbridge_suite

[4]Obrázek je součástí produktu DroCo a je dostupný z [navštíveno 12.05.2023]: http://www.fit.vut.cz/research/product/647/



Obrázek 2.2: Znázornění uživatelského rozhraní aplikace DroCo[7].

Druhou částí je přenos vizuálních dat prostřednictvím streamovacích technologií. Pro navázání spojení je potřeba spustit GStreamer[5] pipeline, která přeposílá vizuální data streamovaná dronem na klientskou aplikaci, kde jsou data zobrazena v reálném čase. Aplikace využívá integrovaného GStreamer klienta[6] pro přenos dat. GStreamer používá pro přenos dat protokol RTP [26] nad protokolem UDP [23]. Jako nevýhodu považuji, že je nutné spouštět tuto pipeline pro převod mezi protokolem, přes který vysílá dron svá multimediální data na protokol RTP, který vyžaduje tato aplikace. Tedy uživatel musí při každém novém připojeném dronu manuálně spustit GStreamer pipeline, aby přenesla multimediální data od dronu k vizualizační aplikaci. Další nevýhodou je, že GStreamer pipeline se musí spouštět ze stejné sítě jako je vizualizační aplikace, protože protokol RTP není schopen překonat NAT (Network Address Translation).

2.3 Primitivní server

V rámci vyzkoušení přenosu letových dat síťového rozhraní, byl již dříve vyvinut jednoduchý Python program[8], který slouží jako broadcast server přes WebSocket [19] protokol. Server naslouchá na zadaném portu a po navázání spojení veškeré přijaté zprávy přeposílá všem ostatním připojeným klientům. Tedy každá aplikace operátora dronu vysílá letové zprávy na tento server, který je přeposílá všem včetně klientským aplikacím, jež tato data využívají.

[5]GStreamer je rozsáhlá open-source aplikace s různými nástroji pro zacházení s multimediálními daty. Aplikace je dostupná z: http://gstreamer.freedesktop.org

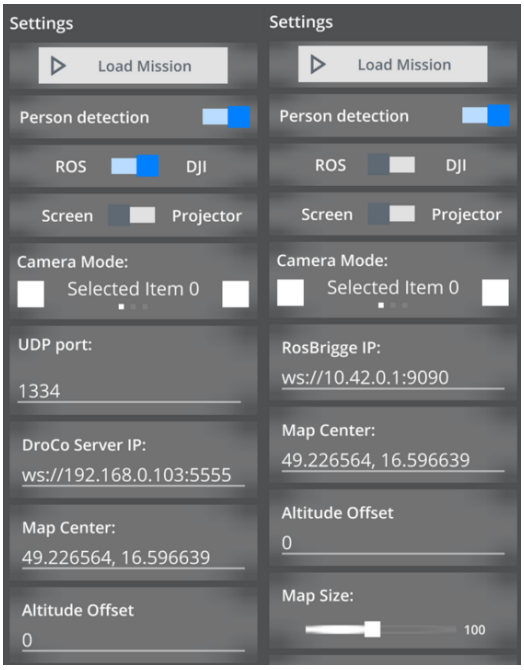
[6]Vizuální aplikace využívá integrace GStreamer klienta pro Unity. Integrace je volně dostupná z: http://github.com/mrayy/mrayGStreamerUnity

[7]Obrázek je součástí produktu DroCo a je dostupný z [navštíveno 12.05.2023]: http://www.fit.vut.cz/research/product/647/

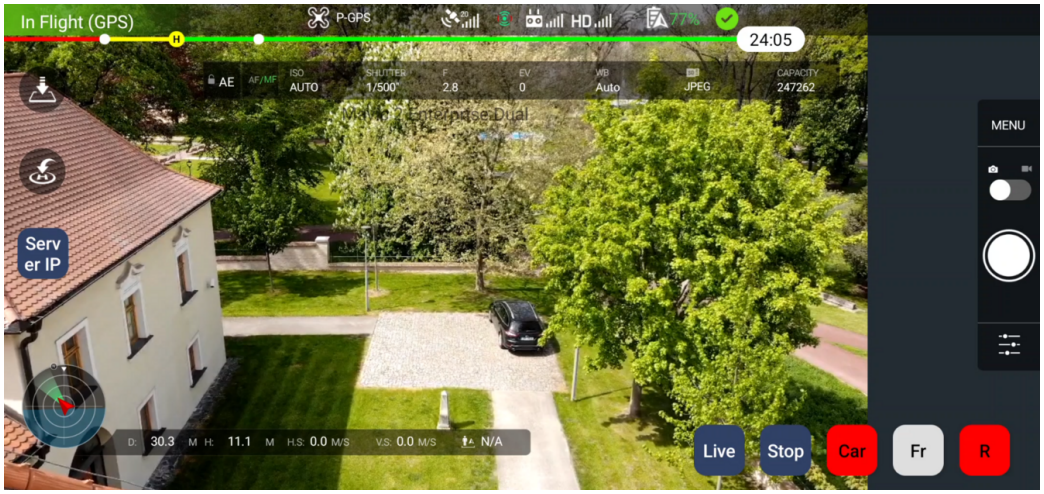
[8]Program je volně dostupný z: http://github.com/robofit/drone_server

```
{
    "DroneId": "test01",
    "Altitude": 225.0,
    "Latitude": 49.22656399999999,
    "Longitude": 16.596639000000005,
    "Pitch": 0.0,
    "Roll": 0.0,
    "Yaw": 90.0,
    "Compass": 0.0
}
```

Blok 1: Ukázka formátu JSON zprávy přenášející letová data.



Obrázek 2.3: Porovnání nastavení aplikace DroCo pro ROS a DJI drony.



Obrázek 2.4: Obrázek zobrazující uživatelské rozhraní aplikace DJI streamer včetně obrazu z kamery dronu.

2.4 Aplikace DJI streamer

Aplikace `DJI streamer`[9] je mobilní aplikace, sloužící pro ovládání dronu s pomocí dálkového ovládání. Využívá vývojových sad `DJI UX SDK`[10] a `DJI Mobile SDK`[11], které zajišťují základní ovládání dronu jako je tomu u oficiální aplikace `DJI Go 4`[12] s tím rozdílem, že je doplněna o komunikační vrstvu pro sdílení dat. Aplikace má, díky vývojovým sadám, k dispozici kromě multimediálních dat i letová data – výška, GPS souřadnice, Eulerovy úhly natočení dronu v prostoru včetně kompasu, rychlost dronu v jednotlivých směrech, Eulerovy úhly kamery a čas. Obrazová data a zvolená letová data jsou periodicky posílána na server.

[9] DJI streamer je mobilní aplikace vyvinutá skupinou Robo@FIT, která je určená pro ovládání dronu a zároveň slouží jako médium pro přeposílání letových a multimediálních dat od dronu serveru. Aplikace je dostupná z: `http://github.com/robofit/drone_dji_streamer`

[10] Vývojová sada od společnosti DJI se základními grafickými prvky využitými při ovládání dronu. Sada je dostupná z: `http://developer.dji.com/ux-sdk/`

[11] Vývojová sada od společnosti DJI disponující mnoha funkcemi pro konfiguraci a ovládání dronů, zpracování dat a mnoha dalšími. Sada je dostupná z: `http://developer.dji.com/mobile-sdk/`

[12] DJI Go je mobilní aplikace pro ovládání dronu značky DJI vyvinutá touto společností. Aplikace je dostupná z: `http://www.dji.com/goapp`

Kapitola 3

Technologie

V této kapitole se zaměřím na technologie používané v problematice přenosu a ukládání letových a multimediálních dat. Představím nejpoužívanější technologie, jejich výhody a nevýhody a porovnám je.

3.1 Přenos letových dat

Letová data se skládají pouze z několika reálných čísel a identifikačního řetězce, tudíž není důvod nevyužít nějakého serializačního formátu, aby byl přenos a zpracování co nejsnadnější. Nabízí se několik z nejrozšířenějších technologií pro serializaci, jako například XML [3] a JSON [2]. Ty jsou v dnešní době hojně využívané pro serializaci přenášených dat, obzvláště pro serverová API. Den ode dne se ale stává JSON populárnějším, neboť je mnohem lépe čitelnějším a hlavně podporovanějším. To je především díky tomu, že odpovídá přesně notaci jazyku Javascript, který je stále více a více využíván, neboť jsou vysoké nároky na responzivní webové stránky. XML se na druhou stranu považuje za bezpečnější a má podporu více kódování. Oba formáty lze přenášet mnoha způsoby.

Na přenos XML většinou vystačí obyčejný protokol HTTP [20]. Popřípadě lze využít protokolu Soap [1] vhodného právě pro přenos serializovaných dat XML většinou přes protokol HTTP. Protokol Soap se řadí mezi starší protokoly, ale stále je dost využíván. I když je pro člověka dokument docela dobře čitelný, tak z pohledu počítače patří k těm náročnějším na parsování vzhledem k délce zprávy a složitosti.

Za zmínku stojí rozšíření nad protokolem HTTP známé jako Rest API, které jednoduše umožňuje přenos dat. Přesně definuje pravidla a omezení, které umožňují snadné a rychlé zacházení s daty vzhledem k dané službě. Protokol definuje snadné způsoby jak vytvořit, získat, aktualizovat nebo smazat data ze vzdáleného serveru přes rozšířený protokol HTTP. Hlavními výhodami je jednoduchost a rozšiřitelnost.

Dalším rozšířeným protokolem je protokol WebSocket [19]. To je rozšiřující technologie nad protokolem HTTP, která slouží pro přenos dat přes TCP [5]. Často je využíván u mnoha serverů, jako protokol pro API. Je to stavový protokol, kterému předchází úspěšné navázání HTTP spojení. Navázání tohoto spojení se považuje za tzv. „handshake“. Při tomto spojení žádá definovanou hlavičkou protokolu HTTP klient server o upgrade na protokol WebSocket. Server odpoví a v závislosti na odpovědi se buď spojení ukončí, nebo je úspěšně navázáno. Společně s informací o upgrade se přenáší i verze protokolu WebSocket, typ požadované komunikace a klíč, což je několik náhodných bytů ve formátu base64 [15]. Po pozitivní odpovědi od serveru se přechází na WebSocket komunikaci. Tedy na komunikaci, kde spojení

je udržováno a data se mohou přenášet v obou směrech. Součástí dat je malá hlavička a tzv. „payload“. Protokol podporuje fragmentaci dat do více payloadů. Dá se využít k přenosu jakéhokoliv typu dat, ať už textových nebo binárních. V praxi se nejčastěji využívá k přenosu dat ve formátu JSON.

3.2 Přenos multimediálních dat

Existuje mnoho technologií, které umožňují živý přenos multimediálních dat, avšak jen některé z nich jsou podporovány technologií dronů.

Real Time Messaging Protocol

Protokol RTMP [22] byl vyvinut společností Macromedia jako proprietární protokol pro živý přenos video, audio a ostatních dat po internetu mezi jejich technologiemi, jako například Flash Player. Po převzetí společností Adobe, byl nekompletní protokol publikován a vzniklo několik jeho alternativ. Protokol je implementován na protokolem TCP [5] a využívá architektury klient-server a je nezabezpečen. To jsou převážně důvody vzniku alternativ, které zabezpečují protokol, anebo zmenšují odezvu přenosu přechodem na protokol UDP [23] a s přechodem na architekturou peer-to-peer. Jelikož drony podporují převážně tento protokol, musel jsem jej využít i přes jeho nevýhody a pečlivě jej popíši.

Při navázání spojení probíhá tzv. handshake. Spojení navazuje klient přenosem 8 bitů reprezentujících verzi protokolu, server odpoví stejnou zprávou. Dále dochází k přenosu vzájemných časů a náhodných čísel. Nedochází zde k žádné autentizaci nebo dohodnutí na šifrování přenosu. Poté se spojení považuje za zahájené. Existují rozšíření RTMP protokolu zvaná RTMPE, která přidává do handshaku výměnu klíčů a zajištění šifrování přenosu nebo rozšíření RTMPS, které využívá SSL [7] protokol. Vznikl i nový protokol RTMFP [29], který se také snaží řešit tyto nedostatky. Protokol pouze zaobaluje a rozšiřuje protokol RTMP.

Protokol přenáší data přes tzv. chunk stream. Tedy jednotlivé datové pakety jsou rozděleny do chunků s několika hlavičkami. První základní hlavička obsahuje formát a identifikační číslo chunk streamu. Velikost této hlavičky je proměnlivá podle tohoto čísla. Celkově tak lze dosáhnout hodnot 2-65599. Ve většině případů si však vystačíme s jednotkami až desítkami chunk streamů. Druhá hlavička se zaměřuje na zprávu. Opět existuje několik typů hlaviček podle stavu chunk streamu. Tento typ se dovíme z formátu z hlavní hlavičky. Protokol se snaží redukovat množství přenesených dat, ignorováním přenosu dat shodných s předchozím chunkem téhož chunk streamu. Přenáší se časová značka, délka zprávy, typ zprávy a identifikační číslo streamu zprávy. Tedy pokud je typ 0, data se přenesou všechna. Pokud je typ 1, data se přesunou bez identifikačního čísla streamu zprávy, atd. V případě rozšířené časové značky, následuje ta, jinak následuje rovnou zpráva. V případě, že zpráva je větší než stanovený limit, tak je chunk rozdělen do několika přenosů. Nejprve se přenese chunk typu 0 s velikostí dat větší než je limit. Následný přenos zbytku dat je pomocí hlavičky zprávy typu 3 neboli bez této hlavičky zprávy. Takto čte data až do přečtení zadaného množství dat. Výchozí limit velikosti chunku je dle [22] stanoven na 128 bytů. Mezi prvními zprávami většinou bývá žádost na zvětšení tohoto limitu.

Existuje několik typů zpráv. Některé zprávy jsou kódovány serializačním formátem od společnosti Adobe AMF0 [9] nebo AMF3 [10]. Vypíšu jen ty nejdůležitější z nich.

- Typ 1 - změna limitu velikosti chunku
- Typ 3 - potvrzení - po odeslání určeného počtu dat o velikosti okna

- Typ 4 - uživatelské příkazy
- Typ 5 - definování velikosti okna
- Typ 6 - definování šířky pásma
- Typ 8 - audio data
- Typ 9 - video data
- Typ 20, 17 - příkazy ve formátu AMF0/3
- Typ 18, 15 - metadata o streamu ve formátu AMF0/3

Klient po připojení k RTMP serveru posílá zprávu typu 20 `connect`, ve které se snaží ohlásit své připojení společně se zaslaným URL a získat tak instanci. Server odpoví a zároveň posílá klientovi zprávu typu 5 a 6 se zvolenými hodnotami. Dále posílá zprávu, že instance streamu je připravena. Poté se klient snaží vysílat data nebo přijímat. Pokud se klient snaží vysílat, tak zasílá zprávu typu 20 `publish`, kde říká, že zahajuje přenos dat. Ale jako první posílá zprávu typu 18 s metadaty o přenášeném médiu. Samotný přenos dat probíhá na jiném chunk streamu a jedná se o zprávy typu 8 a 9. Pokud se klient snaží data získávat, posílá zprávu typu 20 `play` a server od té doby ví, že veškerá přijatá data od jiného klienta se stejným stream id má přeposílat tomuto klientu. Tímto způsobem je jednoduše vysvětlen princip přenosu protokolu RTMP. Server se také musí starat o to, aby se data posílala přes buffer, a aby se řídil přenos dat dle množství posílaných dat a šířkou pásma.

Real Time Streaming Protocol

RTSP [24] je jedním ze starších protokolů postavený na protokolu HTTP [20], ale na rozdíl od něj je stavový. Protokol s architekturou klient-server definuje zprávy pro správu přenosů, ale samotný přenos dat neřeší. K tomu se používá protokol RTP společně s RTCP [26]. RTSP může využívat protokoly UDP [23] i TCP [5] pro přenos transportní vrstvou. Později vyšla verze 2.0 [27] tohoto protokolu, která ale není zpětně kompatibilní. Zmíněná publikace [16] využívá právě tento protokol.

RTP [26] je protokol určený pro posílání audio nebo video dat po síti. Byl navržen v devadesátých letech jako pokus o digitalizaci především telefonního spojení. I když se jedná o starý standard, stále je udržován na nejnovější technologie. Důkazem toho je poslední rozšíření standardu o podporu nového kodeku HEVC [30]. Samotný standard obsahuje nejen popis protokolu RTP, ale také protokolu RTCP. Ten je určen pro ovládání a správu přenosu přes RTP. Charakteristické technologie využívající RTP je VoIP a IP televize. RTP může fungovat jak nad protokolem TCP, tak nad UDP. Často se ale využívá UDP, protože se preferuje rychlost před spolehlivostí.

HTTP Live Streaming

HLS [21] je streamovácí protokol od společnosti Apple, který využívá protokolu HTTP [20]. Jeho hlavní výhodou je podpora mnoha zařízeních, která se podepsala na tom, že se jedná o jeden z nejrozšířenějších způsobů streamování. Na rozdíl od RTP může bez problému projít přes firewall a je také znám svou nízkou odezvou. Díky tomuto protokolu je možné sledovat video přenos v téměř reálném čase.

Web Real-Time Communication

WebRTC [8] protokol je open-source projekt založený na architektuře peer-to-peer. Stejně jako u HLS stačí zařízení s webovým prohlížečem s povoleným jazykem Javascript a můžeme sledovat WebRTC streamy bez potřeby něco stahovat nebo instalovat.

3.3 Ukládání letových dat

Existuje mnoho způsobů, jak ukládat letová data, která se skládají z několika reálných čísel. Klasickým způsobem je využití relační databáze, což patří k jednomu s nejrozšířenějším způsobům ukládání dat. Vzhledem k povaze problému se nabízí i využití NoSQL databází, jako například dokumentové databáze MongoDB.

Relační databáze je roky osvědčený způsob, jak zajistit rychlé a konzistentní uložení dat. Kromě klasických CRUD operací jako vytváření, získání, úpravy a odstranění, nabízí třeba mnoho způsobu vyhledání v datech. Databáze je založena na relačním modelu a tedy mezi jednotlivými entitami se může nacházet vazba. Tento typ databází je znám svou konzistencí umožněnou pomocí transakcí a vysokou integritou dat. Relační databáze kladou větší důraz na strukturu dat a není umožněno mít u stejné entity různé atributy. Tedy hodí se více na neměnná data.

NoSQL databáze jsou se v dnešní době hodně rozšiřují, a to především kvůli rozmachu tzv. „big data“. Za to můžeme považovat obrovské data sety různých dat. Databáze neklade takový důraz na konzistenci a snaží se upřednostnit dostupnost. Výhodou je, že umožňuje škálovat databázi nejen vertikálně, ale i horizontálně. Tedy umožňuje distribuci dat mezi více clusterů, a pak odkazovat žádosti tam, kde se data fyzicky nacházejí. Tímto způsobem může rozložit zátěž na více fyzických zařízení. Příkladem této databáze je MongoDB. Nad touto databází jsem uvažoval, protože je založena na serializačním formátu JSON.

3.4 Ukládání multimediálních dat

U ukládání velkých multimediálních dat mě nenapadá jiný způsob, než záznam video přenosu uložit do souboru a informaci o tomto souboru přiložit do databáze z předchozí sekce. Předmětem kapitoly bude spíše vybrání vhodného kodeku a kontejneru pro tato vizuální data.

Nabízí se tři nejznámější kontejnery. Prvním je AVI [6]. Je to hodně starý standard od společnosti Microsoft, který má na dnešní poměry hodně nedostatků. Hlavním nedostatkem je, že se v té době nepočítalo s tím, aby kontejner podporoval kompresi, u které je potřeba mít přístup k budoucím snímkům. To je vyžadováno u dnešních kompresí, které se bez toho neobejdou.

Dalším je MP4 [13], který je pravděpodobně snad nejrozšířenějším kontejnerem. V poslední době mu ale začal konkurovat kontejner Matroska [18], který je jako jeden z mála open-source[1]. Oba tyto kontejnery jsou po technické stránce na srovnatelné úrovni, ale MP4 se dostává větší multiplatformní podpora.

Za zmínku stojí i kontejner FLV [11], který je vytvořen společností Adobe pro jejich Flash Player. A právě tímto kontejnerem probíhá přenos dat již zmíněným protokolem RTMP z kapitoly 3.2.

[1] Zdrojový kód kontejneru Matroska je dostupný z: http://github.com/Matroska-Org/libmatroska

Co se týče video kodeků, za zmínku stojí dva. Prvním z nich je H.264 (AVC) [12], což je snad zatím nejrozšířenější kodek, který je hojně podporován na spoustě různých zařízeních. A druhým je H.265 (HEVC) [14], který se v porovnání se svým předchůdcem snažil zmenšit objem dat a dosahuje lepších kompresních poměrů. Motivací byl právě trend streamování videa v reálném čase po internetu. Oproti AVC však není tolik podporován různými zařízeními, u kterých se dostala hardwarová podpora hlavně H.264.

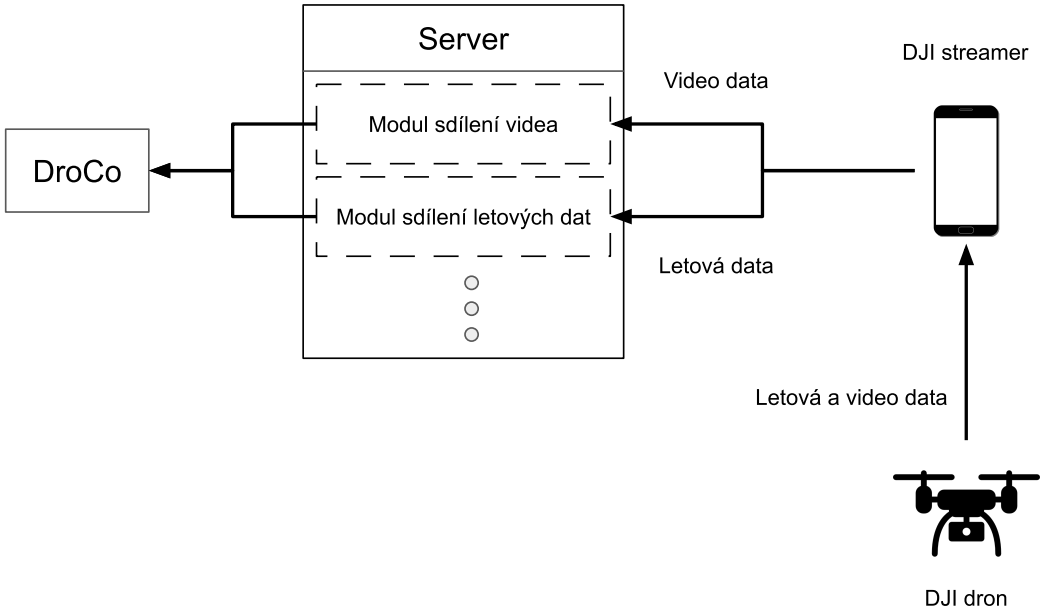
Kapitola 4

Návrh

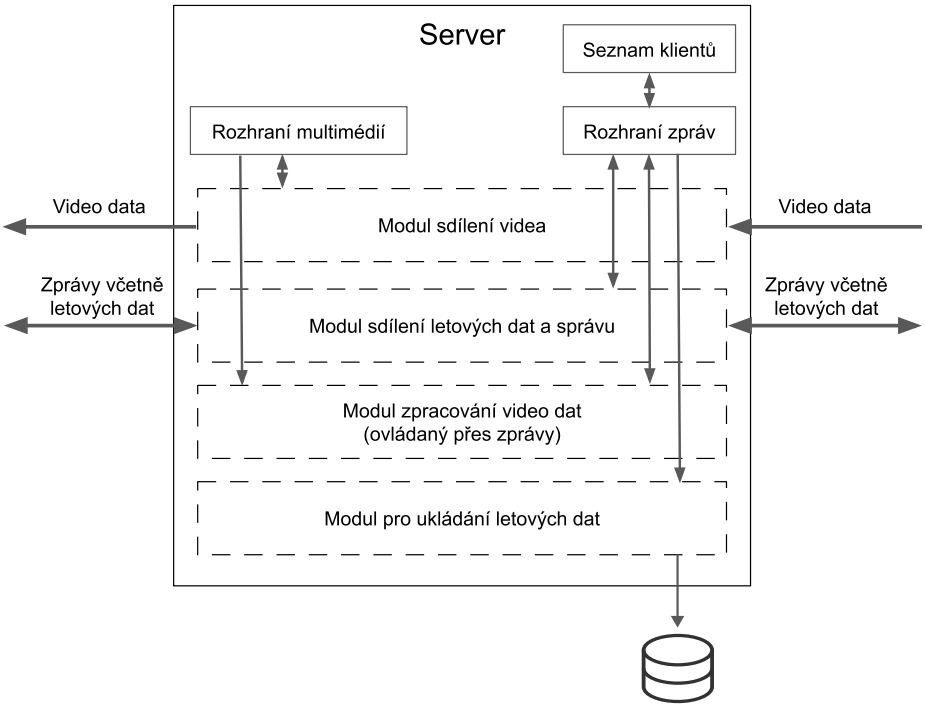
V této kapitole navrhnu architekturu programu s respektem k robustnosti a znovupoužitelnosti. Vyberu technologie z kapitoly 3, které budu využívat a jejich výběr zdůvodním.

4.1 Architektura systému

Z hlediska návrhu se program zabývá letovými a mediálními daty. Server by měl brát v potaz, že poskytuje služby nějakému klientu, se kterým nějak komunikuje a dochází ke sdílení letových a mediálních dat. Vzhledem k požadovaným vlastnostem jsem se rozhodl, že server bude modulární. Jádro serveru bude disponovat přesně a podrobně definovanému rozhraní, pomocí kterého jednotlivé moduly budou komunikovat s klienty. Každý problém řešený tímto serverem by měl být modulem. V následující kapitole 4.2 se budu věnovat právě tomuto rozhraní, které nedefinuje použití jakéhokoliv protokolu nebo serializačních formátů, od toho slouží moduly.



Obrázek 4.1: Schéma nastiňující architekturu celého systému. Aplikace DJI Streamer přijímá data od dronu a přeposílá je serveru ve dvou tocích – video data a letová data. Server data přepošle vizualizační aplikaci DroCo.



Obrázek 4.2: Schéma znázorňující rozhraní serveru včetně příkladů různých modulů a jejich datových toků.

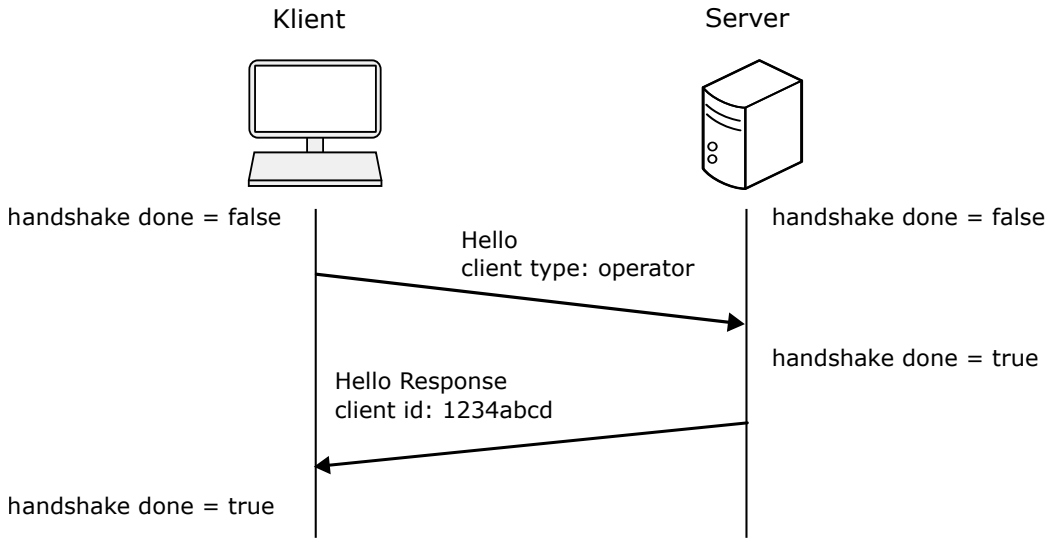
Architektura systému, znázorněná na obrázku 4.1, se skládá ze serveru, dronu, vizualizační aplikace DroCo (kapitola 2) a mobilní aplikace DJI streamer (kapitola 2.4), která slouží z hlediska architektury jako médium při komunikaci s dronem. Data jsou dělena na multimediální a letová právě touto aplikací.

4.2 Modulární rozhraní serveru

Dle architektury systému je navrženo rozhraní serveru pro moduly. Server se věnuje přenosu dvou druhů zpráv – multimediální a letová. Avšak z hlediska návrhu potřebné komunikace mezi dronem a serverem jsou letová data brána jako řetězcové zprávy. Přes tyto zprávy se neposílají pouze letová data, ale i data pro správu a další různá data (např. seznam dronů).

Server tedy musí disponovat rozhraním pro správu, posílání letových a ostatních dat. Předpokládá se architektura klient-server. Jako první je potřeba si definovat co je klient. Ten je definován jako každý koncový bod připojený k serveru. Aby byl klient považován za úspěšně připojeného, musí nejprve provést handshake (kapitola 4.3). Rozlišujeme dva druhy klientů – operátor, dron. Po navázaní spojení dochází k přenosu zpráv. Rozhraní přijímá tyto zprávy od modulů, které je přijímají různými protokoly a technologiemi. Rozhraní po přijetí zprávy je rozpošle mezi další moduly, které o ně stojí. Tedy rozhraní zde hraje roli jako redistributor zpráv.

U multimediálních dat je to obdobně. Modul přijímá multimediální přenos, který je rozdělen na snímky a ty jsou předány rozhraní. Rozhraní je pak dále pošle modulům, které o ně žádají.



Obrázek 4.3: Schéma ukazující handshake komunikaci mezi klientem a serverem v čase včetně změn stavu.

4.3 Komunikační protokol

Komunikační protokol popisuje pravidla komunikace mezi klientem a serverem. Mezi nimi dochází k výměně nejen letových, ale také k přenosu ostatních dat, jako např. seznam dronů. Protokol je stavový, ale to jen protože vyžaduje handshake. Po připojení se klient považuje za nového a server čeká na handshake. Až po úspěšném handshaku je klient úspěšně připojen.

Handshake by měla být první zpráva poslaná klientem a obsahovat informaci o tom, jestli se jedná o dron nebo o aplikaci operátora. Popřípadě by zpráva mohla obsahovat dodatečné informace v závislosti, zda se jedná o dron nebo aplikaci (např. název dronu, sériové číslo dronu, atd.). Server na tuto zprávu odpoví náhodně vygenerovaným číslem, které bude sloužit jako jednoznačný identifikátor tohoto klienta. Rozhraní disponuje generátorem identifikátoru při vyváření klienta. Poté server považuje handshake za úspěšný. Rozhraní serveru si udržuje seznam úspěšně připojených klientů. Úkolem rozhraní serveru není handshake provádět, ale pouze definuje, jak by měl vypadat. Provedení je na jednotlivých modulech.

Konkrétní protokol využitý pro komunikaci s drony a vizualizační aplikací byl navržen a konzultován s vedoucím této diplomové práce, který má na starost zmíněné aplikace. Vzhledem k původnímu řešení (kapitola 2.2) se rozhodlo, že zůstane využit serializační formát JSON.

Všechny zprávy obsahují dvě položky. První je řetězec `type`, který obsahuje typ zprávy. Druhou položkou je JSON datový typ dle typu zprávy. Tato položka je nazvána `data`. V bloku 2 je možné vidět zmíněnou strukturu požadavku a odpovědi handshake zprávy.

U handshake zprávy je typ „hello“ a data musí obsahovat minimálně číslo `ctype` s hodnotou buď 0 nebo 1. První hodnota říká, že se jedná o dron a druhá, že se jedná o operátora. Dron dále obsahuje řetězce `drone_name` a `serial`, které obsahují název dronu a jeho sériové číslo. Server vrací zprávu typu `hello_resp` a data obsahují řetězec `client_id` s přiděleným identifikátorem, a pokud server disponuje RTMP serverem, tak je součástí i

```
{
    "type": "hello",
    "data": {
        ctype: 1
    }
}
```

```
{
    "type": "hello_resp",
    "data": {
        client_id: "1234abcd",
        ?rtmp_port: 1935
    }
}
```

Blok 2: Ukázka formátu JSON zpráv při handshake požadavku (vlevo) a odpověď (vpravo) na něj.

```
{
  "type": "drone_list",
  "data": null
}
```

```
{
  "type": "drone_list_resp",
  "data": [
    {
      client_id: "1234abcd",
      drone_name: "test01",
      serial: "UIOP789"
    },
    {
      client_id: "5678efgh",
      drone_name: "test02",
      serial: "UIOP790"
    }
  ]
}
```

Blok 3: Ukázka formátu JSON zpráv při požadavku (vlevo) na seznam dronů a odpověď (vpravo) na něj.

číslo `rtmp_port`, které obsahuje číslo portu RTMP serveru (blíže popsáno v kapitole 4.5). Příklad tohoto požadavku s odpovědí lze vidět v bloku 2.

Další důležitou zprávou je žádost o seznam dronů. Tuto žádost posílá vizualizační aplikace a server odpoví seznamem všech úspěšně připojených klientů typu dron. Součástí zprávy jsou i informace o dronech (identifikátor, název, sériové číslo). V bloku 3 je opět znázorněna komunikace. Klient posílá žádost s typem `drone_list`, proměnná data zůstává prázdná. Server vrací zprávu s typem `drone_list_resp` a s proměnnou data, která obsahuje seznam se zmíněnými informacemi.

Hlavním druhem zprávy je však posílání letových dat. Zprávu posílá dron serveru a obsahuje letové informace o dronu. Server tyto informace pouze přepošle všem připojeným klientům. Tedy nejedná se o klasickou zprávu typu požadavek – odpověď. Jedná o zprávy typu `data_broadcast`. Letová data se z původního formátu, znázorněného v bloku 1, změnila. Konkrétně položka `DroneId` s hodnotou názvu dronu byla zaměněna za `client_id` s identifikátorem získaným při handshaku. Dále gps souřadnice a orientace dronu v prostoru se vložily do samostatných objektů a přibyly další informace, jako rychlost ve směrech, orientace kamery a čas. Výslednou formu lze vidět v bloku 4.

```
{
    "type": "data_broadcast",
    "data": {
        "client_id": "1234abcd",
        "altitude":234.6,
        "gps":{
         "latitude":49.24031521243185,
         "longitude":16.613207555321484
        },
        "aircraft_orientation":{
         "pitch":-4.0,
         "roll":-1.2,
         "yaw":81.2,
         "compass":81.2
        },
        "aircraft_velocity":{
         "velocity_x":0.1,
         "velocity_y":1.0,
         "velocity_z":0.1
        },
        "gimbal_orientation":{
         "pitch":0.0,
         "roll":0.0,
         "yaw":81.1,
         "yaw_relative":0.0
        },
        "timestamp":"2023-04-05 14:45:06.843"
    }
}
```

Blok 4: Ukázka formátu JSON zprávy přenášející letová data navrženým protokolem.

To jsou pouze základní části komunikačního protokolu, bez kterých by navržená architektura systému nefungovala a každý klient nebo server musí podporovat alespoň tyto tří druhy zpráv. Jednotlivé moduly pak mohou rozšiřovat tento komunikační protokol dle vlastních potřeb a je jen na klientech, zda budou nové funkcionality podporovat.

4.4 Přenos letových dat

Jelikož letová data aplikace DroCo z kapitoly 2 jsou serializovaná formátem JSON a přenášena přes protokol WebSocket, nebyl důvod pro přenos letových dat využít nějakých jiných technologií. Jak bylo navrženo v této kapitole dříve, tak letová data jsou součástí posílaných zpráv. Pro tyto účely by bylo vhodné vytvořit modul, který plní požadavky komunikačního protokolu popsaného v předchozí kapitole 4.3 s využitím protokolu WebSocket. Jak je zmí-

něno v kapitole 4.3, tak došlo ke změně struktury posílaných dat. Příklad formátu JSON zprávy lze vidět v bloku 4 v objektu „data“.

Přenos letových dat je řešen modulem, jehož úkolem je hlavně přenášet letová data dronů ke všem klientům. Modul musí podporovat celý zmíněný komunikační protokol z předchozí kapitoly 4.3, aby mohl připojovat klienty a posílat operátorům seznam dronů. Bez těchto funkcionalit by vizualizační aplikace nefungovala.

4.5 Přenos multimediálních dat

Pro přenos multimediálních dat je využit protokol RTMP popsaný v kapitole 3.2. Ten byl vybrán, protože je to jeden z nejrozšířenějších streamovacích protokolů a hlavně byl již podporován zmíněnu aplikací DJI streamer. Opět přenos multimediálních dat pomocí protokolu RTMP tvoří samostatný modul. K RTMP serveru naslouchajícího na zadaném portu se pak připojuje URI adresou v tomto formátu:

*rtmp://*host[:port]*/live/*client_id,

kde `client_id` značí identifikátor dronu, který vysílá multimediální data. Stejnou URI adresu využívá dron odesílající data a vizualizační aplikace, které tato data přijímají. Tedy vizualizační aplikace musí disponovat identifikátorem dronu, aby se mohla připojit na jeho video stream.

4.6 Ukládání letových dat

Pro ukládání dat jsem zvolil relační databázi. Předpokladem pro tuto volbu bylo, že data se budou především ukládat a nebudou se téměř nikdy měnit. Navíc se předpokládá pevná struktura, kterou tato databáze vyžaduje. Jediným důvodem, proč uvažovat nad NoSQL byla konkrétně MongoDB[1], která přímo podporuje JSON a tím by mi usnadnila práci s daty. Nicméně jsem vybral relační databázi a to konkrétně MariaDB[2]. Vybral jsem si ji, protože s ní mám dobré zkušenosti, je snadno dostupná a hojně podporována.

Databáze běží na zařízení nezávisle na tomto serveru. Pro připojení do databáze modul potřebuje znát čtyři hodnoty – adresu, uživatelské jméno, heslo a název databáze/schématu. Databáze obsahuje pouze jednu tabulku s názvem `flight_data`, která vychází z letových dat. Pokud databáze touto tabulkou nedisponuje, program by ji měl vytvořit. Tabulka má následující strukturu:

- client_id – CHAR(8)
- altitude – DOUBLE
- latitude – DOUBLE
- longitude – DOUBLE
- aircraft_pitch – DOUBLE

[1] MongoDB je dokumentová NoSQL databáze, která využívá formátu JSON. Databáze je dostupná z: http://www.mongodb.com/

[2] MariaDB je relační databáze vzniklá jako open-source nástupce populární MySQL databáze. MariaDB je dostupná z: http://mariadb.org/

- aircraft_roll – DOUBLE
- aircraft_yaw – DOUBLE
- aircraft_compass – DOUBLE
- aircraft_velocity_x – DOUBLE
- aircraft_velocity_y – DOUBLE
- aircraft_velocity_z – DOUBLE
- gimbal_pitch – DOUBLE
- gimbal_roll – DOUBLE
- gimbal_yaw – DOUBLE
- gimbal_yaw_relative – DOUBLE
- timestamp – TIMESTAMP

Modul rozšiřuje protokol o požadavek k ukládání letových dat. Klient požádá server, aby ukládal letová data do databáze. Žádná oprávnění nejsou řešena, kterýkoliv klient může toto nastavení změnit. Konkrétní rozšíření se skládá ze dvou zpráv. První zpráva posílaná klientem s typem `flight_data_save_get` bez žádných dat slouží pro zjištění, jestli v současnosti server nahrává nebo ne. Server odpovídá zprávou typu `flight_data_save_get_resp` a jako data vrací boolean hodnotu. Druhá zpráva je typu `flight_data_save_set` a položka „data“ obsahuje boolean hodnotu. Ta slouží pro nastavení, jestli se mají letová data nahrávat nebo ne. Server na zprávu odpovídá `flight_data_save_set_resp` a položka data odpovídá hodnotě „null“. Příklady zpráv ukazující formát se nachází v blocích 5 a 6.

```
{
  "type":
    "flight_data_save_get",
  "data": null
}
```

```
{
  "type":
    "flight_data_save_get_resp",
  "data": True
}
```

Blok 5: Ukázka formátu JSON zpráv při požadavku (vlevo) na zjištění stavu ukládání letových dat a odpověď (vpravo) na něj.

```
{
  "type":
    "flight_data_save_set",
  "data": False
}
```

```
{
  "type":
    "flight_data_save_set_resp",
  "data": null
}
```

Blok 6: Ukázka formátu JSON zpráv při požadavku (vlevo) na nastavení stavu ukládání letových dat a odpověď (vpravo) na něj.

4.7 Ukládání multimediálních dat

Ukládání multimediálních dat je opět řešeno vlastním modulem. Musí se však vybrat vhodné technologie z kapitoly 3.4. Co se týče kontejneru pro vizuální data, rozhodoval jsem se mezi MPEG-4 [13] a Matroska [18]. Nakonec jsem kvůli multiplatformní kompatibilitě zvolil rozšířený kontejner MP4. Hlavní výhodou kontejneru Matroska je možnost obsahovat více audio nebo titulkových stop, ale to je nevyužitelné pro tuto problematiku. Kontejner AVI [6] nepřipadal v úvahu vzhledem ke zvolenému kodeku.

Při výběru kodeku jsem také následoval cestu kompatibility a zvolil populární H.264. Kodek HEVC má sice lepší kompresi, ale to by bylo parametrem, pokud bychom byli omezeni velikostí fyzického úložiště nebo šířkou pásma při stahovaní záznamů. Vzhledem k potřebám této práce, jsem zvolil v obou případech multiplatformní kompatibilitu.

Modul umožňuje mediální data upravovat před uložením na disk. Konkrétně se jedná o změnu rozlišení, fps nebo bitrate. Díky tomu lze ovlivnit velikosti ukládaných dat za cenu výpočetního výkonu.

Modul, podobně jako ukládání letových dat z kapitoly 4.6, rozšiřuje komunikační protokol o požadavek k ukládání multimediálních dat. Opět se jedná o dvě zprávy typu „get“ a „set“, ale oproti předchozí žádosti zde figuruje parametr, který značí identifikátor klienta vysílající video přenos pro ukládání. První z nich má typ `media_record_get` a data je objekt obsahující identifikátor dronu vysílající video přenos, který chceme ukládat. Tento řetězec se označuje `drone_stream_id`. Server vrací zprávu typu `media_record_get_resp`, kde data obsahují boolean hodnotu, zda se video data nahrávají nebo ne. Druhý požadavek je typu `media_record_set` a data obsahují boolean hodnotu `state`, dle které se buď nahrávání zapne nebo vypne a opět identifikátor video streamu. Server odpoví zprávou typu `media_record_set_resp`, kde data jsou prázdná. Příklady zpráv znázorňující formát se nacházejí v blocích 7 a 8.

```
{
  "type":
    "media_record_get",
  "data": {
    "drone_stream_id":
      "1234abcd"
  }
}
```

```
{
  "type":
    "media_record_get_resp",
  "data": True
}
```

Blok 7: Ukázka formátu JSON zpráv při požadavku (vlevo) na zjištění stavu ukládání video přenosu dle identifikátoru a odpověď (vpravo) na něj.

4.8 Rozpoznávání vozidel

Poslední navržený modul slouží pro rozpoznávání vozidel z video dat přenášených serverem. Výsledek této detekce je poslán operátorům, kteří si o to požádali. Cílem je ukázat praktické využití modulární architektury a celkově účelnost diplomové práce. Pro detekci jsem

```
{
  "type":
    "media_record_set",
  "data": {
    "drone_stream_id":
      "1234abcd",
    "state": False
  }
}
```

```
{
  "type":
    "media_record_set_resp",
  "data": null
}
```

Blok 8: Ukázka formátu JSON zpráv při požadavku (vlevo) na nastavení stavu ukládání video přenosu dle identifikátoru a odpověď (vpravo) na něj.

se rozhodl využít knihovnu OpenCV[3]. Modul získává snímky od rozhraní serveru a jednou za určitý čas provede na snímku detekci vozidel (detekce je náročná na výpočetní výkon – nelze ji provádět v reálném čase). Výsledkem detekce je seznam obdélníků definovaných souřadnicemi x, y, šířkou a výškou, kde souřadnice x, y značí levý horní roh obdélníku. Seznam s obdélníky se poté zašle všem klientům, kteří o to požádali. Detekce se provádí, žádá-li o to alespoň jeden klient. V bloku 9 je ukázka zprávy pro zaslání výsledků detekce klientům. Zpráva je typu `vehicle_detection_rects` a v datech je kromě seznamu obdélníků i identifikátor klienta s video přenosem.

Modul rozšiřuje komunikační protokol z kapitoly 4.3 o žádosti klientů o zasílání výstupu detekcí. Opět se jedná o zprávy typu „get“ a „set“, jejichž formát je podobný rozšíření protokolu modulem ukládání dat z předchozí kapitoly 4.7. Tedy návrh definuje zase dva páry požadavek-odpověď, kde první z nich je typu `vehicle_detection_get` a data je objekt s řetězcem `drone_stream_id`. Server vrací odpověď, zda klient žádá o zasílání výstupu detekce dle zadaného identifikátoru klienta s video přenosem. Odpověď obsahuje typ `vehicle_detection_get_resp` a data v podobě boolean hodnoty. Druhou zprávou žádáme server o změnu, zda se má pro určitý video stream zasílat výsledek detekce. Žádost má typ `vehicle_detection_set` a jako data opět obsahuje objekt s řetězcem `drone_stream_id` a boolean hodnotu s názvem `state`. Odpovědí je zpráva typu `vehicle_detection_set_resp` s prázdnou hodnotou dat. Zprávy zobrazující formát lze vidět v blocích 10 a 11.

[3]OpenCV je open-source knihovna zabývající se počítačovým viděním. Knihovna je dostupná z: http://opencv.org/

```
{
    "type": "vehicle_detection_rects",
    "data": {
        "client_id": "1234abcd",
        "rects": [
            { "x": 40.5, "y": 20.3, "w": 80, "h": 90 },
            { "x": 202.8, "y": 80.9, "w": 20, "h": 20 },
            { "x": 400, "y": 29.6, "w": 40, "h": 60 },
            { "x": 140.5, "y": 760.2, "w": 60, "h": 50 }
        ]
    }
}
```

Blok 9: Ukázka formátu JSON zprávy přenášející výsledky detekce vozidel z video obrazu.

```
{
  "type":
    "vehicle_detection_get",
  "data": {
    "drone_stream_id" :
      "1234abcd"
  }
}
```

```
{
  "type":
    "vehicle_detection_get_resp",
  "data": True
}
```

Blok 10: Ukázka formátu JSON zpráv při požadavku (vlevo) na zjištění, zda klient žádá o zasílání výstupu detekce z konkrétního video streamu a odpověď (vpravo) na něj.

```
{
  "type":
    "vehicle_detection_set",
  "data": {
    "drone_stream_id" :
      "1234abcd",
    "state" : False
  }
}
```

```
{
  "type":
    "vehicle_detection_set_resp",
  "data": null
}
```

Blok 11: Ukázka formátu JSON zpráv při požadavku (vlevo) na nastavení, zda se má klientu posílat výsledek detekce požadovaného video streamu a odpověď (vpravo) na něj.

Kapitola 5

Implementace

V této kapitole se zaměřím na popis implementace samotného jádra programu – modulárního rozhraní. Poté se zaměřím na popis implementace jednotlivých modulů vzhledem k potřebným vlastnostem a protokolu dle návrhu z předchozí kapitoly 4. Nakonec popíši program, který má za úkol propojit funkcionalitu všech modulů dohromady a udělat tak funkční celek. Nejprve však vyjmenuji technologie použité k implementaci.

Rozhodl jsem se program implementovat v jazyku C++, protože jsem s tímto jazykem docela dobře obeznámen a považoval jsem ho za vhodný vzhledem k požadavkům na program. Využil jsem knihovny Boost[1], která nabízí mnoho rozšíření nad standardní knihovny jazyka C++. Sem patří i parsování argumentů nebo jednotné rozhraní pro sockety. Právě sockety využívají vstupně/výstupní služby `io_service`, která je bezpečná vzhledem k více vláknům a podporují synchronní i asynchronní operace. Další důležitou knihovnou je FFmpeg[2], kterou využívám pro manipulaci s mediálními daty. Knihovna podporuje většinu používaných formátů a kodeků.

5.1 Modulární rozhraní

Modulární rozhraní navržené v kapitole 4.2 se skládá ze tří hlavních částí – správa klientů, rozhraní zpráv a rozhraní multimédií. Dále obsahuje třídy pro záznam aktivit a nastavení. Modulární rozhraní disponuje i několika dalšími třídami, které mohou napomoci některým modulům, proto jsou ve společném rozhraní. Těmto třídám se budu věnovat na konci této kapitoly.

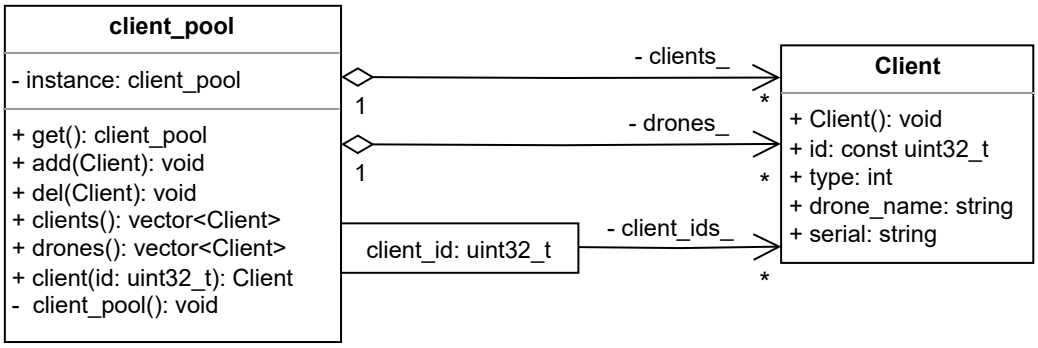
Správa klientů

Rozhraní slouží jako zdroj všech připojených klientů. Skládá se ze třídy `Client`, jedináčku (z angl. *singleton*) `client_pool` a dvěma statickými funkcemi pro převod identifikátoru na řetězec a naopak. Rozhraní pro správu klientů včetně vazeb je znázorněno na UML diagramu tříd na obrázku 5.1.

Třída `Client` obsahuje typ, identifikátor a řetězec s názvem dronu a sériovým číslem. Typ může nabývat tří hodnot. Hodnoty 0 a 1 odpovídají typu klienta. Tyto hodnoty pochází z návrhu protokolu z kapitoly 4.3. Třetí hodnota je -1 a značí nevalidní typ klienta. Touto

[1] Boost je populární multiplatformní open-source knihovna rozšiřující jazyk C++ a jeho standardní knihovny. Knihovna je dostupná z: http://www.boost.org

[2] FFmpeg je knihovna sloužící pro manipulaci s multimediálními daty mnoha různých formátů. Knihovna je dostupná z: http://ffmpeg.org



Obrázek 5.1: UML diagram tříd znázorňující strukturu části modulárního rozhraní spravujícího klienty.

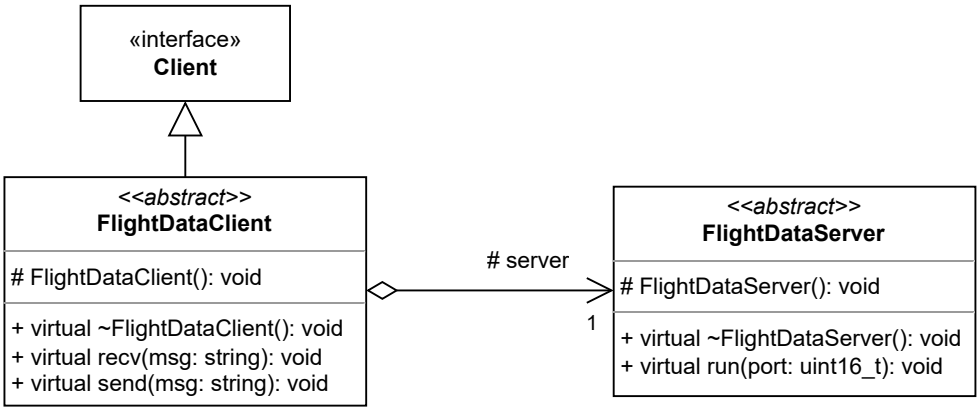
hodnotou je proměnná inicializována. Identifikátor byl zvolen jako 32-bitové nezáporné celé číslo. Při konstrukci této třídy je toto číslo náhodně vygenerováno.

Jedináček `client_pool` slouží jako distributor připojených klientů. Disponuje funkcemi pro přidání a odebraní klientů. Dále funkcemi pro získání seznamu všech klientů, klientů typu dron a konkrétní instance třídy klienta dle identifikátoru.

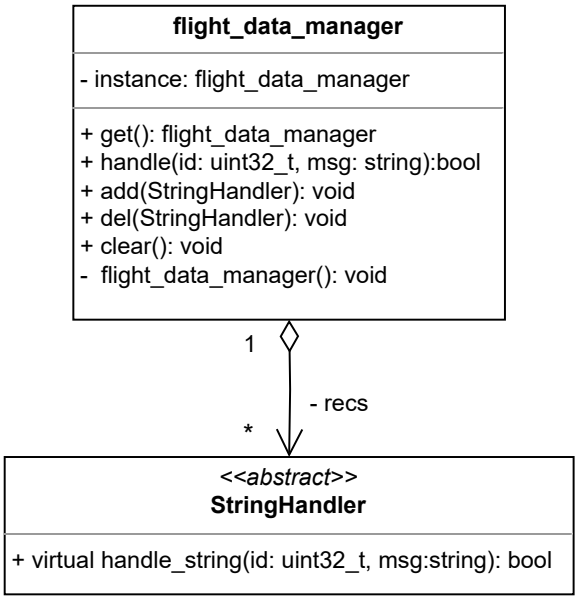
Jsou k dispozici i funkce, které převádí identifikátor z číselné hodnoty na hexadecimální řetězec a naopak. Řetězec má pevnou délku osmi znaků. Funkce se jmenují `cid_from_hex` a `cid_to_hex`.

Rozhraní zpráv

Další je rozhraní pro distribuci zpráv. Skládá se z několika abstraktních tříd a jedináčku. Jak je zmíněno v návrhu, předpokládá se architektura klient–server. První abstraktní třída `FlightDataServer` odpovídá serverovému rozhraní, které disponuje funkcí pro spuštění naslouchaní na zadaném portu. Dále je definována abstraktní třída klienta tohoto serveru (liší se od obecného klienta ze správy klientů) `FlightDataClient`, která definuje funkce pro zaslání a příjem dat. Zároveň je tato třída derivovaná od zmíněné třídy `Client`. Klient obsahuje vazbu na server, ze kterého byl připojen, jak je možné vidět i na UML diagramu na obrázku 5.2.



Obrázek 5.2: UML diagram znázorňující strukturu části modulárního rozhraní, která se věnuje přijímáním a odesíláním zpráv založeno na architektuře klient–server.



Obrázek 5.3: UML diagram znázorňující strukturu části modulárního rozhraní, která umožňuje přenášet zprávy mezi moduly.

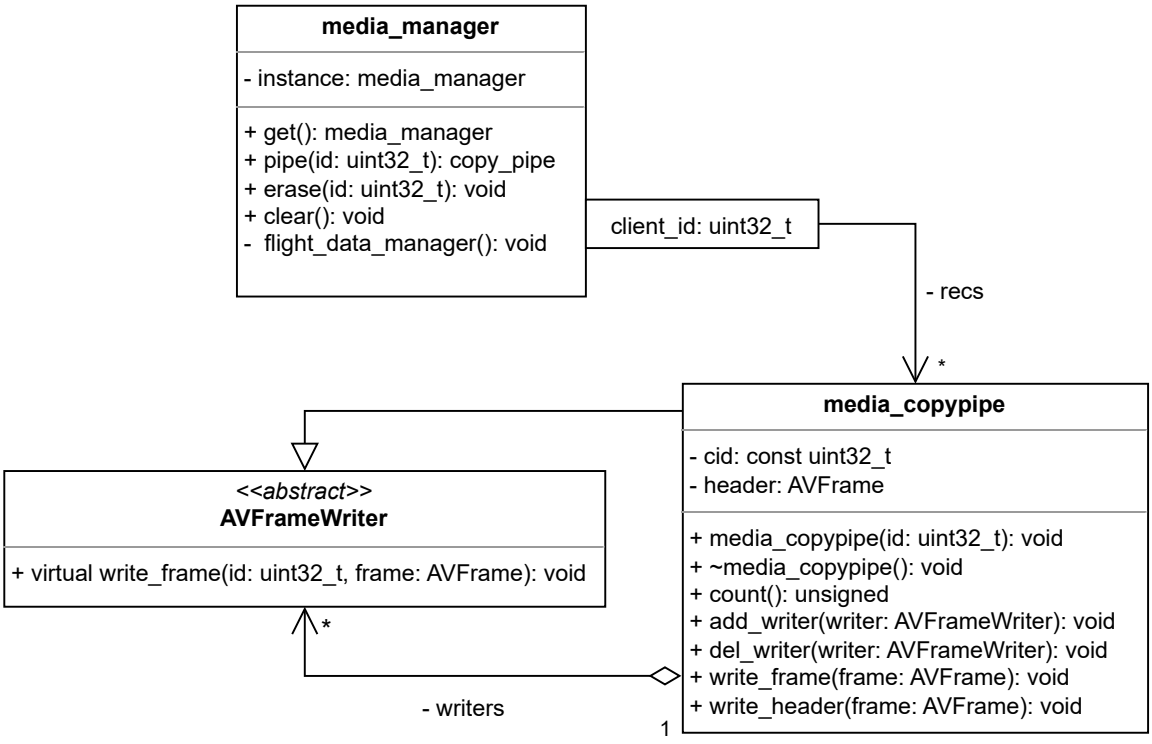
Součástí je i abstraktní třída `StringHandler`, která pouze obsahuje funkci, jež přijímá identifikátor klienta a zprávu a vrací boolean. Třída slouží jako rozhraní pro zpracování zpráv. Další třída `flight_data_manager` je jedináček a slouží pro správu a posílání zpráv třídám, které vycházejí ze zmíněné abstraktní třídy. Tedy třída disponuje seznamem tříd a funkcí, která pro každou abstraktní třídu ze seznamu provede její zpracování nějaké zprávy a vrací logický součet vrácených hodnot. Společně tyto třídy umožňují modulům získávat zprávy posílané serveru a reagovat na ně. Díky tomu mohou moduly samostatně komunikovat s klienty a rozšiřovat tak komunikační protokol. Rozhraní je vyobrazeno na UML diagramu tříd na obrázku 5.3.

Rozhraní multimédií

Rozhraní multimédií je založeno na dvou třídách a jedináčku. Všechny jsou zobrazeny včetně vazeb na UML diagramu tříd na obrázku 5.4. Společně umožňují přeposlání snímku z video streamu k modulům, které o to požádaly. Za snímek se považuje třída `AVFrame` z využité knihovny FFmpeg. Princip funguje podobně, jako u rozhraní zpráv. Je definována abstraktní třída `avframe_writer`, která obsahuje virtuální funkci `write_frame`, jež má parametry – identifikátor klienta a snímek.

Další třída má název `media_copypipe` a slouží jako kontejner pro zmíněné abstraktní třídy `avframe_writer`. Zároveň je od této třídy derivovaná a tyto třídy lze teoreticky řetězit. Třída přepisuje virtuální funkci `write_frame` tak, že všem abstraktním třídám ze seznamu přepošle snímek, který dostala.

Poslední částí rozhraní je jedináček `media_manager`, který dle identifikátoru klienta vysílá mediální přenos na instanci třídy `media_copypipe`. Jedináček disponuje funkcí `pipe` pro získání instance zmíněné třídy dle zadaného identifikátoru. Obsahuje i funkci pro odstranění instance třídy.



Obrázek 5.4: UML diagram znázorňující strukturu části modulárního rozhraní, která zařizuje přenos snímků mezi moduly.

Ostatní třídy součástí jádra

Součástí společného rozhraní jsou i třídy, které mohou být využity různými moduly. Mezi ty nejdůležitější patří `logger` a `settings`.

První z těchto tříd slouží jako jednotné rozhraní pro výpis na standardní výstup. Slouží především k výpisu při nějaké komplikaci nebo při informování uživatele o změně stavu. Disponuje čtyřmi úrovněmi zpráv – ladící informací, obecnou informací, varováním a chybou. Podle zadané úrovně z nastavení, vypisuje pouze ty zprávy, které nejsou nižší úrovně.

Druhá z tříd slouží jako jedináček disponující hodnotami nastavení. Slouží jako dynamické rozhraní, díky kterému mohou jednotlivé moduly přistupovat k nastaveným hodnotám. Jedináček se skládá ze tří seznamů typu klíč-hodnota – `dint`, `dbool`, `dstring`. Každý z nich má klíč typu řetězec a datový typ hodnoty dle názvu proměnné. Moduly mohou přistupovat k tomuto nastavení díky znalostem klíče a datového typu hodnoty. Moduly si nesmí přepisovat nastavení a musí si dávat pozor na použití stejných kombinací klíče a typu hodnoty. Jedináček zároveň disponuje funkcí `set_loglevel`, jež nastaví úroveň rozhraní pro výpis na standardní výstup.

Další důležitou třídou je `async_handler`. Tato primitivní abstraktní třída umožňuje asynchronní provádění operací nad daty, jejichž datový typ je dán dle šablony třídy. Třída obsahuje abstraktní funkci `handle`, která má jeden argument a to data. Disponuje i veřejnou funkcí `add_job`, která umístí data do fronty. Vlákno vzniklé v konstruktoru bere data z fronty a zpracovává je pomocí abstraktní funkce `handle`. Vlákno využívá pasivní čekání pomocí semaforu.

Následující dvě třídy, `tcp_server` a `tcp_session`, se věnují abstraktnímu pojetí serveru protokolu TCP a jeho připojených klientů, jinak známých jako sezení. Třídy využívají socketů od knihovny Boost. Třída `tcp_server` neobsahuje žádné virtuální funkce, ale díky šablonování je možné určit třídu, která se vytvoří při novém připojení a předá jí klienta. Třída staticky kontroluje, jestli klienta předává třídě, která je derivována od abstraktní třídy `tcp_session`. Součástí třídy je funkce `run`, která spustí naslouchání na zadaném portu a verze IP protokolu (výchozí protokol je IP verze 4).

Třída `tcp_session` spravuje připojení konkrétního TCP klienta. Disponuje mnoha funkcemi, z nichž je několik abstraktních, které jsou volány při různých událostech. Třída obsahuje tyto abstraktní funkce:

- `on_connect` – funkce je volána ihned při vytvoření spojení.
- `on_write` – funkce je volána při zapsání dat na socket. Parametrem je počet zapsaných bytů.
- `on_read` – funkce je volána při přečtení dat ze socketu. Parametry jsou data a počet přečtených bytů.
- `on_close` – funkce je volána při ukončení spojení.
- `on_error` – funkce je volána při zachycení chyby. Parametrem je kód chyby. Funkce už obsahuje implementaci, která zapíše všechny chyby na standardní výstup pomocí třídy `logger`.

Mezi veřejné funkce třídy patří funkce `run`, která je volána okamžitě po vytvoření spojení. Dále funkce `close`, jež ukončí spojení. Dalšími jsou vstupně/výstupní funkce, které se starají o zápis a čtení ze socketu. Existují dva páry - blokující a neblokující. Jedná se o funkce `read`, `write` a `async_read` a `async_write`.

Posledními dvěma třídami jsou třídy pro zjednodušení práce s kódováním a dekódováním multimediálních paketů z přenosů na snímky a naopak. Využívá se knihovny FFmpeg. Abstraktní třídy jsou symetrické a nazývají se `media_encoder` a `media_decoder`. Obě třídy přijímají v konstruktoru číslo kodeku, použitého při kódování nebo dekódování. Další společnou funkcí je `open`, která otevře kodek pro použití. Funkce musí být zavolána před zápisem do kodeku. Funkce zapisující data do kodeku se jmenují `encode` a `decode` a přijímají jako argument data v podobě `AVFrame` při kódování a `AVPacket` při dekódování. Třídy disponují abstraktními funkcemi `on_encoded` a `on_decoded`. Ty jsou volány, když třída dostane od kodeku výstupní data. Parametry jsou opačné, než byli u vstupu. Tedy u výstupu kódování to je `AVPacket` a u dekódování `AVFrame`.

5.2 Modul pro přenos letových dat

Jak je zmíněno v návrhové kapitole 4.4, přenos letových dat probíhá protokolem Websocket a pomocí formátu JSON. Modul s názvem `websocket` specifikuje abstraktní třídy `FlightDataServer` a `FlightDataClient`, které jsou pojmenovány jako `WebsocketServer` a `WebsocketClient`. Vazby je možné vidět na UML diagramu tříd na obrázku 5.5. První z nich je Websocket server, který naslouchá na zadaném portu. Při implementaci jsem využil knihovny `websocketpp`[3], která řeší problematiku Websocket protokolu. Součástí tohoto mo-

[3]Knihovna websocketpp je dostupná pod licencí MIT z: http://github.com/zaphoyd/websocketpp

dulu je i single header knihovna[4] pro parsování a generování JSON zpráv. Tu jsem převzal do svého řešení.

První třída `WebsocketServer` provozuje Websocket server schopný komunikovat s klienty. Při konstrukci třídy je potřeba zadat ukazatel na `io_context` knihovny Boost. Ten poskytne kontext, kde se vykonává práce Websocket serveru. Třída disponuje funkcí `run`, která přepisuje virtuální funkci rodičovské třídy `FlightDataServer`. Funkce spustí naslouchání na zadaném portu. Další funkce má název `send` a slouží k zaslání zprávy k jednomu klientu. Parametry této funkce jsou ukazatel na klienta a zpráva ve formě řetězce. Poslední funkce `kick` slouží k odpojení klienta. Parametry jsou kromě ukazatele na klienta také kód statusu ukončení spojení a zpráva s odůvodněním. Websocket server při jakémkoliv problému vypisuje hlášení dle důležitosti. Při navázání spojení vytvoří třídu `WebsocketClient` a uloží si ji do seznamu připojených websocket klientů.

Druhou třídou je `WebsocketClient`. Třída kromě `FlightDataClient` derivuje i od třídy `async_handler`, protože veškeré zpracování zpráv probíhá asynchronně. Třída při konstrukci vyžaduje jeden parametr, a to ukazatel na `WebsocketServer`, který tuto instanci třídy vytvořil. Třída disponuje dvěma veřejnými funkcemi přepisujícími rodičovské funkce. První funkce `recv` slouží pro přijímání zpráv. Je volána Websocket serverem a klient zprávu přidá do fronty ke zpracování. Funkce `recv` pouze předá zprávu k poslání Websocket serveru. Ve třídě se také nachází funkce `handle`, která přepisuje rodičovskou třídu od `async_handle`. Ta je volána asynchronně s parametrem zprávy. V této funkci dochází k handshaku popsanému v kapitole 4.3 a také ke zpracování zpráv typu broadcast dat a výpis seznamu dronů. Ostatní zprávy jsou předány jedináčku. Ten zprávu zapisuje do ostatních modulů, co o to stojí. Podle návratové boolean hodnotě ví, jestli došlo ke zpracování nebo ne. Pokud ne, dojde k chybě a spojení je ukončeno, protože se jednalo o nepodporovanou zprávu.

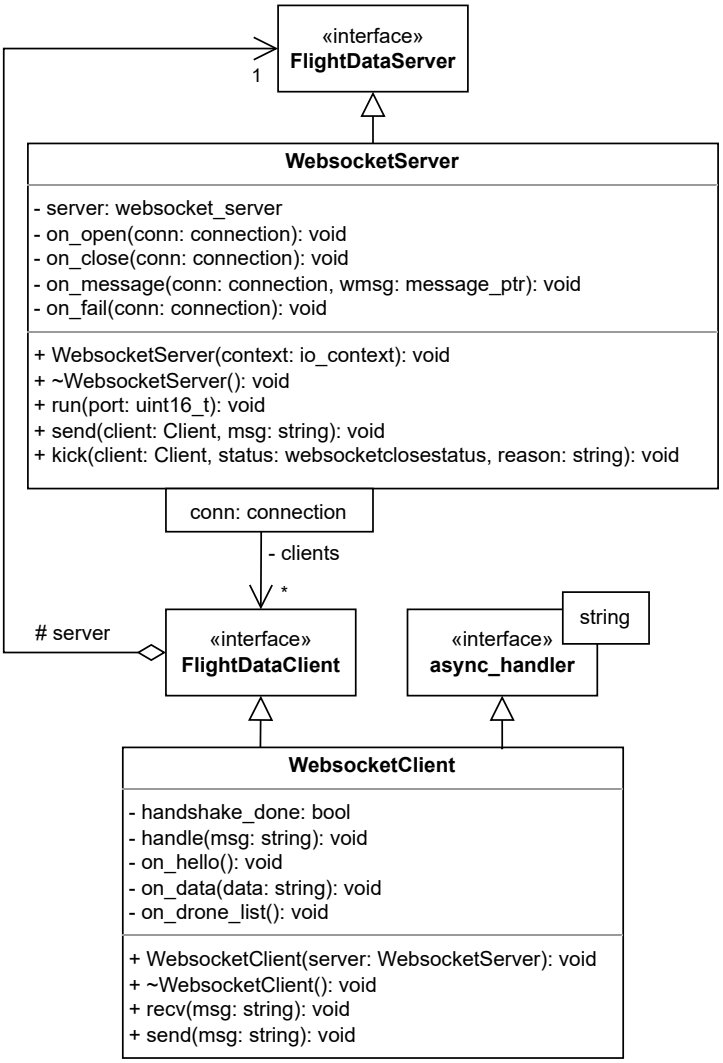
5.3 Modul pro ukládání letových dat

K ukládání letových dat se dle návrhové kapitoly 4.6 využívá externí relační databáze MariaDB, která disponuje vytvořenou databází/schématem. Ukládání je implementováno v modulu `flight_database`, který se stará o získání letových dat ze společného rozhraní a jejich uložení ve zmíněné databázi. Modul se skládá ze dvou tříd nacházejících se na UML diagramu na obrázku 5.6.

První třída `FlightDatabase` zajišťuje připojení k databázi, získání dat z rozhraní a jejich uložení v databázi pomocí vytvořeného spojení. Třída využívá knihovny `mariadb++`[5], která slouží jako konektor k databázi. V konstruktoru dochází k vytvoření spojení a zkontrolování, jestli v databázi/schématu existuje tabulka pro letová data. Pokud se tato tabulka v databázi nenachází, tak dojde k vytvoření tabulky. Třída dědí od abstraktní třídy `StringHandler` a přepisuje její funkci `handle_string`. V této funkci analyzuje zprávy od rozhraní a pokud se jedná o letová data, tak je uloží do databáze.

[4]Single header knihovna (obsahuje pouze jeden hlavičkový soubor) pro manipulaci s formátem JSON v jazyku C++ je volně dostupná pod MIT licencí z: http://github.com/nlohmann/json

[5]Knihovna `mariadb++` je open-source knihovna sloužící jako konektor k databázi MariaDB licencovaná pod BSL-1.0 licencí. Knihovna je dostupná z: https://github.com/viaduck/mariadbpp

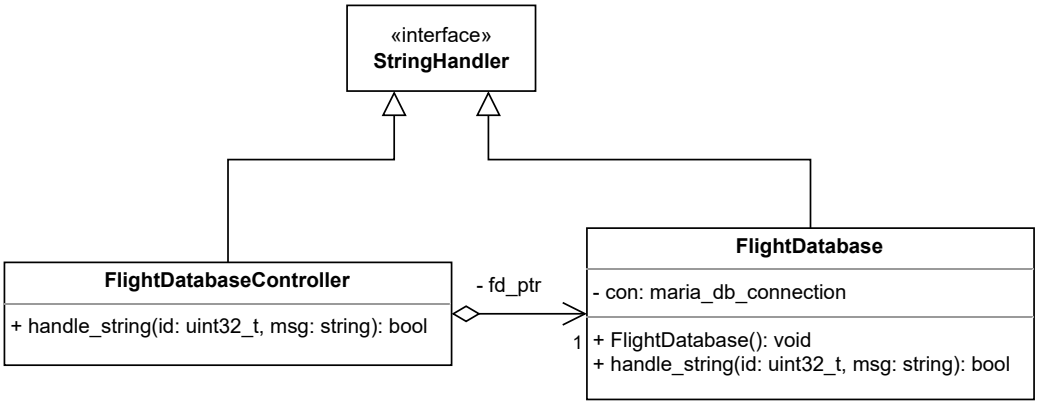


Obrázek 5.5: UML diagram tříd znázorňující strukturu modulu `websocket` včetně vazeb na společné rozhraní.

Informace potřebné pro připojení k databázi získává modul z nastavení ze společného rozhraní (třída `settings`). Jedná se o následující čtyři řetězce:

- mariadb_hostname – adresa zařízení, kde se nachází databáze
- mariadb_username – uživatelské jméno použité pro autorizaci k databázi
- mariadb_password – uživatelské heslo použité pro autorizaci k databázi
- mariadb_database – název konkrétní databáze/schématu, kde se nachází tabulka s daty

Druhá třída s názvem `FlightDatabaseController` slouží pro ovládání instance předchozí zmíněné třídy pomocí zpráv od klientů, dle rozšíření komunikačního protokolu zmíněného v kapitole návrhu. Třída opět dědí od abstraktní třídy `StringHandler` a přepisuje tutéž funkci. V té dochází k vytvoření, nebo ke zničení instance třídy `FlightDatabase` nebo



Obrázek 5.6: UML diagram tříd znázorňující strukturu modulu `flight_database` včetně vazeb na společné rozhraní.

k ověření její existence. Aplikace operátorů mohou zvolit, zda se mají letová data nahrávat nebo ne. Modul nerozlišuje, od jakého dronu tato data jsou, takže modul ukládá všechna letová data nezávisle na identifikátoru dronu. Oprávnění zde nejsou řešena, každý klient může povolit nebo zrušit ukládání letových dat.

5.4 Modul pro přenos multimediálních dat

Rozhodl jsem se přenášet multimediální data pomocí protokolu RTMP. Modul `rtmp` má za úkol vytvořit RTMP server, kde se připojí drony a operátoři. Ti jsou schopni si mezi sebou sdílet multimediální data. Vzhledem k nevyhovujícím požadavkům existujících řešení jsem se rozhodl naimplementovat svůj RTMP server. Tedy pouze ty části, které jsou nezbytné pro multimediální přenos. Protokol jako takový byl určen i pro jiné účely, ale těch není využito, tak postrádalo smysl implementovat všechny schopnosti plného RTMP protokolu. Při implementaci RTMP serveru jsem se inspiroval a částečně i převzal řešení `cpp_media_server`[6]. U jednotlivých tříd tohoto modulu vymezím, kterými částmi jsem se inspiroval, čím jsem přispěl do řešení, a co jsem převzal.

Sezení

Základní třídou je `rtmp_session`, která je derivována od třídy `tcp_session` ze sdíleného rozhraní a plní roli RTMP připojení. Opět využívám knihovny Boost. Jediným argumentem této třídy při konstrukci je socket protokolu TCP knihovny Boost, který je rovnou předán abstraktní třídě. Server je pak třída `tcp_server` ze společného rozhraní, která díky šablonování používá třídu `rtmp_session` jako třídu pro správu připojeného klienta. Třída obsahuje vnitřní stavy a řeší problematiku spojení protokolem RTMP. Jak je zmíněno v kapitole 3.2, tak veškerý přenos protokolem je prováděn pomocí několika proudů částí zvaných „chunk“. Třída disponuje jednou virtuální funkcí `on_stream_create`, která je volána při vytvoření nového streamu pro přenos mediálních dat.

Spojení může nabývat sedmi různých stavů. V prvním počátečním stavu se čeká na začátek handshaku od klienta. Jelikož RTMP protokol provádí čtyřcestný handshake, tak

[6] cpp_media_server je volně dostupná multimediální knihovna, která disponuje RTMP serverem. Je licencována jako MIT a nachází se na: http://github.com/grandi23/cpp_media_server-1

odpoví první část handshaku a přejde do druhého stavu, kde čeká na odpověď klienta. Po úspěšné odpovědi je proveden handshake a přejde se do třetího stavu, kde už je klient v pořádku připojen a může nastat přenos dat přes chunk streamy. Pak následuje zpráva typu 20 `connect`. Klient žádá o připojení ke konkrétní URL cestě, označující stream a opět se změní stav. Přenosem dat přes chunk streamy se očekává, že klient požádá server o vytvoření instance streamu. Server žádost splní a přejde do dalšího stavu, kde se čeká na žádost klienta o přenos dat. Existují dvě žádosti o přenos dat. První je zpráva typu 20 `publish`, kde klient oznamuje, že bude data vysílat. Druhá typu 20 `play`, kde bude klient data přijímat. Tím dojde ke změně stavu na předposlední stav. Poslední stav značí ukončení spojení.

Třída obsahuje instance třídy, provádějící handshake a zpracování chunk streamů. Úkolem této třídy je získat data ze socketu, na základě stavu buď předat data třídě, která se stará o handshake nebo přečte první hlavičku s formátem a číslem chunk streamu a data předá chunk stream třídě s odpovídajícím číslem. Třída podporuje pouze hlavičku, kde chunk stream dosahuje pouze hodnot 2 až 63. S využitím více chunk streamů jsem se nesetkal. Pro účely přenosu medií si klienti vždy vystačili s pěti. Po vyparsování dat těmito třídami se zvolí reakce v podobě zpracování dat, popřípadě změně stavu. Třída přepisuje rozhraní třídy `tcp_session`, což umožňuje například číst data ze socketu. Zápis se provádí jednou z veřejných funkcí. Funkce ze svých parametrů vytvoří chunk a ten se odešle klientu. Zpracování dat z chunk streamu jsem rozdělil do tří částí. Buď se jedná o data pro ovládání protokolu, data pro příkazy nebo o přenos mediálních dat.

Ovládání sezení protokolu se týká zpráv typu 1 až 6, kde se zprávy věnují například změně limitu velikosti chunku, potvrzení nebo nastavení velikosti potvrzovacího okna. Tedy tato část třídy se věnuje správě protokolu. Server na tyto zprávy patřičně odpovídá. Mezi nejdůležitější patří výše zmíněné. Zprávu o změně velikosti chunku posílá server okamžitě po přijetí zprávy typu 20 `connect` a zvětšuje ji ze 128 bytů na 4096. Společně s ní posílá i zprávu na změnu velikosti okna pro potvrzení, kterou nastavuje na 2.5 MB. Klient nebo server po přijetí chunků přesahující tuto velikost, posílá potvrzující zprávu. Server si musí počítat množství přijatých dat a posílat potvrzení.

Příkazy protokolu se zabývají zprávami typu 20. Jedná se o již zmíněné příkazy/žádosti/oznámení o připojení, vytvoření streamu, oznámení o vysílání nebo přijímání dat. Existují i další zprávy, které server přijímá, ale ty jsou převážně ignorovány, protože nenesou žádnou důležitou informaci a dokonce nejsou součástí dokumentace a chování se u různých klientů liší.

Zprávy typu 8, 9 a 18 jsou považovány za přenos mediálních dat. Tato data jsou pouze předána patřičné instanci třídy, zajišťující přenos dat ostatním klientům, připojeným ke stejnému streamu.

Ostatní typy zpráv nejsou podporovány. Jedná se hlavně o typy 17 a 15, které nesou stejné informace jako 20 a 18, ale používají serializační formát AMF3 [10] namísto AMF0 [9]. Tyto zprávy nejsou podporovány, protože server obsahuje pouze parser formátu AMF0. Implementace parseru formátu AMF3 mě přišla zbytečná, protože jsem se nesetkal s klientem, který by ji využíval. Právě parser formátu AMF0 je částí mého programu, který jsem převzal z existujícího řešení, což bylo zmíněno na začátku této kapitoly 5.4.

Handshake

Handshake je implementován ve třídě `rtmp_handshake`. Dle oficiální dokumentace protokolu, stačí přenášet ve volných polích dat náhodné hodnoty. Existují rozšíření (kapitola 3.2),

která se snaží přenos zabezpečit. Kromě základního řešení byla naimplementována i část rozšíření RTMPE. To využívá algoritmů HMAC [17] s využitím hashe SHA256 [4] a Diffie-Hellman [25]. Pokud se nepovedou kontrolní součty, tak se vrátí k základní implementaci a klienta přijme. Řešení nemá sloužit k zabezpečení. Jde pouze o prvek kompatibility s možnými klienty. Pro zmíněné algoritmy jsem využil knihovnu OpenSSL[7].

Chunk stream

Veškeré zacházení s chunky řeší třída `rtmp_chunk_stream`. Při implementaci jsem se inspiroval řešením, zmíněným na počátku této kapitoly 5.4. Každý chunk stream může nabývat dvou stavů – přijímání hlavičky chunku a přijímání těla chunku. Chunk stream si také ukládá naposledy použité hodnoty v hlavičkách, aby mohl redukovat množství přenesených dat použitím menších hlaviček. Více se o tomto lze dočíst v teoretické kapitole 3.2. Důležité však je, aby třída správně parsovala všechny hlavičky, aby nemohlo dojít k narušení čtení z chunk streamu. Třída podporuje i rozšířenou časovou značku.

Media stream

Stream pro přenos multimediálních dat je implementován v třídě `rtmp_stream`. Ta disponuje seznamem klientů, kteří odebírají data ze streamu a jedním vysílajícím klientem. Třída přijímá data od tohoto klienta v podobě chunk streamu. První věci, co udělá třída po přijetí dat, že je převede na strukturu s názvem `media_packet`. Struktura mediálního paketu, po vzoru zmíněné knihovny, obsahuje informace o RTMP přenosu, druhu dat (audio, video, metadata), použitého kodeku, časové známky, informace o tom, jestli se jedná o klíčové snímky nebo sekvenční hlavičku a data. Některé informace jsou dostupné z RTMP hlavičky a pro některé další se musí parsovat hlavičky přenášených multimediálních dat. Protokol RTMP využívá formátu FLV pro přenos multimediálních dat, takže dochází k získání informací z jeho hlavičky. Získáme tak informace o tom, jestli se jedná o klíčový snímek, sekvenční hlavičku a dostaneme i časovou známku. Po vytvoření mediálního paketu ho stačí přeposlat všem klientům, kteří odebírají data ze streamu. Třída `rtmp_session` disponuje funkcí pro zapsání mediálního paketu přímo na chunk stream. Před tím, než dojde k rozeslání mediálního paketu klientům, se ukládá do cache paměti, převádí se na snímek a posílá se sdílenému rozhraní serveru.

Cache paměť je implementována třídou `gop_cache`. Tuto třídu jsem vytvořil s inspirací ze zmíněného řešení na začátku kapitoly 5.4. Třída slouží pro snížení času čekání na klíčový snímek a metadata při prvotním navázání spojení. Třída dostává mediální pakety a ukládá si je. Pokud se jedná o sekvenční hlavičku nebo metadata, tak si je uloží zvlášť. Všechny ostatní pakety si ukládá do seznamu. Pokud přijde klíčový snímek, tak inkrementuje čítač klíčových snímků. Když přesáhne dané hodnoty, tak seznam vyprázdní a začne ho plnit znovu. Pokud se na stream připojí nový klient, jsou mu naráz poslány všechny hlavičky, metadata a seznam paketů od klíčového snímku včetně po nejnovější paket. Při navázání spojení tedy dochází k největší zátěži na síť.

Poslední třídou tohoto modulu je `flv_demuxer`. Tato třída se stará o převod mediálního paketu na snímek `AVFrame` z knihovny FFmpeg, který je podporován společným rozhraním serveru, které tento snímek posílá dalším modulům na požádání. Současně je dceřinou třídou třídy z rozhraní `media_decoder` a přepisuje funkci `on_decoded`, která je

[7] OpenSSL je populární open-source knihovna disponující mnoha algoritmy pro zabezpečení především komunikací. Knihovna je dostupná z: http://www.openssl.org

volána po dekódování snímku. Snímek je následně předán rozhraní. Cílem je dekódovat pomocí kodeku přenášený paket na snímek. Třída podporuje pouze převod video paketu na video snímek, což vzhledem k požadavkům zadání stačí. Za zmínku stojí obalovací třída `flv_demuxer_wrap`, která vytváří zmíněnou třídu na základě kodeku z paketu. Jsou podporovány pouze kodeky H264 a HEVC.

5.5 Modul pro ukládání multimediálních dat

Modul slouží pro ukládání multimediálních dat do souborů. Využívá se zde knihovny FFmpeg. Modul má název `recorder` a skládá se ze dvou tříd. Ty je možné vidět na UML schématu na obrázku 5.7. První třídou je `media_recorder`, která dědí od abstraktní třídy `avframe_writer`. Zároveň dědí od třídy `media_encoder`, která je součástí tříd ze sdíleného rozhraní. Tato třída umožňuje kódovat získané snímky dle zadaného kodeku. Konstruktor této třídy přijímá jeden parametr, a to řetězec s cestou k souboru. Třída disponuje veřejnou funkcí `write_frame`, která přepisuje funkci z abstraktní třídy. Díky tomu lze třídu vložit do seznamu jedináčka, `media_manager`, který přeposílá získané snímky od ostatních modulů, jako například modulu z předchozí kapitoly 5.4, do ostatních modulů, které si žádají o data. Třída přijímá snímky obrazu ve formátu `AVFrame` a identifikátor klienta, jež tento stream vysílá. Snímek se kóduje a ukládá do souboru. Lze nastavit rozlišení, fps a bitrate. Rozlišení musí být menší nebo rovno rozlišení příchozího obrazu. Modul umožňuje definovat i kodek a formát pixelů. Nastavení těchto hodnot probíhá pomocí jedináčka `settings` ze sdíleného rozhraní.

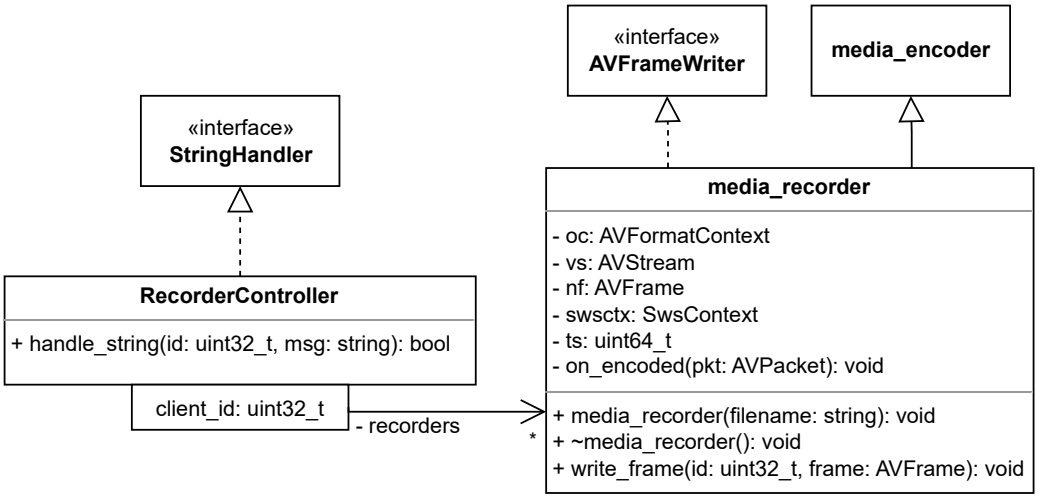
Tyto číselné hodnoty načítá modul od nastavení sdíleného rozhraní:

- record_width – šířka výstupního obrazu
- record_height – výška výstupního obrazu
- record_fps – počet snímků za sekundu výstupního videa
- record_bitrate – množství dat obrazu

Druhou třídou je `RecorderController`, který se stará o přijímání a zpracování žádostí o nahrání video přenosu. Využívá abstraktní třídy `StringHandler`, u které přepisuje funkci `handle_string` a umožňuje tak dostávat zprávy od společného rozhraní. Obdobně jako u třídy `FlightDatabaseController` z kapitoly 4.6, třída zpracovává žádosti o vytvoření nebo smazání instance třídy `media_recorder` a také ověření existence této instance. Pro každý video přenos se vytváří nová instance zmíněné třídy, která je umístěna do třídy společného rozhraní `media_copypipe` odpovídajícího video přenosu. Takto instance dostává video data a provádí nahrávání. Takže hlavní rozdíl mezi ukládáním letových a multimediálních dat je, že nastavení ukládání letových dat platí pro všechny zdroje, zatím co nastavení ukládání video dat se provádí pro každý zdroj zvlášť. Název vytvořeného souboru má formát složený z aktuálního času a identifikátoru klienta:

YYYYMMDD-HHMMSS-CID.*mp4*,

kde YYYY značí rok, MM měsíc, DD den, HH hodinu, MM minutu, SS sekundu a CID značí identifikátor klienta, který vysílá video přenos (např. `20220417-131233-1234abcd.mp4`).



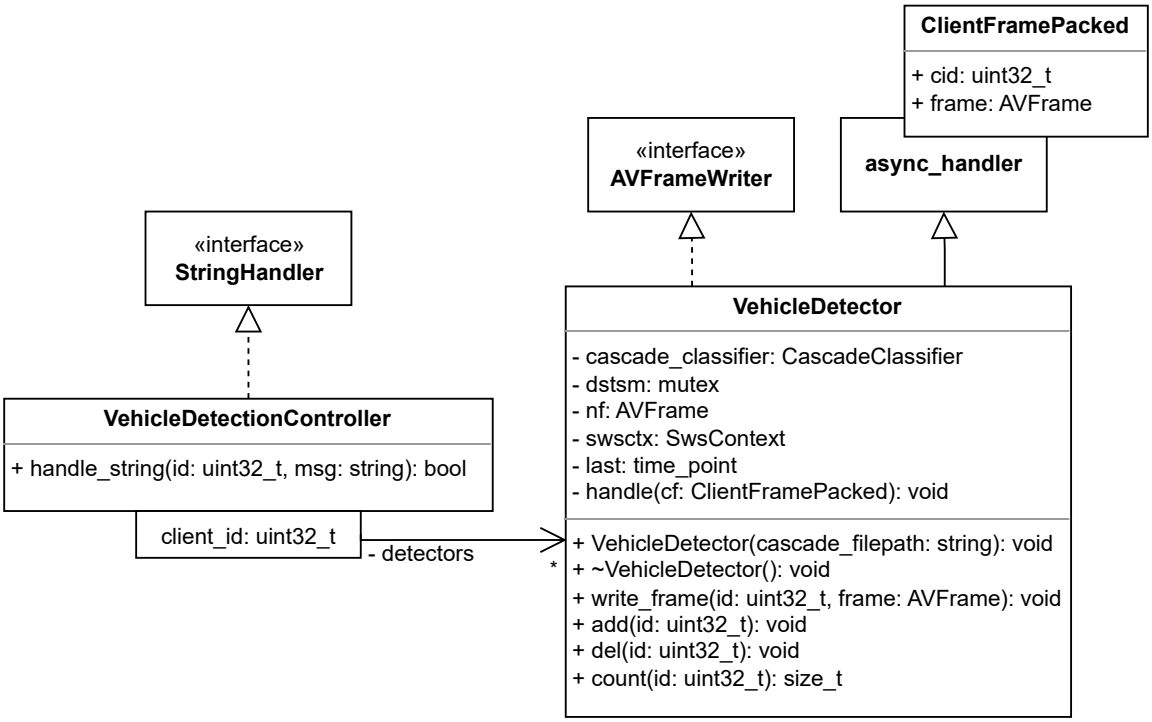
Obrázek 5.7: UML diagram tříd znázorňující strukturu modulu `recorder` včetně vazeb na společné rozhraní.

5.6 Modul pro detekci vozidel z obrazu

Dle návrhu z kapitoly 4.8 byl implementován modul s názvem `vehicle_detection` pro detekci vozidel z obrazových dat video přenosu. Jak je v návrhu zmíněno, modul sloužil pro demonstraci možností mé práce. Modul měl ukázat, že lze snadno propojit části programu a využít je pro rozumné, v praxi využitelné, řešení. Modul se skládá ze dvou tříd (viz UML diagram tříd na obrázku 5.8).

První třída je `vehicle_detection`, která provádí detekci vozidel ze snímků. Ta derivuje od dvou tříd. První je abstraktní třída `AVFrameWriter`, takže se očekává, že třída bude přijímat obrazová data pomocí přepsané funkce *write_frame*. Druhou derivovanou třídou je `async_handler`, díky které se provádí detekce paralelně vzhledem ke komunikaci přes RTMP. Konstruktor této třídy přijímá jeden parametr a to řetězec s cestou k souboru s předtrénovanými daty. Dále třída disponuje dvěma veřejnými funkcemi pro přidání a odebrání klientů, kteří mají zájem o výsledná data. Parametrem je vždy identifikátor klienta. Přepsaná funkce `write_frame` umístí snímek jednou za sekundu do fronty ke zpracování. Na jiném vláknu se paralelně zpracovávají snímky a výsledek se posílá přes společné rozhraní všem klientům, kteří si o to požádali. Zkráceně třída přijímá snímky z video přenosu a jednou za sekundu jeden z nich zpracuje. Výsledné obdélníky s detekovanými vozidly pošle vybraným klientům.

Druhou třídou je `VehicleDetectionController`. Ta se stará o zpracování požadavků klientů, o zapnutí nebo vypnutí posílání detekčních dat. Cílem je přistoupit k instanci předchozí třídy `vehicle_detection` a přidat nebo odebrat klienta z jejího seznamu. Třída využívá společného rozhraní pro přenos zpráv. Je derivovaná od `StringHandler` a přepisuje funkci `handle_string`. V podstatě naslouchá požadavkům klientů a pokud se některý z nich týká žádostí o zapnutí nebo vypnutí detekce, tak jej zpracuje. Třída vytváří instance `vehicle_detection`, pokud alespoň jeden klient požádal o detekci. Poté je do seznamu, ve vytvořené třídě, přidán uživatel jako zájemce o detekční data. Když klient přestane mít zájem o výstup detekce, tak pošle požadavek a je ze seznamu odstraněn, ale instance se neruší. Tímto přístupem se liší od předchozích kontrolérů. Při vytváření instance pro de-



Obrázek 5.8: UML diagram tříd znázorňující strukturu modulu `vehicle_detection` včetně vazeb na společné rozhraní.

tekci používá cestu k souboru s předtrénovanými daty, z nastavení ze společného rozhraní `settings`. Řetězcová položka se jmenuje `vehicle_detection_filepath`.

5.7 Program spojující moduly dohromady

Po naimplementování všech potřebných modulů zbývá vytvořit spustitelný program, který obsahuje všechny moduly a řeší jejich funkcionalitu. Program má za úkol načíst parametry dle dostupných modulů, spustit všechny služby a udržovat je spuštěné a propojené. Zároveň musí řešit signály od operačního systému. V programu je kladen důraz na robustnost a na řešení možných chyb. Pro parsování argumentů programu jsem využil opět knihovnu Boost. Program má k dispozici makra a ta říkají, které moduly jsou dostupné a program se podle toho se chová.

Program obsahuje pouze jednu třídu, která je derivovaná od třídy `rtmp_session`. Přepisuje její jedinou virtuální funkci – `on_stream_create`. Pokud je argumenty programu vynucené ukládání, tak v této přepsané funkci dojde k zajištění ukládání bez ohledu na požadavky klientů a to rovnou při začátku mediálního streamu. Pro detekci vozidel byl použit soubor *cars.xml*[8] s předtrénovanými daty.

[8]Soubor s předtrénovanými daty pro detekci vozidel s využitím kaskádové klasifikace je volně dostupný z: http://github.com/andrewssobral/vehicle_detection_haarcascades/blob/master/cars.xml

Kapitola 6

Experimenty a jejich vyhodnocení

Nedílnou součástí práce je i experimentální vyhodnocení mého řešení. Soustředil jsem se na tři klíčové vlastnosti, kterým odpovídají následující podkapitoly. U každé z nich popíši způsob a nástroje získání a zpracování dat. Následně představím výsledky experimentů a zhodnotím je. Pro vygenerování grafů v této kapitole jsem využil jazyk Python s knihovnami `pandas` a `matplotlib`. Všechny použité skripty, programy a naměřená data jsou součástí řešení a lze je najít dle přílohy A. Experimenty byly provedeny na zařízení Intel NUC, které disponuje procesorem `Intel(R) Core(TM) i7-10710U`. Procesor má šest jáder a celkem dvanáct vláken. Zařízení má k dispozici 16 GB RAM.

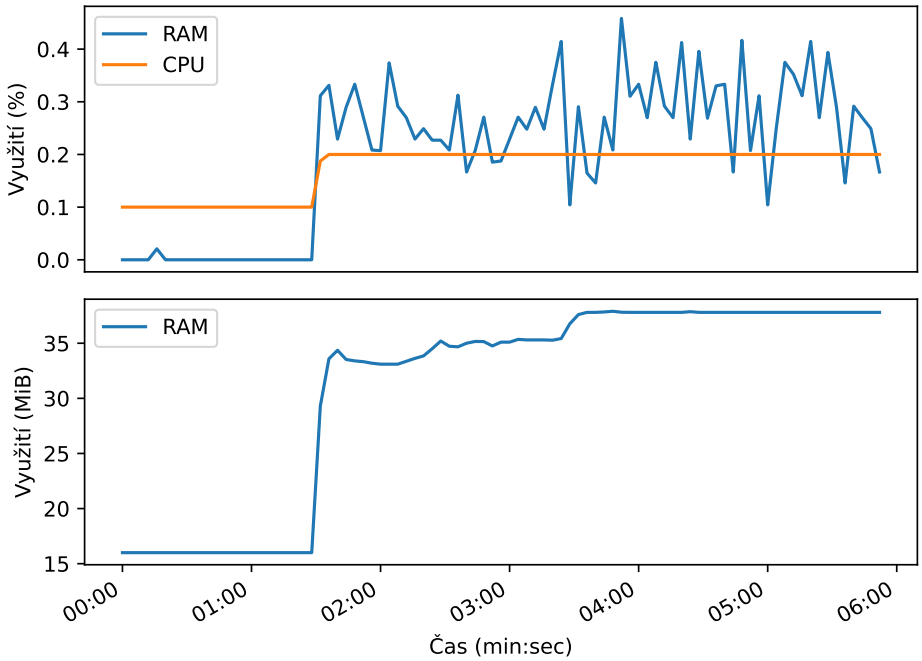
6.1 Hardware vytížení

Věnoval jsem se měření využitých výpočetních zdrojů během několika různých situací. Konkrétně jsem se zaměřil na využití procesoru a paměti. U procesoru pouze na jeho procentuální využití, ale u paměti byla měřena i velikost využité paměti. Použitým nástrojem pro měření byl unixový program `top`. Měření bylo prováděno dvakrát za sekundu a výsledek byl zaznamenán do textového souboru. Velikost využitých pamětí byla zaznamenána v MiB. Pro záznam byl využit skript *capture_hw.sh*, který spustí aplikaci dodanou parametrem, včetně následujících argumentů a provádí záznam do souboru s názvem *hw.txt*. Pro zpracování dat do formátu `csv` jsem využil unixový program `awk`. Experimenty probíhaly během čtyř různých situací:

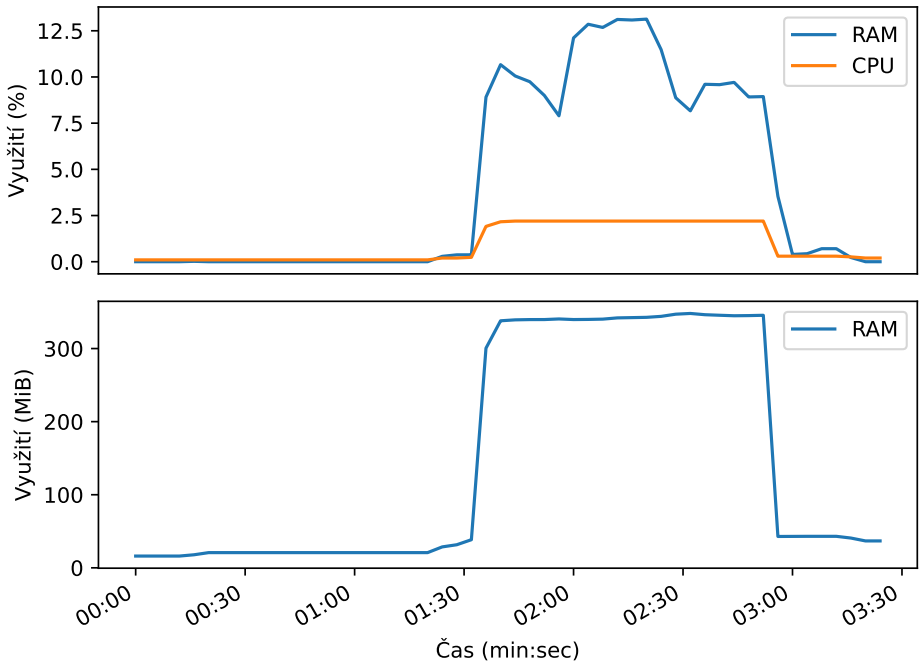
- Připojený jeden dron a vizualizační aplikace.
- Připojený jeden dron a nahrávání obrazu.
- Připojený jeden dron a vizualizační aplikace, přičemž se provádí detekce.
- Připojeny dva drony a vizualizační aplikace. Provádí se nahrávání i detekce zároveň.

Výsledek prvního scénáře je možné vidět na obrázku 6.1. Grafy nedisponují tolika zajímavými informacemi. Je zde jasně vidět začátek přenosu dat a také, že pouhé přeposílání letových a multimediálních dat skoro vůbec nezatěžují zařízení.

U druhého scénáře je výsledek zajímavější. Jak lze vidět na obrázku 6.2, tak od začátku až do konce nahrávání dojde ke skokovému nárůstu využití procesoru i paměti. V části experimentu dojde k využití až 12.5% procesoru. To je především způsobeno překódováním obrazu při ukládání video dat v závislosti na zadaném nastavení. Z grafů je patrný začátek a konec nahrávání.



Obrázek 6.1: Grafy znázorňující vytížení procesoru a paměti serveru v čase pro jeden připojený dron a vizualizační aplikaci. Pro přehlednost jsou data seskupena po čtyřech sekundách a zprůměrována.



Obrázek 6.2: Grafy znázorňující vytížení procesoru a paměti serveru v čase pro jeden připojený dron a spuštěné nahrávání video přenosu. Pro přehlednost jsou data seskupena po čtyřech sekundách a zprůměrována.

Třetí scénář se hodně podobá předchozímu. Opět dojde ke skokovému nárůstu vytížení procesoru a paměti, jak je možné vidět na obrázku 6.3, ale není tak znatelný jako u nahrávání. To je především tím, že detekci provádíme pouze jednou za sekundu a i tak lze zřetelně vidět nárůst využití. Opět lze jasně vyčíst začátek a konec detekce ve využití procesoru. V případě paměti zůstávají data uložena v RAM. To je dáno tím, že instance pro detekci zůstává v paměti, kdyby se nějaký klient znovu rozhodl zapnout detekci.

V posledním případě dochází ke kombinaci všech předchozích, ale se dvěma drony. Takže dochází k přenosu dvou multimediálních přenosů a současně je u obou zapnuto nahrávání a detekce. Podle grafů z obrázku 6.4 dochází k využití téměř 30% procesoru a přibližně 800 MiB paměti. Z grafů lze také vyčíst, že první došlo k zapnutí detekce okolo druhé minuty. Paměť po vypnutí zůstává na úrovni z toho času. Detekce nevytíží procesor tolik co nahrávání. Nahrávání je jasně patrné na první pohled, dokonce dle menších zubů okolo třetí a půl páté minuty lze rozpoznat, kdy došlo k zapnutí a vypnutí nahrávání druhého video přenosu.

6.2 Potřebná šířka pásma

Další experimenty byly zaměřeny na množství přenesených RTMP paketů za jednotku času. Cílem bylo zjistit potřebnou šířku síťového pásma pro přenos multimediálních dat. K zaznamenání RTMP paketů byl využit program `TShark`[1]. Program byl nastaven, aby odposlouchával data na výchozím RTMP portu 1935 a zaznamenával komunikaci do souboru s příponou *.pcap*. Tento soubor byl zpracován stejným programem, který ze zaznamenaných paketů extrahoval informace o relativním čase, zdrojové a cílové IP adresy a payload TCP protokolu v hexadecimálním řetězci do souboru s formátem `csv`. Výstupem experimentů bylo množství přenesených dat v čase. Experimenty jsem provedl pro dvě situace:

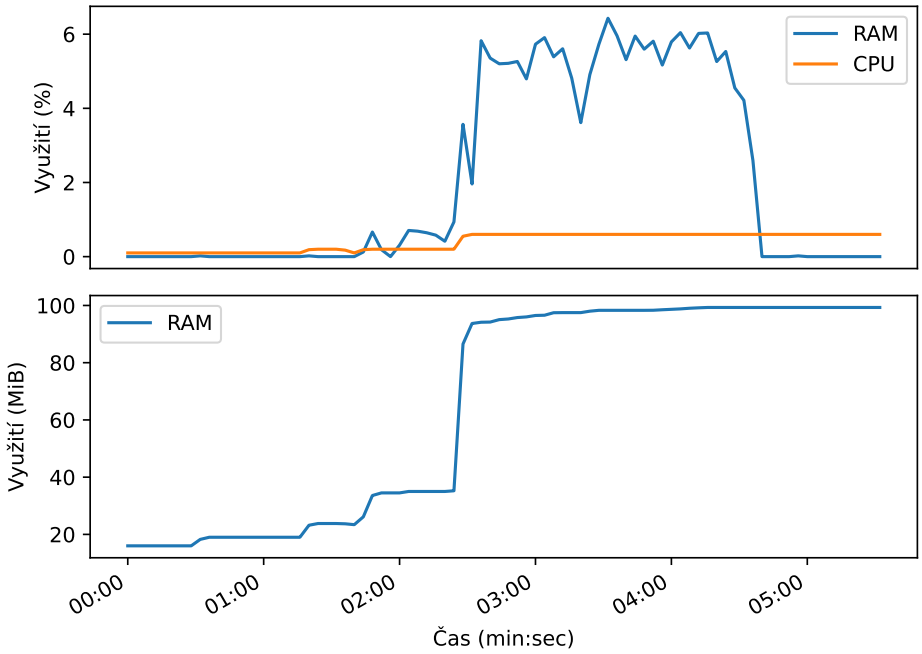
- Připojený jeden dron a vizualizační aplikace.
- Připojeny dva drony a vizualizační aplikace.

V prvním scénáři došlo k přenosu celkem 329.5 MiB dat po dobu 317 s. Tedy využitá šířka pásma internetového připojení odpovídá průměrně 1.0 MiB/s. V následujícím grafu 6.5 lze vidět, že docházelo k výkyvům, kdy během jedné sekundy bylo přeneseno až čtyřikrát více, než je zmíněný průměr. Tyto výkyvy by měly značit přenos klíčových snímků. Z grafu je také patrné, kdy byl přenos zahájen. Téměř nulové hodnoty na začátku značí, že se první připojila vizualizační aplikace, která čekala na video data. Dron se připojil a začal vysílat až po první minutě. Při ignorování počátku, kdy nedocházelo k přenosu je pak průměrná šířka využitého pásma přibližně 1.28 MiB/s.

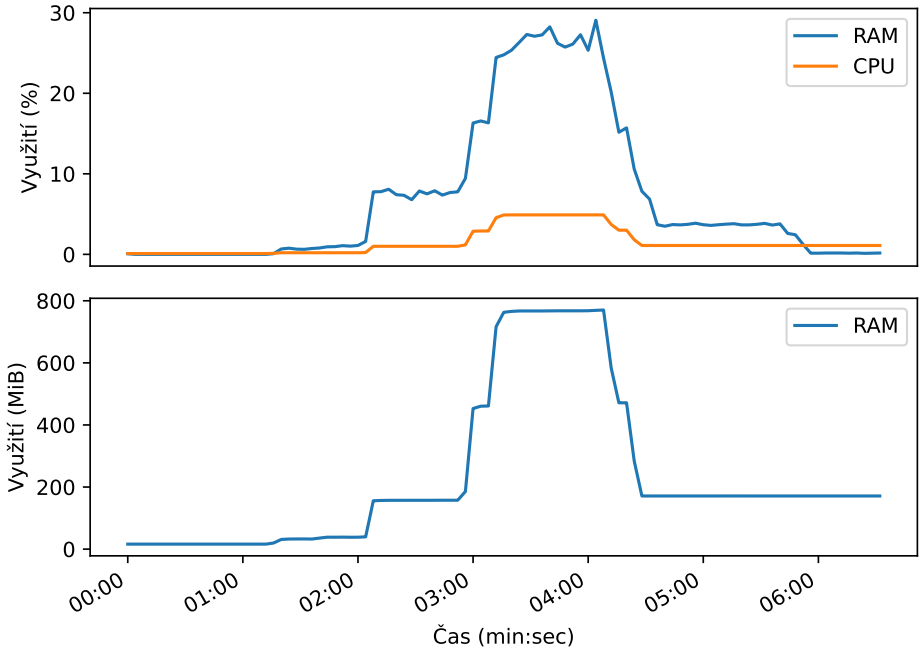
Ve druhém scénáři lze sledovat podobné výkyvy. Bylo přeneseno celkem 1.0 GiB dat po dobu 298.6 s, což znamená, že využitá šířka pásma připojení byla průměrně 3.5 MiB/s. Jak je možné vidět na grafu obrázku 6.6, přenos začíná hned ze začátku, protože se připojil dron jako první a hned vysílal.

Ze získaných informací lze například přibližně vypočítat, jaké množství dat by mohlo být přeneseno za jeden let do vybití baterie dronu (cca 20 minut). Pro případ nasazení vizualizační aplikace a serveru v reálné misi dvou dronů by bylo přeneseno přibližně 4 GiB dat do vybití baterií.

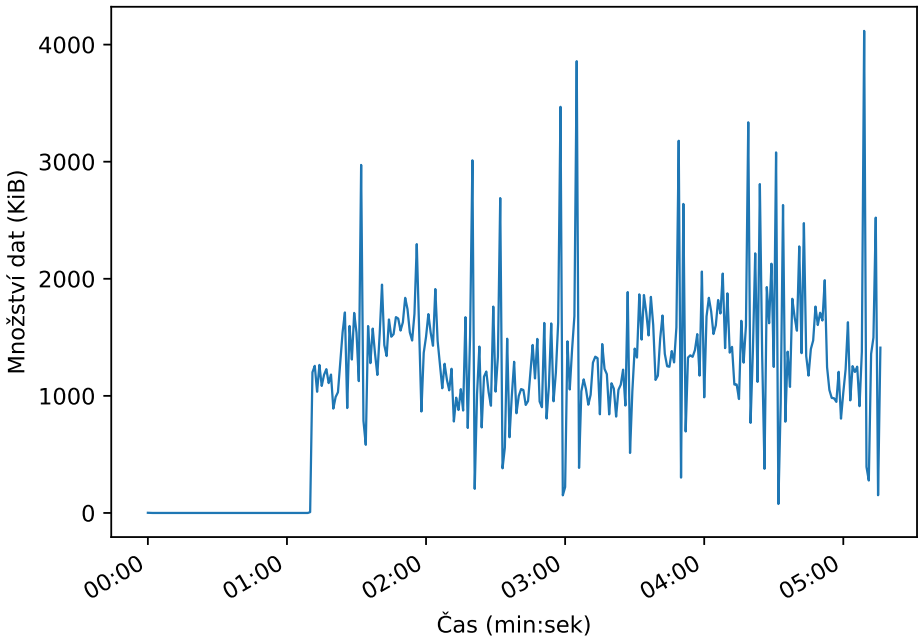
[1] TShark je program pro analyzování a filtrování síťového provozu ať už ze souboru nebo z živého přenosu. Program je dostupný z: http://tshark.dev/



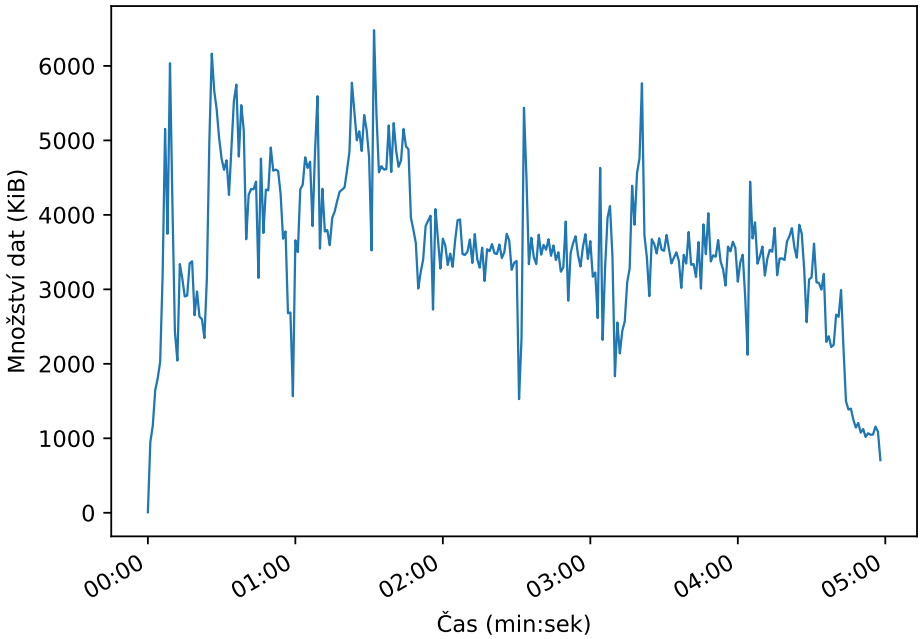
Obrázek 6.3: Grafy znázorňující vytížení procesoru a paměti serveru v čase pro jeden připojený dron, vizualizační aplikace a spuštěnou detekci z video přenosu. Pro přehlednost jsou data seskupena po čtyřech sekundách a zprůměrována.



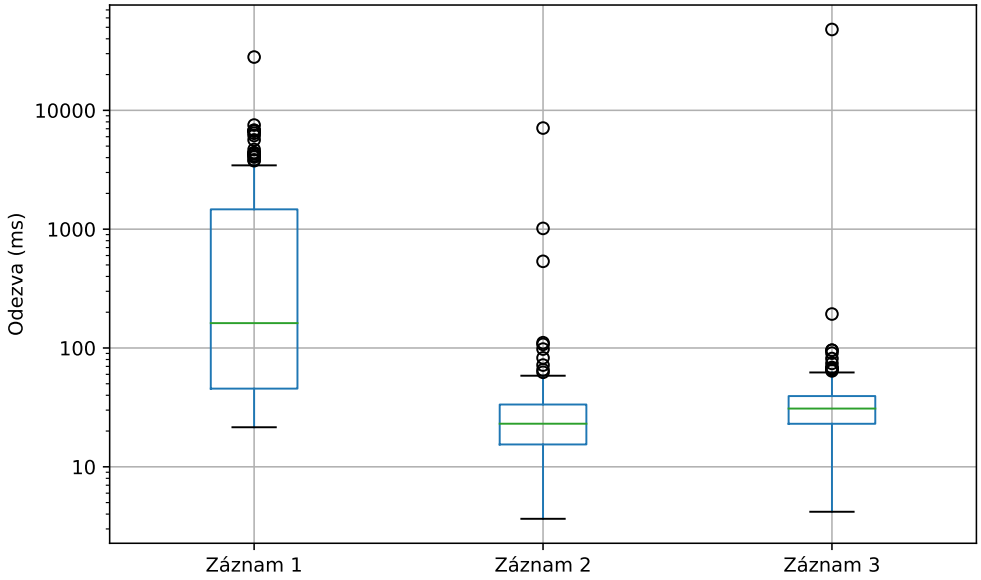
Obrázek 6.4: Grafy znázorňující vytížení procesoru a paměti serveru v čase pro dva připojené drony, vizualizační aplikace a spuštěnou detekci a nahrávání obou video přenosů. Pro přehlednost jsou data seskupena po čtyřech sekundách a zprůměrována.



Obrázek 6.5: Graf znázorňující množství přenesených dat z TCP payloadů posílaných přes port 1935 za čas na síťovém rozhraní serveru během připojení jednoho dronu a vizualizační aplikace.



Obrázek 6.6: Graf znázorňující množství přenesených dat z TCP payloadů posílaných přes port 1935 za čas na síťovém rozhraní serveru během připojení dvou dronů a vizualizační aplikace.



Obrázek 6.7: Krabicový graf[2]znázorňující odezvy pro jednotlivé záznamy.

6.3 Odezva

Poslední experiment byl zaměřen na odezvu RTMP paketů na serveru. Tedy jak dlouho trvá získání, zpracování a odeslání RTMP paketů. Byly využity záznamy přenosů použité z minulé kapitoly 6.2. Výpočet odezvy byl proveden analýzou TCP payloadů z `csv` souborů. Pro každý paket byla extrahována část dat, která se hledala ve všech ostatních. Pokud se nalezl alespoň jeden další paket obsahující zvolený řetězec a měli odlišné IP adresy, tak se jednalo o data, která byla přeposlána. Odezva těchto paketů byl rozdíl relativních časů. Nicméně zpracování všech paketů tímto způsobem bylo časově neúnosné, tak bylo náhodně vybráno 20000 paketů. Výstupem je množství časových odezv v průběhu času. Záznamy odpovídají následujícím třem scénářům:

- Připojený jeden dron a vizualizační aplikace.
- Připojeny dva drony a vizualizační aplikace.
- Připojeny dva drony a vizualizační aplikace. Provádí se nahrávání i detekce zároveň.

Z krabicovém grafu z obrázku 6.7 je možné vyčíst, že během prvního záznamu se klientská aplikace a dron potýkaly s problémy s připojením, protože odezva se pohybovala ve stovkách milisekund. V průběhu dalších záznamů bylo připojení stabilnější a odezva se pohybovala v desítkách milisekund, ale na grafu lze vidět, že docházelo k výkyvům, které jsou znázorněny odlehlými hodnotami v řádu stovek milisekund až sekund. Důvodem této nestability, hlavně u prvního záznamu, bylo pravděpodobně mobilní připojení použité během měření. Tedy program je schopen v řádu desítek až stovek milisekund přijmout paket video přenosu a odeslat jej vizualizačním aplikacím a to i v případě provádění nahrávání a detekce zároveň. Lze tedy říci, že vizualizační aplikace dostává relativně živý obraz.

[2]Krabicový graf vychází ze statistiky a vizualizuje číselná data podle jejich kvartilů. Vršek obdélníku značí 3. kvartil a spodek 1. kvartil. Zelená čára v obdélníku značí 2. kvartil, jinak známý jako medián. Tedy v obdélníku se nachází polovina hodnot ze záznamu. Čáry nad a pod obdélníkem značí minimum a maximum. Některé odlehlé hodnoty byly vyjmuty a jsou znázorněny kroužky.

Kapitola 7

Závěr

Cílem diplomové práce bylo navržení a vytvoření programu schopného sdílet letová a obrazová data mezi drony a jejich operátory. Následně jsem měl provést experimenty, které vyhodnotily vlastnosti mého řešení. Dalším úkolem bylo zhotovení prezentačního videa zobrazujícího klíčové vlastnosti řešení.

Nejprve jsem se však musel obeznámit s existující vizualizační aplikací DroCo, mobilní aplikací DJI Streamer a primitivním serverem, jež jsem měl nahradit. Ponořil jsem se tak do problematiky zacházení s drony a potřeb operátorů. Dokonce mi byl školou zapůjčen dron a já si tak mohl vyzkoušet řešení.

Dále jsem musel prostudovat problematiku tvorby serverů a analyzovat metody sdílení letových a multimediálních dat. Obeznámil jsem se tak s mnoha protokoly. U letových dat to byl především protokol WebSocket a u multimediálních to byl protokol RTMP. Seznámil jsem se také s protokoly RTSP, HLS nebo WebRTC. Navíc jsem také analyzoval možnosti ukládání těchto dat na serveru. Kromě připomenutí informací o relačních a NoSQL databází, jsem se dozvěděl mnoho o kodecích AVC a HEVC, nebo o kontejnerech MPEG-4 a Matroska.

Na základě získaných informací, jsem sestavil architekturu systému určeného pro sdílení letových a multimediálních dat mezi drony a operátory. Navrhnul jsem modulární rozhraní serveru, které zajišťuje snadnou rozšiřitelnost. Dále jsem vymyslel komunikační protokol pro přenos zpráv mezi serverem a jeho klienty s ohledem na možná rozšíření protokolu moduly. Rozdělil jsem si řešenou problematiku do jednotlivých modulů, u kterých jsem následně vybral technologie, které využívám při implementaci řešení na základě informací z kapitoly 3.

Po návrhu následoval popis implementace. Jako první jsem na začátku kapitoly 5 vypsal hlavní využité technologie, které jsou součástí společného rozhraní programu. Právě tomuto rozhraní jsem se dále věnoval. Popsal jsem správu klientů, rozhraní zpráv, rozhraní multimédií a ostatní součásti rozhraní programu. Poté jsem přešel k popisu implementace jednotlivých modulů, včetně jejich vazeb na společné rozhraní. Kromě modulů pro sdílení letových a multimediálních dat, jsem také naimplementoval moduly pro ukládání dat a detekci vozidel z obrazu. Na následujícím obrázku 7.1 je možné vidět aplikaci DroCo, která přijímá obraz dronu včetně provedené detekce. Následoval popis spustitelného programu, který svazuje moduly dohromady.

Pro zjištění kvality implementovaného řešení, jsem provedl několik experimentů zaměřených na vytížení procesoru a RAM, využití šířky pásma připojení a odezvy programu (doba od přijetí do odeslání RTMP paketu na serveru). Popsal jsem zařízení, na kterém byly prováděny experimenty, metody získání a zpracování dat, znázornil jsem výsledky v grafech a zhodnotil je.

Obrázek 7.1: Aplikace DroCo s obrazem od dronu včetně vykreslených obdélníku detekce vozidel.

Literatura

[1] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N. et al. *Simple Object Access Protocol (SOAP) 1.1* [online]. Květen 2000 [cit. 2023-05-12]. Dostupné z: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[2] Bray, T. *The JavaScript Object Notation (JSON) Data Interchange Format* [online]. RFC Editor, prosinec 2017 [cit. 2023-05-12]. DOI: 10.17487/RFC8259. Dostupné z: https://www.rfc-editor.org/info/rfc8259.

[3] *Extensible Markup Language (XML) 1.0 (Fifth Edition)* [online]. Listopad 2008 [cit. 2023-05-12]. Dostupné z: https://www.w3.org/TR/2008/REC-xml-20081126/.

[4] Dang, Q. *Secure Hash Standard (SHS)*. Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, 2012-03-06 2012. DOI: 10.6028/NIST.FIPS.180-4.

[5] Eddy, W. *Transmission Control Protocol (TCP)* [online]. RFC Editor, srpen 2022 [cit. 2023-05-12]. DOI: 10.17487/RFC9293. Dostupné z: https://www.rfc-editor.org/info/rfc9293.

[6] Fleischman, E. W. *WAVE and AVI Codec Registries* [online]. RFC Editor, červen 1998 [cit. 2023-05-12]. DOI: 10.17487/RFC2361. Dostupné z: https://www.rfc-editor.org/info/rfc2361.

[7] Freier, A. O., Karlton, P. a Kocher, P. C. *The Secure Sockets Layer (SSL) Protocol Version 3.0* [online]. RFC Editor, srpen 2011 [cit. 2023-05-12]. DOI: 10.17487/RFC6101. Dostupné z: https://www.rfc-editor.org/info/rfc6101.

[8] Hickson, I. *WebRTC: Real-time communication in browsers* [online]. Mar 2023 [cit. 2023-05-12]. Dostupné z: https://www.w3.org/TR/webrtc/.

[9] Inc, A. S. *Action Message Format - AMF 0* [online]. 2006 [cit. 2023-05-12]. Dostupné z: https://rtmp.veriskope.com/pdf/amf0-file-format-specification.pdf.

[10] Inc, A. S. *Action Message Format - AMF 3* [online]. 2006 [cit. 2023-05-12]. Dostupné z: https://rtmp.veriskope.com/pdf/amf3-file-format-spec.pdf.

[11] Inc, A. S. *Adobe Flash Video File Format Specification Version 10.1* [online]. 2010 [cit. 2023-05-12]. Dostupné z: http://download.macromedia.com/f4v/video_file_format_spec_v10_1.pdf.

[12] *Information technology — Coding of audio-visual objects — Part 10: Advanced Video Coding*. Standard. Geneva, CH: International Organization for Standardization, prosinec 2003.

[13] *Information technology — Coding of audio-visual objects — Part 14: MP4 file format*. Standard. Geneva, CH: International Organization for Standardization, listopad 2003.

[14] *Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 2: High efficiency video coding*. Standard. Geneva, CH: International Organization for Standardization, srpen 2020.

[15] Josefsson, S. *The Base16, Base32, and Base64 Data Encodings* [online]. RFC Editor, říjen 2006 [cit. 2023-05-12]. DOI: 10.17487/RFC4648. Dostupné z: https://www.rfc-editor.org/info/rfc4648.

[16] Kim, S., Jung, H., Lee, S., Park, J., Yu, S. et al. A Study of real-Time 4K drone images visualization to rescue for missing people base on web. In: *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*. 2022, s. 1594–1596. DOI: 10.1109/ICTC55196.2022.9952607.

[17] Krawczyk, D. H., Bellare, M. a Canetti, R. *HMAC: Keyed-Hashing for Message Authentication* [online]. RFC Editor, únor 1997 [cit. 2023-05-12]. DOI: 10.17487/RFC2104. Dostupné z: https://www.rfc-editor.org/info/rfc2104.

[18] Lhomme, S., Bunkus, M. a Rice, D. *Matroska Media Container Format Specifications* [online]. Internet-Draft draft-ietf-cellar-matroska-16. Internet Engineering Task Force, duben 2023 [cit. 2023-05-12]. Work in Progress. Dostupné z: https://datatracker.ietf.org/doc/draft-ietf-cellar-matroska/16/.

[19] Melnikov, A. a Fette, I. *The WebSocket Protocol* [online]. RFC Editor, prosinec 2011 [cit. 2023-05-12]. DOI: 10.17487/RFC6455. Dostupné z: https://www.rfc-editor.org/info/rfc6455.

[20] Nielsen, H., Mogul, J., Masinter, L. M., Fielding, R. T., Gettys, J. et al. *Hypertext Transfer Protocol – HTTP/1.1* [online]. RFC Editor, červen 1999 [cit. 2023-05-12]. DOI: 10.17487/RFC2616. Dostupné z: https://www.rfc-editor.org/info/rfc2616.

[21] Pantos, R. a May, W. *HTTP Live Streaming* [online]. RFC Editor, srpen 2017 [cit. 2023-05-12]. DOI: 10.17487/RFC8216. Dostupné z: https://www.rfc-editor.org/info/rfc8216.

[22] Parmar, H., Thornburgh, M. a Adobe. *Adobe's Real Time Messaging Protocol* [online]. 2012 [cit. 2023-05-12]. Dostupné z: https://rtmp.veriskope.com/pdf/rtmp_specification_1.0.pdf.

[23] Postel, J. *User Datagram Protocol* [online]. RFC Editor, srpen 1980 [cit. 2023-05-12]. DOI: 10.17487/RFC0768. Dostupné z: https://www.rfc-editor.org/info/rfc768.

[24] Rao, A., Lanphier, R. a Schulzrinne, H. *Real Time Streaming Protocol (RTSP)* [online]. RFC Editor, 1. duben 1998 [cit. 2023-05-12]. DOI: 10.17487/RFC2326. Dostupné z: https://www.rfc-editor.org/info/rfc2326.

[25] Rescorla, E. *Diffie-Hellman Key Agreement Method* [online]. RFC Editor, červen 1999 [cit. 2023-05-12]. DOI: 10.17487/RFC2631. Dostupné z: https://www.rfc-editor.org/info/rfc2631.

[26] Schulzrinne, H., Casner, S. L., Frederick, R. a Jacobson, V. *RTP: A Transport Protocol for Real-Time Applications* [online]. RFC Editor, červenec 2003 [cit. 2023-05-12]. DOI: 10.17487/RFC3550. Dostupné z: https://www.rfc-editor.org/info/rfc3550.

[27] Schulzrinne, H., Rao, A., Lanphier, R., Westerlund, M. a Stiemerling, M. *Real-Time Streaming Protocol Version 2.0* [online]. RFC Editor, prosinec 2016 [cit. 2023-05-12]. DOI: 10.17487/RFC7826. Dostupné z: https://www.rfc-editor.org/info/rfc7826.

[28] Sedlmajer, K., Bambušek, D. a Beran, V. Effective Remote Drone Control Using Augmented Virtuality. In: *Proceedings of the 3rd International Conference on Computer-Human Interaction Research and Applications 2019*. SciTePress - Science and Technology Publications, 2019, s. 177–182. DOI: 10.5220/0008349401770182. ISBN 978-989-758-376-6. Dostupné z: https://www.fit.vut.cz/research/publication/12006.

[29] Thornburgh, M. C. *Adobe's Secure Real-Time Media Flow Protocol* [online]. RFC Editor, listopad 2013 [cit. 2023-05-12]. DOI: 10.17487/RFC7016. Dostupné z: https://www.rfc-editor.org/info/rfc7016.

[30] Wang, Y.-K., Sanchez, Y., Schierl, T., Wenger, S. a Hannuksela, M. M. *RTP Payload Format for High Efficiency Video Coding (HEVC)* [online]. RFC Editor, březen 2016 [cit. 2023-05-12]. DOI: 10.17487/RFC7798. Dostupné z: https://www.rfc-editor.org/info/rfc7798.

Příloha A

Obsah přiloženého DVD

- `/`
  - `data` — Adresář obsahující naměřená data pro experimenty.
    - `hw_*.txt` — Naměřené hodnoty vytížení HW pro různé scénáře.
    - `rtmp_*.txt` — Zaznamenané RTMP přenosy pro různé scénáře.
  - `dokumentace`
    - `dip_cprint.pdf` — Písemná zpráva DP pro barevný tisk.
    - `dip_is.pdf` — Písemná zpráva DP.
    - `src` — Zdrojové soubory písemné zprávy DP.
  - `program`
    - `compiled` — Spustitelné soubory programu.
    - `src` — Zdrojové soubory programu.
  - `skripty`
    - `capture_hw.sh` — Skript pro záznam HW vytížení.
    - `capture_rtmp.sh` — Skript pro záznam RTMP přenosu.
    - `convert_hw.sh` — Skript pro převedení dat HW vytížení do csv.
    - `convert_rtmp.sh` — Skript pro převedení pcap souboru do csv.
    - `calc_bandwidth.py` — Skript pro výpočet šířky pásma z RTMP csv souboru.
    - `calc_response.py` — Skript pro výpočet odezv z RTMP csv souboru.
    - `graph_bandwidth.py` — Skript pro vytvoření grafu využití šířky pásma.
    - `graph_hw.py` — Skript pro vytvoření grafu HW vytížení.
    - `graph_resp.py` — Skript pro vytvoření grafu odezv.
  - `video`
    - `2023-xherrg00-drone-server.mp4` — Prezentační video.