

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ŽIVÝ PŘENOS VIDEA PRO APLIKACI ZÁCHRANKA

LIVE VIDEO TRANSMISSION FOR ZACHRANKA APP

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **RADEK JESTŘABÍK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAN PLUSKAL, Ph.D.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

165207

Ústav: Ústav informačních systémů (UIFS)
Student: **Jestřabík Radek**
Program: Informační technologie
Název: **Živý přenos videa pro aplikaci Záchranka**
Kategorie: Mobilní aplikace
Akademický rok: 2024/25

Zadání:

1. Nastudujte možnosti přenosu obrazu a zvuku vhodné pro využití v mobilní aplikaci Záchranka. Zaměřte se na technologie a protokoly, které splňují požadavky stanovené konzultantem, včetně aspektů jako šířka pásma, latence, bezpečnost a kompatibilita s různými mobilními platformami.
2. Navrhněte vhodné řešení pro realizaci přenosu obrazu a zvuku podle požadavků konzultanta. V návrhu zohledněte integraci do stávající infrastruktury aplikace Záchranka, optimalizaci pro různé síťové podmínky a uživatelskou přívětivost. Návrh konzultujte s vedoucím práce a případně upravte dle jeho doporučení.
3. Implementujte navržené řešení přenosu obrazu a zvuku s důrazem na efektivitu, bezpečnost a spolehlivost. Zajistěte kompatibilitu s různými mobilními zařízeními a operačními systémy.
4. Otestujte implementované řešení důkladně. Zaměřte se na robustnost a spolehlivost přenosu v různých síťových podmínkách, včetně omezené šířky pásma a nestabilního připojení. Vyhodnoťte výkonnostní parametry jako je latence, kvalita obrazu a zvuku, a zhodnoťte celkovou uživatelskou zkušenost. Navrhněte případná vylepšení na základě získaných výsledků.

Literatura:

- Y. Xing *et al*. *A Low-Latency MPTCP Scheduler for Live Video Streaming in Mobile Networks | IEEE Journals & Magazine | IEEE Xplore* [online] [vid. 2024-10-13]. Dostupné z: doi: 10.1109/TWC.2021.3081498
- HILLESTAD, Odd Inge, Ola JETLUND a Andrew PERKIS, 2006. RTP-based broadcast streaming of high definition H.264/AVC video: An error robustness evaluation. *Journal of Zhejiang University-SCIENCE A* [online]. **7**(1), 19–26. ISSN 1862-1775. Dostupné z: doi:10.1631/jzus.2006.AS0019
- MEDEIROS, Iago, Lucas PACHECO, Denis ROSÁRIO, Cristiano BOTH, Jéferson NOBRE, Eduardo CERQUEIRA a Lisandro GRANVILLE, 2021. Quality of experience and quality of service-aware handover for video transmission in heterogeneous networks. *International Journal of Network Management* [online]. **31**(5), e2064. ISSN 1099-1190. Dostupné z: doi:10.1002/nem.2064

Při obhajobě semestrální části projektu je požadováno:

- body 1 a 2.

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

Vedoucí práce: **Pluskal Jan, Ing., Ph.D.**
Konzultant: Dybal Martin
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2024
Termín pro odevzdání: 14.5.2025
Datum schválení: 28.3.2025

Abstrakt

Cílem této práce bylo navrhnout, implementovat a otestovat systém pro živý přenos videa. Tato aplikace bude sloužit pro aplikaci Záchranka, ve které se bude využívat pro komunikaci mezi operátory tísňové linky 155 a uživateli. Systém byl implementován jako multimediální server, přes který je přenášen obraz i zvuk od uživatele do klientské aplikace operátorovi. Celý přenos je nahráván a ukládán na serveru. Pro tvorbu tohoto serveru byla využita knihovna mediasoup, která využívá technologii WebRTC. Testováním bylo zjištěno, že přenos obrazu i zvuku je kvalitní a plynulý. Latence je dostatečně nízká pro přenos v reálném čase. Díky těmto parametrům je zajištěna dobrá uživatelská zkušenost.

Abstract

The aim of this thesis was to design, implement and test a system for live video streaming. This system will serve the Záchranka application, in which it will be used for communication between 155 emergency line operators and users. The system has been implemented as a multimedia server through which video and audio is transmitted from the user to the operator's client application. The entire transmission is recorded and stored on the server. The mediasoup library, which uses WebRTC technology, was used to create this server. Testing showed that the video and audio transmission is of good quality and smooth. The latency is low enough for real-time transmission. These parameters ensure a good user experience.

Klíčová slova

Živý přenos videa, WebRTC, Javascript, Mediasoup, Spolehlivost, Bezpečnost

Keywords

Live video streaming, WebRTC, Javascript, Mediasoup, Reliability, Security

Citace

JESTŘABÍK, Radek. *Živý přenos videa pro aplikaci Záchranka.* Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Pluskal, Ph.D.

Živý přenos videa pro aplikaci Záchranka

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Pluskala, Ph.D. Další informace mi poskytl pan Martin Dybal. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Radek Jestřabík
13. května 2025

Poděkování

Rád bych poděkoval Ing. Janu Pluskalovi, Ph.D. za vedení mé bakalářské práce a odbornou pomoc. Dále děkuji panu Martinu Dybalovi za cenné rady a asistenci při konzultacích. Chtěl bych také poděkovat přítelkyni a rodině za podporu v průběhu celého studia.

Obsah

1 Úvod **3**

2 Současné technologie pro videokomunikaci a živý přenos médií **4**
2.1 Srovnání jednotlivých technologií 4
2.2 Multimediální a datový přenos ve WebRTC 6
2.3 Mediasoup 8
2.4 Signalizace 10
2.5 Proces navázání spojení 11
2.6 Zabezpečení komunikace 14

3 Návrh **15**
3.1 Požadavky 15
3.2 Architektura systému 17
3.3 Uživatelské rozhraní 19

4 Implementace **21**
4.1 Serverová část 21
4.2 Klientská aplikace 24
4.3 Pořizování snímků 25
4.4 Přenos médií mezi serverem a klienty 25
4.5 Příprava na integraci do aplikace Záchranka 26
4.6 Výzvy při implementaci a jejich řešení 27

5 Testování **28**
5.1 Kompatibilita zařízení a prohlížečů 28
5.2 Testování v omezených síťových podmínkách 29
5.3 Výkonnostní parametry přenosu a uživatelská zkušenost 29
5.4 Analýza přenosu 30
5.5 Zátěžové testování 31
5.6 Kontrola uzavírání portů 32
5.7 Omezení zjištěná při testování 32

6 Závěr **33**

Literatura **34**

A Posudek konzultanta **36**

Seznam obrázků

2.1 Využití různých typů snímků pro snížení množství přenášených dat. I-snímky jsou celé snímky, P-snímky označují změnu od minulého I-snímku a B-snímku kombinují předchozí P-snímek a následující I-snímek. 8
2.2 Architektura knihovny *mediasoup* při využití tří procesů `Worker`. 10
2.3 Ukázka komunikace přes TURN server. 12

3.1 Diagram toků informací v systému s připojeným jedním operátorem a uživatelem. 17
3.2 Průběh fungování systému při vytvoření nového hovoru s jedním operátorem a jedním uživatelem. 19
3.3 Vzhled aplikace pro operátora při režimu náhledu (části jiné než zobrazení videa v pravém horním rohu jsou již součástí aplikace Záchranka a nejsou náplní této práce). 20
3.4 Vzhled aplikace pro operátora při sledování záznamu v režimu celé obrazovky. 20

5.1 Grafy z nástroje *webrtc-internals*, na kterých je ukázán pokles přenášených paketů při přepnutí z živého vysílání na záznam. 30
5.2 Průběh využití procesoru a paměti během testování s třemi současně probíhajícími hovory. 31

A.1 Posudek konzultanta Martina Dybala. 36

Kapitola 1

Úvod

V dnešní době se živé přenosy videa a zvuku stávají nedílnou součástí každodenního života ve formě digitální komunikace. Ať se jedná o pracovní schůzky, distanční výuku, osobní rozhovory, ale také o kritické aplikace jako zdravotnictví. Přenos multimediálních dat zásadně mění způsob, jakým lidé komunikují. Díky těmto technologiím se odstraňují bariéry vzdálenosti a je možné téměř ihned komunikovat.

Motivací této práce je zlepšení kvality služeb aplikace Záchranka, která rozšiřuje možnosti při tísňové komunikaci mezi operátorem tísňové linky a uživatelem. V současné podobě aplikace poskytuje možnost živého přenosu videa, což operátorům umožňuje mnohem rychleji zhodnotit situaci a přesněji navigovat uživatele, co má dělat. Nicméně systém, pomocí kterého je přenos videa řešen, již není pro aplikaci dostačující a vznikly na něj nové požadavky. Jeden z důvodů, proč systém nevyhovuje, je, že v aktuální situaci není možné přetáčet video zpět v rámci živého přenosu. Tuto funkcionalitu by nově vzniklý systém měl umožňovat bez nutnosti ukončovat hovor. Zároveň jsou kladeny požadavky na podporu více operátorů v jednom hovoru, aktivaci mikrofonu uživatele a přepínání mezi kamerami.

Cílem této práce je navrhnout, vytvořit a otestovat nové řešení, které vyhovuje požadavkům a nahradí aktuální řešení. Jako první je provedena analýza současných technologií a jsou zvoleny ty technologie, které splňují požadavky. Poté je navržena architektura systému a způsob fungování, který obsahuje požadovanou funkcionalitu, jako je záznam přenosu, přetáčení videa a příprava na integraci do stávající aplikace.

Při implementaci se vycházelo z návrhu. Byl kladen důraz na efektivitu, bezpečnost přenosu dat a kompatibilitu s co nejvyšším počtem různých mobilních zařízení. Zároveň byly sledovány výkonnostní parametry, jako je latence, kvalita obrazu a zvuku a celková uživatelská zkušenost. Byly vytvořeny a otestovány jednotlivé části systému, včetně uživatelského rozhraní, které bylo navrženo tak, aby bylo co nejvíce intuitivní.

V této práci jsou nejprve analyzovány a popsány technologie, které umožňují přenos multimediálních dat v reálném čase, a je provedeno jejich srovnání. Následně je popsán návrh nového řešení, jeho implementace a příprava na integraci do aplikace Záchranka. Poté je ukázán průběh testování a jeho výsledky. Nakonec jsou zhodnoceny dosažené výsledky a navrženy možnosti dalšího rozvoje systému.

Kapitola 2

Současné technologie pro videokomunikaci a živý přenos médií

Tato kapitola se zaměřuje na technologie a protokoly, které umožňují přenos multimediálních dat v reálném čase. Důležitým aspektem je výběr správných technologií, které umožní splnit požadavky na tento systém. Proto je v této kapitole také popsáno, jaké jsou dostupné možnosti, a které jsem po průzkumu a srovnání usoudil, že budou ideální. Také jsou zde popsány způsoby řešení efektivity, spolehlivosti a bezpečnosti přenosu, což jsou nedílné součásti takových moderních aplikací.

Dále je vysvětleno, jak fungují důležité části zvolených protokolů a technologií. Jako například proces navázání spojení, který obsahuje výměnu informací mezi klientem a serverem, výběr nejlepšího možného způsobu spojení a inicializaci zabezpečené komunikace. Tento proces je důležitý, protože je pomocí něj dosažena vysoká kompatibilita mezi různými typy zařízení a způsoby připojení. Další zásadní částí, která je rozebrána, jsou problémy, které je nutné vyřešit při tvorbě systému pro živý přenos médií, a jak je možné je řešit.

2.1 Srovnání jednotlivých technologií

V této podkapitole je popsáno, jaká technologická řešení by teoreticky mohla být využita a která jsem se rozhodl zvolit. Na základě analýzy požadavků na tento systém jsou zde popsány výhody a nevýhody jednotlivých řešení. Mezi hlavní požadavky patří mimo jiné přenos multimediálních dat v reálném čase, bezpečnost, spolehlivost přenosu a kompatibilita mezi různými zařízeními. Hlavním cílem je vysvětlit a zdůvodnit, která technologie je nejvhodnější pro tento projekt. Porovnávány jsou protokoly a technologie jako SIP a RTP, WebRTC a řešení využívající multimediální servery.

2.1.1 SIP a RTP

Protokol SIP (Session Initiation Protocol, česky protokol pro inicializaci relace, nadále bude využívána pouze zkratka) slouží k navázání, správě a ukončení relací pro přenos multimediálních dat. Protokol RTP (Real-Time Transport Protocol, česky protokol pro přenos v reálném čase, nadále bude využívána pouze zkratka) je využíván pro samotný přenos multimediálních dat [13].

Nevýhodou těchto technologií je vyšší komplexnost při vývoji. Náročnost je primárně v tom, že je nutné vyřešit všechny problémy, které se mohou vyskytnout. Jeden z problémů je například průchod přes zařízení využívající konkrétní typy překladu síťových adres (NAT) nebo firewall, což je systém, který rozhoduje, zda příchozí a odchozí pakety mohou být poslány dále. Tento problém musí být pro účely této aplikace vyřešen z důvodu zajištění co nejvyšší kompatibility. Další věc, která není v rámci těchto protokolů řešena, je zabezpečení, pro to musí být využity další protokoly, které by musely být integrovány při vývoji [16].

2.1.2 WebRTC bez serveru

Zkratka WebRTC znamená webová komunikace v reálném čase, odvozeno z anglického Web Real-Time Communication (nadále bude využívána pouze zkratka). Je to open source technologie, která je navržena pro peer-to-peer komunikaci mezi zařízeními v reálném čase [5]. Peer-to-peer označuje takový typ komunikace, kdy jsou klienti propojeni přímo. WebRTC je API (aplikační programové rozhraní) a zároveň protokol. Protokol WebRTC definuje pravidla pro bezpečnou a obousměrnou výměnu dat, nejen multimediálních, ale libovolných typů (prostý text, binární objekty a další). WebRTC API, jenž je vytvořeno pro programovací jazyk *JavaScript*, poskytuje vývojářům prostředky pro využití tohoto protokolu přímo v aplikacích. Protokol WebRTC je dostupný i pro jiné jazyky než je *JavaScript* [3].

WebRTC používá mnoho dříve vyvinutých protokolů, které spojuje dohromady, a tím řeší velké množství problémů, které by se jinak musely při implementaci řešit samostatně. Příklad toho, co implicitně řeší, může být šifrování komunikace a navazování spojení. Agent WebRTC označuje spojení těchto protokolů [3]. Další velkou výhodou je, že WebRTC je integrováno do většiny moderních webových prohlížečů [5].

Nevýhodou peer-to-peer komunikace je její omezená škálovatelnost. Při větším množství klientů, kteří spolu komunikují, není udržitelná, protože každý klient musí navázat přímé spojení s každým dalším. Toto omezení ale nezpůsobuje problém pro tuto aplikaci, protože podle požadavků popsaných v sekci 3.1 by neměla nastávat situace, že by komunikovali více než 4 klienti v rámci jednoho hovoru.

Poměrně zásadní problém při peer-to-peer komunikaci nastává u vytváření a distribuce záznamu, který by se podle požadavků měl vytvářet a být dostupný již během živého přenosu. Jedním z možných přístupů je, že operátor bude nahrávat příchozí přenos od uživatele. Toto řešení má ale dvě velká omezení:

1. **Omezení prohlížečů**: Běžná aplikační rozhraní v prohlížečích neumožňují nahrávat takovým způsobem, aby byl záznam dostupný již při přenosu. Prohlížeče běžně umožňují vytvářet soubory pouze ve formátech MP4 a WebM[1], které není možné přehrávat před dokončením nahrávání.

2. **Závislost na přítomnosti operátora**: Pokud by žádný z operátorů nesledoval přenos, záznam by vůbec nevznikal. V případě, že by se operátor v průběhu hovoru odpojil, záznam by byl neúplný.

Nahrávání na straně uživatele také není ideální řešení. V této aplikaci se totiž nemůže předpokládat, že uživatel má dostatečně silné a stabilní připojení, aby mohl odesílat záznam operátorům nebo na externí úložiště. Z těchto důvodů nebylo zvoleno řešení pomocí WebRTC bez serveru.

[1] https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder/isTypeSupported_static

2.1.3 Řešení založená na multimediálním serveru

Alternativou k peer-to-peer komunikaci jsou řešení, která využívají multimediální servery. Jsou to technologie, které v sobě obsahují předpřipravené řešení pro multimediální komunikaci. Jejich výhodou je, že není potřeba navrhovat a implementovat vlastní systém od základu. Existují open source varianty, které je možné spustit a provozovat na vlastním serveru, nebo také varianty, které jsou proprietární a běží na serveru třetí strany. Příkladem mohou být *Janus*[2], *Kurento*[3] nebo *mediasoup*[4].

Varianty, které jsou provozovány na serveru třetí strany, způsobují závislost na poskytovateli a také mohou být placené. Další nevýhodou těchto řešení může být vyšší latence způsobená nutností přenosu všech dat přes centrální server [3].

Rozhodl jsem se pro využití open source knihovny *mediasoup*, pomocí které je možné vytvořit multimediální server, který je možné provozovat na vlastním serveru. Toto řešení bylo zvoleno z důvodu, že v porovnání s ostatními technologiemi představovalo nejmenší počet komplikací. Mezi výhody této knihovny patří flexibilita, kdy na straně serveru je možné řídit komunikaci a směrování mediálních toků. Tím je umožněna plná kontrola nad průběhem přenosu a také nad správou jednotlivých hovorů a klientů [2]. Centrální server také řeší problém s vytvářením a distribucí záznamu, který byl u WebRTC bez serveru. Uživatel odesílá mediální tok na server, který jej přeposílá připojeným operátorům. Na rozdíl od přímého přenosu mezi klienty může server vytvářet záznam, bez ohledu na to, jestli operátoři sledují přenos. Vytvářený záznam si může operátor po připojení jednoduše získat ze serveru. Detailnější popis *mediasoup* se nachází v kapitole 2.3.

2.2 Multimediální a datový přenos ve WebRTC

V této podkapitole je rozebráno, jaké technologie využívá WebRTC pro přenos multimediálních a libovolných dat v reálném čase. Jsou zde popsány protokoly RTP a pomocný protokol RTCP. Protokol RTP obsahuje časové označení paketů a jejich pořadí, což umožňuje plynulé přehrávání. Pomocný protokol RTCP se používá pro monitorování kvality a získávání statistik o přenosu. Pro přenos libovolných dat, což mohou být například textové zprávy nebo binární data, využívá WebRTC protokol SCTP [3].

Důležitou částí, která je nutná k zajištění dobré uživatelské zkušenosti při přenosu videa a audia, je efektivní přenos multimediálních dat. V této podkapitole je také popsáno, jak funguje komprese videa a audia a jaké jsou způsoby komprese.

2.2.1 Protokoly pro přenos a řízení médií

Tato podsekce je převzata z technické zprávy [7]. K přenosu multimediálních dat se ve WebRTC využívá protokol RTP. Tento protokol byl vytvořen z důvodu, že existující protokoly v té době (TCP a UDP) nebyly uzpůsobeny pro přenos multimediálních dat. Například jim chyběly informace o pořadí snímků nebo informace o čase. V protokolu RTP jsou nejen tyto informace, ale také navíc informace o typu dat, což popisuje, jaký kodek je použitý. Také se v protokolu přenáší identifikátor zdroje, což slouží například k rozlišení jednotlivých kamer nebo mikrofonů. Každý zdroj má unikátní identifikátor, proto je možné rozlišovat jednotlivé přenosy, i když se přenášejí současně. RTP slouží pouze pro přenos dat, neřeší

[2] https://janus.conf.meetecho.com/
[3] https://kurento.openvidu.io/
[4] https://mediasoup.org/

řazení příchozích paketů, pokud přijdou ve špatném pořadí, nebo nějaké chybí, musí to být vyřešeno na aplikační vrstvě.

Společně s protokolem RTP se také používá RTCP (RTP Control Protocol, česky řídicí protokol pro přenos v reálném čase, nadále bude využívána pouze zkratka). Tento protokol slouží jako doplněk pro protokol RTP. Nejdůležitější funkce tohoto protokolu jsou:

- **Monitorování kvality služeb a řízení zahlcení**: RTCP pakety obsahují informace pro monitorování kvality služeb, jako je zpoždění, ztrátovost paketů a změna zpoždění mezi jednotlivými pakety (jitter). Tyto údaje jsou předávány prostřednictvím zpráv odesílatele (Sender Reports, SR) a zpráv příjemce (Receiver Reports, RR).
- **Identifikace**: Poskytuje informace o zdroji médií (jeho identifikátor), podobně jako u RTP.
- **Odhad velikosti relace a škálování**: Objem řídicího provozu se zvětšuje téměř úměrně tak, že vždy zabírá kolem 5 % celkového objemu dat.

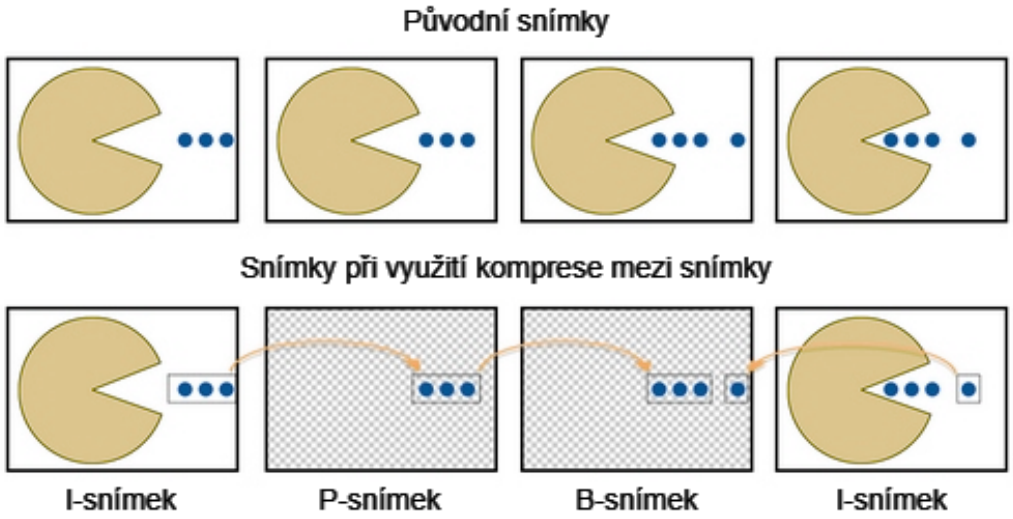
Pomocí RTCP je možné vypočítat dobu zpoždění cesty paketu tam a zpět (anglicky round-trip time delay). Výpočet probíhá tak, že se vezme časová značka z poslední zprávy odesílatele a zpoždění od poslední zprávy odesílatele, a obojí se odečte od časové značky, která byla získána při přijetí zprávy příjemce odesílatelem. Další užitečná informace o průběhu komunikace, kterou můžeme získat z RTCP zpráv, je počet ztracených paketů. Tato hodnota může být využita pro řízení zahlcení.

2.2.2 Efektivní přenos multimediálních dat

Tato podsekce je převzata z knihy [3]. Pro efektivní přenos multimediálních dat, zejména videa, je nutné provádět kompresi dat. Komprese se dělí na ztrátovou a bezeztrátovou. U ztrátové komprese se odstraněním méně podstatných informací dosahuje úspory dat. Při bezeztrátové kompresi zůstává kvalita zachována, ale může dojít ke zmenšení velikosti dat kvůli efektivnějšímu uložení. Další způsob dělení je podle toho, zda komprese probíhá pouze v rámci jednotlivých snímků videa, nebo zda se může kombinovat více snímků za sebou. Při kompresi jednotlivých snímků odděleně probíhá komprese jako u obrázků. Způsob kombinování informací z více snímků funguje tak, že předpokládá, že se ve většině videí nemění všechny pixely mezi jednotlivými snímky, a proto není nutné posílat ty pixely, které se nezměnily. Způsoby komprese v rámci snímku i mezi snímky se kombinují dohromady. Při kompresi mezi snímky se posílají tyto typy snímků:

- **I-snímek**: Celý, nezávislý snímek, který obsahuje všechny informace potřebné k dekódování a zobrazení. Jsou využívány jako referenční snímky (anglicky keyframes) pro další typy, které neposílají celé snímky. Například při připojení dalšího uživatele do videohovoru musí daný uživatel získat celý snímek, než může začít zobrazovat video.
- **P-snímek**: Tento typ snímku odkazuje na předchozí I-snímek, ze kterého využívá části snímku, které se nezměnily. Navíc také posílá ty části, které se změnily v porovnání s referenčním I-snímkem.
- **B-snímek**: Umožňuje využít data z předchozího P-snímku a následujícího I-snímku. Dekodér může vytvořit aktuální snímek pomocí těchto dvou snímků. Využitím tohoto typu snímku je možné získat ještě menší velikost snímku v porovnáním s P-snímkem.

Na obrázku 2.1 je zobrazeno, jakým způsobem se různé typy snímků kombinují dohromady při využití komprese mezi snímky. Obrázek převzat z [8].



Obrázek 2.1: Využití různých typů snímků pro snížení množství přenášených dat. I-snímky jsou celé snímky, P-snímky označují změnu od minulého I-snímku a B-snímku kombinují předchozí P-snímek a následující I-snímek.

2.2.3 Přenos libovolných dat

Ve WebRTC se pro přenos libovolných dat (například obrázků a textu) používají datové kanály (anglicky data channels). Jsou založeny na transportním protokolu SCTP (Stream Control Transmission Protocol). Tento protokol vznikl za účelem vytvoření kompromisu mezi spolehlivým přenosem dat u protokolu TCP a nespolehlivým přenosem u protokolu UDP. Protokol TCP zajišťuje doručování dat v pořadí odeslání, což může způsobovat zbytečné zpoždění pro aplikace, které tuto funkcionalitu nevyžadují [15].

Protokol DCEP (Data Channel Establishment Protocol) slouží pro vytváření datových kanálů ve WebRTC. Tento protokol definuje pouze dvě zprávy, `DATA_CHANNEL_OPEN` a `DATA_CHANNEL_ACK`. První zpráva je poslána agentem na definování parametrů datového kanálu. Druhý agent pro potvrzení odpovídá pouze zprávou `DATA_CHANNEL_ACK` [4].

Parametry datového kanálu umožňují nastavit, jestli bude zaručeno doručení všech paketů v pořadí, nebo kolikrát se má pokoušet znovu poslat nedoručený paket, a nebo, jestli mohou pakety přicházet v pořadí, nebo ne. Tato nastavení jsou vhodná pro různé účely, například pokud je potřeba zaručit doručení dat v reálném čase. Není vhodné několikrát data zkoušet posílat, protože by se zdržela a zhoršila by se uživatelská zkušenost. U jednotlivých datových kanálů je také možné si nastavit prioritu, tím se zaručí, že kanály s vyšší prioritou nebudou zpožděny těmi s nižší prioritou. Mezi dvěma agenty může být otevřeno až 65534 datových kanálů [4].

2.3 Mediasoup

Mediasoup je *Node.js* knihovna, pomocí které je možné vytvořit multimediální server, který využívá technologii WebRTC pro přenos zvuku, videa a dat v reálném čase. Vzhledem k tomu, že *mediasoup* využívá WebRTC, automaticky přebírá klíčové bezpečnostní a síťové mechanismy. To znamená, že využívá protokoly jako ICE pro navazování spojení a

SRTP pro šifrování komunikace mezi klienty a serverem [2]. Jedná se o tzv. SFU (Selective Forwarding Unit, nadále bude využívána pouze zkratka), což znamená, že dokáže přijímat více mediálních toků a rozhodovat, které toky budou přeposlány kterým uživatelům [17]. Tímto dokáže zajistit efektivní přenos médií mezi více účastníky při zachování nízké latence. Díky flexibilitě knihovny *mediasoup* je možné, aby probíhalo více nezávislých hovorů najednou. Díky nízkoúrovňovému aplikačnímu rozhraní umožňuje odesílání RTP paketů (bez připojování skrze WebRTC), což může být vhodné například při tvorbě záznamu [2].

2.3.1 Architektura knihovny mediasoup

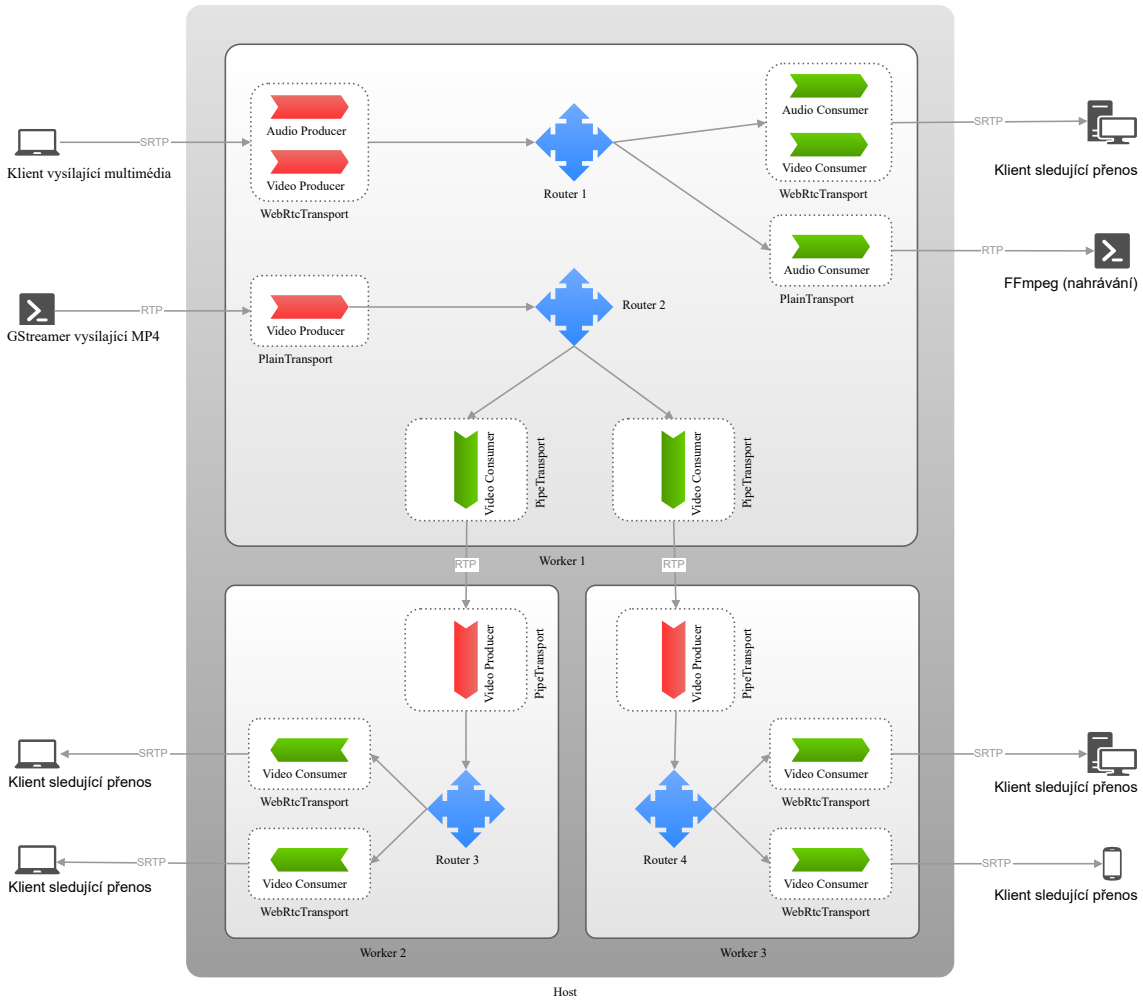
Mediasoup je navržen jako výkonný multimediální server, který běží nad jádrem napsaným v jazyce *C++*. *Mediasoup* poskytuje knihovny pro jazyk *JavaScript* a *Rust* [2]. Zvolil jsem knihovnu pro jazyk *JavaScript*, jelikož s tímto jazykem mám předchozí zkušenosti a zároveň jej zná i vývojový tým aplikace Záchranka. *Mediasoup* jsem zvolil pro tento systém z důvodu, že nabízí velmi kvalitní dokumentaci a příklady, které pokrývaly potřeby tohoto systému. Mezi ně patřilo například nahrávání přenosu, které je v *mediasoup* navrženo velmi flexibilně, a ukázky umožňovaly snadné přizpůsobení. Alternativní řešení, jako jsou *Janus* a *Kurento* také nabízejí mnoho možností, z analýzy ale vyplynulo, že neposkytují žádné funkce, které by byly přínosnější při tvorbě tohoto systému. Zároveň je jejich architektura komplexnější, což by vedlo k delší době nutné pro seznámení a implementaci.

Mediasoup se skládá z několika klíčových komponent, kde každá z nich plní specifickou roli v procesu přenosu a směrování médií mezi klienty (tento přehled je převzat z oficiální dokumentace knihovny *mediasoup* [2]):

- **`Worker`:** Je to *C++* proces, který běží na serveru, zajišťuje správu objektů typu `Router` a běží na jednom jádře procesoru. Z důvodu škálovatelnosti je možné vytvořit pro každé jádro procesoru jeden běžící `Worker`.
- **`Router`:** Zjednodušeně může být `Router` považován za reprezentaci jednoho hovoru (jedné místnosti) mezi více účastníky, kteří spolu komunikují. Pro každý hovor se tedy vytváří nový `Router`. `Router` může obsahovat několik objektů `Transport`.
- **`Transport`:** Slouží jako spojení mezi klientem a `Router` na multimediálním serveru. Je to abstrakce nad `RTCPeerConnection` z WebRTC a definuje, jaký způsobem budou data přenášena. Slouží k přenosu dat jak směrem od klienta (pomocí `Producer`), tak směrem ke klientovi (pomocí `Consumer`).
- **`Device`:** Používá se na straně klienta a představuje bod, který se připojuje k `Router` na serveru za účelem přijímat nebo odesílat multimediální data. `Transport` na klientské straně se vytváří z `Device`.
- **`Producer`:** Objekt, který reprezentuje zdroj videa nebo zvuku, který je přenášeny pomocí `Transport` do `Router` na server.
- **`Consumer`:** Objekt, který reprezentuje zdroj videa nebo zvuku, který server přeposílá klientovi. Je vytvořen na základě existujícího `Producer` jiného účastníka.

Na diagramu 2.2 lze vidět prvky architektury knihovny *mediasoup*. Je zde ukázán příklad, kdy se využívají tři procesy `Worker`. Každý z nich může obsahovat jeden nebo více `Router`. `PipeTransport` je speciální druh objektu `Transport` určený k přenosu multimediálních nebo libovolných dat mezi objekty `Router`. Lze také vidět, že *mediasoup* je velmi

flexibilní a umožňuje přijímat a odesílat data nejen do klientských aplikací, ale také do procesů, jako jsou *FFmpeg* a *GStreamer*. Na jednom objektu `Router` může být vytvořeno více různých objektů `Transport`, což umožňuje, aby více klientů sledovalo jeden přenos [2]. Obrázek převzat z [1].



Obrázek 2.2: Architektura knihovny *mediasoup* při využití tří procesů `Worker`.

2.4 Signalizace

Signalizace označuje proces, kdy mezi dvěma agenty (agentem může být klient nebo server) probíhá výměna informací, které jsou potřebné k vytvoření přímého spojení mezi nimi. Proces signalizace nezajišťuje spojení agentů, ale pouze získání informací, které může využít jiný protokol ke spojení. V kontextu WebRTC se pro předávání informací využívá protokol SDP (Session Description Protocol, neboli protokol pro popis relace) [5].

Důležité je zmínit, že signalizace neprobíhá skrze WebRTC, což znamená, že musí být implementován jiný kanál předávání informací. WebRTC nemá žádné omezení na použití jiného kanálu pro předání informací. Může být použit jakýkoliv kanál, který dokáže předávat

informace popisující relace (například REST API nebo WebSocket) [3]. V rámci tohoto systému je možné, aby signalizační server byl přímo součástí multimediálního serveru.

2.4.1 Protokol SDP – protokol pro popis relace

SDP je prostý textový protokol. Pro výměnu informací mezi agenty se posílají SDP zprávy. Každá SDP zpráva má několik záznamů. Každý záznam má klíč, který označuje typ záznamu, a hodnotu. SDP zpráva obsahuje například následující informace [3]:

- *IP adresy a porty, na kterých je možné se ke agentovi připojit*,
- *kodeky pro audio a video, které agent podporuje*,
- *počet a typ médií (audio a video)*,
- *informace použité pro zabezpečení komunikace (hash certifikátu pro DTLS).*

WebRTC používá SDP tím způsobem, že jeden WebRTC agent posílá nabídku (`SDP Offer`) druhému agentovi a ten mu odešle odpověď (`SDP Answer`), pokud je ochoten přijmout to, co nabízí. Tento způsob je vhodný z důvodu, že si navzájem agenti mohou oznámit, které kodeky a formáty podporují a které ne [3].

2.5 Proces navázání spojení

V této podkapitole je popsáno, jak probíhá shromažďování IP adres a portů, na které se mohou jiná zařízení připojit. Jsou zde popsány problémy, které mohou nastat při připojování, způsobené například systémem firewall, který může blokovat komunikaci. V procesu navazování spojení se využívá protokol ICE [16].

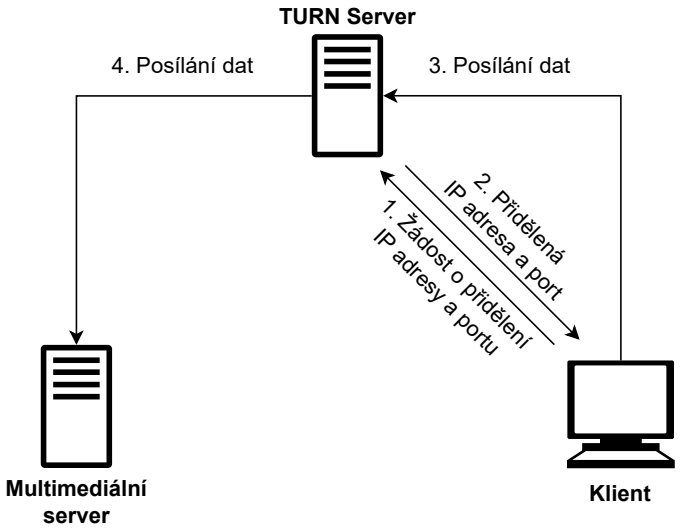
Pro získání všech dostupných adres, na které by se bylo teoreticky možné k agentovi připojit, se využívá protokol STUN (Session Traversal Utilities for NAT, nadále bude využívána pouze tato zkratka). Je to „klient-server“ protokol, který slouží jako nástroj pro průchod přes firewall nebo směrovače používající překlad síťových adres. V rámci STUN existují 2 typy zařízení, které implementují STUN protokol. Prvním typem je STUN klient, což je zařízení, které se chce dozvědět, jakou má přidělenou IP adresu a port. Druhým typem je STUN server, který slouží k získání požadovaných informací od klienta a jejich poslání zpět [10]. V rámci knihovny *mediasoup* pracuje multimediální server jako STUN server [9]. Získání klientovy adresy je jednoduchý proces, kdy server zjistí zdrojovou adresu z příchozího paketu, uloží ji podle stanovených pravidel protokolu do STUN zprávy a odešle ji klientovi (struktura STUN zprávy je popsána v RFC 8489 [10], na straně 8). Mezi klientem a STUN serverem se pro zjištění IP adresy a portu posílají zprávy `Binding Request`, které slouží k zjištění vlastní veřejné IP adresy a portu, a `Binding Response`, která je využívána k poslání zjištěné IP adresy a portu klienta zpět klientovi [10].

2.5.1 Překonání problémů při navazování spojení

Při navazování spojení mezi klientem a serverem může dojít k problémům kvůli některým nastavením firewallu, což je systém, který kontroluje příchozí a odchozí pakety a případně je blokuje. K tomu slouží protokol TURN (Traversal Using Relays around NAT, nadále bude využívána pouze tato zkratka), neboli přenos pomocí přeposílání okolo NAT. Tento protokol je rozšířením protokolu STUN [11].

Proces využití TURN serveru vypadá následovně: Klient si od serveru vyžádá dočasnou IP adresu a port, předá tyto informace klientovi nebo serveru, se kterým chce komunikovat, a všechny pakety, které budou přicházet na tuto IP adresu a port, budou přeposílány klientovi. Pro získání IP adresy a portu je vyžadována autentizace. Následně jsou posílána data pomocí zprávy `Send indication` nebo `ChannelData` [11]. Rozdíl mezi těmito zprávami je, že zpráva `Send indication` obsahuje vždy navíc IP adresu a port, kam chceme data posílat. `ChannelData` posílá data po předem vytvořeném kanálu, a proto není nutné posílat tyto informace v každé zprávě [3].

Diagram na obrázku 2.3 zobrazuje proces vyžádání a získání IP adresy a portu od TURN serveru, výměnu těchto hodnot přes signalizační server a posílání dat skrze TURN server. O IP adresu a port klient žádá pomocí zprávy `Allocate request`, dostává je přidělené ve zprávě `Relayed address`. Z diagramu je pro zjednodušení vynecháno zjištění veřejné IP adresy a portu od STUN serveru, zde je zobrazen případ, kdy není možné komunikovat přímým propojením [11].



Obrázek 2.3: Ukázka komunikace přes TURN server.

2.5.2 Protokol ICE

Protokol ICE (Interactive Connectivity Establishment, česky protokol pro interaktivní konektivitu) umožňuje spojení mezi dvěma agenty ICE. Tito agenti mohou být za směrovačem, který provádí překlad síťových adres, nebo za firewallem. ICE umožňuje vytvořit přímé spojení nebo použití prostředníka, pokud přímé spojení není možné [6].

Proces připojení a udržování spojení

Pro umožnění komunikace je nejdříve nutné vytvořit agenty ICE pro každou komunikující stranu (klient nebo server). Agent ICE je buď řídicí, nebo řízený. Řídicí agent je ten, který rozhoduje o výběru vhodného páru IP adres a portů (bude popsáno později v této podkapitole), většinou je to ten agent, který zahajuje spojení. Každý agent má hodnoty `ufrag` a `pass`. Hodnota `ufrag`, neboli username fragment (část uživatelského jména), je používána

k identifikaci relace ICE. Je zaslána pomocí SDP druhému agentovi. Hodnota `pass` slouží pro autentizaci zpráv, každý paket obsahuje `MESSAGE-INTEGRITY`, což je atribut, pomocí kterého se dá zkontrolovat autenticita zprávy, a je vytvořen pomocí `pass`. `pass` i `ufrag` jsou vygenerované pseudonáhodné hodnoty [3].

V dalším kroku se získávají všechny dostupné adresy, na které by se bylo teoreticky možné k agentovi připojit. Tyto adresy jsou nazývány kandidáti. Jsou definovány čtyři druhy kandidátů (tento přehled je převzat z [6], [3]):

- **Host kandidát** (Host candidate): Je to IP adresa a port zařízení na jeho lokálním síťovém rozhraní. Je využíván, v případě, že se oba účastníci nacházejí ve stejném lokální síti nebo mají přímé připojení.
- **Serverový reflexivní kandidát** (Server Reflexive candidate): Je to IP adresa a port zjištěný pomocí STUN serveru. Používá se, pokud je zařízení v síti, jejíž směrovač provádí jiný než symetrický překlad síťových adres.
- **Klientský reflexivní kandidát** (Peer Reflexive candidate): Tento kandidát je získán způsobem, že klient odešle paket druhému klientovi, se kterým dříve nekomunikoval, druhý klient tedy pošle tuto adresu zpět prvnímu pomocí STUN paketu.
- **Přeposílaný kandidát** (Relayed candidate): Je to kandidát poskytnutý TURN serverem, který funguje jako prostředník, pro přenos dat. Využíván například v případě použití symetrického mapování síťových adres u jednoho z klientů.

Dalším krokem je vytvořit páry kandidátů, na kterých se bude následně zkoušet, jestli je mezi nimi možná komunikace. Každý ze získaných kandidátů od druhého agenta je přiřazen ke každému z kandidátů prvního agenta. To znamená, pokud by první kandidát získal x kandidátů a druhý agent by získal y kandidátů, vzniklo by $x \cdot y$ dvojic [3].

Po získání všech možných párů kandidátů oba agenti posílají data na všechny páry. Všechny páry, které jsou schopny posílat data, jsou zvoleny jako validní kandidátní páry. Řídicí agent následně vybere jeden z párů a rozhodne, že tento pár bude zvolen pro komunikaci. Ale je nutné ještě před potvrzením znovu vyzkoušet, jestli je opravdu možné pomocí tohoto páru komunikovat. Pokud v pořádku proběhne komunikace od obou agentů, stane se tento pár zvoleným kandidátním párem (Selected Candidate Pair). Pokud v průběhu komunikace přestane tento pár fungovat, agenti ICE se restartují a tento proces začne znovu od získávání kandidátů [3].

ICE lite

Protokol ICE je možné implementovat buď jako plnou verzi, která podporuje řídicí i řízené agenty, nebo jako zjednodušenou verzi označovanou jako *lite*. Tato zjednodušená implementace podporuje pouze roli řízeného agenta, tedy neřídí proces navazování spojení. Účelem této implementace je usnadnění implementace protokolu ICE oběma zařízením, které se k sobě připojují. A to z důvodu, že pro využití benefitů, které tento protokol poskytuje, je nutné, aby jej obě komunikující strany podporovaly. Jednou z mála dalších věcí, kterou tato implementace podporuje, jsou odpovědi na zprávy STUN s dotazy na IP adresu odesílatele. Podmínkou pro využití této implementace je, že zařízení, které ji využívá, musí mít veřejnou IP adresu, která není za směrovačem využívajícím překlad síťových adres [6]. Tuto implementaci je tedy možné využít na multimediálních serverech, pokud splňují podmínky. To platí i v případě tohoto systému, kdy knihovna *mediasoup*, která je využita k tvorbě multimediálního serveru, implementuje zjednodušenou verzi protokolu ICE [2].

2.6 Zabezpečení komunikace

Každé WebRTC spojení je autentizováno a šifrováno. Dva nejdůležitější protokoly, které jsou použity pro zabezpečení komunikace, jsou DTLS a SRTP. Každý z těchto protokolů se využívá pro zabezpečení jiné části komunikace.

Protokol DTLS (odvozeno z anglického Datagram Transport Layer Security, nadále bude využívána pouze zkratka) slouží k zabezpečenému navázání spojení mezi komunikujícími účastníky. Tento protokol také pomáhá zabránit odposlouchávání, podvržení nebo úpravě komunikace. Využívá se při UDP komunikaci. Je podobný protokolu TLS (Transport Layer Security), který je však využíván pro TCP komunikaci [12].

Pro zabezpečení přenosu se používá protokol SRTP (odvozeno z anglického Secure Real-time Transport Protocol). Každý RTP paket je šifrován. Pro vyšší zabezpečení je k obsahu přidáno sekvenční číslo z RTP paketu a hodnota počítadla (rollover counter), která se zvýší vždy po přetečení sekvenčního čísla. To způsobí, že i když se posílají stejná data, výsledná zašifrovaná data budou rozdílná [3].

Kapitola 3

Návrh

Tato kapitola se zaměřuje na návrh systému a specifikaci požadavků. Nejprve jsou stanoveny funkční a nefunkční požadavky na navrhovaný systém. Následně je popsána architektura systému včetně podsekce týkající se návrhu pro řešení tvorby záznamu pomocí nástroje *FFmpeg*. Dále je popsán návrh uživatelského rozhraní klientských aplikací s důrazem na jednoduchost a zachování podobného vzhledu jako dosavadní aplikace.

Cílem bylo navrhnout systém, který bude možné v budoucnu integrovat do aplikace Záchranka a zároveň splňuje požadavky jako kompatibilitu s různými mobilními platformami, spolehlivost a uživatelskou přívětivost. Návrh zároveň sloužil jako podklad k implementaci, která vychází z jednotlivých komponent a jejich interakcí popsaných v této kapitole.

3.1 Požadavky

V této sekci jsou popsány funkční a nefunkční požadavky na tento systém. Funkční požadavky popisují jednotlivé funkcionality systému, neboli to, co má systém dělat. Nefunkční požadavky jsou vlastnosti, které musí systém splňovat. Příkladem mohou být požadavky na efektivitu, zabezpečení a přenositelnost.

3.1.1 Funkční požadavky

Funkční požadavky byly navrženy tak, aby odpovídaly potřebám operátorů tísňové linky. Důraz je kladen na zachování funkcionality stávající aplikace a její rozšíření. Funkční požadavky na tento systém jsou následující:

1. **Přenos videa od uživatele k operátorovi**: Při spojení mezi uživatelem a operátorem bude operátorovi dostupný živý přenos videa od uživatele.
2. **Více operátorů**: Systém musí umožnit, aby se k přenosu mohlo současně připojit více operátorů. Při konzultacích bylo zjištěno, že nenastává situace, kdy by přenos sledovali více než tři operátoři.
3. **Zapnutí mikrofonu uživatele**: Zvuková komunikace probíhá skrze tísňovou linku 155. Je také časté, že odkaz na zahájení videohovoru, který se posílá SMS zprávou uživateli, je zaslán jinému člověku než tomu, kdo volal na tísňovou linku. Proto bude mít operátor také možnost zapnout mikrofon od uživatele, se kterým probíhá videohovor.
4. **Možnost otočení kamery uživatele**: Pro operátora je také důležité, aby měl možnost otočit uživateli kameru, to znamená, že přenos videa bude probíhat z přední

namísto zadní kamery, či naopak. Tato funkcionalita může být užitečná v případě, že uživatel například ztratí vědomí, operátor poté může otočit kameru a může tak získat větší přehled o situaci.

5. **Pořízení fotografie aktuálního snímku**: Operátor bude mít možnost vyfotit aktuální snímek videa. Tyto fotografie budou uloženy pro možné budoucí přezkoumání důležitých okamžiků.
6. **Podpora mobilních platforem**: Systém musí fungovat na různých mobilních zařízeních a operačních systémech.
7. **Posun v čase v rámci živého přenosu**: Operátor bude moci posouvat video zpět a vpřed, bez nutnosti ukončení živého přenosu.
8. **Ukončení hovoru**: Hovor může být kdykoliv ukončen ze strany operátora.
9. **Přístup k uloženým záznamům a fotografiím**: V průběhu hovoru, i po ukončení incidentu v rámci aplikace Záchranka budou operátorům dostupné uložené záznamy a pořízené fotografie.
10. **Integrace do aplikace Záchranka**: Systém by měl být navržen takovým způsobem, aby byl připraven na budoucí integraci do aplikace Záchranka.
11. **Jednoduché uživatelské rozhraní pro uživatele**: Uživatel bude mít k dispozici jednoduché rozhraní, kde bude mít možnost otočit kameru a ukončit hovor. Toto rozhraní je důležité pro minimalizování času, který uživatel potřebuje pro zorientování se při používání aplikace.

3.1.2 Nefunkční požadavky

Nefunkční požadavky se zaměřují na kvalitu systému z hlediska spolehlivosti, bezpečnosti a použitelnosti. Zásadním aspektem u přenosu multimédií v reálném čase je latence, která má přímý dopad na uživatelskou zkušenost. Požadované nefunkční vlastnosti systému jsou:

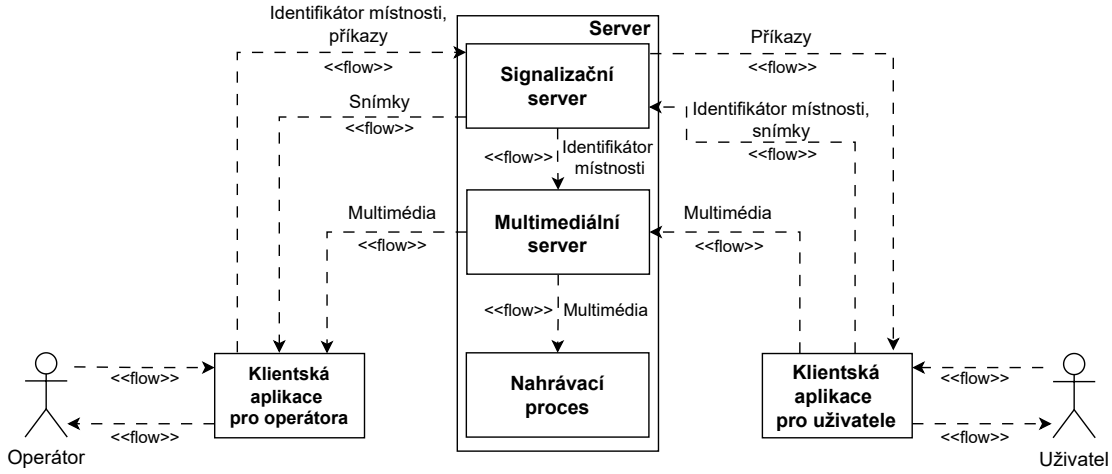
1. **Kvalita přenášeného videa a fotografií**: Pořízení snímku obrazovky by mělo být v co nejvyšší možné kvalitě.
2. **Latence přenosu**: Zpoždění přenosu by mělo být co nejnižší, aby byla zajištěna dobrá uživatelská zkušenost při komunikaci.
3. **Dostupnost videohovoru**: Systém musí spolehlivě fungovat i v případě pomalejších nebo nestabilních internetových připojení.
4. **Zajištění konektivity**: Mělo by být možné se spojit bez ohledu na typ připojení, ať už se jedná o mobilní síť nebo Wi-Fi.
5. **Bezpečnost přenosu**: Všechna data musí být šifrována, aby byla chráněna proti neoprávněnému přístupu.

3.2 Architektura systému

Systém bude využívat klient-server architekturu. To znamená, že operátor i uživatel budou komunikovat vzájemně prostřednictvím centrálního serveru. Všechna data budou nejprve zasílána na server, který je bude přeposílat dále. Pro tvorbu serveru bude využita *Node.js* knihovna *mediasoup* 2.3. Na serveru bude kromě samotného multimediálního serveru běžet také server pro signalizaci. Pro signalizaci bude využita knihovna *Socket.IO*, která nabízí jednoduché API pro komunikaci přes WebSocket. Mezi alternativy se může zařadit využití čistých WebSocketů, například s pomocí knihovny *ws*. *Socket.IO* však nabízí vyšší úroveň abstrakce, což usnadňuje práci s komunikací. Primárním důvodem pro zvolení *Socket.IO* ale byla podpora místností, kterou by jinak bylo nutné navrhnout a implementovat samostatně.

Klientská část bude realizována jako webová aplikace. Aplikace bude vytvořena pomocí technologií *HTML*, *CSS* a *JavaScript* bez přidaných aplikačních rámců pro tvorbu uživatelských rozhraní z důvodu zjednodušení integrace do aplikace Záchranka.

Diagram na obrázku 3.1 ukazuje přenos dat a informací mezi jednotlivými komponentami systému. Z klientských aplikací uživatele i operátora se budou předávat identifikátory místností, do kterých se chtějí připojit. K předávání budou sloužit signalizační zprávy. Signalizační server zařadí klienta do signalizační místnosti a předá identifikátor multimediálnímu serveru. Tam bude klient zařazen do dané místnosti, která reprezentuje hovor. Po připojení klienta k multimediálnímu serveru se začnou multimediální data odesílat do nahrávacího procesu, který bude ukládat. Příkazy odesílané z klientské aplikace budou signalizační zprávy, které budou označovat příkazy k otočení uživatelovy kamery nebo pořízení snímku či aktivaci mikrofonu. Pořízené snímky se budou od uživatele odesílat přes signalizační server, kde se budou ukládat, k operátorovi.



Obrázek 3.1: Diagram toků informací v systému s připojeným jedním operátorem a uživatelem.

Průběh fungování tohoto systému by měl vypadat následovně (jsou zde zmíněny hlavně části týkající se tohoto systému, ne celé funkcionality aplikace Záchranka):

1. Operátor otevře aplikaci a automaticky proběhne připojení k signalizačnímu serveru.

2. Pokud má operátor v URL parametr s identifikátorem existující místnosti (hovoru), server jej přidá do místnosti na signalizačním serveru. Způsob řešení pomocí URL parametrů je nyní zvolen pouze jako ukázka[1].

3. Operátor spustí inicializaci hovoru stisknutím tlačítka „Spustit hovor“. V případě, že se jedná o nový hovor (v URL není parametr s identifikátorem místnosti), server vytvoří nový hovor s identifikátorem, který vrátí do klientské aplikace operátora. Poté se operátor začne připojovat k serveru pomocí WebRTC s využitím klientské knihovny *mediasoup-client*.

4. Po integraci do aplikace Záchranka se bude odesílat SMS zpráva, která obsahuje odkaz na připojení se do aplikace, ze serveru aplikace Záchranka. V této aplikaci je tento krok vynechán a předání odkazu musí proběhnout jiným způsobem.

5. Uživatel otevře aplikaci, v průběhu toho operátor čeká na spojení.

6. Uživatel povolí přístup k mikrofonu a kameře na svém zařízení.

7. Uživatel se připojí zároveň k signalizačnímu serveru i k multimediálnímu serveru. Pro připojení k multimediálnímu serveru používá uživatelská aplikace klientskou knihovnu *mediasoup-client*, která pracuje s WebRTC. Podle identifikátoru místnosti z URL parametru je přiřazen do dané místnosti na signalizačním serveru.

8. Po připojení uživatele k serveru se začnou přenášet multimediální data od uživatele k serveru a následně od serveru k operátorovi.

9. Operátor vidí živý přenos a může přepínat mezi sledováním živého přenosu a dynamicky vytvářeného záznamu. Zároveň také může zachycovat snímky z živého přenosu, aktivovat mikrofon na zařízení uživatele a otáčet jeho kameru.

10. Operátor ukončí hovor.

11. Uživatel je přesměrován na jinou stránku mimo tuto aplikaci.

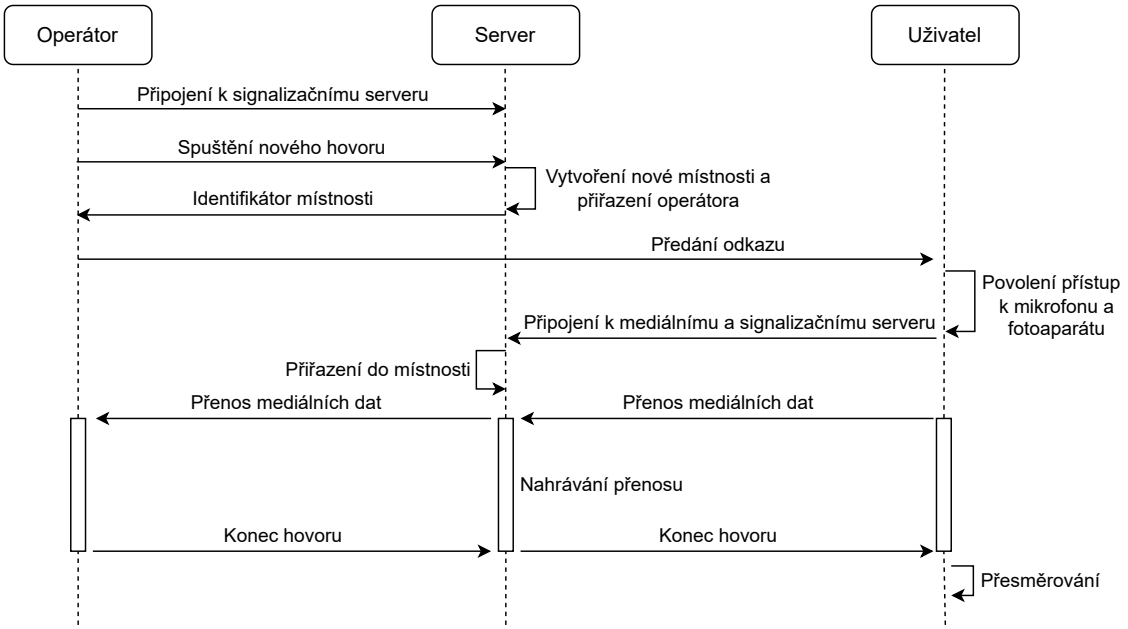
12. Operátorovi zmizí živý přenos a má k dispozici záznam a zachycené snímky.

Na diagramu 3.2 lze vidět výše popsaný průběh fungování (pro zjednodušení jsou vynechána potvrzení o provedení akce od serveru):

3.2.1 Tvorba záznamu

Pro záznam živého přenosu videa a zvuku bude využit nástroj *FFmpeg*, který dokáže přijímat multimediální data ve formě RTP toku. *Mediasoup* umožňuje přeposílat multimediální data nejen jiným klientům (pomocí WebRTC), ale také jako RTP pakety. Na serveru se vytvoří SDP zpráva, ve které budou mimo jiné informace o kodecích, typu přenášených dat a IP adrese, na kterou bude server odesílat RTP pakety. IP adresa bude `127.0.0.1` protože proces *FFmpeg* bude spuštěn na stejném zařízení jako server. Tato SDP zpráva se předá při vytvoření procesu *FFmpeg*. Z RTP toku, jenž bude přicházet z multimediálního serveru do procesu *FFmpeg*, se bude vytvářet výstup, který je vhodný pro následnou distribuci pomocí protokolu HLS (HTTP Live Streaming). Protokol HLS byl zvolen proto, že

[1] Po integraci do aplikace Záchranka se identifikátor bude předávat jinými způsoby. To by nebylo možné ukázat bez vytvoření další aplikace, která by předávala tento parametr.



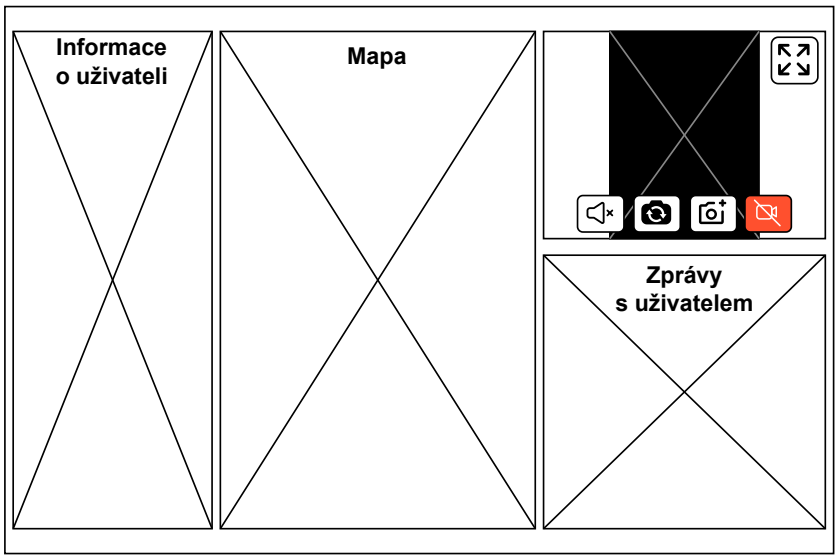
Obrázek 3.2: Průběh fungování systému při vytvoření nového hovoru s jedním operátorem a jedním uživatelem.

je pomocí něj možné přehrávat záznam ještě před tím, než je tvorba záznamu ukončena (v průběhu hovoru). Výstupem budou soubory ve formátu *.ts* (Transport Stream), což jsou několikasekundové segmenty, které budou obsahovat video i zvuk. Zároveň se na začátku také vytvoří soubor ve formátu *.m3u8*, do kterého se dynamicky přidávají segmenty po jejich vytvoření. Velkou výhodou tohoto řešení je, že je možné vytvářet záznam i v případě, že přenos nesleduje žádný operátor. To je zajištěno tím, že se od uživatele posílají data na server bez ohledu na to, jestli přenos nějaký operátor sleduje.

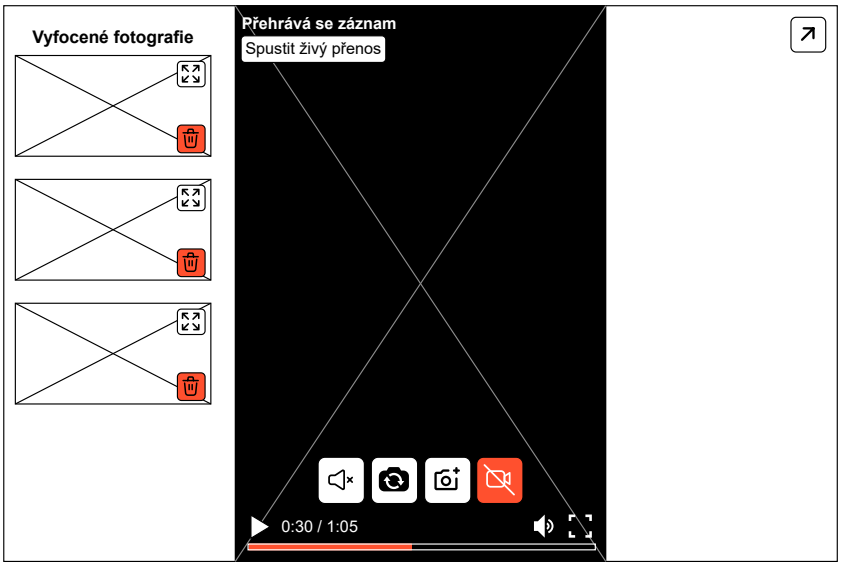
3.3 Uživatelské rozhraní

Vzhled byl inspirován existujícím vzhledem aplikace. Operátor může využívat dva druhy zobrazení, režim náhledu a režim celé obrazovky. Režim náhledu je základní zobrazení, které je integrováno na obrazovce incidentu, zobrazeno na obrázku 3.3. Pomocí tlačítka v pravém horním rohu se dostane operátor do režimu celé obrazovky a zobrazí se modální okno, které překryje tuto obrazovku, zobrazeno na obrázku 3.4. Z tohoto režimu se dá odejít pomocí tlačítka v pravém horním rohu bez ukončení videohovoru.

V porovnání s původním existujícím vzhledem byl režim celé obrazovky rozšířen o sloupec s vyfocenými fotografiemi a tlačítkem pro vyfocení fotografie. Navíc bylo také přidáno do levého horního rohu tlačítko na přepínání mezi živým přenosem a záznamem. Nad ním se nachází indikátor zobrazující, zda operátor právě sleduje živý přenos nebo záznam. Při sledování záznamu se zobrazují prvky pro ovládání videa, jako je posuvník pro přetáčení videa vzad i vpřed, tlačítko na pozastavení a ovládání hlasitosti. Pro přehrávání bude využitý výchozí přehrávač prohlížeče, což znamená, že jeho vzhled se v každém prohlížeči může mírně lišit.



Obrázek 3.3: Vzhled aplikace pro operátora při režimu náhledu (části jiné než zobrazení videa v pravém horním rohu jsou již součástí aplikace Záchranka a nejsou náplní této práce).



Obrázek 3.4: Vzhled aplikace pro operátora při sledování záznamu v režimu celé obrazovky.

Kapitola 4

Implementace

Tato kapitola se zabývá implementací systému pro živý přenos videa, jehož cílem je umožnit efektivní a spolehlivou komunikaci mezi operátorem tísňové linky a uživatelem. Vývoj systému vycházel z návrhu představeného v kapitole 3. Celý systém se skládá ze serverové části, do které patří multimediální server a signalizační server, a z klientské aplikace.

4.1 Serverová část

Serverová část je vytvořena pomocí jazyka *TypeScript*, který je překládán do jazyka *JavaScript*. Jazyk *TypeScript* byl zvolen z důvodu podpory statické typové kontroly, která umožňuje odhalení chyb již při vývoji. Server obsahuje dvě hlavní části, signalizační server a multimediální server. Signalizační server zajišťuje výměnu zpráv mezi klienty, multimediální server se stará o směrování mediálních toků a nahrávání. Pro tvorbu multimediálního serveru byla využita knihovna *mediasoup*, která je určena pro použití v běhovém prostředí *Node.js* a umožňuje manipulaci s WebRTC a tvorbu SFU. Multimediální i signalizační server jsou součástí jedné *Node.js* aplikace.

4.1.1 Signalizační server

Pro vytvoření signalizačního serveru byla využita knihovna *Socket.IO*. Tato knihovna umožňuje obousměrnou komunikaci mezi klienty a serverem prostřednictvím WebSocketů. Umožňuje navázat trvalé spojení, díky kterému mohou být zprávy odesílány rychle a efektivně. Signalizační server slouží k posílání kontrolních a synchronizačních zpráv. Pomocí zpráv například probíhá připojování do místností a ovládání uživatelovy kamery a mikrofonu. Přenos vyfocených snímků také probíhá skrze signalizační zprávy (detailněji popsáno v sekci 4.3). Dále je také možné posílat zprávy pouze specifickým klientům, což je využito například při synchronizaci, kdy se zprávy odesílají pouze operátorům v dané místnosti.

Socket.IO využívá koncept místností, do kterých je možné přidávat jednotlivé klienty a hromadně jim zasílat zprávy. Místnosti jsou v tomto systému využity tak, že shromažďují všechny operátory a jednoho uživatele. Jsou vytvářeny na základě identifikátoru hovoru. Tyto místnosti se ale liší od místností spravovaných multimediálním serverem, popsaných v podsekci 4.1.2. Místnosti v *Socket.IO* jsou specifické pro tuto knihovnu a slouží pouze k signalizaci.

4.1.2 Správa místností a připojených klientů

Každá místnost v systému představuje jeden probíhající hovor, do kterého se připojují operátoři a jeden uživatel. Pro správu místností se v systému pracuje s asociativním polem, kde je klíčem identifikátor dané místnosti a hodnotou je typ `Room` 4.3. Nová místnost je vytvořena po stisknutí tlačítka „Spustit hovor“ operátorem. Každá místnost má vlastní `Router` (popsáno v podsekci 2.3.1), který je přiřazen do atributu `router` dané místnosti. Operátor je po vytvoření do místnosti automaticky přidán.

Při vytvoření místnosti je také nastaven stav místnosti (typu `RoomState` 4.1), který obsahuje informace používané pro synchronizaci mezi operátory. Po připojení k serveru je operátorovi poslána signalizační zpráva, která obsahuje právě stav dané místnosti. Při každé změně jednoho z těchto atributů se posílá signalizační zpráva ostatním operátorům v místnosti pro synchronizaci. Stav místnosti obsahuje tyto informace:

- **Stav hovoru (atribut `callState`):** Po vytvoření místnosti nastaven na hodnotu „not-started“, která označuje, že místnost byla vytvořena, ale uživatel se do ní ještě nepřipojil.
- **Stav přenosu zvuku (atribut `audioEnabled`):** Výchozí hodnota atributu je „false“, protože při připojení uživatele se zvuk neposílá, dokud není spuštěn od operátora.
- **Dostupnost záznamu (atribut `recordingReady`):** Indikace, zda je první segment videa k dispozici. Tento atribut je nutný, protože několik sekund trvá než se vytvoří první segment videa a společně s ním soubor „playlist.m3u8“.
- **Pořízené snímky (atribut `snapshots`):** Tento atribut je využíván pro ukládání pořízených snímků. Tento atribut má dvě různá využití. Po připojení nového operátora se zde posílají všechny snímky, které byly pořízeny před jeho připojením. Druhé využití je posílání operátorům, kteří jsou připojeni v místnosti, ale zatím nejsou připojeni k hovoru (stav, kdy má operátor zobrazeno tlačítko „Připojit se k hovoru“). Těmto operátorům se posílají ale pouze nově pořízené snímky z důvodu efektivity, protože není potřeba posílat duplicitní snímky, které dostal při připojení. Všechny snímky jsou ukládány v jiné struktuře na serveru mimo `RoomState`, ze které se při prvním připojení operátora načtou.
- **Indikace operačního systému uživatele (atribut `isProducerIOS`):** Označuje, jestli uživatel používá operační systém iOS. Využití je popsáno v sekci 5.1.

```
export type RoomState = {
  callState:
    | 'not-started'
    | 'producer-connecting'
    | 'producer-connected'
    | 'ended',
  audioEnabled: boolean,
  recordingReady: boolean,
  snapshots?: Record<string, Uint8Array>,
  isProducerIOS: boolean,
}
```

Výpis 4.1: Typ popisující stav místnosti.

Ve zdrojovém kódu se namísto názvů „user“ a „operator“ používají názvy „producer“ a „consumer“, což odpovídá konvencím používaným v knihovně *mediasoup*.

Typ `PeerInfo` 4.2 slouží k uchování informací o připojeném klientovi. Klientem může být buď operátor, nebo uživatel. Atribut `isProducer` slouží k rozlišení, o jakého klienta se jedná. Každý klient má `Transport`, který zastupuje jeho připojení k serveru. Na straně uživatele jsou z transportu vytvořeny dva objekty typu `Producer`, jeden pro přenos audia a druhý pro přenos videa. Operátor využívá dva objekty typu `Consumer`, které slouží pro příjem těchto médií. U operátora je navíc atribut `isWatching`. Ten rozlišuje, zda je operátor připojen k multimediálnímu serveru, nebo pouze k signalizačnímu serveru. V případě, že je připojen pouze k signalizačnímu serveru, dostává synchronizační zprávy, podobné tomu, jako by se nově připojil. Tím se zajistí, že po připojení k multimediálnímu serveru bude mít okamžitě aktuální stav hovoru, včetně všech snímků.

```
export type PeerInfo = {
  isProducer: boolean,
  producerData?: {
    producerTransport?: Transport,
    videoProducer?: Producer,
    audioProducer?: Producer,
  },
  consumerData?: {
    consumerTransport?: Transport,
    videoConsumer?: Consumer,
    audioConsumer?: Consumer,
    isWatching: boolean,
  },
};
```

Výpis 4.2: Typ popisující připojeného klienta (operátora i uživatele).

4.1.3 Tvorba záznamu

Pro každou místnost se spouští proces nahrávání probíhajícího hovoru. Nahrávání je prováděno pomocí nástroje *FFmpeg*, který přijímá RTP toky videa a audia přes objekty typu `PlainTransport` z knihovny *mediasoup*. Každá místnost má také atributy pro nahrávání, například `PlainTransport` jsou v typu `Room` 4.3 ukládány v atributu `plainVideoConsumerTransport` a `plainAudioConsumerTransport`. V každé místnosti běží pouze jeden nahrávací proces, který je uložen v atributu `recordingProcess`. Atributy `remoteRtpVideoPort` a `remoteRtpAudioPort` označují porty, na kterých bude proces očekávat příchozí RTP pakety s videem a audiem.

Tento proces je spuštěn po připojení uživatele k serveru. O nahrávání se stará třída `FFmpeg`. Ta nejprve připraví SDP zprávu, která popisuje přenášená média. Následně spustí proces *FFmpeg*, kterému hned po spuštění tuto zprávu předá pomocí roury. Při vytváření procesu se předávají argumenty, které nastavují chování procesu a zpracování toků. Argument `-c:v libx264` nastavuje formát kódování videa na H.264. Tento formát kódování byl zvolen z důvodu širší podpory v běžně používaných prohlížečích[1] ve srovnání s novějším formátem H.265[2]. Argument `-vf "transpose=1, scale=trunc(iw/2)*2:trunc(ih/2)*2"`

[1] https://caniuse.com/mpeg4
[2] https://caniuse.com/hevc

rotuje video o 90° a upravuje rozlišení tak, aby počet pixelů byl sudý, protože `libx264` nepodporuje liché rozlišení. Rotace videa o 90° probíhá pouze, pokud video přichází ze zařízení s operačním systémem iOS (detailněji popsáno v sekci 5.1). Výstupem jsou segmenty ve formátu *.ts* a argumenty `-hls_time`, `-hls_list_size` a `-hls_flags` určují délku segmentů a způsob jejich generování.

```
export type Room = {
  router: Router,
  peers: Record<string, PeerInfo>,

  plainVideoConsumerTransport?: PlainTransport,
  plainAudioConsumerTransport?: PlainTransport,
  plainRtpVideoConsumer?: Consumer,
  plainRtpAudioConsumer?: Consumer,

  recordingProcess?: FFmpeg,
  remoteRtpVideoPort?: number,
  remoteRtpAudioPort?: number,

  state: RoomState,
};
```

Výpis 4.3: Typ popisující místnost (hovor).

Alternativou pro tvorbu záznamu byl nástroj `GStreamer`[3]. Tento nástroj byl využit v počátečních fázích implementace, protože také nabízí velmi kvalitní funkce pro práci s multimédii. Přestože `GStreamer` umožňuje detailní konfiguraci a manipulaci s jednotlivými toky, během vývoje se nepodařilo dosáhnout záznamu, který by spolehlivě kombinoval jak video, tak audio. Proto bylo nutné zvolit nástroj `FFmpeg`, pomocí kterého toho bylo dosaženo.

4.2 Klientská aplikace

Klientská aplikace také využívá *TypeScript*, který je překládán do *JavaScriptu*, a knihovnu *mediasoup-client*. Pro vytvoření jednoduchého vzhledu byly použity technologie *HTML* a *CSS*. Na komunikaci se signalizačním serverem je využita knihovna *Socket.IO-client*. V rámci vývoje se používá prostředí *Node.js*, ale díky nástroji pro balíčkování (anglicky bundler) je možné aplikaci se všemi knihovnami sloučit do jednoho balíčku, který je možné spouštět bez *Node.js*.

Aplikace je z pohledu logiky rozdělena do tří hlavních částí. Soubor *common.ts* obsahuje sdílenou logiku pro uživatele a operátora, zajišťuje připojování k serveru a vytváření objektů typu `Transport` a celkovou správu hovoru ze strany klienta. Tato struktura byla zvolena, protože velká část kódu týkajícího se komunikace s multimediálním i signalizačním serverem je podobná, tím pádem není nutné kód duplikovat na straně operátora i uživatele. Soubor *operator.ts* zajišťuje práci s uživatelským rozhraním na straně operátora, obsahuje logiku pro zobrazování přijatých snímků a inicializaci knihovny *hls.js* pro přehrávání záznamu pomocí protokolu HLS. V souboru *index.ts* je obsažena logika pro získání přístupu k mikrofonu a kameře. Zajišťuje také vytváření a odesílání snímků na vyžádání operátora (detailněji popsáno v sekci 4.3).

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

4.3 Pořizování snímků

Jedním z požadavků zmíněných v návrhu v podsekci 3.1.1 bylo pořizování fotografií aktuálních snímků. Pro přenos těchto snímků od uživatele k serveru a následně dalším uživatelům byl využit signalizační server. Alternativním řešením by bylo posílat snímky přes datové kanály WebRTC. Tato varianta nebyla zvolena z důvodu, že propojení se signalizačním serverem již bylo implementováno a nemělo žádné nevýhody, kvůli kterým by se nedalo použít. Naopak výhoda využití signalizačního serveru je v tom, že je možné operátorům posílat obrázky předtím, než se připojí k probíhajícímu hovoru na multimediálním serveru. Tento přístup je v aplikaci využíván a umožňuje odeslání většího množství dat ještě před zahájením přenosu médií k operátorovi, a tím se snižuje zatížení sítě v průběhu hovoru. Obě metody, tedy posílání přes signalizační server i skrze datové kanály, zaručují doručení zpráv ve správném pořadí. Toto je důležitá vlastnost z důvodu, že obě metody mají limity na velikost zpráv. Signalizační server má limit 1 MB, a proto se kvalitnější snímky musí rozdělovat na více zpráv. Příchod ve správném pořadí umožní serveru skládat zprávy v pořadí příchodu a ušetří čas zpracování, protože není nutné kontrolovat pořadí.

Implementace pořizování snímků probíhá přímo na straně klienta pomocí *HTML* elementů `video` a `canvas`. Při obdržení zprávy s žádostí o pořízení snímku je aktuální snímek vykreslen na `canvas` element. Poté je výsledný snímek převeden na objekt typu `Blob`. Tento objekt je preferován před použitím kódování base64 pro převod obrázku, jelikož je dle dokumentace MDN více efektivní[4]. Po vytvoření objektu typu `Blob` započne proces odesílání.

Odesílání probíhá tím způsobem, že se posílají signalizační zprávy tří různých typů:

- **`start-of-snapshot`:** Úvodní zpráva, ve které nejsou posílána data. Tato zpráva pouze oznamuje, že snímek byl pořízen a bude postupně přicházet. Je zde také obsaženo identifikační číslo daného snímku, které je generováno jako časové razítko. Identifikační číslo slouží k rozpoznání více souběžně posílaných snímků.
- **`snapshot-data`:** Tato zpráva obsahuje data části, nebo celého snímku a identifikační číslo daného snímku. Data jsou přenášena jako objekt typu `Uint8Array`.
- **`end-of-snapshot`:** Ukončující zpráva, která označuje, že přenos byl dokončen. V tento moment je snímek uložen na serveru jako obrázek ve formátu JPEG a zobrazen u operátora. Poté je možné smazat data o obrázku z datových struktur na serveru, které dočasně uchovávají části snímku dokud nepřijdou všechny a obrázek je uložen.

4.4 Přenos médií mezi serverem a klienty

V této sekci je popsáno, jakým způsobem se připojují klienti k serveru a jak probíhá přenos dat pomocí knihovny *mediasoup*. Proces vychází z doporučené architektury této knihovny. Pro zjednodušení se zde předpokládá, že proces `Worker` běží a `Router` (významy typů knihovny *mediasoup* jsou popsány v podsekci 2.3.1) pro danou místnost je již vytvořen.

Pro spojení se serverem přes WebRTC potřebuje klient znát, jaké kodeky server podporuje pro příjem audia a videa. Proto klient nejdříve zašle signalizační zprávu serveru, kde si o tyto informace zažádá. Server z vytvořeného `Router` získá tyto informace a pošle je zpět klientovi. Po obdržení těchto informací si může klient vytvořit `Device`.

[4] https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/toDataURL

Odesílání dat ze strany uživatele

Jakmile má klient vytvořený objekt typu `Device` a získaný přístup ke kameře a mikrofonu, může zahájit proces zahájení vysílání multimediálních dat. Nejprve odešle signalizační zprávu, ve které žádá o vytvoření `Transport` na serveru. Server tento `Transport` vytvoří a odešle klientovi zpět zprávu s parametry daného `Transport`. Tato zpráva obsahuje parametry pro DTLS 2.6 a ICE 2.5.2, kandidáty ICE a identifikátor `Transport`. Klient pomocí těchto informací vytvoří vlastní `Transport`, který bude určen pro odesílání dat. Pomocí událostí, které jsou vyvolány na lokálním `Transport` klient získá vlastní DTLS parametry, které pošle na server. Poté také posílá informace o médiích, které má v plánu odesílat. Server jako odpověď vrací identifikátor `Producer`. Tento identifikátor se předá lokálnímu `Transport`, tím se potvrdí, že se data přenesla správně. Po tomto kroku začínají proudit data od klienta k serveru.

Přijímání dat ze strany operátora

Ze začátku je při přijímání postup stejný jako při odesílání médií. Klient požádá o vytvoření serverového `Transport`, přijme jeho parametry a vytvoří lokální `Transport`. Tento `Transport` bude určen pro přijímání dat. Následně informuje server o tom, že chce začít přijímat data a jaký formát dat umí přijímat. Server zkontroluje, jestli je formát kompatibilní s přijímanými daty, a pokud ano, vytvoří se serverový `Consumer`. Při vytváření se musí předat identifikátor objektu typu `Producer`, ten má server interně uložený a využije jej. Poté pošle klientovi parametry potřebné pro vytvoření lokálního `Consumer`, parametry obsahují formát dat a identifikátor serverového `Producer`. S těmito informacemi je možné vytvořit lokální `Consumer`. Z vytvořeného `Consumer` vezme atribut `track` a předá jej při vytvoření objektu `MediaStream`. Poté může klient začít sledovat přenos tak, že přiřadí `MediaStream` k video elementu v *HTML*.

4.5 Příprava na integraci do aplikace Záchranka

Jedním z cílů této práce bylo také vytvořit takový systém, který umožní jednoduchou integraci do aplikace Záchranka. V této sekci je popsáno, jaké kroky byly provedeny k tomu, aby byla integrace co nejsnadnější.

Serverová část systému je přizpůsobená tomu, aby mohla běžet jako kontejner v prostředí *Docker*. Díky tomu je možné nasadit systém bez nutnosti ruční konfigurace prostředí. Zároveň je tím zajištěno konzistentní chování napříč různými systémy.

Při spuštění nového hovoru operátorem je na serveru vygenerován unikátní identifikátor hovoru. Tento identifikátor je následně vrácen zpět do klientské aplikace pomocí signalizační zprávy. Posílání identifikátoru zpět je jednoduchý způsob, jak po integraci do aplikace Záchranka z klientské aplikace ukládat identifikátor do perzistentního úložiště pro připojení dalších operátorů. Pokud by bylo potřeba, je možné upravit způsob zahajování hovoru, aby identifikátor generovala aplikace Záchranka a předávala jej při spouštění hovoru. Systém je připraven i na tento způsob a vyžadoval by pouze malé změny v logice na straně klienta i serveru.

Klientskou aplikaci je po vytvoření balíčku možné spustit samostatně, bez závislostí na jiných knihovnách. A to z důvodu, že nástroj pro balíčkování vytvoří soubory typu *HTML*, *CSS* a *JavaScript*, kde *JavaScriptové* soubory obsahují všechny zdrojové kódy knihoven

nutné k funkčnosti aplikace. Tento přístup zjednodušuje integraci do aplikace Záchranka, protože výsledný kód neobsahuje žádné závislosti.

4.6 Výzvy při implementaci a jejich řešení

Během implementace systému se vyskytlo několik technických omezení, která bylo potřeba vyřešit, aby bylo zajištěno spolehlivé fungování na různých zařízeních a prohlížečích. Většina komplikací se týkala nahrávání, kde bylo nutné zajistit, aby se nahrávalo vždy a nestalo se, že se nahrávací proces nečekaně ukončí nebo nebude správně fungovat.

Jedním z problémů bylo, že záznam nebylo možné přehrát a namísto rozdělených segmentů se vytvářel pouze jeden velký segment. Problém byl, že se chybu nedařilo replikovat a vyskytovala se náhodně, proto bylo náročné zjistit, co ji způsobuje a jestli se ji podařilo opravit. Bylo odhaleno, že chyba byla pravděpodobně způsobena tím, že přenos neobsahoval informace o snímkové frekvenci nebo je proces *FFmpeg* neuměl interpretovat. V těchto případech proces *FFmpeg* chybně nastavil hodnotu 90 000 snímků za sekundu. Tato hodnota byla převzata ze vzorkovací frekvence přenosu. Tento problém byl vyřešen předáním argumentu `-r 30`, který zajistí, že všechny záznamy budou mít 30 snímků za sekundu.

Další výzva souvisela také s nahráváním přenosu. Nahrávání pomocí procesu *FFmpeg* funguje tak, že přijímá RTP pakety, které neobsahují informaci o rotaci snímků videa. Rotace videa je řešena v rámci WebRTC, což při přenosu pouze RTP paketů není možné zachovat. Z toho důvodu nebylo možné vytvořit nahrávku, která by správně zachycovala rotaci videa podle toho, jak uživatel držel telefon. Výsledný záznam je tady vždy orientovaný na výšku a pokud by uživatel v průběhu otočil telefon, rotace nebude změněna. Po konzultaci s panem Dybalem ale bylo zjištěno, že to nepředstavuje problém, protože hovory s operátory probíhají na výšku. A v případě jiného otočení budou operátoři navádět uživatele, aby tak telefon drželi.

Jiná komplikace nastala při připojování druhého operátora do hovoru. Při návrhu bylo zamýšleno, že operátor se okamžitě připojí do hovoru, který již probíhá. Bylo ale zjištěno, že většina prohlížečů nepovoluje přehrávat audio na stránce, se kterou uživatel neinteragoval (například nestiskl tlačítko). Což představovalo problém, že když se operátor připojil do hovoru, kde byl aktivovaný audio přenos, tak viděl, že přenos je aktivní, ale zvuk se nepřehrával. Jako řešení bylo tedy zvoleno, že se před připojením do hovoru zobrazí text s informací, že hovor probíhá a tlačítko „Připojit se k hovoru“. Po stisknutí je operátor připojen k multimediálnímu serveru a zvuk může být přehráván, protože proběhla interakce se stránkou.

Kapitola 5

Testování

Testování celého systému probíhalo lokálně v průběhu celého vývoje. Klíčové bylo otestovat funkčnost, kompatibilitu a stabilitu. Vzhledem k tomu, že výsledná aplikace má rozhraní velmi podobné existující aplikaci, nebylo nutné provádět uživatelské testování zaměřené na použitelnost. Užitečným nástrojem při lokálním testování byla možnost zobrazení podrobných výpisů do konzole od obou klíčových knihoven *mediasoup* a *FFmpeg*. Pomocí těchto výpisů bylo možné snáze řešit problémy a ověřovat, že se systém nechová v rozporu s předpoklady knihoven.

5.1 Kompatibilita zařízení a prohlížečů

Pro tento systém je velmi důležité, aby byl kompatibilní s co nejvyšším počtem zařízení a prohlížečů. Proto se velká část testování zaměřila na tuto část. Aplikace pro uživatele je webová, ale je výhradně používána z mobilního telefonu, proto testování probíhalo primárně na mobilních telefonech (iOS 18, Android 12, Android 14 a Android 15). Naopak aplikace pro operátory je používána na počítači, z tohoto důvodu probíhalo testování hlavně na počítačích (Windows 11 a macOS Sequoia). Aplikace byly testovány na těchto prohlížečích:

- **Safari (iOS, macOS)**
- **Google Chrome (Windows, macOS, iOS, Android)**
- **Mozilla Firefox (Windows, macOS, iOS, Android)**
- **Microsoft Edge (Windows, macOS, iOS, Android)**
- **Opera (Windows, macOS, iOS, Android)**
- **Brave (Windows, macOS, iOS, Android)**
- **DuckDuckGo (Windows, macOS, iOS, Android)**
- **Samsung Internet (Android)**

Z testování vyšlo, že fungují tyto všechny kombinace kromě prohlížeče DuckDuckGo na macOS, ve kterém knihovna *mediasoup-client* nefunguje. Podle statistik ze stránky statcounter [14] pokrývají testované prohlížeče v České republice velkou většinu (alespoň 98 %) používaných prohlížečů. Při testování bylo také odhaleno, že prohlížeč Mozilla Firefox ignoruje metadata o rotaci videa. To způsobovalo, že přenos od uživatelů ze zařízení s operačním

systémem iOS byl otočený o 90°. Tento problém byl vyřešen v klientské aplikaci u operátora přidáním CSS třídy, pokud je uživatel na zařízení s operačním systémem iOS. Spolu s touto třídou bylo využito pravidlo CSS, které platí pouze pro Firefox a správně otočí video. Stejné chování bylo odhaleno při tvorbě záznamu, že video přicházející ze zařízení s operačním systémem iOS je otočeno o 90°. Řešení proběhlo tak, že se v takovém případě před spuštěním procesu *FFmpeg* přidá argument na rotaci videa.

Dále bylo také zjištěno, že Firefox může při lokálním testování způsobovat chybu, kdy se není možné připojit k serveru. Tato chyba se ale vyskytuje pouze při lokálním testování, protože Firefox ve výchozím nastavení ignoruje adresu lokálního počítače (*127.0.0.1*) při získávání kandidátů ICE. Problém je možné vyřešit v nastavení (*about:config*) přiřazením hodnoty `true` do `media.peerconnection.ice.loopback`.

5.2 Testování v omezených síťových podmínkách

Dalším důležitým požadavkem na tuto aplikaci bylo, aby fungovala i ve zhoršených síťových podmínkách. Pro simulaci těchto podmínek byla využita aplikace *Clumsy*[1]. Tato aplikace umožňuje mimo jiné simulovat omezenou šířku pásma, ztrátu a duplikaci paketů, latenci. Aby bylo možné otestovat zhoršené síťové podmínky i ze strany operátora, i ze strany uživatele, byly obě aplikace testovány z počítače. Rozlišení přenášeného videa bylo Full HD. Byly testovány následující scénáře:

- **Ztráta paketů**: Do zhruba 3 % ztracených paketů probíhal přenos bez znatelnějších problémů, při vyšší hodnotě se začínal přenos zasekávat.
- **Šířka pásma**: Při nižší šířce pásma než cca 3 Mbps docházelo k výraznějšímu poklesu kvality a zasekávání.
- **Zvýšená latence**: Při zpoždění do zhruba 500 ms byl přenos poměrně kvalitní, pokud bylo zpoždění větší přenos už nebyl příliš kvalitní.
- **Duplikace a změna pořadí paketů**: Tato omezení neměly zásadní dopad na kvalitu přenosu.

Z výsledků testování vyplývá, že aplikace je schopna fungovat i ve zhoršených síťových podmínkách při zachování poměrně dobré kvality přenosu.

5.3 Výkonnostní parametry přenosu a uživatelská zkušenost

Parametry přenosu byly testovány na reálném serveru v běžných síťových podmínkách. Testování bylo zaměřeno primárně na latenci, kvalitu obrazu a zvuku, a s tím spojenou uživatelskou zkušenost.

Latence mezi uživatelem a operátorem byla odhadnuta přibližně na 250 ms. Tato hodnota byla určena pomocí jednoduché metody, při které byl spuštěn časovač na samostatném zařízení, které uživatel nahrával. Vedle telefonu byla spuštěna klientská aplikace operátora. Pomocí dalšího zařízení byly nahrávány zároveň obě klientské aplikace a ve zpomaleném záběru bylo možné vidět, jaký je v daný moment zobrazený čas na obou zařízeních. Na základě rozdílu v zobrazených časech v různých momentech byla odhadnuta celková latence.

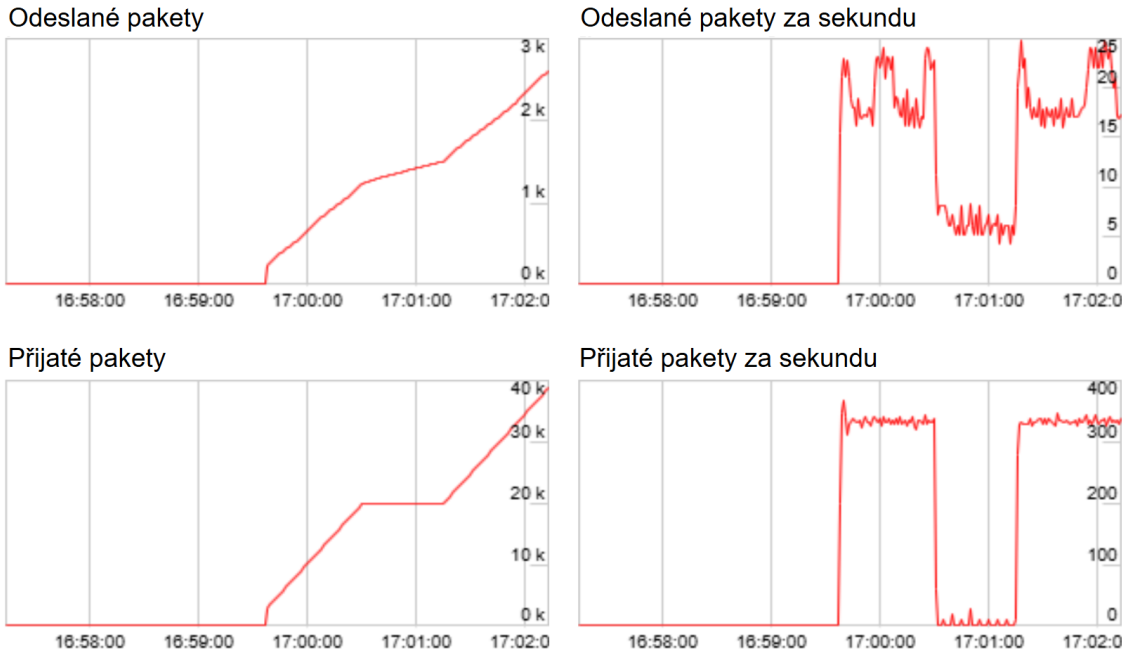
[1] https://jagt.github.io/clumsy/

Kvalita obrazu a zvuku byla hodnocena jako velmi dobrá. Nedocházelo k zasekávání ani k velkým rozdílům v latenci. Testována byla také rychlost odezvy na ovládací tlačítka, která po stisknutí reagují téměř okamžitě a nezpůsobují zasekávání přenosu, ani neovlivňují kvalitu. Proto by se dalo říct, že uživatelská zkušenost z používání aplikace je kvalitní.

Vzhledem k tomu, že samotný přenos je realizován pomocí technologie WebRTC, která je uzpůsobena pro nízkou latenci a vysokou kvalitu obrazu i zvuku, není v tomto směru mnoho možností pro vylepšení na úrovni tohoto systému.

5.4 Analýza přenosu

Pro analýzu datových toků během přenosu byl využit nástroj *webrtc-internals*, který je součástí prohlížeče Microsoft Edge. Pomocí tohoto nástroje je možné zkoumat velké množství informací o přenosu a spojení. Při testování byl využit pro ověření pozastavení přenosu při sledování záznamu operátorem. To znamená, že pokud operátor v průběhu přenosu spustí přehrávání záznamu, není potřeba přenášet multimediální data s živým přenosem k operátorovi. Správnost chování lze vidět na grafu na obrázku 5.1. Je zde ukázán počet odeslaných a přijatých paketů během hovoru od serveru k operátorovi. Hovor byl spuštěn cca 16:59:40 a je patrné, že zhruba 17:00:30 byl přenos pozastaven a počet přijatých paketů klesá téměř na nulu. U odesílaných paketů jde také vidět pokles, ale ne tak výrazný. Část paketů je odesílána i při pozastaveném přenosu, může se jednat o RTCP pakety nebo udržování spojení ICE aktivní. V momentě, kdy se operátor vrátí ke sledování živého přenosu, je na grafu (asi v čase 17:01:15) vidět opětovný nárůst přenášených paketů.



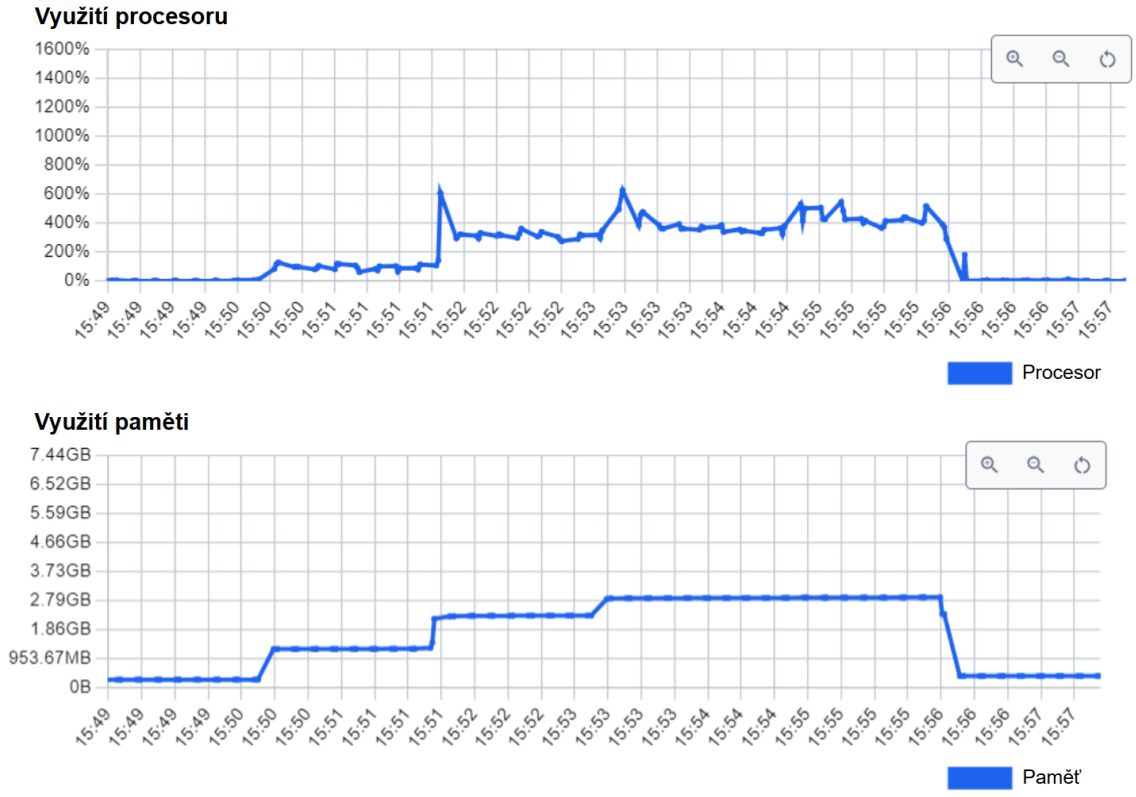
Obrázek 5.1: Grafy z nástroje *webrtc-internals*, na kterých je ukázán pokles přenášených paketů při přepnutí z živého vysílání na záznam.

5.5 Zátěžové testování

Zátěžové testování proběhlo spuštěním tří hovorů současně. Každý z přenosů byl samostatně nahráván, jako při běžném provozu. Navíc se z každého hovoru pořizovalo několik snímků. Všechny tři přenosy probíhaly bez problémů, nenastávalo zasekávání ani zpoždění. Systém byl testován v prostředí Docker. Při spuštěném kontejneru s běžící aplikací bez probíhajících hovorů bylo využití prostředků procesoru zhruba 1–5 % a využití paměti asi 250 MB. Test byl prováděn na notebooku s procesorem Intel i7 13. generace (10 jader, 16 vláken).

Na grafech na obrázku 5.2 je vidět průběh využití procesoru a paměti během testování (obrázek je pořízen z aplikace Docker Desktop). Na grafech lze vidět nárůst po 15:50, který odpovídá spuštění prvního hovoru. Následně přibližně v 15:52 a 15:53 přibývají další dva hovory, což je patrné z dalšího navýšení zátěže procesoru a paměti. Zvýšené využití kolem 15:55 bylo způsobeno povolením přenosu audia u všech tří přenosů. Maximální vytížení se pohybovalo kolem 600–700 %, což odpovídá 6–7 využitým jádrům procesoru. S každým nově zahájeným hovorem došlo ke zvýšení spotřeby paměti přibližně o 1 GB. Celkové využití dosáhlo zhruba 3 GB. Po ukončení všech hovorů došlo opět k poklesu využití prostředků na původní úroveň.

Při konzultacích s panem Dybalem bylo zjištěno, že v aktuální situaci se většinou nestává, že by probíhalo více hovorů najednou. Proto se dá předpokládat, že systém je dobře připraven i na budoucí provoz, kdy by se mohl zvýšit počet současně probíhajících hovorů.



Obrázek 5.2: Průběh využití procesoru a paměti během testování s třemi současně probíhajícími hovory.

5.6 Kontrola uzavírání portů

Pro ověření správného uzavírání portů, které se využívají pro přenos médií nebo pro nahrávání, byl využíván příkaz *netstat -peanut*. Testování probíhalo způsobem, že se spustilo několik hovorů po sobě a kontrolovalo se, které porty přibyly jako využité. Po ukončení každého hovoru se ověřilo, že se dané porty uzavřely a nevzniká problém s hromaděním neuzavřených portů. Bylo zjištěno, že porty jsou správně uzavírány a nehromadí se. To znamená, že systém může pracovat dlouhodobě a nenastane vyčerpání dostupných portů.

5.7 Omezení zjištěná při testování

Při testování bylo zjištěno, že pokud uživatel opustí prohlížeč na déle než 10–15 sekund, nahrávání se přeruší. Živý přenos stále zůstává funkční i po delší době. Na některých zařízeních (Vivo Y01) se tento problém neprojevuje vůbec, zatímco na jiných (Samsung A52s, iPhone 12 mini) ano.

Dalším zjištěným omezením bylo, že pokud u uživatele probíhal telefonní hovor a následně se připojil do tohoto systému, nebylo vidět video. Pokud operátor ale otočil kameru, video se správně spustilo. Zvuk nebyl přenášen do doby, než byl ukončen telefonní hovor, což je očekávané chování, protože mobilní operační systémy nepřidělí mikrofon více aplikacím najednou.

Kapitola 6

Závěr

Cílem této práce bylo zanalyzovat současné technologie pro videokomunikaci a živý přenos médií a navrhnout systém, který splňuje stanovené požadavky. Tento systém bude součástí již existující aplikace Záchranka, kde nahradí současné řešení.

V tomto textu jsou srovnány aktuální technologie, které je možné využít pro vývoj systému pro videokomunikaci. Ze srovnání vyšlo, že nejlepší způsob bude vytvořit multimediální server, který s pomocí knihovny *mediasoup* využívá technologii WebRTC. Tyto technologie a detaily jejich fungování, které jsou nutné pro vývoj této aplikace, jsou zde popsány. V kapitole návrhu je popsána architektura a fungování tohoto systému. Při analýze a návrhu bylo zohledněno, že aplikace bude integrována do aplikace Záchranka.

Na základě návrhu systému proběhla implementace, která umožňuje spojení uživatele aplikace Záchranka s operátorem tísňové linky 155. Přenos audia a videa probíhá přes multimediální server, který jej nahrává a ukládá s možností zpětného přehrávání v průběhu živého přenosu nebo po jeho skončení. Operátor má také možnost vyfotit aktuální snímek videa, který je pořízen u uživatele z důvodu vyšší kvality. Tyto snímky jsou také ukládány na serveru pro možné pozdější přezkoumání. Jednou z obtížnějších částí byla implementace nahrávání, kdy muselo být zajištěno, aby byl přenos vždy nahráván a aby byl dostupný již před skončením živého přenosu. Z toho důvodu musely být zvoleny vhodné způsoby zpracování a protokoly, aby bylo možné tohoto chování dosáhnout.

Testování probíhalo formou nejen lokálního testování, ale také jako konzultace s panem Dybalem, který poskytoval rady ohledně fungování systému. Testování s budoucími uživateli této aplikace nebylo nutné, vzhledem k tomu, že klientské aplikace vypadají velmi podobně s již existujícími. Dá se tedy předpokládat, že aplikace uživatelům vyhovuje. Hlavní požadavky, které měl systém splnit, byly kompatibilita s různými zařízeními a prohlížeči, efektivita, bezpečnost a stabilita. Z testování vyplynulo, že se systém chová podle očekávání a splňuje požadované vlastnosti.

Systém by v budoucnu mohl být rozšířen o ukládání záznamů a snímků na externí úložiště. Další možné rozšíření by mohlo spočívat v oboustranném přenosu zvuku. Systém by také mohl filtrovat hluk na pozadí a zobrazovat přepis řeči v reálném čase, případně i její překlad do jiného jazyka.

Literatura

[1] Castillo, I. B. *Mediasoup architecture*. 2015. Dostupné z: https://mediasoup.org/images/mediasoup-v3-architecture-01.svg.

[2] Castillo, I. B. *Mediasoup Documentation*. 2015. Dostupné z: https://mediasoup.org/documentation/.

[3] DuBois, S.; Mogren, C.; Zhukov, A. et al. *WebRTC for the Curious* online. https://webrtcforthecurious.com/, červenec 2020. Dostupné z: https://webrtcforthecurious.com/docs/webrtc-for-the-curious.pdf. [cit. 2024-12-07].

[4] Jesup, R.; Loreto, S. a Tüxen, M. *WebRTC Data Channel Establishment Protocol* Internet Requests for Comments. RFC 8832. Internet Engineering Task Force, leden 2021.

[5] Kashif, M. *WebRTC Implementation: A Comprehensive Guide*. 2024. Dostupné z: https://medium.com/@kashif00527/webrtc-implementation-a-comprehensive-guide-26132cee835e.

[6] Keränen, A.; Holmberg, C. a Rosenberg, J. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal* Internet Requests for Comments. RFC 8445. Internet Engineering Task Force, červenec 2018.

[7] Koistinen, T. Protocol overview: RTP and RTCP. *ResearchGate*, Květen 1999, s. 8.

[8] Kong, C. H. *EOS Movie Compression Options: All-I and IPB*. 2014. Dostupné z: https://www.canon.com.hk/cpx/public/images/tech/article/video/EOS_Movie_Compression_Options_All_I_and_IPB/IPB_Compression_EN_s.jpg.

[9] Nozhenko, S. *STUN and TURN servers*. 2022. Dostupné z: https://mediasoup.discourse.group/t/stun-turn-servers/4108/3.

[10] Petit Huguenin, M.; Salgueiro, G.; Rosenberg, J. et al. *Session Traversal Utilities for NAT (STUN)* Internet Requests for Comments. RFC 8489. Internet Engineering Task Force, únor 2020.

[11] Reddy, T.; Johnston, A.; Matthews, P. et al. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)* Internet Requests for Comments. RFC 8656. Internet Engineering Task Force, únor 2020.

[12] Rescorla, E. a Modadugu, N. *Datagram Transport Layer Security Version 1.2* Internet Requests for Comments. RFC 6347. Internet Engineering Task Force, leden 2012.

[13] Schulzrinne, H.; Casner, S. L.; Frederick, R. et al. *RTP: A Transport Protocol for Real-Time Applications* Internet Requests for Comments. RFC 3550. Internet Engineering Task Force, červenec 2003.

[14] Statcounter. *Mobile Browser Market Share Czech Republic*. 2025. Dostupné z: https://gs.statcounter.com/browser-market-share/all/czech-republic/#monthly-202403-202503-bar. Licencováno pod Creative Commons Attribution-Share Alike 3.0 Unported License.

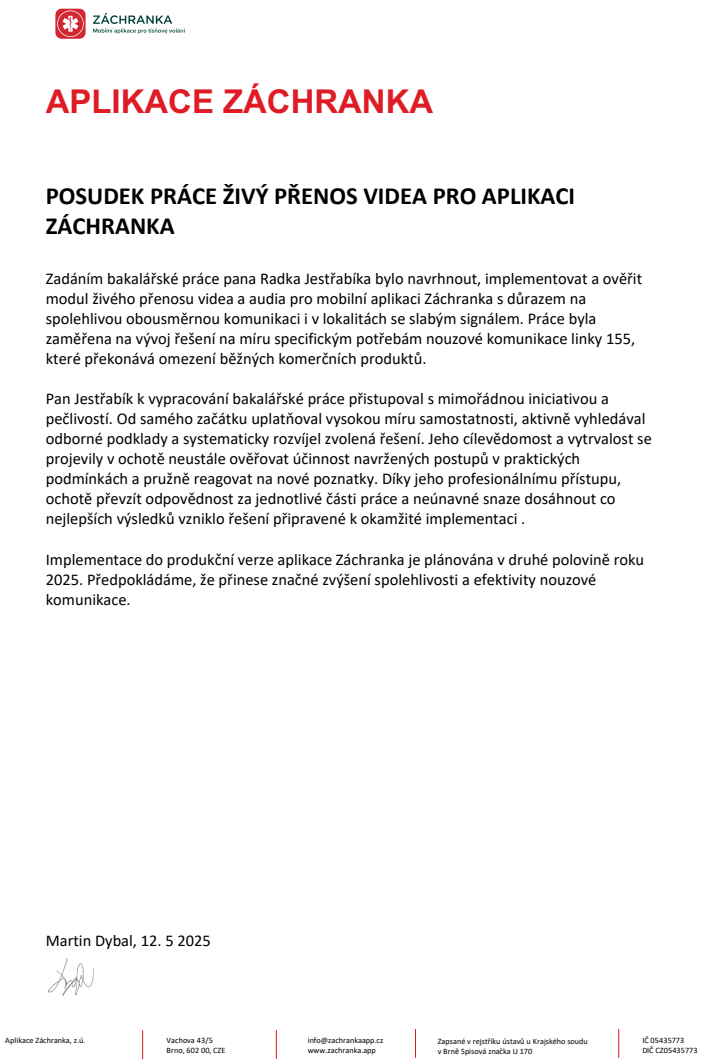
[15] Stewart, R. R. *Stream Control Transmission Protocol* Internet Requests for Comments. RFC 4960. Internet Engineering Task Force, září 2007.

[16] Technologies, E. *WebRTC vs. Traditional Communication Protocols: Unveiling the Future of Digital Connectivity*. 2023. Dostupné z: https://medium.com/@ecosmobtechnologies/webrtc-vs-traditional-communication-protocols-unveiling-the-future-of-digital-connectivity-8e758cd69594.

[17] Westerlund, M. a Wenger, S. *RTP Topologies* Internet Requests for Comments. RFC 7667. Internet Engineering Task Force, listopad 2015.

Příloha A

Posudek konzultanta



ZÁCHRANKA
Mobilní aplikace pro tísňové volání

APLIKACE ZÁCHRANKA

POSUDEK PRÁCE ŽIVÝ PŘENOS VIDEA PRO APLIKACI ZÁCHRANKA

Zadáním bakalářské práce pana Radka Jestřabíka bylo navrhnout, implementovat a ověřit modul živého přenosu videa a audia pro mobilní aplikaci Záchranka s důrazem na spolehlivou obousměrnou komunikaci i v lokalitách se slabým signálem. Práce byla zaměřena na vývoj řešení na míru specifickým potřebám nouzové komunikace linky 155, které překonává omezení běžných komerčních produktů.

Pan Jestřabík k vypracování bakalářské práce přistupoval s mimořádnou iniciativou a pečlivostí. Od samého začátku uplatňoval vysokou míru samostatnosti, aktivně vyhledával odborné podklady a systematicky rozvíjel zvolená řešení. Jeho cílevědomost a vytrvalost se projevily v ochotě neustále ověřovat účinnost navržených postupů v praktických podmínkách a pružně reagovat na nové poznatky. Díky jeho profesionálnímu přístupu, ochotě převzít odpovědnost za jednotlivé části práce a neúnavné snaze dosáhnout co nejlepších výsledků vzniklo řešení připravené k okamžité implementaci .

Implementace do produkční verze aplikace Záchranka je plánována v druhé polovině roku 2025. Předpokládáme, že přinese značné zvýšení spolehlivosti a efektivity nouzové komunikace.

Martin Dybal, 12. 5 2025

Aplikace Záchranka, z.ú. | Vachova 43/5, Brno, 602 00, CZE | info@zachrankaapp.cz, www.zachranka.app | Zapsané v rejstříku ústavů u Krajského soudu v Brně Spisová značka U 170 | IČ 05435773, DIČ CZ05435773

Obrázek A.1: Posudek konzultanta Martina Dybala.