Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Optimalizace procesu nasazení technologie WebRTC

Diplomová práce

Bc. Jiří Dušek

Školitel: PhDr. Milan Novák, Ph.D.

České Budějovice 2020

Bibliografické údaje

Dušek, J., 2020: Optimalizace procesu nasazení technologie WebRTC. [Optimization of application of WebRTC technology. Mgr. Thesis, in Czech]. Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Diplomová práce se zabývá optimalizací nasazení a infrastrukturou systémů založených na WebRTC technologii pro streamování videa a audia prostřednictvím internetové sítě, ve které tato technologie umožňuje vytváření P2P spojení mezi koncovými body. Práce popisuje problémy spojené s P2P přístupem a nalézá alternativu v podobě konkrétního SFU serveru. Do poskytované klientské aplikace je implementován P2P způsob komunikace, který je porovnáván s SFU díky implementaci vlastního měřícího aparátu a realizuje automatizované přepínání obou přístupů a navrhuje další optimalizační změny pro plynulou konferenci s cílem snížení zátěže koncových bodů.

Annotation

The diploma thesis deals with the optimization of deployment and infrastructure of systems based on WebRTC technology for streaming video and audio over the internet network, in which this technology enables creating P2P connections between endpoints. The thesis describes problems related to P2P access and finds an alternative in the form of a specific SFU server. The provided client application implements P2P communication method, which is compared with SFU thanks to the implementation of its own measuring apparatus and implements automated switching of both modes in the background of the ongoing conference and proposes further optimization changes for a continuous conference in order to reduce endpoint load.

Prohlášení

Prohlašuji, že svoji diplomovou práci jsem vypracoval/a samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 19. 5. 2020

Bc. Jiří Dušek

Poděkování

Tímto bych rád poděkoval svému vedoucímu práce, PhDr. Milanu Novákovi, Ph.D., zejména za věcné připomínky při vedení a trpělivost při tvorbě této práce. Poděkování patří také těm, kteří mě v průběhu její realizace podporovali zejména přátelům a užší i širší rodině zvláště pak Petře Janě, Marii, Petru a Pavlovi Duškovím a Jiřímu, Lucii a Martinovi Machiánovím.

Obsah

Seznam zkratek

ACR – Absolute Category Rating

AVC – Advanced Video Coding

AV1 – AOMedia Video 1

CDN – Content delivery network

DTLS – Datagram Transport Layer Security

HD – High definition

HEVC – High Efficiency Video Coding

HTML – Hypertext Markup Language

ICE – Interactive Connectivity Establishment

ISC – Internet Systems Consortium

ISO/IEC – International Organization for Standardization / International Electrotechnical Commission

ITU-T – Telecommunication Standardization Sector of the International Telecommunications Union

MCU – Multipoint control unit

MPEG – Moving Picture Experts Group

NAT – Network address translation

ORTC – Object Real-Time Communications

P2P – Peer to peer

RTC – Real-Time Communications

RTP – Real-time Transport Protocol

SDP – Session description protocol

SFU – Selective Forwarding Unit

TCP – Transmission Control Protocol

TLS – Transport Layer Security

TURN – Traversal Using Relays around NAT

UDP – User Datagram Protocol

VCEG – Video Coding Experts Group

1 Úvod

Uveřejnění zdrojových kódů technologie WebRTC jako open-source provedla společnost Google s cílem vytvořit unifikované API rozhraní pro sdílení médií v internetové síti, které bude dostupné ve všech prohlížečích. [1] Tato technologie je využívána například v aplikacích jako jsou Messenger nebo Facebook Live společnosti Facebook nebo Hangouts společnosti Google, ale také v mnohých dalších jako jsou například platformy pro online konference Amazon Chime, Gotomeeting, Appear.in apod. [3] Javascriptové API současných majoritně používaných prohlížečů jako jsou Google Chrome, Mozilla Firefox, Safari, iOS Safari, Edge, Opera a dalších (viz. kapitola 2.8 Podpora WebRTC technologie) poskytují možnost práce s připojenou web kamerou a mikrofonem a současně také disponují podporou API WebRTC technologie na úrovni, která vývojářům dává možnost vyvíjet aplikace pro streamování médií reálném čase.

Její postavení posiluje také současná snaha o implementaci API pro rozšířenou (AV) a virtuální realitu (VR) v podobě WebXR Device API [4]. V oblasti budoucího vývoje existuje již návrh organizace W3C rozšiřující případy užití pro tuto technologii tzv. "WebRTC Next Version Use Cases"dle [5].

Optimalizace procesů technologie WebRTC je komplexní záležitostí zejména uvážíme-li, že se jedná o komunikaci v reálném čase. Způsob nasazení je závislý na zvoleném případu užití. Hlavními požadavky jsou vysoká kvalita služby a nízké náklady na její provoz, byť se tyto požadavky do jisté míry vzájemně vylučují. Náklady mohou tvořit například spotřeba elektrické energie, nákupy potřebných licencí, hardwaru či softwaru, náklady spojené s vývojem a mzdami nebo administrativou apod. Jedním z cílů vývojářů a zřizovatele dané služby je disponovat maximálně jednoduchým koncovým řešením, které je automatizované a snadno se udržuje a vyvíjí.

1.1 Východiska práce

Autor se v bakalářské práci [2] věnoval problematice webově založených online konferenčních systémů. Výstupem práce byla na základě provedených analýz implementace konferenčního systému konstruovaného právě na této technologii. Šlo o systém, který umožňoval vzájemnou komunikaci účastníků v konferenční místnosti. Účastníkovi umožňoval streamovat vlastní audio a video všem členům v místnosti.

Práce poukázala zejména na nedostatky P2P přístupu při realizaci konferenční místnosti, které se projevovaly markantním snížením kvality či úplným rozpadem streamovaného obrazu a zvuku. V návaznosti na práci bakalářskou bude zvoleným případem užití vzájemná konference účastníků jakožto jedním z východisek.

1.2 Cíle práce

Cílem práce je v návaznosti na autorovu bakalářskou práci najít řešení realizace konference pomocí WebRTC technologie, které překlene její nedostatky. V praxi tak práce nabídne vhled do optimalizačních oblastí WebRTC a zmapuje možnosti jejich řešení. Klíčová je optimalizace z pohledu topologie a propojení mediálních streamů mezi účastníky pro plynulou konferenci a poukáže na výhody a nevýhody jednotlivých přístupů. Dále poukáže na implementační problémy, které souvisí s vývojem takového systému. V případě nasazení mediálních přenosových serverů bude cílem zvolit vhodné řešení, které bude pokud možno reflektovat potřebu nižšího zatížení serverových zdrojů při udržení dobrého uživatelského dojmu z konference.

Další oblastí jsou také video kodeky, které mají svojí kompresní efektivitou přímý vliv na množství přenášených dat v internetové síti a na kvalitu streamu. Cílem bude zjistit chování a možnosti manipulace s kodeky v nasazované WebRTC technologii a vyvodit případné závěry.

Vzhledem k velké škále funkcí, kterými běžný konferenční systém disponuje, bude po komplexní analýze problematiky pro účely práce realizován systém, jež bude implementovat konferenční místnosti pro relace s přenosem audia a videa.

1.3 Metody práce

Teoretická část práce bude zpracovaná jako rešerše daných tématických oblastí: WebRTC technologie, mediální servery pro WebRTC, kodeky a související oblasti. Důležitým zdrojem budou studie a analýzy vypracované organizacemi, které s touto technologií pracují.

Výběr vhodného mediálního serveru bude podložen výsledky komparativní studie těchto serverů. Dalšími kritérii pro výběr serveru jsou možnosti nasazení a integrovatelnost do jiných aplikací a existence dalšího podpůrného software v podobě testovacích nástrojů apod.

Dále proběhne analýza zdrojových kódů a funkčního konceptu zvoleného řešení. S těmito poznatky budou spojeny výstupy teoretického základu v návrhu, který bude implementovat dvě techniky přenosu médií. Pro porovnání bude nutné vytvořit v aplikaci měřící logiku. Porovnávány budou přístupy připojení P2P všech klientů v topologii zvané Mesh a zvolenou technikou vyplývající z teoretické části práce. Kvalita streamování bude testována v prostředí internetové sítě. Na základě vytvořené platformy a provedených testů bude rozhodnuto o preferenci konečného řešení, nasazení P2P techniky a dalších možnostech optimalizace vyplývajících z nabytých poznatků.

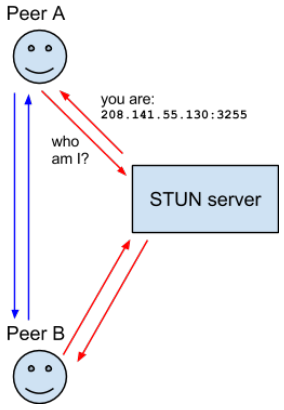
2 WebRTC technologie – koncept funkce a základní komponenty

WebRTC technologie je koncipována jako technologie umožňující přímou komunikaci dvou koncových bodů, kterými jsou často webové prohlížeče, ale může se jednat o jakéhokoli jiného klienta implementujícího potřebná rozhraní, která jsou definována standardem organizace W3C: "WebRTC 1.0: Real-time Communication Between Browsers"[65].

V reálném prostředí internetové sítě je však nutné překlenout některé obtíže. WebRTC si klade za cíl realizovat přímou P2P komunikaci. Máme tedy dva koncové body. Aby mohlo být navázáno spojení, musejí si tyto body nějakým způsobem předat potřebné údaje jako porty, IP adresy a další parametry spojení, protože se doposud navzájem neznají. Tato činnost je zajištěna komunikací skrze centrální server. V terminologii WebRTC hovoříme o signálním mechanizmu. V běžné praxi se využívají webové sockety (nebo již vyjímečně cyklicky opakované ajaxové požadavky). Implementace signálního mechanizmu jakožto i samotný formát předávaných zpráv je čistě v režii vývojáře.

2.1 STUN server – Session Traversal Utilities for NAT

Dalším aspektem je získání veřejné IP adresy, pod kterou je daný klient nalezitelný z prostředí internetu. To je nutné zejména proto, že většina dnešních zařízení jsou od internetové sítě odděleny jedním či více překladači adres (NAT – Network Address Translation) z bezpečnostních důvodů či kvůli limitům IPv4 protokolu. Veřejná IP adresa daného klienta není aplikaci známá a tak je do architektury zařazen veřejně přístupný STUN server, který na přijatý dotaz vrací IP adresu iniciátora dotazu samozřejmě po nezbytné autorizaci příchozího požadavku. Schéma je zobrazeno na obrázku č. 1 převzatého z [62]. STUN je definován dle RFC 5389 [18].



Obrázek 1: STUN server

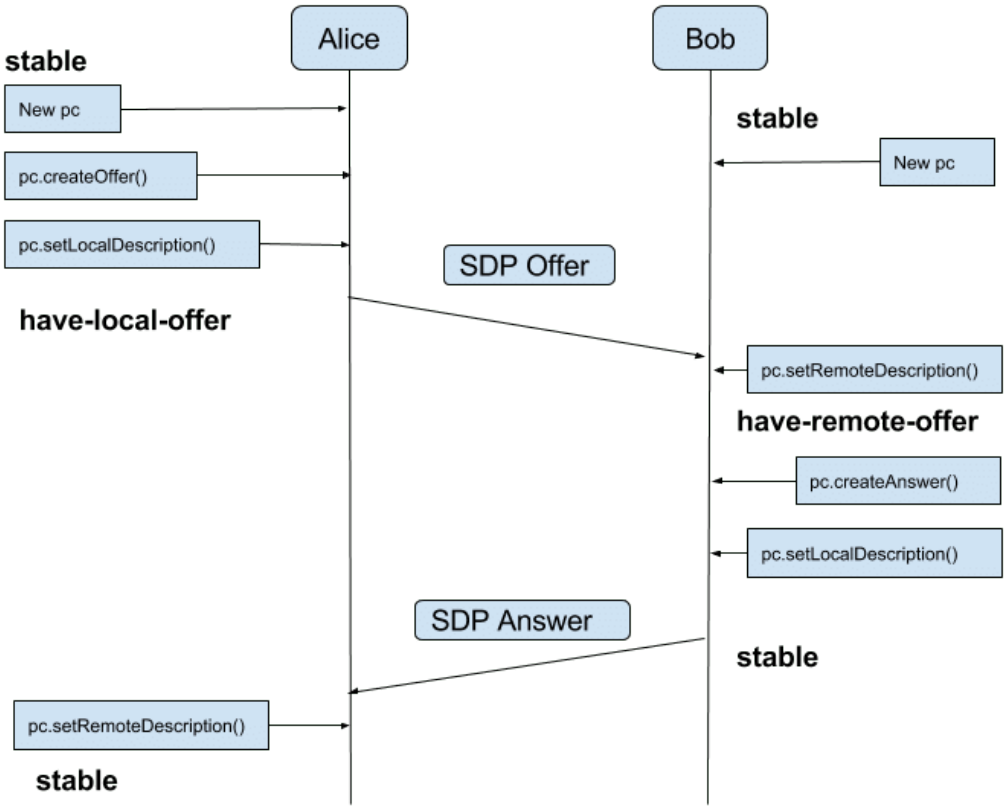
2.2 Protokol popisu relace – SDP a WebRTC

Jak uvádí specifikace WebRTC [65], oba koncové body spolu musejí vyjednat parametry multimediální relace. Jejich součástí je i zjišťovaná IP adresa přes STUN server nebo také lokálně přiřazená IP adresa – druhý koncový bod se může nacházet též ve stejné lokální síti. K těmto údajům ale patří také čísla portů nebo zejména údaje o možnostech streamovaných médií tj. o kodecích, spojení atp. Tyto údaje je třeba předávat ve standardizovaném formátu tak, aby je mohl automatizovaně převzít a aplikovat i druhý koncový bod WebRTC. Formátem této

zprávy je vygenerovaný popis tzv. SDP (Session description protocol). V současné době se jedná fakticky o textový řetězec, nad kterým je v javascriptu vytvořen objekt `RTCSessionDescription`.

Prvním nutným krokem ke vzniku spojení je vytvoření instancí `RTCPeerConnection` objektu obou párů koncových bodů. Dále je na vývojáři, aby stanovil iniciátora komunikace. Ten zasílá prostřednictvím signálního mechanizmu druhému účastníkovi tzv. 'Offer' – popis jeho možností pro vytvoření relace. Tu vytváří uživatel pomocí metody `createOffer()`. Proces výměny zpráv je patrný z obrázku č. 2 [9]. Zobrazuje také signální stavy objektu. Druhý koncový bod reaguje obdobně a zasílá zpět tzv. 'Answer', jež je vytvořena metodou `createAnswer()`. Objekt `RTCSessionDescription` disponuje zejména vlastností `type`, jejímž obsahem je určující text 'offer' či 'answer' a dále disponuje samotným popisem relace v podobě textového řetězce.

Signální mechanizmus je potřebné zajistit jako obousměrnou komunikaci, neboť zajišťuje datovou výměnu zpráv mezi klienty. Častým iniciátorem komunikace je tedy server sám, jež předává zasílaná data. Vhodnou technikou je websocketová komunikace či případně obligádní a z limitujících důvodů spíše zastaralou technikou jako je AJAX long polling, jež představuje cyklické zasílání dotazů na server aj.



Obrázek 2: Schéma realizace výměny zpráv a signálních stavů RTCPeerConnection

2.2.1 Struktura SDP

Popis relace je složen z řádků textu v UTF-8, z nichž každý začíná jendím znakem, za kterým je znak '=' následovaný hodnotou v podobě textu specifického formátu. Řádky textu, které začínají zadaným písmenem, jsou obecně označovány jako "řádky písmen"např. řádky popisu médií mají typ "m", takže tyto řádky jsou označovány jako "m-řádky". [62] Detailnější obsah SDP dle pojednání o složení SDP [61]:

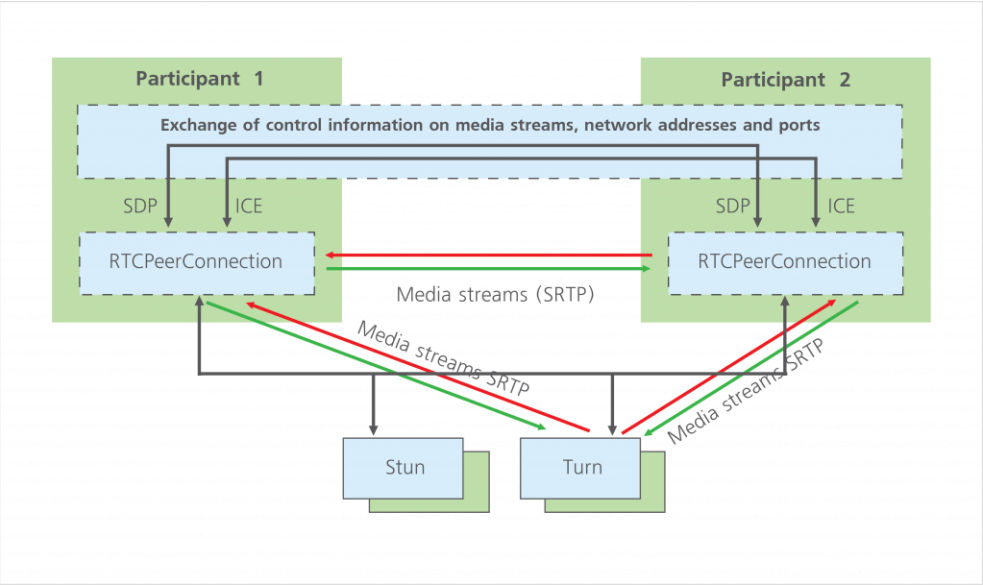
- Globální informace jako jsou:
  - Unikátní identifikátor relace
  - Verze protokolu IP adresy a adresa samotná
  - Číslo udávající, kolik již bylo vzájemně dohodnutých relačních popisů vyjednáno – v průběhu relace může docházet k nutnosti změnit parametry a vyjednat nové
  - Nabídka, jaká média chce daný koncový bod sdílet (audio a video)
  - Unikátní identifikátory pro každý WebRTC Media Stream – každý obsahuje Media Tracky, složky s jednotlivými médii. Každý RTP paket obsahuje tento identifikátor, aby bylo možné určit k jakému streamu dat náleží.
- Informace jak pro video, tak pro audio složku (pro každé zvlášť):
  - Transportní protokoly, formát popisu média
  - Informace o ICE kandidátech – priorita daného kandidáta, přenosové protokoly, porty a IP adresy a bezpečnostní tokeny proti zamezení iniciace komunikace s neautorizovaným koncovým bodem
  - Informace o kodecích
  - Hash certifikátu, který by měl být ověřován při zřizování zabezpečeného spojení
  - SSRC parametry pro identifikaci zdrojů používaných k odesílání médií v RTP.

2.3 Relay TURN server – Traversal Using Relays around NAT

Jak uvádí RFC 5766 [67]: pokud je jeden z koncových bodů umístěn za jedním či více překladači adres, pak v určitých situacích není možné, aby komunikoval s druhým přímo (P2P). V těchto situacích je nutné, aby používal služby mezilehlého uzlu, který funguje jako komunikační prostředník. TURN protokol umožňuje koncovému bodu řídit výměnu paketů skrze tohoto prostředníka se svými protilehlý koncovými body (peery). TURN se liší od některých jiných protokolů v tom, že umožňuje řízení přenosu k více koncovým bodům při použití jedné adresy. Schématické vyjádření je na obrázku č. 3 (převzato ze webové stránky [64]). TURN protokol byl navržen pro použití jako součást ICE komponenty, ačkoli může také být používán bez ICE více viz. v následující kapitole.

Nedojde-li tedy ke zřízení P2P spojení mezi oběma koncovými body kvůli firewallu, specifické konfiguraci atp. některého z aktivních prvků v dané síti, WebRTC zahájí pokus o přenos mediálních streamů skrze přenosový TURN server. V praxi to s sebou také nese nutnost na tomto serveru realizovat z bezpečnostních důvodů systém autorizace, který musí být implementován, aby nedocházelo k nepovolenému přístupu k přenášeným médiím, nebo zneužívání kapacity serveru cizí entitou.

Mezi volně dostupné implementace TURN serveru patří například: Coturn server [10], XTurn - Xirsys TURN Server in Elixir [11], Restund server [12] nebo Pion TURN server[13] napsaný v jazyce Go. Nejznámějším je Coturn, jehož vývoj iniciovala společnost Google. Představuje implementaci vycházející z již zmiňovaného RFC 5766, ale v dnešní době již implementuje celou řadu dalších standardů jako komunikaci na protokolu IPv6 dle RFC 6156 [14], zajištění autentizace prostřednictvím oAuth dle RFC 7635 [15], komunikaci s různými typy databází apod.

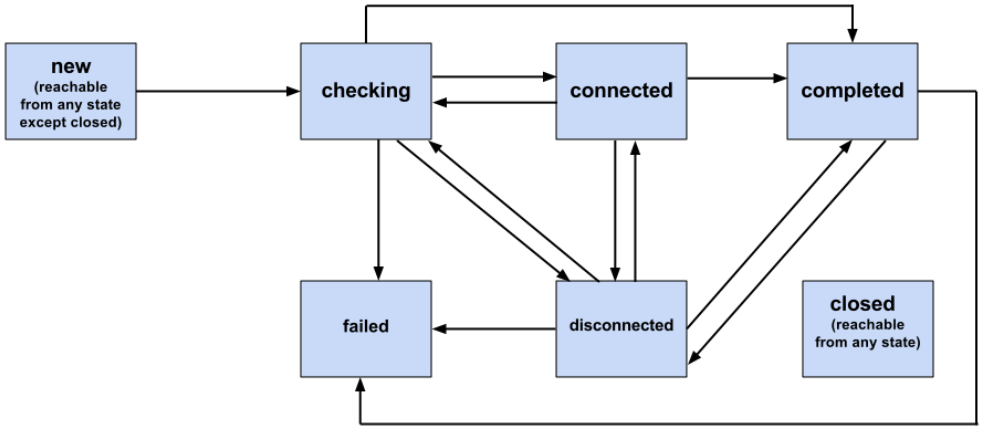


Obrázek 3: Schéma základní architektury P2P komunikace WebRTC

2.4 ICE – Interactive Connectivity Establishment

Jedná se o komponentu zajišťující síťové spojení mezi koncovými body napříč internetovou sítí. Jak uvádí RFC 5245 [69] o ICE, na začátku procesu ICE agenti neznají topologii sítě, v níž se nacházejí. Mohou a nemusí být za jedním čí více překladači adres. Nicméně ICE umožňuje těmto klientům zjistit dostatek informací o jejich topologiích k tomu, aby mohli potenciálně najít jednu či více cest, kterými mohou být propojeni. ICE definuje 3 hlavní typy ICE síťových kandidátů spojení dvou bodů:

- **Host candidate** – využívá nastavení lokálního síťového rozhraní daného zařízení – druhý koncový bod může být v téže lokální síti. Tento typ kandidáta při pokusu o navázání spojení má nejvyšší prioritu.
- **Server reflexive candidate** – zde je využito STUN serveru pro získání veřejné IP adresy, aby doručená SDP nabídka druhé straně obsahovala IP adresu kontaktovatelnou za překladači adres – z vnějšího prostředí internetu.
- **Relayed candidate** – jako poslední možnost, pokud se nezdaří navázat komunikaci přímo, některým z postupů výše, je zvolena volba přenosového TURN serveru, který přemostí komunikaci jako prostředník.



Obrázek 4: Přechodový diagram ICE stavů

Standard technologie WebRTC, z něhož je též přejat obrázek č. 4, definuje výčet stavů ICE komponenty, jež jsou detailněji popsány v seznamu níže [65, 71, 76]. Jak bylo testováno v Chrome 80.0 (Windows 10), posledním zasílaným kandidátem opačné straně je hodnota `null`, jež má ukončit proces zasílání kandidátů.

- **new** – ICE komponenta čeká na příjem ICE kandidáta druhé strany prostřednictvím volání metody `addIceCandidate()`.
- **checking** – ICE komponenta přijala jednoho či více kandidátů druhé strany prostřednictvím signálního mechanizmu a ověřuje je vůči svým lokálním – je-li možné dosáhnout síťového spojení mezi danými páry. Testování zřízení síťového spojení ICE komponentou mezi prohlížeči různých platforem je k nahlédnutí na obrázku č. 28 [53] v přílohách.
- **connected** – Síťové spojení mezi klienty bylo na základě páru kandidátů nalezeno a zřízeno. Nicméně na pozadí může stále probíhat proces ověřování dalších párů pro lepší spojení.
- **completed** – Proces ověřování párů byl dokončen.
- **failed** – Byly ověřeny vůči sobě všechny páry ICE kandidátů a komunikační cesta nebyla nalezena.
- **disconnected** – Jedná se o stav, kdy došlo k selhání spojení některých komponent WebRTC, ale stále je možné, aby se automaticky zotavilo. Pokud nedojde k zotavení, spojení upadne do stavu 'failed'. Důvodem může být např. uzavření prohlížeče, záložky, chyba v komunikaci, pád spojení – změna sítě (např. přechod na wifi) či úplné odpojení druhé strany. Další příčinou může být zahazování paketů šifrovaného P2P spojení firewallem.
- **closed** – aktivita ICE aparátu byla aplikací ukončena.

2.4.1 ICE restart

Může nastat situace, při které se v průběhu inicializace vůbec nezdaří spojení zřídit či během existence WebRTC relace se změní podmínky sítě. Jeden z uživatelů by mohl například přejít z mobilní sítě do sítě WiFi či dojde ve zvolené komunikační trase k problémům, jež znemožní pokračování spojení atp., jak je popsáno ke stavu "disconnected"viz. výše.

V takovém případě může být jedinou možností provedení tzv. restartování ICE komponenty. Jedná se o proces, při kterém je síťové připojení znovu vyjednáno a to stejným způsobem, jako při počátečním vyjednávání ICE s jedinou výjimkou: tok médií pokračuje přes původní síťové připojení, dokud není nové spuštěno. Pak se média přesunou na nové síťové připojení a staré se uzavře.[66]

ICE restart je možné realizovat pomocí metody `pc.restartIce()`. K přikročení doporučeno [70] provádět v momentě, kdy stav ICE komponenty v javascriptovém objektu PeerConnection (v proměnné `pc`) nabyde hodnoty "failed"viz. ukázka kódu události. Restart je vynucen zasláním nové nové vygenerované SDP nabídky.

```
pc.oniceconnectionstatechange = (evt) => {
  if (pc.iceConnectionState === "failed") {
    pc.createOffer({ iceRestart: true })
      .then(pc.setLocalDescription)
      .then(sendOfferToServer);
  }
}
```

2.5 Shrnutí problematiky stavů RTCPeerConnection

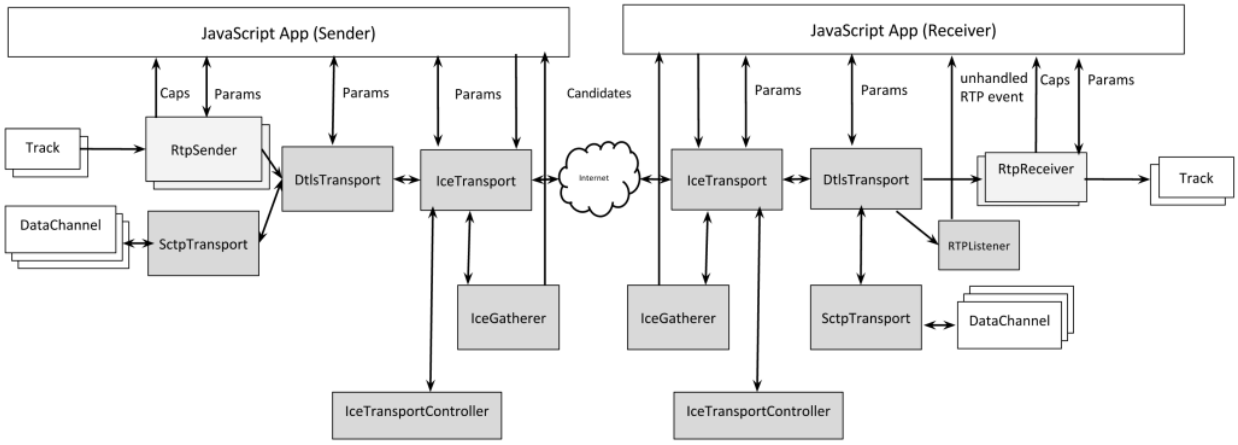
`RTCPeerConnection`, nabývá několika různých typů stavů. Nejprve jde o tzv. signální stav, jež reprezentuje stav spojení z pohledu výměny SDP zpráv. Spojení se nachází ve stabilním stavu tj. při vytvoření instance spojení, dokud výměna zpráv nezapočala, nebo již skončila a zprávy jsou pro daný okamžik oboustranně vyměněny. Hodnotu aktuálního stavu uchovává vlastnost `signalingState` tohoto objektu. Událost změny stavu se jmenuje `onsignalingstatechange`.

V okamžiku nastavení SDP popisu dochází k hledání ICE kandidátů lokálního koncového bodu, jež by mohly být signálním mechanizmem zaslány druhé straně ke spuštění mediální relace. Zda-li probíhá tento sběr, indikuje stav přístupný pod vlastností `iceGatheringState` a událost, jež zachycuje jeho změnu, se jmenuje `onicegatheringstatechange`. Tento stav není z pohledu nutnosti obsluhy a implementace logiky pro realizaci spojení příliš podstatný.

Dále je zde stav, jež souhrně reflektuje mediální síťové spojení a ne jen jeho část jako tomu je v případě předcházejícím. Každé jednotlivé síťové spojení je reprezentováno objektem `RTCIceTransport` a souhrně mluvíme o tzv. ICE komponentě. Její stavy zachytává událost `oniceconnectionstatechange` a aktuální hodnotu uchovává vlastnost `iceConnectionState`.

A stav `connectionState` v podstatě reprezentuje agreaci stavů všech ICE transportů [72]. To je pro vývojáře důležitá informace. Událostí je `onconnectionstatechange`.

2.6 Šifrované odesílání a příjem mediální stopy – jednotlivé komponenty



Obrázek 5: Propojení RTCRtpSender a RTCRtpReceiver objektů – odesílání mediální stopy

Specifikace WebRTC vyžaduje, aby všechna přenášená data - audio, video a data - byla během přenosu šifrována. Protokol Transport Layer Security (TLS) by byl samozřejmě dokonalým řešením, ale nelze jej použít přes UDP, protože se spoléhá na jistotu doručení paketů a jejich správné řazení, které poskytuje TCP. Místo toho WebRTC používá DTLS, což poskytuje rovnocenné bezpečnostní záruky [73].

Obrázek č. 5 je převzat z webové stránky, jež odkazuje na ORTC technologii. Z tohoto důvodu je také v tomto schématu nadbytečným objektem `RTPListener`, který se v implementované WebRTC technologie nenachází. Jinak ale tento obrázek představuje podklad pro vykreslení propojenosti jednotlivých tříd, jež zapouzdřuje `RTCPeerConnection` v souvislosti s šifrovaným odesláním mediální stopy a jejím přijetím druhým koncovým bodem. ORTC představovala objektový přístup k manipulaci s médii nad WebRTC relací a implementoval jej pouze prohlížeč Edge ve verzích 13-18 [74].

Strana odesílatele vytvoří `RTCRtpSender` popisující kódování videa nebo audia mediální stopy (`MediaStreamTrack`). `RTCRtpSender` je připojen k `RTCDtlsTransport`. `RTCDtlsTransport` odpovídá za vytvoření bezpečné relace DTLS s přijímačem a za šifrování média. `RTCDtlsTransport` je připojen k `RTCIceTransport`. Role `RTCIceTransport` je objevit a vytvořit nejlepší síťovou cestu u příjemce [75].

Strana přijemce vytvoří `RTCRtpReceiver` popisující, jak vzdálená strana kódovala audio a video `MediaStreamTrack`. `RTCRtpReceiver` poskytuje `MediaStreamTrack`, který lze připojit k renderovací ploše pro video nebo k přehravači audio. `RTCRtpReceiver` je připojen k `RTCDtlsTransport`. `RTCDtlsTransport` příjemce je zodpovědný za vytvoření odesílatele zabezpečeného připojení a za dešifrování přicházejících médií. Dále je připojen k `RTCIceTransport` [75].

2.7 Uživatelská média v klientském javascriptu

2.7.1 MediaStreamTrack

`MediaStreamTrack` představuje objektovou obálku nad konkrétní mediální stopou. Na základě dokumentace [8] můžeme jmenovat základní vlastnosti takové stopy. Je určitého typu. Nabývá konkrétního stavu. Může být tzv. ztišena. Má vlastní identifikátor, popisek a určení, jedná-li se o médium lokální či cizí (např. přijaté prostřednictvím internetové sítě). Hodnotami 'audio' či 'video' je nastavována jeho vlastnost `kind` určující typ. Vlastnost `readyState` specifikuje akutální stav tj. textový řetězec 'live', jež indikuje, že příjem média ze vstupního zařízení je aktivní, nebo je stopa tzv. ukončena tj. hodnota 'ended'. `muted`, ztišení, je typu boolean stejně jako `remote`, jež udává již zmiňovaný původ zdroje.

Každý takový objekt je také určen i parametry daného média. Zde rozlišujeme parametry požadované při získání mediální stopy ze vstupního zařízení a parametry aktuální, reálné. Požadované parametry jsou součástí dané stopy a to voláním metody `getConstraints()`, ale jak bylo testováno v Chrome 80.0 a Mozilla Firefox 74.0 (Windows 10), informace jsou dostupné pouze v rámci lokálních stop. Stopy, které jsou přijímány ze vzdáleného zdroje tyto informace neposkytují.

Dále disponují aktuálními parametry. Ty se mohou od požadovaných lišit, neboť vstupní zařízení a softwarové rozhraní nemuselo být schopno přesně vyhovět daným požadavkům. V případě přenosu videa přes internetovou síť také dochází ke změnám okamžitého rozlišení v důsledku adaptace datového toku celkovým podmínkám a v případě audia dochází ke změnám vzorkovací frekvence. Načtení aktuálních parametrů lze provést metodou `getSettings()`.

Prostřednictvím API viz. níže můžeme získat informace o podpoře dané vlastnosti pro konfiguraci stopy týkající se média. Požadavky, jež prohlížeč podporuje, můžeme aplikovat metodou `applyConstraints(configConstraints)` i nad existující stopou. Specifikace uvádí celou řadu možných nastavení pro danou stopu. V případě video stopy to může být například počet zobrazovaných snímků za vteřinu, šířka a výška videa, zoom, světelnost, kontrast apod. Nicméně je třeba říci, že zdaleka ne všechny možnosti konfigurace, jež specifikace uvádí, jsou implementovány.

```
navigator.mediaDevices.getSupportedConstraints();
```

2.7.2 Načtení médií

Pro pozdější implementaci P2P techniky je třeba rámcově představit také způsob načítání médií. Načtená média a to ať už jedna či více mediálních stop jsou vždy vráceny metodou `getUserMedia` z objektu `navigator.mediaDevices` jako mediální stream tj. objekt `MediaStream`, který vytváří synchornizovaný konterjner pro načtené stopy. Podobně jako metodu getUserMedia lze volat i další. Jedná se o jednu z částí náležící právě k WebRTC technologii. Stream je v javascriptu proměnné video1 uchovávající právě tento element nastavován následujícím způsobem: `video1.srcObject = stream;`.

Konfigurační objekt definuje preference pro získání specificky nastavených médií. Umožňuje také například načíst konkrétní zařízení respektive konkrétní kameru na základě jejího identifikátoru. Seznam zařízení lze získat

metodou `enumerateDevices()` nad objektem `mediaDevices`. Pro načtení videa s ideálně požadovanými parametry lze využít například takovýto konfigurační objekt.

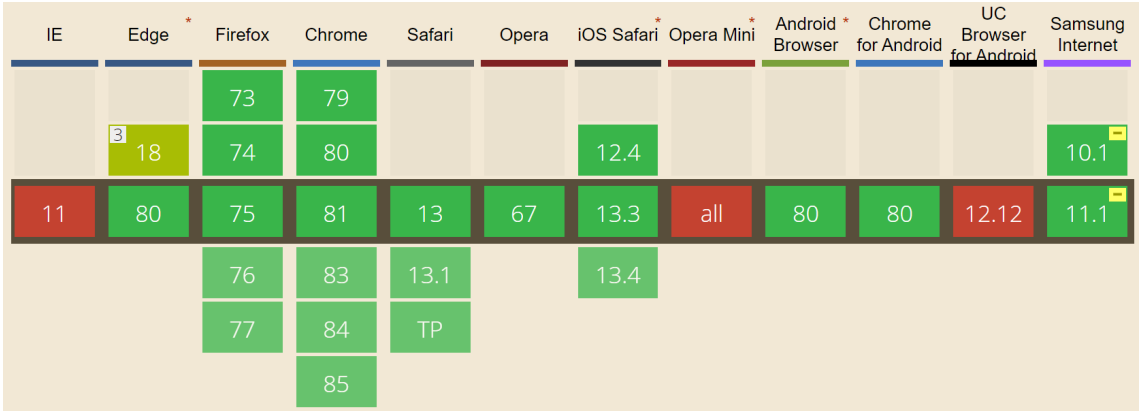
```
{
    video: {
        width: { ideal: 1280 },
        height: { ideal: 720 },
        deviceId : { exact: "7304e1b95cc57c7637fa3a349542aa345fb1" }
    }
}
```

V rámci realizace systému se vyskytla situace, ve které prohlížeč Google Chrome odmítal načíst mediální stream definovaný kódem následující ukázky, byť k tomu nebyl sebe menší důvod a tato implementace je jinak funkční. Při spuštění tohoto kódu docházelo k chybě. Po restartování prohlížeče se však již tento problém neobjevil. Vzhledem k těmto situacím je vhodné takovéto volání zapouzdřit ještě konstrukcí `try` a náležitě vše ošetřit.

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true
});
```

2.8 Podpora WebRTC technologie

Dříve bylo v rámci původního API využíváno tzv. `callback` funkcí. Tento přístup byl s nástupem `Promise` objektů a přístupem async/await změnen, což přináší jednodušší čitelnost v implementaci WebRTC aplikací. Také samotná implementace API se v průběhu doby vyvíjí. Některé metody byly označeny za zastaralé. Jmenujme například metodu `addStream` nad `RTCPeerConnection`, jež byla nahrazena metodou `addTrack` apod. Podporu technologie WebRTC vyobrazuje obrázek č. 6 [68]. Také prohlížeče v různých verzích implementují chování odlišně. V obecné rovině však tato technologie podporu má, ale je třeba brát v úvahu hledisko odlišnosti implementace.



Obrázek 6: Podpora WebRTC tehcnologie

2.8.1 WebRTC a desktopové aplikace postavené na Electronu

WebRTC je také možné implementovat v desktopových aplikacích, které disponují implementací požadovaného API. Electron je open-source framework vyvinutý společností GitHub pro budování desktopových aplikací na platformách HTML, CSS a javascriptu, jež byl iniciován v roce 2013. Electron tohoto dosahuje kombinací projektů Chromium a Node.js v jedné aplikaci, která může běžet na operačních systémech Windows, Linux i Mac. [77]

Slack představuje komunikační nástroj pro týmovou spolupráci v podobě online služby a aplikace. Jeho desktopová verze je postavena na Electronu [79, 80] a umožňuje i audiovizuální konferenci postavenou právě na technologii WebRTC.

2.9 Internetová síť a latence přenosu médií

V internetové síti mezi koncovými body může docházet ke ztátě paketů, jejich poškození, modifikaci nebo zpožedění či desynchronizaci přenášených datových toků apod. V případě realizace jakéhokoli řešení zajišťujícího funkci konferenčního systému a streamování médií v internetové síti obecně obvykle nelze počítat s možností jakkoli upravovat či optimalizovat síťová propojení jako taková. Prostředí sítě je tedy pro vývojáře do značné míry daností – z pohledu možnosti změny konkrétních síťových prvků, jejich konfigurace, současné zátěže či fyzických přenosových médií v trase WebRTC spojení. Nicméně vývojář může aktivně ovlivnit infrastrukturu a celkové řešení vlastního streamovacího systému.

Důležitým faktorem je přenosová rychlost vyjádřená množstvím bitů přenesených za vteřinu. Přímo určuje limity streamování, nicméně tento faktor vývojář obvykle také nemůže ovlivnit (navýšit). Zde nezáleží pouze na rychlosti připojení v koncových bodech, které zajišťuje poskytovatel připojení, ale závisí na možnostech po celé trase spojení. Dalším faktorem, který ovlivňuje výsledný uživatelský dojem, je latence, nebo-li doba prodlevy přenosu médií. Latenci způsobenou vlastnostmi sítě lze rozdělit do čtyř skupin jak je uvedeno níže [82]. Nicméně, jak se ukáže později, nemusí nutně představovat vypovídající hodnotu o kvalitě uživatelského dojmu více viz. kapitola č. 4.3 SFU – Selective Forwarding Unit.

- **Processing delay** – zpoždění způsobené prodlevou při zpracování paketů. Představuje zejména čas spojený se síťovými uzly, jež analyzují záhlaví paketů a určují, kam mají být odeslány. To závisí do značné míry na položkách ve směrovací tabulce či konkrétní implementaci hardwaru a softwaru.

- **Queueing delay** – zpoždění čekáním paketů ve frontě. Představuje dobu mezi umístěním paketů do fronty přenosu a jejich skutečným odesláním. Reálná doba se liší v závislosti na objemu síťového provozu přes dané uzly či implementovaných algoritmech front směrovačů. Svojí roli také hraje spesicifké nastavení či preference propustnosti pro některé typy paketů.

- **Transmission delay** – zpoždění přenosu paketů. Představuje čas potřebný k posunu datových bitů paketů do přenosového média z jednotlivých uzlů.

- **Propagation delay** – zpoždění šíření paketů. Představuje zpoždění vznikající na (fyzických) médiích, která spojují jednotlivé uzly a koncové body. Zpoždění šíření je závislé na typu média a je vyjádřeno jako zlomek rychlosti světla ve vakuu. Například optické vlákno má nižší zpoždění šíření ve srovnání s většinou měděných vodičů.

Každý uzel, kterým jednotlivé pakety musí projít, aby dosáhly koncového bodu, zvyšuje latenci způsobenou zpracováním, čekáním a přenosem. Čím více uzlů existuje mezi koncovými body, tím vyšší zpoždění je. V komunikaci v reálném čase zahrnují běžné příklady uzlů proxy servery, TURN či jiné mediální servery, směrovače a firewall brány. [82]

3 Kodeky, mediální streamy a kontejnery

Kodek představuje algoritmus implementovaný v softwarovém programu pro kompresi či dekompresi, při které dochází k transformaci multimediálních dat. Ta jsou buď kódována nebo dekódována. Zakódovaná data jsou uchovávána v souborech na disku nebo přenášena v bitovém streamu. [23] V javascriptu mohou být skrze WebRTC spojení distribuovány mediální streamy, které reprezentují objekty `MediaStream`. Ty obsahují `MediaStreamTrack` objekty, jež představují obálku datových toků (stop) pro každou z mediálních složek – audia a videa.

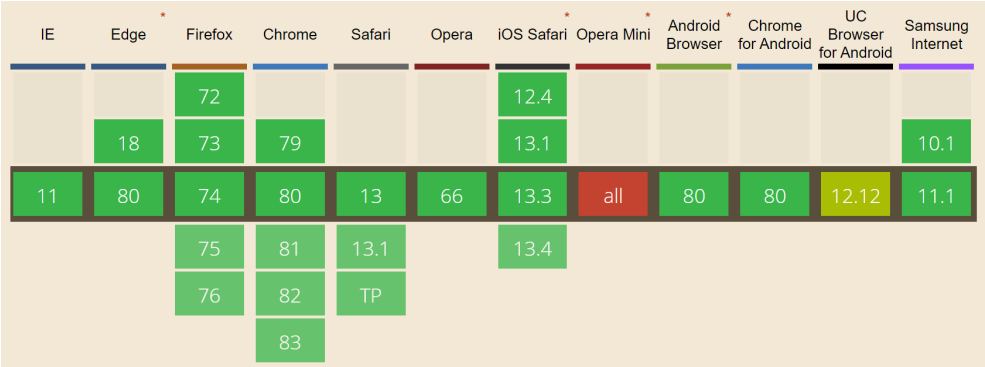
Vylepšená komprese videa je důležitá pro rychlejší a kvalitnější poskytování digitálního obsahu tj. zejména videosouborů a to jak z důvodu úspory jejich velikosti v uložištích tak při přenosu v internetové síti. Všechno od streamování filmů ve 4K až po videohovory na chytrých zařízeních nebo sdílení obrazovky přenosných počítačů může být vylepšeno tím, že se obrazové soubory zmenší pomocí lepších kompresních kodeků. [24] V roce 2021 bude IP video přenos tvořit 82 % ze všech spotřebitelských internetových přenosů a to oproti 73 % v roce 2016 [25].

Soubory pro uchování médií jsou tvořeny tzv. kontejnery. Formát kontejneru (avi, ogg, mpg, ...) definuje celkovou strukturu souboru včetně toho, o jaký soubor se jedná, jeho metadata a další informace o datových stopách. Ty odkazují na skutečně zaznamenané video, audio a další (například titulky) uložené v kontejneru. Audio a video je zakódováno jedním z mnoha kodeků, které jsou specifikovány v metadatech kontejnerového formátu, takže přehrávací zařízení může soubor správně dekódovat. [23]

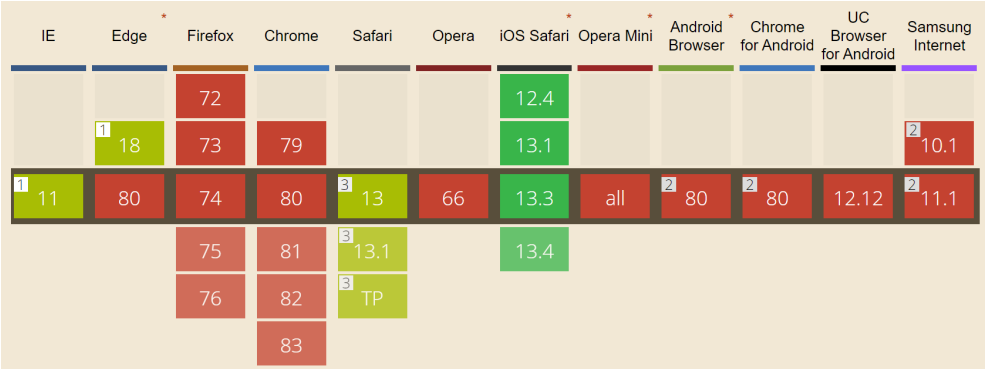
Dle analýzy publikované na webu "Smashing Magazine"[42], který se zabývá tématy vývoje webových technologií, bylo zjištěno, že kolem 50% (testovaných) webových stránek uchovává video obsah na svých vlastních serverech a druhá polovina využívá služeb jako je CDN (Content delivery network). V případě uchovávání souborů na vlastních serverech je jen ve 4% případů formátem webm, ale v 53% případů je formátem mp4, což je kontejnerový formát, který je obvykle kódovaný kompresními kodeky ze skupiny H.26x.

3.1 Kodek H.264/AVC a H.265/HEVC

Kodek H.264 neboli AVC (Advanced Video Coding) byl vyvinut ve spolupráci Mezinárodní expertní skupiny pro telekomunikační unii (ITU-T), expertní skupiny pro video kódování (VCEG) a expertní skupiny ISO / IEC Moving Picture Experts Group (MPEG). Cílem bylo vytvořit kompresní standard, který zajistí efektivní kompresi, distribuci a nahrávání video obsahu. V současnosti je jedním z nejrozvinutějších standardů v oblasti kodeků. [43, 38] Finální návrh kodeku byl představen v roce 2004. Jeho nástupcem je kodek H.265 neboli HEVC (High Efficiency Video Coding), který byl představen roku 2012 na kongresu mobilních zařízení "Mobile World Congress"a jeho první finální verze byla vydána roku 2013. Licence používání těchto kodeků je zpoplatněna. [38, 39] Na obrázcích č. 7 a č. 8 (převzatých z webových stránek [34] a [35]) můžeme vidět podporu těchto kodeků v prohlížečích.



Obrázek 7: Podpora kodeku H.264 v prohlížečích

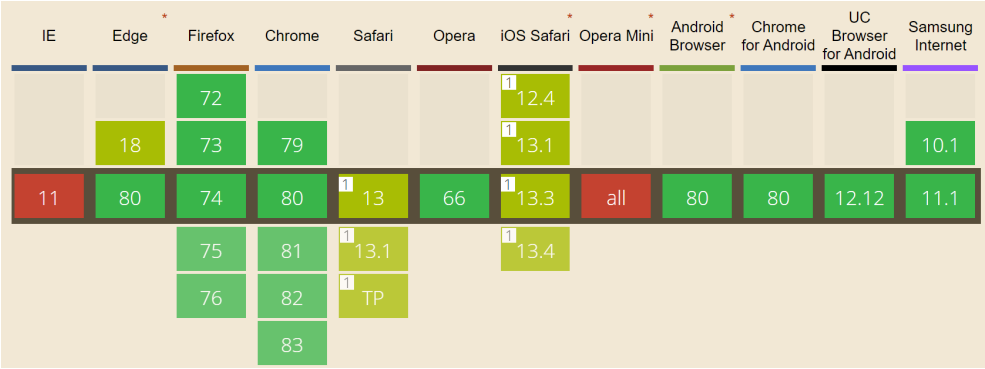


Obrázek 8: Podpora kodeku H.265/HEVC v prohlížečích

3.2 WebM projekt a kodeky VP8 a VP9

WebM je open-source projekt podporovaný společností Google iniciovaný roku 2010, který představuje formát video souboru určeného pro web. Soubory webM se skládají z video streamů komprimovaných video kodekem VP8 nebo jeho nástupcem VP9, audio streamů komprimovaných open-source audio kodekem Vorbis či Opus a textovými stopami WebVTT. Struktura souboru WebM je založena na kontejneru Matroska (.mkv). [36]

Kodeky VP8 a VP9 jsou vysoce efektivní technologie pro kompresi videa. Kodeky mohou být využívány kýmkoli bez licenčních poplatků stejně jako audio kodeky Vorbis a Opus, jež jsou vyvinuty organizací Xiph Foundation [36]. Podpora kodeků je znázorněna na obrázku č. 9 převzatého z portálu Can i use. [37].



Obrázek 9: Podpora WebM formátu – kodeků VP8 a VP9 v prohlížečích

3.3 Srovnání efektivity kodeků

Dle studie [40] zveřejněné portálem "ResearchGate"pro publikaci vědeckých studií v oblasti technologií kompresní standardy VP9 i HEVC poskytují vyšší kompresi ve srovnání s průmyslovým standardem H.264. Současně HEVC kodek poskytuje lepší kompresi než VP9, ale VP9 lze na rozdíl od HEVC použít bez licenčních poplatků. Experimenty ukázaly, že hlavní příčinou vyšší účinnosti oproti kodeku VP9 je způsoben lepší predikcí.

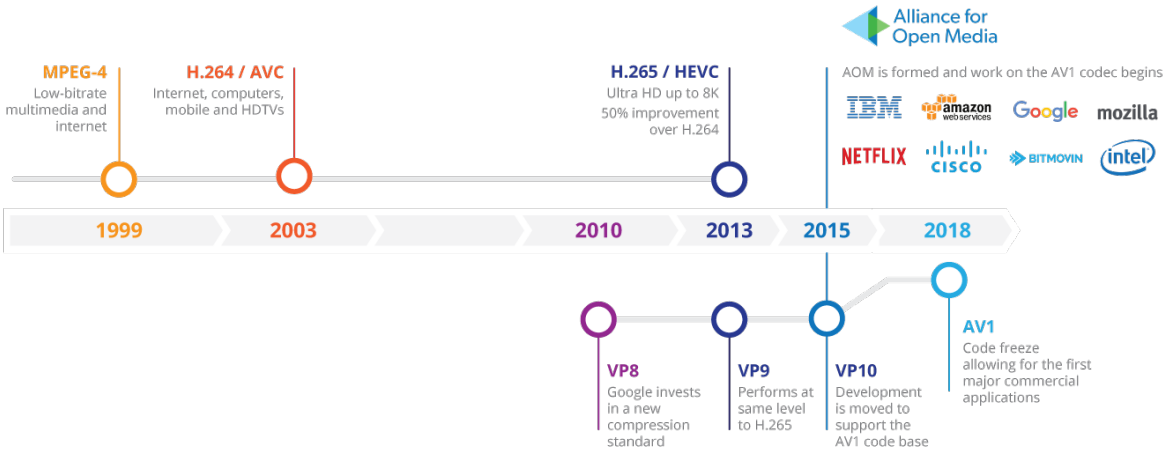
Taktéž další studie [41] zveřejněná v online repozitáři švýcarské technické vysoké školy "École Polytechnique Fédérale de Lausanne (EPFL)"ukazuje, že kodek HEVC nabízí vylepšení komprese ve srovnání s kodeky VP9 a H.264. Na základě objektivního měření bylo zjištěno, že HEVC dosahuje průměrných úspor ve výši 57,3 % oproti H.264 a až 33,6 % oproti VP9, zatímco průměrné snížení bitového toku při použití HEVC na základě subjektivního měření vnímané kvality dosahuje 59,5 % oproti H.264 a 42,4 % oproti VP9.

3.4 Kodek AOMedia Video 1 – AV1

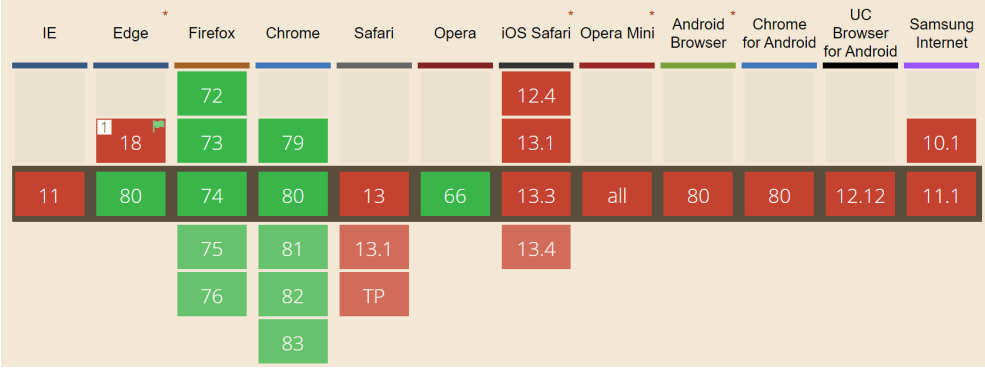
Kodek AV1 je nový univerzální video kodek vyvinutý společností Alliance for Open Media. Aliance zahájila jeho vývoj v roce 2015. Jeho výchozím bodem byly projekty kodeků Google VPx, Thor společnosti Cisco a kodeku Daala Mozilla / Xiph.org. Překonává svými parametry všechny z používaných kodeků včetně VP9 a HEVC v závislosti na parametrech až o desítky procent, což z něj potenciálně činí kodek příští generace. AV1 je open-source projekt, jehož použití je bez licenčních poplatků. V současné době je participujícími organizacemi vyvíjeno úsilí o jeho průmyslové nasazení a využívání [24, 26, 27]. Vývoj kodeků je patrný na obrázku č. 10 převzatého z [31].

První oficiální verze kodeku byla představena v březnu 2018. Na jeho vývoji se podílejí zaměstnanci organizací Google, Microsoft, Cisco, IBM, Mozilla, Netflix, Amazon, Apple, Facebook, Nvidia, Intel a Arm. [26, 28] Prohlížeč Google Chrome v desktopové variantě ve výchozím nastavení od verze 70 disponuje podporou tohoto kodeku [32]. Prohlížeč Firefox ve variantě Nightly jím disponuje už od verze 58, nicméně ale až po zapnutí příslušného příznaku `media.av1.enabled` v sekci prohlížeče `about:config` [33]. Obrázek č. 11 převzatý z webové stránky [44] poukazuje na současnou podporu prohlížečů této technologie. Nicméně během práce s WebRTC se ukázalo, že objekt `RTCRtpSender` určující možnosti kódování odesílaných médií nenabízí možnost nastavení tohoto kodeku jako jednoho z volitelných.

Na obrázcích 12 a 13 (převzatých z webové stránky [49]) je srovnání efektivity kodeku AV1 s kodeky HEVC a



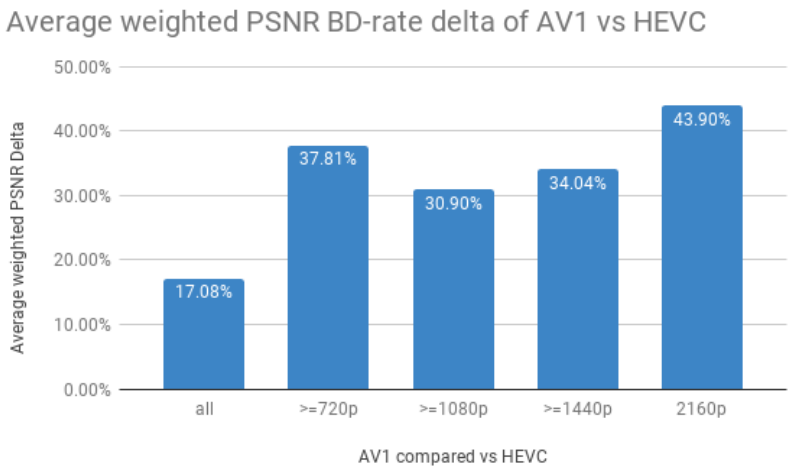
Obrázek 10: Vývoj video kodeků v čase s potenciálním nástupcem AV1



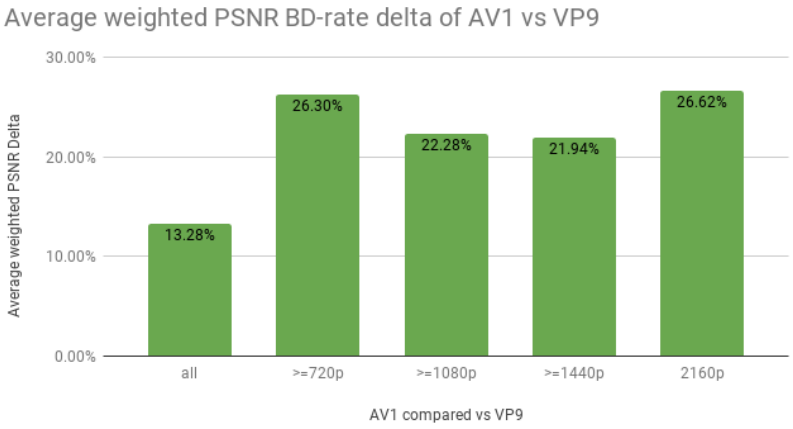
Obrázek 11: Podpora kodeku AV1 v prohlížečích

VP9. Reflektují procentuálně vyjádřenou redukci datového toku zjištěného na základě PSNR metody v závislosti na rozlišení videa. Při výpočtu v celém rozsahu datového toku byla naměřena průměrná redukce AV1 ve srovnání s HEVC o 17 % a ve srovnání s VP9 o 13 %. Když se zaměříme na videa s vyšším rozlišením, s AV1 snížení datového toku ve srovnání s VP9 vzroste na 22 - 27 % a ve srovnání s HEVC se snížení zvyšuje na 30 -43 % [49].

PSNR (The peak signal to noise ratio) neboli poměr špičkového signálu k šumu je běžně přijímaná a akceptovaná metrika používaná experty v oblasti kompresních kodeků pro objektivní měření účinnosti kódovacích algoritmů. Současně je ale známo, že PSNR nereflektuje přesně lidské vnímání vizuální kvality obrazu. Vypočítává poměr signálu k šumu v decibelech mezi dvěma obrazy. Tento poměr se často používá jako měření kvality mezi původním a komprimovaným obrazem přičemž BD-rate (The Bjontegaard rate difference) představuje rozdíl efektivity kódování dvou rozdílných kompresních algoritmů [50].



Obrázek 12: Vyjádření snížení datového toku za použití AV1 oproti HEVC kodeku



Obrázek 13: Vyjádření snížení datového toku za použití AV1 oproti VP9 kodeku

3.5 Detekce podporovaných kodeků v prohlížeči

Detekce podporovaných kodeků pro přehrávání videa je v prohlížeči možná a to javascriptovým voláním metody `canPlayType` nad objekty `HTMLMediaElement`, které reprezentují video či audio HTML elementy.

```
const videoEl = document.createElement('video');
const isPlayable = videoEl.canPlayType('video/webm; codecs="vp8, vorbis"');
```

Podpora této metody je rozšířena ve všech verzích standardních prohlížečů. V Internet Exploreru od verze 9 a v prohlížeči Mozilla Firefox od verze 3.5. Metodá má 3 návratové typy [52]:

- '**probably**' – médium bude s největší pravděpodobností možné přehrát.
- '**maybe**' – jestli bude možné médium přehrát, se ukáže až při přehrávání.
- Prázdný textový řetězec značí, že médium není možné přehrát.

Příklady možných MIME typů jako argumentů metody `canPlayType` [52, 45] doplněný právě i o nový AV1 kodek viz. níže.

```
  let formats = {
    ogg: 'video/ogg; codecs="theora"',
    h264: 'video/mp4; codecs="avc1.42E01E"',
    webm: 'video/webm; codecs="vp8, vorbis"',
    vp9: 'video/webm; codecs="vp9"',
    hls: 'application/x-mpegURL; codecs="avc1.42E01E"',
    av1: 'video/mp4; codecs=av01.0.05M.08,opus'
  };
```

Příklad HTML elementu se souborem načítajícím video kódovaným pomocí AV1 by vypadal takto:

```
<video controls width="800" height="600">
  <source src="video.av1.mp4" type="video/mp4; codecs=av01.0.05M.08,opus" />
</video>
```

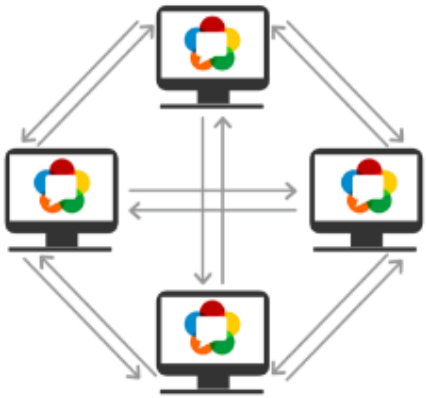
4 Topologie propojení koncových bodů

I přes to, že je možné uskutečňovat videohovory s více účastníky pomocí komunikace P2P, i za optimálních síťových podmínek videohovor nefunguje s více než pěti účastníky. V takovém okamžiku je užitečný mediální server, protože pomáhá snížit počet streamů s médii, které klient odesílá, a dokonce může snížit počet streamů, které klient přijímá – v závislosti na možnostech mediálního serveru. [7] Topologie propojení bude tedy klíčovým faktorem při optimalizaci WebRTC ale i jakékoli jiné aplikace pro streamování v reálném čase v prostředí internetu. Dle zdrojů [6, 7, 19, 20, 21] můžeme rozdělit topologie do několika obecných modelů. Do konference je připojeno N účastníků:

- **(Full) Mesh** — účastník konference typicky odesílá N-1 streamů vlastního média ostatním účastníkům. Stejný počet streamů i přijímá.
- **MCU – Multipoint control unit** — účastník konference typicky odesílá jeden a přijímá také jeden mediální stream. MCU server zpracovává všechny streamy, ze kterých v reálném čase vytváří pro každého ze zúčastněných speciální stream, který mu zasílá.
- **SFU – Selective Forwarding Unit** — účastník konference odesílá typicky jeden (či více) streamů a přijímá N-1 streamů ostatních účastníků.

4.1 (Full) Mesh

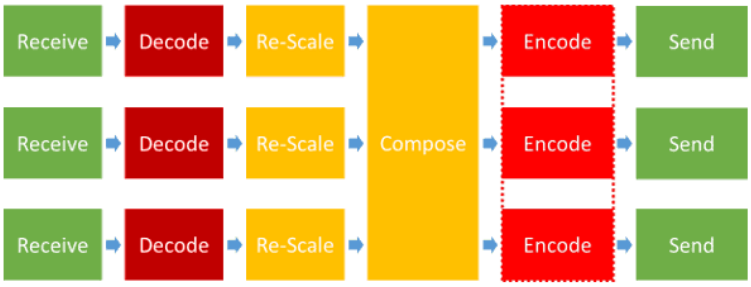
P2P WebRTC je decentralizovaný mediální protokol, který umožňuje výměnu médií a dat přímo mezi koncovými body. Ve standardním modelu připojení P2P se všichni účastníci navzájem spojují v síti známé jako Mesh (viz. obrázek 14 převzatý z webové stránky [7]). Veškeré zpracování média probíhá na hardwaru obou koncových bodů. To funguje až do okamžiku, kdy procesor začne vyčerpávat svoji kapacitu pro zpracování všech mediálních streamů, které jsou odesílány k ostatním a přijímány od všech ostatních účastníků. Jakmile je příliš mnoho koncových bodů navzájem spojeno tímto způsobem, začne se kvalita spojení rozpadat. [6] Některé prvky sítě jsou nakonfigurovány tak, že neumožní zřízení přímého P2P spojení mezi danými koncovými body (např. ochranou bezpečnostním firewallem apod.), a proto musí být do architektury zařazen mediální přenosový TURN server více viz. v kapitole Relay TURN server – Traversal Using Relays around NAT.



Obrázek 14: WebRTC konference – P2P Mesh topologie

4.2 MCU – Multipoint control unit

Při videokonferenčním hovoru přijímá mediální server MCU audio a video streamy, dekóduje je a propojuje dekódované rámce od ostatních účastníků. Případně je transformuje a poté opět kóduje, aby výsledné streamy mohly být poslány ke svému cíli. [16, 19] S nadcházejícími změnami v kvalitě videa a displejů se brzy výkonové požadavky na MCU servery opět zvýší. 4K a 8K displeje a kamery se stávají realitou a nejvíce zatěžující činností MCU serveru je právě kódování a dekódování videa. V případě zajištění MCU serverů je třeba zohlednit výkonný hardware, spotřebu energie, prostory či administrativní náklady atp. [19] Schématicky je funkce MCU serveru znázorněna na obrázku č. 15 (obrázek přejat z článku z portálu Gloggeek.com [19]) a počet odesílaných a přijímaných streamů klienty na obrázku č. 16 z článku z portálu Medium.com [20].

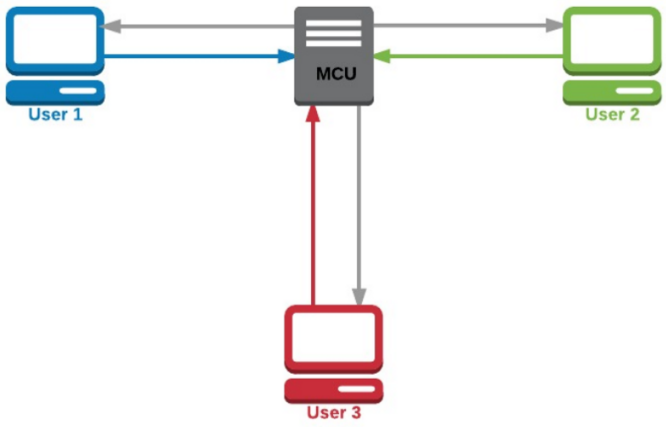


Obrázek 15: Schématické zpracování médií MCU serverem

Základní vlastnosti architektury MCU:

- Šetří výkon klientských koncových zařízení, která reprodukují média, protože dochází typicky k dekódování i kódování jednoho příchozího a odchozího streamu.
- Snižuje oproti SFU a P2P řešení síťový provoz.

- Principiálně se jedná o centralizované řešení s delší dobou odezvy než v případě SFU řešení.
- Náročnost na zajištění serverové kapacity a výpočetního výkonu je vyšší oproti SFU řešení.
- Na serveru umožňuje provádět s médii dodatečné úpravy jako filtrování šumu a ozvěny, úpravy audio stop, vylepšení kvality obrazu apod. [20]
- Softwarová implementace je náročnější než v případě SFU řešení
- Každý stream je překódován na MCU mediálním serveru a není tak nutné, s protilehlým účastníkem sdílet stejný kodek nebo rozlišení.



Obrázek 16: WebRTC konference – MCU server

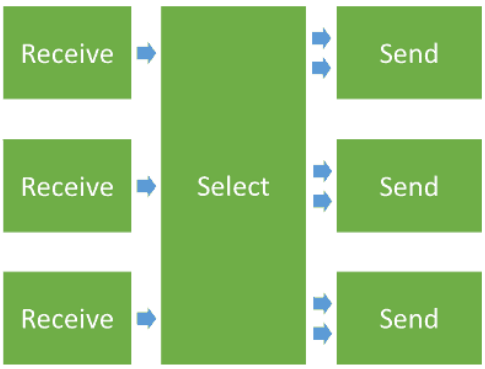
4.3 SFU – Selective Forwarding Unit

Využití SFU serveru představuje topologii umožňující klientům odesílat streamy na centralizovaný mediální server, odkud jsou pak směrovány k ostatním klientům viz. obrázek č. 17 z článku pojednávajícím o SFU jako optimálním řešení [17]. Topologie SFU je atraktivním přístupem k řešení problému výkonu serveru, protože nezahrnuje náklady na výpočet dekódování a kódování videa. Latence přidaného mediálního serveru SFU je oproti MCU serveru minimální. [22] Obrázek č. 18 z článku o modelech komunikace [20] znázorňuje principiální činnost SFU jednotky. Ta přijímaný stream rozesílá koncovým bodům.

Základní vlastnosti architektury SFU:

- Principiálně se jedná o centralizované řešení, které funguje jako směrovač, a proto je oproti MCU jednotce výkonově úspornější a streamy jsou distribuovány s minimální dobou odezvy.
- Klientská zařízení přijímají N-1 streamů a odesílají jeden (či více – v případě simulcast)
- V určitých případech šetří zdroje klientských zařízení oproti P2P přístupu.
- Klienti mají plnou kontrolu nad přijímanými streamy.
- Klienti mohou mediálnímu serveru zasílat simulcast, tedy více obsahově totožných streamů o různých technických parametrech zejména různého rozlišení. Například pro klientské zařízení, které nepodporuje

Obrázek 17: WebRTC konference s SFU serverem



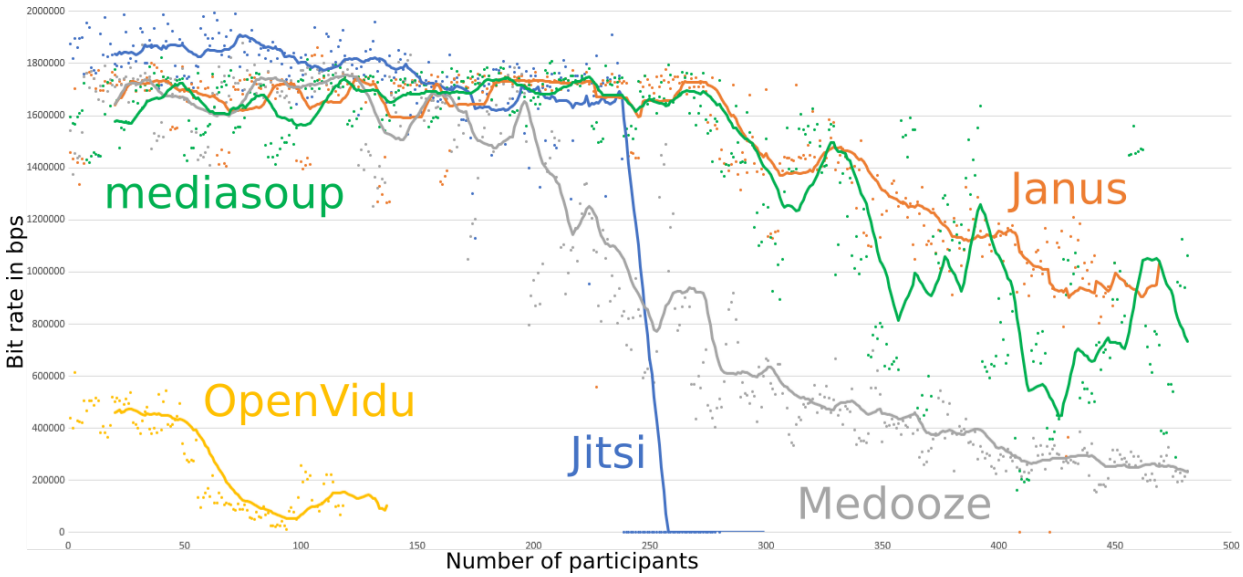
Obrázek 18: Schématické zpracování médií SFU serverem

přehrávání videa ve 4K rozlišení, nebude tento stream server vůbec odesílat a má-li k dispozici méně kvalitní stream, pošle jej místo něj. [21] Jednotka tak může na základě informací o klientských zařízeních provádět inteligentní rozhodnutí pro optimalizaci přenosu.

5 Výběr mediálního SFU serveru

V současné době existuje řada implementací mediálních SFU serverů. Tato práce se zabývá zejména open-source projekty. Klíčové je, zda-li daný server bude vhodný pro námi zvolený případ užití, kterým je konference a přenos audia a videa mezi všemi účastníky současně. Důležitým aspektem výběru je množství účastníků, které je jedna instance přenosového serveru schopna obsloužit, jednoduchá nasaditelnost a rozšiřitelnost či licence.

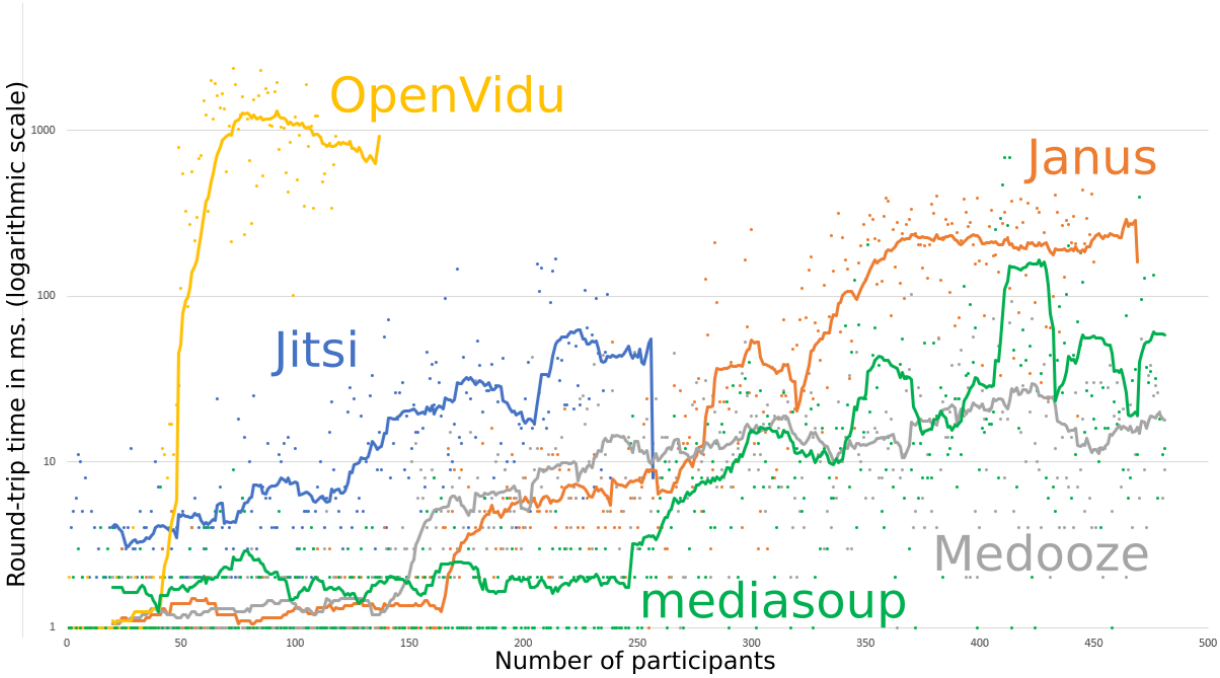
V rámci této práce jsou použity výsledky studie komparativních zátěžových testů: "Comparative Study of WebRTC Open Source SFU's for Video Conferencing"[46], které ukazují na některé servery jako na vhodné kandidáty. Sady testů pro open-source implementace SFU serverů byly provedeny organizací CoSMo Software [54] pod vedením Dr. Alexandra Gouaillarda, ředitele této společnosti a přispěvatele na potálu webrtchacks.com. Organizace CoSMo Software se zabývá problematikou WebRTC technologií, jejího nasazení a testování.



Obrázek 19: Datový tok přenášených médií k počtu účastníků zatěžujících přenosové jednotky

Výsledky testů jsou znázorněny na následujících grafech. Obrázek č. 19 [47] udává datový tok přenášející danou serverovou instancí k počtu účastníků pro každý server.

Je zde vidět, že algoritmy řídící přetížení WebRTC začínají upravovat datový tok přibližně při 250 připojených účastnících. Dále obrázek č. 20 [47] však ukazuje, že latence spojení se zvyšuje lineárně. Navzdory snižujícímu se datovému toku a zvyšující se latenci, metrika kvality videa uvedená na obrázku **??** hlásí snížení kvality mnohem později. To ukazuje, že přenosová rychlost a latence nejsou dobré ukazatele pro měření kvality videa. [47] Jak vyplývá z grafů, nejlepších výsledků dosahují řešení Janus, Mediasoup a Medooze.



Obrázek 20: RTT (doba odezvy) k počtu účastníků zatěžujících SFU jednotku

5.1 Uvedení nejlepších testovaných serverů

5.1.1 Janus

Janus [55] představuje implementaci mnohoúčelného SFU WebRTC serveru od společnosti Meetecho zabývající se streamováním médií v reálném čase. Jeho hlavním autorem je Lorenzo Miniero. Licencován je pod GNU General Public License ve verzi 3. Projekt Janus je realizován v jazyce C a jeho architektura je řešena zásuvnými moduly, které poskytují pouze požadovanou funkcionalitu SFU jednotky, kterou lze dále rozšiřovat.

5.1.2 Mediasoup

Mediasoup projekt [59] představuje skupinu 3 projektů: klientskou část, serverovou SFU část a uživatelskou aplikaci konferenčního systému. Autory projektu jsou Iñaki Baz Castillo a José Luis Millán. Všechny části spadají pod ISC licenci. Celý koncept fungování serveru je precizně dokumentován. Projekt je vyvíjen v jazyce C++ a javascriptu dle API ECMAScriptu verze 6. Jednotlivé části projektu lze nasadit jako NODE moduly.

5.1.3 Medooze

Dalším projektem spadajícím do testování, jež vykazoval oproti jiným dobré výsledky zatížitelnosti, byl Medooze [60], jehož autorem je Sergio Garcia Murillo. Projekt je licencován jako GNU General Public License ve verzi 2 pro nasazení implementace v jazyce C++ a GO nebo jako MIT licence při využití implementace Node modulu.

5.2 Výběr SFU serveru

Vzhledem k tomu, že Medooze server dosahuje ve zmíněné studii horších výsledků než dva další, je tento server z výběru vyřazen. Zbývajícími možnostmi je Janus a Mediasoup řešení. Tyto dvě vykazují při stovkách připojených účastníků v konferencích nejlepší vlastnosti. Jako řešení pro další optimalizaci bude zvolen Mediasoup SFU server. Výhodou je možnost přímo jeho nasazení jako Node modulu. Tak jej lze snadno integrovat do libovolné aplikace běžící na Node jako její součást a lze tak jednoduše kombinovat mnohé již existující knihovny a řešení pro dosažení požadavků na aplikaci. V neposlední řadě je výhodou jeho licence a dále k aplikaci existuje také vlastní aplikace, která poslouží jako základ pro další možnosti optimalizace nasazení této technologie.

6 Aplikace Mediasoup

Cílem je porovnat a ověřit chování P2P a SFU spojení, tak abychom mohli rozhodnout o využití obou přístupů, zejména P2P a případně přistoupit k dalším optimalizačním změnám. Samotnému porovnání se věnuje kapitola č. 9. Zapotřebí bude tedy obou přístupů s jednotným prostředím pro testování spojení. Je možné buď vytvořit vlastní prostředí či využít možnosti již existujícího řešení SFU aplikace konferenčních místností a v rámci ní implementovat i P2P techniku. K tomuto kroku bude přistoupeno.

Vzhledem k tomu, že aplikace Mediasoup (tzv. demo) ve verzi 3.0.0., která bude využita, je poskytována na serveru Github s licencí ISC, jsou její zdrojové soubory otevřené a může být její kód modifikován pro vlastní účely.

Aplikace je přístupná na serveru Github pod účtem Versatica. Existuje k ní i dokumentace na webových stránkách [59]. Ke své funkci potřebuje samostatné knihovny `mediasoup-client` pro klientskou část a `mediasoup-server` jakožto serverovou SFU jednotku. Struktura aplikace je taktéž rozdělena a to do dvou hlavních adresářů. Prvním je `app`, jež slouží klientské části a adresář `server`,jež spouští aplikaci mediálního SFU jednotky a websocketový server.

Aplikace představuje online konferenční místnost pro přenos audia a videa. V levém dolním rohu jsou zobrazena média daného uživatele. Tj. video element o velikosti přibližně 300x250 pixelů. Vizuálně v jeho horní části jsou zobrazena tlačíka pro zakázání a povolení vlastních médií a změnu webkamery. V dolním rohu tohoto elementu je jméno daného uživatele a verze jeho prohlížeče s ikonou. Jméno je generováno náhodně ze seznamu. Uživatel disponuje možností úpravy svého jména. Zobrazované jméno ale může být také nastaveno z URL adresy.

Všichni připojení uživatelé, které klient vidí, jsou centrováni na střed. Každý má vyhrazenou plochu maximálně do 450x380 pixelů. Na mobilních zařízeních je tato velikost upravována. V horní části se zobrazují ikony notifikující o zakázaném přenosu média tj. zvuku a videa. V případě chybějícího videa je místno něj zobrazen obrázek siluety. V dolní části pak jméno účastníka a verze prohlížeče s jeho ikonou. Dále také každý obsahuje příslušné informace o probíhajícím přenosu.

Aplikace nevyužívá databázového uložiště. K dané konferenci se účastník připojí prostřednictvím linku s definovým identifikátorem místnosti, které udržuje běžící serverová instance. Pokud v místnosti není žádný uživatel, je zrušena.

Aplikace jako taková je též navržena i pro jiný případ užití a tím je tzv. ”One-to-many”, kdy jeden uživatel odesílá mediální obsah a ostatní jej přijímají.

Projekt je postaven na knihovně React-Redux a disponuje tak jednotným uložištěm pro stavy aplikace. Kombinuje kód psaný čistě v javascriptu (js) a v tzv. jsx tj. soubory psané v rozšířené syntaxi oproti běžnému javascriptu (syntax extension to JavaScript). V projektu je dále využíváno konstrukcí ES6 například `async/await` funkcí, jež umožní pozdržení zpracování daného kódu a přináší tak možnost serializace provádění příkazů.

6.1 Reactové komponenty

Klíčovým objektem, který zajišťuje veškerou logiku a inicializaci procesů konference je třída `RoomClient`. Ta je také využívána k obsluze událostí, jejichž zachytávání realizují reactové komponenty. Všechny reactové komponenty v podobě tříd dědí od třídy `Component` z balíčku `react`.

Stěžejní komponentu představuje třída `Room`, která umisťuje všechny ostatní komponenty a HTML části. Do ní je vsazena komponenta zajišťující notifikace, jež se každá po určitou dobu zobrazuje v pravém dolním rohu konferenční místnosti. V horní části na středu je také umístěn odkaz, přičemž pokud uživatel na něj klikne, je mu do schránky zkopírována URL adresa pro jeho snažší sdílení. Dále je vložena komponenta `Peers`, která představuje klienty jež jsou právě nyní připojeni.

Komponenta `Peers` vytváří jednotlivé `Peer` komponenty. Ty představují všechny připojené klienty do místnosti. Na obrázku č. 23 jsou takto reprezentováni dva klienti a to na panelech Klient 2 a 3. Komponenta `Peers` je tak eventuálně největší komponentou co do plochy, pomineme-li místnost jako takovou. Všichni připojení účastníci jsou centrováni na střed. Základní komponentu jakéhokoli zobrazení médií tvoří komponenta `PeerView`. Ta je nasazena jako základ jak pro `Peer`, tak pro `Me` tj. pro média lokálního účasníka.

6.2 Mediální přenos a RTCPeerConnection

Klientská aplikace Mediasoup implementuje v komunikaci se serverem dvě `RTCPeerConnection` spojení. Ta jsou uchovávána ve ve vlastnostech `_sendTransport` a `_recvTransport`, které jsou součástí hlavní třídy `RoomClient`. Jedno spojení slouží pouze pro příjem mediálních dat a druhé je pouze odesílá.

Výhodou tohoto přístupu je udržení částečného přenosu dat v případě pádu jednoho ze spojení. Pokud selže spojení, jež přijímá data, odesílání vlastního média všem přítomným v místnosti zůstane aktivní a naopak. Dále nově příchozí a odchozí účastníci z konference stále způsobují změny ve vyjednávání popisů relací neboť ty je třeba upravovat, když je na straně serveru nad spojením volána metoda `addTrack` a `removeTrack`.

6.3 Třída Handler

Objekty `_sendTransport` a `_recvTransport` nezapouzdřují objekt `RTCPeerConnection` přímo, ale ve třídě `Handler`. Ta zajišťuje překlenutí implementace WebRTC různých odlišností prohlížečů. Při analýze kódu bylo zvažováno, zda-li je možné využít současné implementace realizující mediální komunikaci mezi serverem a klientem případně s určitými úpravami ke komunikaci dvou klientů vzájemně. Tím by se usnadnila otázka implememntace P2P a bylo by využito kódů ve smyslu tohoto řešení. Bohužel to ale prakticky možné není, neboť jeden z klientů by musel de facto kopírovat chování serveru a muselo by být přeprecováno veliké množství kódu s nejistým výsledkem.

6.4 Producer a Consumer

Mediasoup aplikace vytváří nad každou mediální stopou obálku vlastní třídou. Všechny `MediaStreamTrack` objekty odesílané na server jsou jednotlivě zapouzdřeny ve třídě `Producer`. Odtud jsou dále směrovány jednotlivým klientům konference, které pro ně, příchozí mediální stopy, vytváří objekty `Consumer`.

6.5 Inicializace klientské části

Klientský kód je iniciován funkcí `run()` souboru `index.jsx` v adresáři `app`. Tato funkce nejprve zpracuje parametry v URL adrese. Kromě dalších klíčový parametr celé konfrence představuje její identifikátor, který je reprezentován alfanumerickým textovým řetězcem. Pokud není definován, aplikace vytvoří náhodný, čímž

následně způsobí vytvoření nové místnosti též na serveru. Identifikátor daného klienta je náhodně vytvářen vždy. Volitelným parametrem může být i zobrazované jméno uživatele.

Po sběru všech informací je vytvořena instance třídy `RoomClient`. Ta představuje stěžejní třídu klientské části. Během vykreslování reactových komponent se již počítá s vytvořenou instancí této třídy a je volána její funkce `join()`, jež iniciuje vytvoření websocketového spojení se vzdáleným serverem. To zajišťuje knihovna `protoo`. Jsou registrovány jednotlivé funkce obsluhující standardní události spojení. Dochází k registraci událostí tohoto spojení a při jeho otevření se následně spustí inicializace mediální komunikace.

Na počátku inicializace komunikace se serverem je vytvořena instance třídy `Device` balíčku `mediasoup-client`. Mezi serverem a klientem dojde k výměně parametrů pro zřízení mediální komunikace se serverem. Vyměněny jsou například ICE parametry, ICE kandidáti a parametry pro DTLS šifrované komunikace a další. Dále jsou vytvořeny prostřednictvím instance třídy `Device` objekty pro příjem a odesílání mediální komunikace z kapitoly č. 6.2 Mediální přenos a RTCPeerConnection. Po zřízení obousměrného spojení je klient připojen do místnosti a následně jsou přidáni ostatní klienti místnosti. Ti jsou udržováni ve stavu aplikace v objektu `peers` viz. ukázka: `store.getState().peers`. Následně jsou načtena lokální média a přiřazena odpovídajícímu transportu v proměnné `_sendTransport`, odkud jsou přeposílána na server a odtud dalším klientům v místnosti.

7 Využití Mediasoup aplikace na dlouhé vzdálenosti

Zvažovanou otázkou optimalizace také byla vzdálenost mezi klienty a způsobená latence. Mezi koncovými body samozřejmě nezáleží na vzdálenosti jako takové nýbrž zejména na technologii přenosu, množství uzlů na trase, použitých platformách, zatížení sítě apod. Nicméně obecně streamování při online konferenci z druhé strany země se patrně na kvalitě konference projeví.

Představme si konferenci dvou účastníků. Oba dva se nacházejí v České republice (případně kdekoli ve Střední Evropě) – relativně blízko u sebe. Přenosový SFU server s obslužnou aplikací se bude nacházet ve Spojených státech amerických v Chicagu. Média jednoho účastníka tak musí putovat přes celý svět skrze tento server a poté zase zpět do Evropy. Taktéž i média druhého klienta.

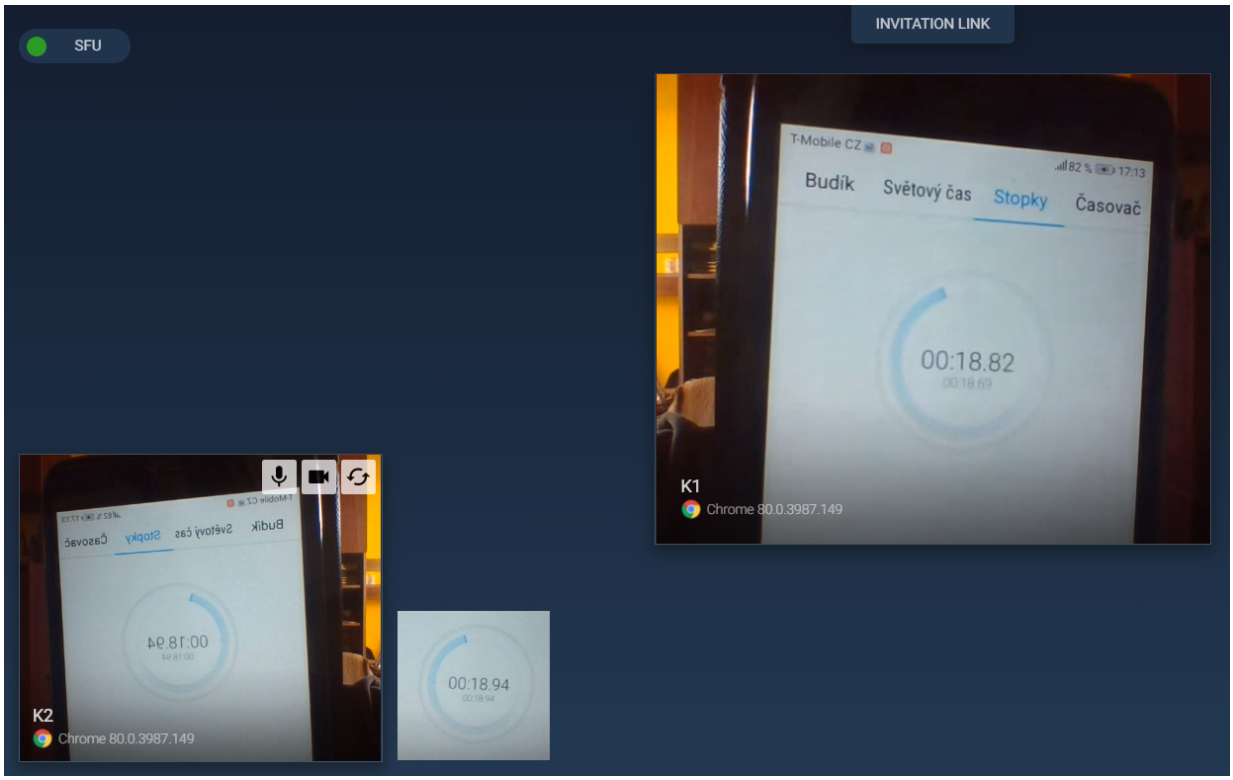
V tomto případě tak dochází zbytečně k dvojnásobně dlouhému cestování paketů přes internet z důvodu přítomnosti SFU serveru. Server je špatně umístěn resp. další možností by bylo zajistit síť serverů v předem stanovených lokalitách, které by obsluhovaly uživatele v dosahu. Byť tato forma pokrytí představuje technicky přínosné řešení, je třeba nejprve zjistit, v jakých mezích se skutečně pohybuje zpoždění, je-li třeba v našem případu užití přistoupit k dalším opatřením v této oblasti.

Obrázek 21: Účastníci konference připojováni z Evropy přes server v Chicagu

Pro otestování takové latence byla objednána a zřízena VPS v Chicagu prostřednictvím firmy VPSSERVER [56]. Odtud byla spuštěna konference ze zařízení, které se vyskytovalo v Českých Budějovicích. V případě serveru se jednalo se o platformu Debian 9.5 x64, na které byla zprovozněna prostřednictvím Node balíčků aplikace Mediasoup. Video bylo nastaveno na rozlišení 1280x720 pixelů. Obrázek č. 21 [57] je čistě ilustrací k tomuto případu, neboť koncové body A a B se viditelně nenacházejí na stejném místě.

Otázkou bylo, jak provést měření jako takové. Bylo by jistě možné hledat řešení tohoto problému v oblasti implemetnace měřící logiky a sběru dostupných dat. Nicméně nabízí se i další varianta. Přenášet číselné vyjádření času v obrazu videa a porovnat tak snímky lokálního videa a přijímaného skrze vzdálený server pro

stejný okamžik, jak uvádí zdroj [58]. Rozdíl čísel na snímcích nám tak ukáže skutečnou latenci. Toto měření lze provést více způsoby. Jednak můžeme jako odesílaný stream použít již připravené video, jež bude složené ze sekvence snímků měnících číselnou hodnotu, tak jak se mění čas či jiná varianta tohoto způsobu, nebo využít externí zdroj časomíry jako například stopky jako aplikace mobilního telefonu. Pro porovnání pak stačí udělat tzv. ”print screen”, na kterém jsou hodnoty zaznamenány. V případě lokálních médií je obraz osově převrácený.



Obrázek 22: Měření zpoždění SFU komunikace

Konference byly uskutečněny v prohlížeči Google Chrome ve dvou záložkách – tak bylo dosaženo streamování přes oceán a zpět při použití jediného zařízení (a mimo měření i jiných). Celé testování bylo několikrát opakováno s obdobnými výsledky. Kvalita médií byla subjektivně na vysoké úrovni s hodnotami sedm a osm – dle hodnot ACR škály viz. kapitola č. 9.1 Způsob měření kvality videa. Média se nezasekávala ani netrpěla jinými nedostatky. Pozorovatel měl možnost subjektivně porovnávat lokální média a média zasílaná. Opticky bylo zpoždění lokálních médií vůči přijímaným pozorovatelné, ale nebylo rušivé. Byly měřeny hodnoty v rámci stovek miliskend s hodnotami nejčastěji okolo 500 ms. Měřený rozptyl se pohyboval v mezích od 450 - 650 ms. Vzhledem k tomu, že proběhlé testování nepoukázalo na zásadní problémy při streamování ani v tomto speciálním případě a implementace optimalizace zpoždění by byla velmi náročná, práce se bude dále soustředit na optimalizaci zejména pomocí P2P techniky jako takové a další možnosti současné implementace WebRTC.

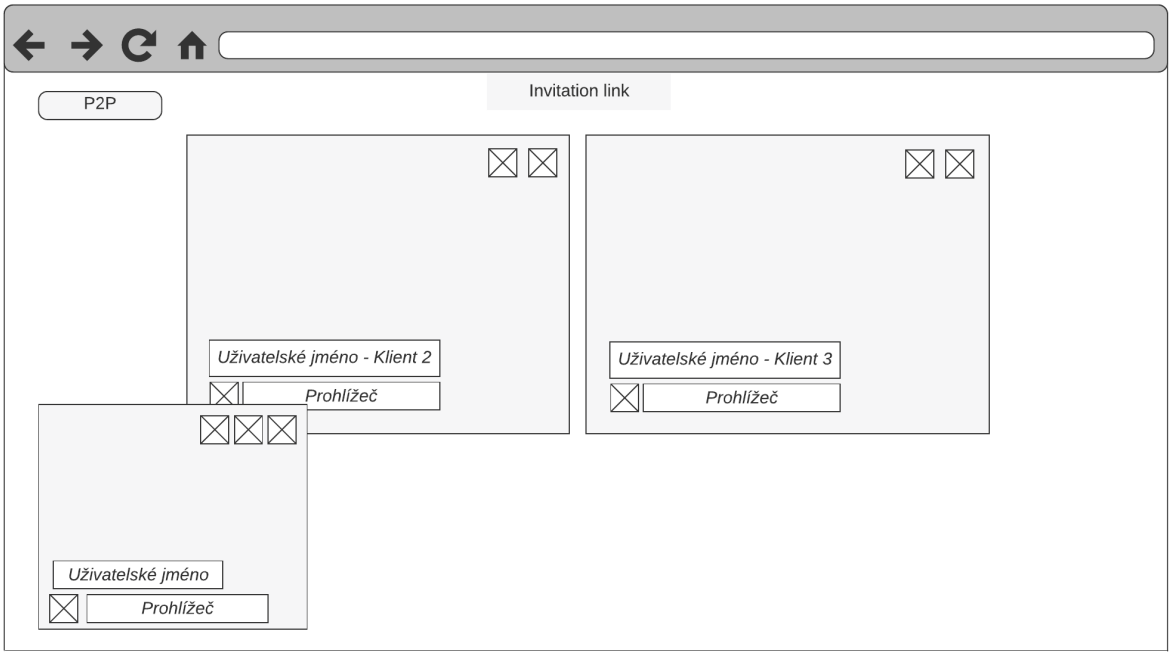
Příklad použití měření se stopkami na mobilním telefonu ukazuje obrázek č. 22. Zobrazuje měření, u kterého se SFU server nachází v Hluboké nad Vltavou a připojený klient v Jindřichově Hradci. Pozorované zpoždění činí kolem 120 ms.

8 Implementace P2P přístupu v rámci aplikace Mediasoup

8.1 Funkční požadavky na systém

Systém bude založený na audiovizuální komunikaci s SFU a P2P přístupy a bude rozšiřovat již existující aplikaci Mediasoup, která byla vyhodnocena jako vhodný základ pro další práci. Koncoví uživatelé připojovaní P2P technikou budou v systému zobrazováni vizuálně stejným způsobem jako ti, kteří jsou připojování SFU přístupem. Systém bude umožňovat spravovat vlastní klientská média tj, zakázat a povolit jejich přenos v průběhu relace. Dále umožní přepnutí používané kamery dostupné v rámci prohlížeče. Každý ze vzdálených klientů bude při zakázání přenosu videa či audia zobrazovat informační ikonu. Ty budou zobrazovány v horní části zobrazovaného videa.

Celý projekt je realizován v anglickém jazyce. Tudíž i texty a notifikační hlášení jsou taktéž v anglickém jazyce. Takto zůstanou i v upravované aplikaci. Na obrázku č. 23 vidíme navrhované rozvržení místnosti, které vychází z původního systému, ale představuje jednotnou formu zorbazování pro obě techniky. Média lokálního uživatele zůstávají v levé spodní části. U každého klienta bude zobrazováno vlastní jméno a ikona a verze prohlížeče. Připojovaní klienti budou v samostatných panelech v centrální části. V levém horním rohu bude indikace současného módu. V horní části na středu bude jako v původním systému možnost zkopírovat současný link do schránky. Notifikace se zobrazují nezměněným způsobem v pravém dolním rohu místnosti.



Obrázek 23: Návrh rozložení částí v upravovaném systému

8.2 Klientská část

Klíčová je hlavní klientská třída `RoomClient` v adresáři `app/lib`. Z ní je volána metoda `join()`, jež iniciuje jak websocketovou tak v návaznosti i mediální komunikaci. Tato metoda je volána z reactové komponenty

vykreslující místnost v obligádní metodě `componentDidMount()`. Jako výchozí mód je nastavováno P2P. Po připojení k serveru je tedy nejprve kontrolován a případně změněn, dle aktuální hodnoty módu v instanci místnosti na serveru, jež je pochopitelně autoritou. Indikace současného módu je taktéž tlačítkem, jež v závislosti na existenci parametru 'm' jako moderátor s hodnotou jedna v URL adrese umožňuje změnu tohoto módu. Změna je chápána jako požadavek zasílaný serveru, jež centrálně iniciuje tuto změnu u všech připojených klientů a to pomocí zprávy typu 'systemMode'. Spuštění klientské aplikace je možné z adresáře `app` a to příkazem `npm start`.

8.2.1 Reprezentace koncových bodů – třída P2PPoint

Každý jednotlivý koncový bod P2P komunikace je reprezentován třídou `P2PPoint`. Jeho data se vztahují k opačné straně tj. druhému klientovi v páru komunikace. Tato třída uchovává instanci `RTCPeerConnection` objektu. Výčet dalších uchovávaných položek je v seznamu viz. níže. Důležitou otázkou bylo, jak implementovat podporu WebRTC, neboť v rámci prohlížečů a jejich verzí se implementace jednotlivých API metod liší. Bylo přistoupeno k využití knihovny WebRTC Adapter, jež jej sjednotí a to verzi 7.3. Nainstalována byla konzolovým příkazem `npm install webrtc-adapter`. Třída také disponuje přístupem ke společnému reduxovému uložišti, a proto implementuje statickou metodu `init` pro přijetí a nastavení tohoto objektu.

Dědí od třídy `EventEmitter`, což jí umožňuje vyvolávat události, jejichž obsloužení je zajištěno vyšší vrstvou tj. stěžejní třídou `RoomClient`, která objekty `P2PPoint` vytváří. Díky tomuto mohou být například odstíněny některé závislosti. Například pokud `P2PPoint` potřebuje odeslat požadavek druhému koncovému bodu, je volána událost 'request' takto: `this.emit('request', { method, data });`. Parametr `method` určuje typ zprávy. Mechanizmus odeslání samotné zprávy tak nemusí být implementován touto třídou. Této události je využito při odesílání popisu mediální nabídky a odpovědi, ICE kandidátů (nebo při odesílání údajů z měření).

- Unikátní identifikátor klienta v podobě textového řetězce koncového bodu (shodného pro obě techniky po celou dobu existence místnosti)
- Zobrazované jméno je v aplikaci generované náhodně ze slovníku (převzato z aplikace Mediasoup)
- Dvě Boolean hodnoty pro indikaci je-li aktuálně odesíláno audio a video druhého koncového bodu.
- Informace o typu a verzi prohlížeče druhého koncového bodu.
- Číslo iniciátora určuje (dle jeho velikosti) kdo z páru koncových bodů zahájí iniciaci zřizování jako první.
- Boolean hodnota indikující, je-li vzdálený klient iniciátorem mediální komunikace či nikoli.
- `MediaStreamTrack` objekty médií vzdálených koncových bodů.
- Počet pokusů o navázání mediální komunikace v případě jeho pádu

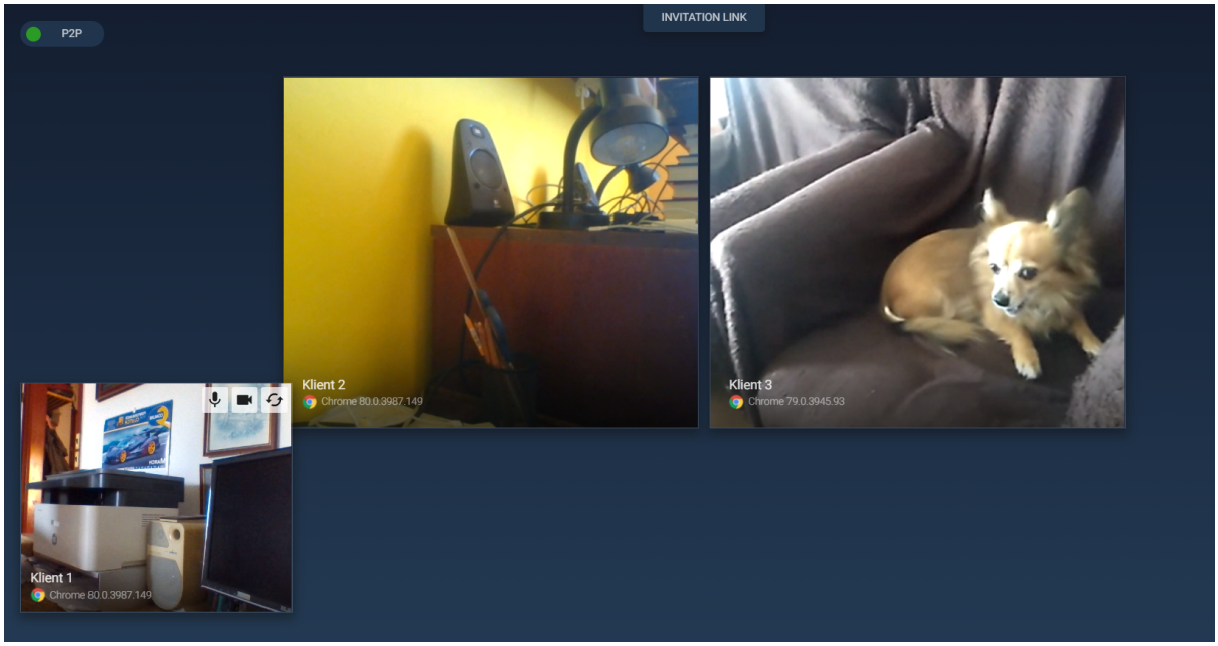
Při inicializaci objektu dochází k nastavení proměnných zaslaných prostřednictvím serveru mimo hodnoty určující, jedná-li se o iniciátora mediální komunikace a média druhé strany, která ještě nemohou být v tomto okamžiku přenášena. Nová instance se vytváří při příchodu resp. při přidání nového klienta do seznamu připojených

danou technikou v konferenční místnosti. Je třeba implementovat logiku jak z pohledu nového klienta, kterému je zaslán aktuální seznam klientů s jejich daty, aby mohly být u tohoto klienta vytvořeny instance třídy `P2PPoint`, tak zajistit vytvoření nové instance bodu u každého stávajícího účastníka v již existující místnosti.

Konstruktor iniciuje samotný objekt `RTCPeerConnection`. Ten vyžaduje připojení lokálních médií, která jsou již načtena a dostupná prostřednictvím proměnné z datového uložiště. S nastavovanou video stopou je konfigurováno rozlišení odesílaného videa viz. ukázka níže. V případě testování je přednastaven kodek použitý při testování.

```
const local = store.getState().localPoint;
const videoTrack = local.videoTrack;
if (videoTrack) {
    this.videoSender = this.pc.addTrack(videoTrack);
    const scalingFactor = this._calculateScaleResolution();
    this._setVideoSenderParams(scalingFactor);
}
```

Poslední akce konstruktoru je registrace funkcí zachytávajících události spojení. Mezi využité patří jednoznačně generování nových ICE kandidátů lokálním objektem pro vyjednání síťového spojení tj. `onicecandidate`. Událost vyžaduje implementaci zaslání těchto dat druhému klientovi. Na konci procesu zřizování komunikace je `RTCPeerConnection` objektem volána událost `ontrack`, která zajišťuje obsluhu nově přijatých médií druhé strany. Zde jsou média přiřazena k `P2PPoint` objektu a je iniciováno překreslení reactové komponenty s médii. Poslední registrovanou událostí je `onconnectionstatechange`, jež poukazuje na aktuální stav z pohledu vyjednaného síťového spojení.



Obrázek 24: Konferenční místnost v P2P módu

8.2.2 Možné kolize při zahájení P2P komunikace

Spojení `RTCPeerConnection` objektů jsou vytvářena při spuštění místnosti tj. při načítání webové stránky při příchodu nového účastníka konference a při přepínání z módu SFU do P2P. Jak uvádí kapitola č. 2 o WebRTC technologii, jeden z bodů spojení vytváří relační nabídku a druhý vytváří odpověď v podobě SDP popisů nutných k zahájení relace. Při práci s párem dvou `RTCPeerConnection` objektů je třeba provádět jednotlivé operace nastavování těchto zpráv, ICE kandidátů apod. ve správných stavech a nesmí dojít ke kolizním situacím. Uvažme situaci přepínání módu do P2P. V takovém případě jsou totiž všichni v místnosti připojováni prakticky v totožný okamžik. Vzhledem k tomu, že zpracování kódu v javascriptu dochází asynchronně, zahájení mediální komunikace z pohledu jednoho nově připojovaného bodu nemůže začínat automaticky odesláním 'offer', protože by mohlo docházet ke kolizním stavům. Nedošlo by k odeslání nabídky a po jejím zpracování druhou stranou následné vygenerování odpovědi, ale k zasláním dvou nabídek, což by znemožnilo komunikaci. Klíčovým aspektem ve dvojici připojovaných klientů je určit iniciátora komunikace. Musíme uvážit, že každý z koncových bodů implementuje tutéž logiku. Pokud se oba pokusí nějakým mechanizmem výměn zpráv nastavit sebe či druhou stranu jako iniciátora, dojde ke zmatení, protože oba provádí totéž. Musí zde být určena autorita, která rozhodne. Tou autoritou může být v našem případě náhodně generované číslo či upravený čas v milisekundách prvotního připojení koncového bodu do místnosti. Spouštěný kód koncového bodu je současně spouštěn i u ostatních bodů. Probíhá současně ve stejném čase i na jiných strojích. Každý klient odesílá na server svojí časovou známku a ten ji distribuuje opačnému konci. Iniciátor komunikace je jednoduše ten, kdo z dvojice má větší číslo. Objekt `P2PPoint` disponuje vlastnostmi `isInitiator` a `initiatorNumber`, které jsou k tomuto využity.

8.3 STUN server

Aplikace vyžaduje ke svému chodu běžící STUN server. Použitým řešením je tzv. Coturn [10]. Vzhledem k tomu, že využitým serverem je Debian, je možné aplikaci stáhnout jako běžný balíček prostřednictvím příkazu `sudo apt install coturn`. Pro testovací účely lze také využít některých ze serverů, které provozuje společnost Google. Konfiguraci lze nalézt v souboru `/etc/turnserver.conf`. Zde lze provést detailní nastavení, která umožňují spustit pouze STUN, neboť obsahuje i TURN implementaci. Toto nastavení je provedeno direktivou `stun-only`. Nastaveným portem je 5349. STUN server je pak v klientské aplikaci konfigurován jako vlastnost objektu, předávaného konstruktoru `RTCPeerConnection`. Jako parametr pro spuštění aplikace je vyžadován název domény, na kterém bude server spuštěn. Spuštění aplikace lze reliazovat následujícím příkazem. `sudo turnserver -r <doména>`.

8.4 Serverová část

Serverová část zajišťuje samotný mediální přenos a websocketovou komunikaci mezi klienty resp. mezi klienty a serverem. Je tvořena javascriptovými soubory s hlavním souborem `server.js`. Oproti původní implementaci jsou přidány další třídy a některé části jsou upraveny. Zejména je přidána obsluha P2P komunikace.

Po spuštění hlavního souboru a jeho funkce `run` dochází k vytvoření tzv. `Worker` objektů z balíčku `mediasoup`, jež reperezentuje SFU jednotku v podobě Node balíčku. Na základě údaje z konfigurace je vytvořen příslušný počet těchto `Workerů`. Každý zajišťuje samotnou mediální komunikaci a to v kombinaci s

klientským balíčkem, jež zajišťuje realizaci tohoto spojení u koncových bodů. Dále se spustí webový server pro websocketovou komunikaci. Při události `connectionrequest` se načítá buď existující instance reprezentující místnost z mapy místností, nebo se vytváří nová. Toť původní chování, které bude zachováno. Místnost reprezentovala třída `Room`, která nesla jak své informace, tak zajišťovala potřebnou logiku. Po uvážení bude však tento koncept pozměněn následovně. Klíčovou třídou bude `ConferenceRoom`, která bude reprezentovat zejména datovou entitu. Ta bude existovat po dobu připojení alespoň jediného klienta v místnosti. Jejími atributy budou identifikátor, aktuální mód a třídy `P2P` a `SFU`, jež zajišťují zpracování websocketové komunikace pro každý jednotlivý přístup. Společnou třídou, od které tyto dvě třídy budou dědit je třída tvořící komunikační základ `CommBase`. Ta obsahuje metody společné oběma přístupům.

Obě třídy mají vlastní implementaci metody `handleProtooConnection` a `_handleProtooRequest`. První zajišťuje vytvoření objektu `Peer` knihovny `protoo` či zahození spojení v případě již existujícího duplicitního. Jsou v ní zejména registrovány události `request` na příchozí požadavek a `close` při uzavírání spojení s daným klientem. Při zpracování požadavku je dále volána druhá zmiňovaná metoda, která obsahuje všechny typy zpráv a jejich zpracování. Jak třída `P2P` tak `SFU` vznikají tak, aby logicky rozdělily příslušnou komunikaci. SFU část podle svého předchůdce, tj. třídy `Room`, zajišťuje vytváření tříd `Consumer`, obálek nad mediálními stopami, jež mohou být instancovány na základě zaslaných informací od klienta (jež tato média produkuje jako `Producer` třídy) a zasílány ostatním koncovým bodům místnosti.

Serverovou aplikaci lze spustit v adresáři `server` příkazem `node server.js`. Její součástí je i soubor `config.js`, jež představuje konfiguraci i pro klientskou část. Mezi klíčové parametry patří nastavení domény či certifikátu pro šifrovaný přenos.

9 Porovnání P2P a SFU přístupu

K porovnání obou přístupů potřebujeme mít k dispozici údaje vypovídající jednak o uživatelském dojmu a kvalitě streamovaných médií, tak o parametrech přenosu. V této kapitole je předložen souhrn poznatků z testování a měření.

9.1 Způsob měření kvality videa

V oblasti klasifikace kvality videa a jeho streamování se setkáváme s dvěma základními přístupy tj. subjektivní a objektivní způsob hodnocení dle [85]. Základní charakteristikou subjektivních metod je hodnocení provedené člověkem a to přímým pozorováním skutečných výstupů. Naopak objektivní měření udává své hodnocení na základě charakteristik hodnot technických parametrů uskutečněného spojení. Stěžejní výhodou je zde možnost automatizace těchto testů vzhledem ke komplikovanosti realizace testování se skutečnými uživateli nejen z časových důvodů. Nevýhodou může být však náročnost implementace či v některých případech vzniklá nepřesnost mezi výsledky těchto testů a skutečným dojmem pozorovatele.

V každé z oblastí existuje řada standardizovaných metod pro hodnocení kvality videa. Pro realizaci testování v této práci byla vybrána metoda subjektivního hodnocení ACR (Absolute Category Rating) dle doporučujícího dokumentu Subjective video quality assessment methods for multimedia applications [86] organizace ITU zabývající se telekomunikacemi. Základní škála klasifikace kvality popisuje výčet hodnot 1-5. Nejlepší hodnocení je oceněno pěti body a představuje tak vynikající přenos. Naopak nejhoršího hodnocení dosahuje to, které získá pouze jeden bod.

Zejména pro posouzení videa s nízkou bitovou rychlostí je často nutné použít hodnotící stupnici s více než pěti stupni. Vhodnou stupnicí pro tento účel je devítistupňová stupnice, kde je pět verbálně definovaných kategorií kvality. [86] Tato škála bude využita při testování konferenčního systému.

- 9 – Excellent
- 8
- 7 – Good
- 6
- 5 – Fair
- 4
- 3 – Poor
- 2
- 1 – Bad

9.2 Možnosti měření mediálního přenosu

9.2.1 Statistiky přenosu RTCPeerConnection objektu

Obecně přenos multimédií je v javascriptu možný prostřednictvím objektu `RTCPeerConnection`. Ten zajišťuje relaci s dalším koncovým bodem. Disponuje metodou `getStats()`, která vrací objekt typu `Promise` a při úspěšném načtení dat poskytuje informace o přenosu uskutečněném k okamžiku volání této metody od počátku existence spojení. Statistiky jako takové jsou reprezentovány objektem `RTCStatsReport`, jež lze procházet cyklem pomocí metody `forEach()`. V následující ukázce kódu je znázorněn postup, jakým lze přistupovat k jednotlivým záznamům statistik. Každý záznam je určitého typu a definuje tak svůj obsah.

```
/**
 * @param {RTCPeerConnection} point
 */
async logStats(point)
{
        const stats = await point.getStats();
        stats.forEach(report => {
                console.log(report.type, report);
        });
}
```

9.2.2 Některé parametry přenosu ze statistik

Jednotlivé objekty představující statistiku určitého výčtu informací a každý je určen svým typem. Objekt typu `transport` nabízí informace o množství odeslaných a přijatých bytů na tomto spojení k aktuálnímu okamžiku nad všemi médii (objekty `MediaStreamTrack`) přiřazenými k tomuto spojení metodou `addTrack()`. Hodnoty jsou k dispozici pod klíči `bytesSent` a `bytesReceived`.

V případě videa hraje důležitou roli jeho rozlišení, které přímo ovlivňuje množství přenášených dat. WebRTC technologie implementovaná ve webových prohlížečích automaticky mění parametry audio a video relace v průběhu mediálního spojení v závislosti na možnostech sítě a koncových zařízeních. Dochází tak například právě k automatickým změnám velikosti rozlišení videa, aby plynulost streamování byla zachována se sníženou kvalitou i při horších podmínkách sítě. Jeho aktuální šířku a výšku v pixelech lze zjistit z objektu typu `track` statistiky pod klíči `frameWidth` a `frameHeight`. Objektů `track` statistika obsahuje obvykle více – lokální i z druhého koncového bodu, a proto je nutné je vzájemně odlišit. Tento objekt proto disponuje klíčem `remoteSource`, který určuje jedná-li se o mediální stopu lokálního zařízení, která je přiřazena k danému spojení či je daná stopa přenášena z druhého koncového bodu.

Aktuální rozlišení však lze také získat přímo z přijímaného `MediaStreamTrack` objektu druhého bodu, jak ukazuje následující ukázka.

```
// track {MediaStreamTrack} obsahující video
```

```
const settings = track.getSettings();
// settings.width - šířka přenášeného videa
// settings.height - výška přenášeného videa
```

9.2.3 Konfigurace kodeků pomocí RTCRtpTransceiver

Chceme-li relevantně porovnat komunikaci P2P a SFU propojení, necháme aplikaci nastavit v obou případech stejné kodeky jak pro audio tak i video. Objekt `RTCRtpSender` umožňuje nastavení způsobu odesílání dat konkrétního `MediaStreamTrack` objektu. Podobně `RTCRtpReceiver` představuje obálku nad příchozími audio či video stopami druhého koncového bodu. Jak uvádí dokumentace API tohoto objektu na MDN serveru [89], oba tvoří společnou dvojici daného typu média, jež je zapouzdřena objektem `RTCRtpTransceiver`. Pomocí tohoto objektu lze nastavit kodeky, které může mediální relace využít – nutně ale ještě před jejím vznikem. Po vyjednání SDP popisů již kodeky nelze měnit pouhým přenastavením tohoto objektu, ale je třeba opětovně nastavit danou relaci a vyměnit nové SPD popisy.

```
//  point {RTCPeerConnection}
const transceivers = point.getTransceivers();
for (const index in transceivers) {
        const tr = transceivers[index];
        if (tr.sender.track.kind === "video") {
                tr.setCodecPreferences(videoCodecs);
        }
        if (tr.sender.track.kind === "audio") {
                tr.setCodecPreferences(audioCodecs);
        }
}
```

Otázkou je, jakým způsobem zjistíme podporované kodeky, byť ve většině možných případů budou podporovány kodeky VP8 či h.264 pro video a kodek Opus pro audio a právě tyto jsou také nastavovány částí aplikace Mediasoup, která zřizuje SFU komunikaci, a proto budou konkrétně tyto (VP8 a Opus) nastaveny též při testování P2P spojení. `RTCRtpSender` disponuje statickou metodou `setCapabilities()`, jejíž návratovou hodnotou je objekt zahrnující i výčet podporovaných kodeků.

```
// audio {Object[]} - pole objektů jednotlivých kodeků
const audio = RTCRtpSender.getCapabilities("audio").codecs;
```

9.2.4 Konfigurace RTCRtpSender objektů pro měření

Chceme-li porovnat komunikaci P2P a SFU propojení, je nutné, abychom v obou případech měli stejně konfigurovaný způsob odesílání mediálních stop. Zde je podstatné zmínit, že zde nehovoříme o změně vyjednaných popisů relace SDP, jež určují například kodeky jako takové. `RTCRtpSender` je vrácen jako návratová hodnota metody přidání mediální stopy k objektu `RTCPeerConnection` a lze s ním dále pracovat.

```
//  point {RTCPeerConnection}
const videoSender = point.addTrack(videoTrack);

// ...

// Získání pole všech RTCRtpSender objektů
const senders = point.getSenders();
```

Volání metody `getParameters()` příslušného `RTCRtpSender` vrací objekt s klíčem encoding (také objekt). Ten obsahuje parametry aplikované pro odesílání dané mediální stopy. Ty lze upravit a následně překonfigurovat kdykoli v průběhu relace. Změny parametrů jsou možné v rámci mezí vyjednaných SDP zprávami. Před změnou parametrů je nutné jejich načtení právě metodou `getParameters()`, i kdybychom již měli tento objekt připravený v jiné proměnné, což vytváří z nastavování parametrů určitou transakci. Nicméně je třeba reflektovat, zda-li daná verze prohlížeče toto API podporuje, neboť se nejedná o prvotní způsob implementace WebRTC v prohlížečích.

V rámci našeho spojení nastavíme profil kódování odesílaných dat tj. tzv. maximální bitrate `maxBitrate` neboli horní hranici pro množství odesílaných bitů média za vteřinu. Tento parametr lze nastavit jak pro audio tak pro video `RTCRtpSender`. Dalším podstatným parametrem je `scaleResolutionDownBy`, jež určuje poměr v jakém má být sníženo rozlišení a tím pádem datová velikost odesílaného videa. Hodnota 1 představuje poměr původní tj. nesnížené rozlišení.

Při vývoji a opakovaném testování se však ukázalo, že byť je hodnota `maxBitrate` aplikována, neodpovídá zcela zadané hodnotě, ale reálná implementace funguje spíše orientačně, často totiž zadanou hodnotu reálná měření překračují dle množství odeslaných bytů `bytesSent` podle statistiky spojení. Současně se ale jednoznačně prokázalo, že změna konfigurace byla okamžitě aplikována a změny se viditelně projevily.

```
/**
 * Přenastavení parametrů
 * @param {RTCRtpSender} sender
 * @param {RTCPeerConnection} point
 * @param {Number} i - index
 */
async setVideoSenderParams(sender, point, i = 0)
{
        const params = sender.getParameters();
        params.encodings[i].maxBitrate = 1500000;
        params.encodings[i].scaleResolutionDownBy = 1;
        await this.sender.setParameters(params);
}
```

Cílem úprav těchto parametrů může být mimo testování řízeně korigovat zátěž koncových bodů – nepřetížit odesílatele ani stranu přijímající. Z této a předešlé kapitoly také vyplývá, že objekt `MediaStreamTrack` s

audiem či videem lze nastavit danému `RTCPeerConnection` opakovaně a s jinými parametry pro odesílání, protože ty určuje právě `RTCRtpSender` nikoli mediální stopa jako taková nebo SDP popis. Tohoto poznatku bude dále využito. V aplikaci Mediasoup jsou definovány pro SFU komunikaci 3 profily, jež slouží pro nastavení `RTCRtpSender` objektů. Pro porovnání přístupů P2P a SFU musíme nastavit v době měření totožný profil.

```
// Konfigurační profily využívané SFU částí
const VIDEO_SIMULCAST_ENCODINGS =
[
        { maxBitrate: 180000, scaleResolutionDownBy: 4 },
        { maxBitrate: 360000, scaleResolutionDownBy: 2 },
        { maxBitrate: 1500000, scaleResolutionDownBy: 1 }
];

// Parametry nastavované pro měření statistik
const STATS_ENCODINGS =
{
        maxBitrate: 1500000,
        scaleResolutionDownBy: 1
};
```

9.3 Implementace měření statistik

Cílem je implementovat mechanizmus, který na povel jednoho z klientů spustí v rámci všech klientů konferenční místnosti měření statistiky ve stejný čas, sesbírá naměřená data a zašle je iniciátorovi měření. Zde je tedy třeba rozdělit realizaci měření v rámci jednoho klienta a obslužný synchronizační mechanizmus.

Implementace samotného měření představuje proces, na jehož počátku načteme statistiku `RTCPeerConnection` objektu. Uložíme některá její data a spustíme časovač s intervalem měření. Po uplynutí zadané doby je provedeno sekundární měření a uložení hodnot. Finální sumarizace údajů se různí podle zvolené techniky, protože jak P2P tak SFU jsou v aplikaci implementovány vlastním, odlišným způsobem.

Metoda pro měření statistik P2P se bude jmenovat `recordP2PStats` a v případě SFU to bude `recordSFUStats`. Obě metody budou mít 2 parametry, které budou stejné pro obě: `statsInterval`, jenž je číslem vyjadřujícím počet milisekund měření – výchozím nastavením bude jedna minuta. Parametr `all` bude typu `Boolean` a bude určovat, jedná-li se o měření lokální či měření iniciované jiným klientem a jedná se tak o kompletní testování všech účastníků.

Spuštění měření jedné z výše uvedených metod bude představovat proces, ve kterém budou nejprve nastaveny parametry `RTCRtpSender` objektů pro totožné podmínky měření (možný výběr kodeků je nastaven před inicializací spojení a vyjednán procesem Offer/Answer SDP popisů). Konečný výpočet odeslaných a přijatých bytů je získán odečtením hodnot statistiky měřené později od té předchozí, protože statistika `RTCPeerConnection` obsahuje vždy data platná k okamžiku měření od počátku relace.

Základem pro měření obou technik bude vytvoření objektu `result` pro sběr výsledků.

```
// objekt pro výsledky měření
const result = {
        total: {
                bytesSent: 0,
                bytesReceived: 0
        }
};
```

9.3.1 Implementace P2P měření v klientské aplikaci

Ve třídě `RoomClient` je definovaná metoda `recordP2PStats`, která realizuje měření. Z globálního reduxového uložiště `store` jsou načteny objekty reprezentující vzdálené účastníky konference. Proměnná `points`, která je objektem, obsahuje instance třídy `P2PPoint` uložených pod jedinečnými identifikátory účastníků.

Metoda `recordStats` třídy `P2PPoint` z následující ukázky jednotlivě spouští privátní metodu pro získání statistik každého z `RTCPeerConnection` objektů. Prvním parametrem této metody je již zmiňovaný objekt `result` pro výsledky. Získané informace o každém spojení budou uchovány v objektu, jenž bude do `result` uložen klíčem, kterým je identifikátor vzdáleného klienta. `result` v sekci `total` také provádí inkrementaci výsledků. Tato sekce obsahuje v případě P2P navíc ještě proměnnou `points`, která je taktéž postupně inkrementována a určuje, kolik spojení již měření provedlo. V případě, že byly provedeny již všechny a předávaný parametr `all` je hodnoty `true`, dojde po ukončení měření k odeslání dat na server. Parametr `statsInterval` určuje dobu mezi počátečním a koncovým měřením v milisekundách.

```
const points = store.getState().remotePoints;
for (const id in points) {
        points[id].recordStats(result, statsInterval, complete);
}
```

V následující úkázce kódu můžeme vidět stěžejní část implementace měření, kterou realizuje funkce `setTimeout` s asynchronní callback funkcí metody `recordStats`. Mimo přijatých a odeslaných bytů jsou uchovávány i další parametry jako rozlišení přenášeného videa.

```
// počátek měření - načtení statistik
const start = await this._recordStats();
setTimeout(async () => {
        // druhé měření
        const end = await this._recordStats();

        // výpočet odeslaných a přijatých dat v MB
        const sent =
            (end.bytesSent - start.bytesSent) / 1000000;
        const received =
            (end.bytesReceived - start.bytesReceived) / 1000000;
```

```
            /* .. uložení do result objektu a odeslání na servrer .. */
    }, statsInterval);
```

Data měření jsou načítána metodou `_recordStats` třídy `P2PPoint`. Tato metoda již přímo načítá statistiky z `RTCPeerConnection` objektu a cyklem prochází jednotlivé datové objekty a řídící strukturou `switch` na základě typu objektu zpracovává požadované objekty. To jsou objekty týkající se přenosu a médií tedy `transport` a `track`. V ukázce níže vidíme uložení jednoho z objektů statistiky do výstupního objektu `data` jakožto návratové hodnoty metody.

```
if (report.kind === 'video' && report.remoteSource) {
        data.video = report;
}
```

9.3.2 Implementace SFU měření v klientské aplikaci

Ve třídě `RoomClient` je definovaná metoda `recordSFUStats`, která realizuje měření. Jak je zřejmé z kapitoly č. 6 Aplikace Mediasoup, aplikace Mediasoup využívá k odesílání médií na server objekt uložený pod vlastností `_sendTransport` této třídy, jež zapouzdřuje `RTCPeerConnection` objekt. Z něj je možné získat statistiky dvěma způsoby. Buď přímo ze zapouzdřeného objektu spojení, nebo pomocí jeho implementované metody `getStats()`, jež na spojení odkazuje. V případě příchozích médií je obdobně využito objektu `_recvTransport`.

Dále je třeba získat seznam všech `RTCRtpSender` objektů, jež obsahují video stopy. Každému jednotlivě jsou nastaveny parametry přenosu, jak je popsáno.

```
for (const index in videoSenders) {
    await this._setStatsSendersParams(videoSenders[index]);
}
```

Následuje samotné načtení statistik ze zmíněných objektů. Výsledky jsou uloženy do proměnných. Následně je spuštěn časovač, po jehož uplynutí dojde k výpočtu skutečně odesílaných a přijímaných dat obdobně jako je tomu v případě P2P varianty.

9.3.3 Implementace mechanizmu měření všech klientů

Při ladění programu je v globální proměnné `CLIENT` uchována instance třídy `RoomCLient`, která tvoří stěžejní část klientské aplikace. Voláním `CLIENT.recordCompleteStats()` spustí a provede měření statistik u všech připojených klientů ve stejném okamžiku.

Prohlížeč iniciátora měření odesílá na server websocketový požadavek ”fireStats”. Třída zpracovávající komunikační požadavky na serveru dle typu komunikace tj. buď objekt `P2P` nebo `SFU` v rámci třídy `ConferenceRoom` disponuje instanční proměnnou `_peerStatsCollector`, která je určena pro uchování identifikátoru websocketového spojení koncového bodu (sběratele statistik), který byl iniciátorem volání metody `recordCompleteStats()` a na závěr procesu obdrží naměřená data všech účastníků konference, jenž jsou jako objekt ukládána do globální proměnné prohlížeče `completeResult` a v podobě textového řetězce do

`completeResultStr` přičemž uložení je hlášeno zprávou v konzoli prohlížeče. V případě, že měření neprobíhá, hodnota je nastavena jako `NULL`. V daném okamžiku může probíhat v rámci dané konference pouze jediné měření.

Celý proces websocketové komunikace je následující. Zpráva typu "fireStats"o počátku měření je od iniciátora zaslána na server a ten odesílá dané místnosti broadcast stejného typu. Každý z klientů provede měření a zašle data serveru: typ "statsData". Ten po sběru dat všech účastníků finální podobu odesílá iniciátoru měření: typ "resultStats". Server tedy sleduje počet přijatých statistik od jednotlivých klientů. Klientská metoda zpracovávající přijatá data

```
_resultStatsVars(result)
{
        window.completeResult = result;
        logger.debug("RESULT COMPLETE: ", result);
}
```

9.4 Měření konkrétních relací dle ACR a WebRTC statistik

Pro každé měření byla stanovena doba jedné minuty. To je čas, po který bude probíhat jak hodnocení streamovaných médií uživateli dle škály hodnot ACR (Absolute Categry Rating viz. kapitola č. 9.1 Způsob měření kvality videa), tak případně sběr dat javascriptových statistik. Test bude vždy proveden pro každou techniku zvlášť. Účastník hodnotí bodovým ohodnocením celkového dojmu – tj. zejména plynulost, chybovost, rozpad obrazu, zpomalování či proměny rozlišení způsobené automatickou adaptací WebRTC na změny podmínek sítě a zařízení atp.

Hodnota jedna značí nejhorší možný přenos a hodnota devět značí nejlepší možný přenos. Číslo s vykřičníkem znamená, že po jistou dobu měření přenos probíhal v dané kvalitě, ale v průběhu došlo k jeho nenadálému přerušení. Číslem nula jsou značena ta spojení, jež se vůbec nepodařilo navázat. Každý připojený klient hodnotí každého. Uživatel samozřejmě nehodnotí sám sebe, a proto budou tato pole označena písmenem x. Řádky tabulky představují to, jak daný klient na řádku hodnotí ostatní klienty a sloupce vyjadřují to, co daný klient odesílá resp., jak ostatní hodnotí tohoto klienta.

9.4.1 Porovnání datové zátěže dvou a tří klientů

V prvním měření otestujeme datovou náročnost obou technik a to nejprve se dvěma připojenými klienty a posléze se třemi. Porovnání ukáže, kolik dat je mezi nimi vzájemně odesíláno a přijímáno. Pro oba testové případy konference probíhá na jediném zařízení lokálně v rámci záložek téhož prohlížeče, což zajišťuje snížení vlivů internetové sítě oproti běžné konferenci.

P2P	Klient 1	Klient 2
Uživatelské hodnocení	9	9
Rozlišení příchozího videa	1280x720	1280x720
Celkem odesláno (MB)	11,994333	11,940721
Celkem přijato (MB)	11,941898	11,996543
SFU		
Uživatelské hodnocení	9	9
Rozlišení příchozího videa	1280x720	1280x720
Celkem odesláno (MB)	10,660724	10,20418
Celkem přijato (MB)	10,133684	10,588562

Tabulka 1: *Konference 2 účastníků v Chrome 78.0 (Windows 10) na zařízení Dell Inspiron 15 5000*

P2P	Klient 1	Klient 2	Klient 3
Uživatelské hodnocení	8	8	8
Rozlišení příchozího videa	1280x720	1280x720	1280x720
Celkem odesláno (MB)	23,812571	24,008729	23,99411
Celkem přijato (MB)	24,043099	23,936283	23,824291
SFU			
Uživatelské hodnocení	8	8	8
Rozlišení příchozího videa	1280x720	1280x720	1280x720
Celkem odesláno (MB)	7,270817	7,866668	8,385286
Celkem přijato (MB)	16,125295	15,53522	15,042336

Tabulka 2: *Konference 3 účastníků v Chrome 78.0 (Windows 10) na zařízení Dell Inspiron 15 5000*

Přenos klientů byl bezproblémový a nevykazoval známky chyb, rozlišení zůstalo po celou dobu měření stejné. Jedinným negativním vlivem bylo občasné mírné a krátké zadrhnutí obrazu. Hodnocená kvalita médií dosahovala maximálních stupňů. Druhé měření v tabulce č. 2 ukázalo na markantní datovou zátěž při nasazení P2P oproti SFU technice.

9.4.2 Konference čtyř klientů přes internetovou síť

Cílem měření je ověřit chování P2P a SFU spojení. Vzhledem k tomu, že výsledky měření jsou zcela závislé na aktuálnách podmínkách sítě a okamžitém vytížení koncových zařízení, bude celý test ihned po jeho skončení opakován, abychom měli srovnání. Měření poukáže zejména na četnost zřízení komunikace oběma přístupy v dané sestavě, ověří, zda-li dosahují uspokojivé kvality a plynulého chodu a v neposlední řadě poukáže na datovou náročnost spojení při určité kvalitě.

P2P	Klient 1	Klient 2	Klient 3	Klient 4
Klient 1	x	0	0	4
Klient 2	0	x	0	5
Klient 3	0	0	x	4
Klient 4	3	4	2	x
SFU				
Klient 1	x	8	7	6
Klient 2	8	x	7	5
Klient 3	7	9	x	6
Klient 4	7	4	7	x

Tabulka 3: *Konference 4 účastníků*

P2P	Klient 1	Klient 2	Klient 3	Klient 4
Klient 1	x	0	8	6
Klient 2	0	x	0	0
Klient 3	8	0	x	6
Klient 4	8	0	7	x
SFU				
Klient 1	x	8	8	5
Klient 2	8	x	8	6
Klient 3	7	8	x	6
Klient 4	8	8	7	x

Tabulka 4: *Konference 4 účastníků – opakování měření*

Bylo realizováno měření čtyř klientů, jež se propojovali přes internetovou síť. V tomto měření probíhalo jak uživatelské hodnocení se škálou hodnocení ACR, tak měření na základě implementovaného mechanizmu. Hodnotitelé byli před zahájením testu seznámeni se způsobem hodnocení. Zúčastněnými byli klienti nacházející se v České republice konkrétně z Jihočeského a Středočeského kraje. Klienty jedna až čtyři tvořila v tomto pořadí přenosná zařízení: Dell Inspiron 15 5000 Series, Lenovo V130-15IKB, HP 15-ba062nc a Lenovo B50-80. Každý z klientů byl připojen prostřednictvím prohlížeče Google Chrome ve verzi 80.0 na platformě Windows 10. Společně nastavovaným kodekem byl VP8.Video bylo konfigurováno na rozlišení 640x480 pixelů.

Z tabulky č. 5 lze jednoznačně potvrdit již známý závěr, že přímé spojení nelze považovat za spolehlivou metodu. Z tabulek č. 3 a 4 je zjevné, že byť P2P spojení se nepodaří navázat vždy, k navázání komunikace s opačným koncovým bodem může dojít i později resp. při opakovaném měření ke zřízení dochází. Tímto příkladem je právě spojení klientů jedna a tři v P2P módu z těchto tabulek. Zejména pak spojení klienta dva bylo žádné či slabé oproti ostatním v obou případech v P2P. Je také třeba říci, že většina spojení, jež se nepodařilo navázat v prvním případě,

se nepodařilo navázat ani ve druhém případě.

Při testování P2P techniky v běžné lokální síti v průběhu vývoje k těmto výpadkům prakticky nedocházelo. Dále P2P v případech existence spojení se při opakovaném měření projevilo celkově jako plynulejší a bez výpadků. Podobně tomu bylo i s SFU přístupem, který ve druhém případě byl klienty hodnocen také pozitivněji. SFU přístup poukázal v porovnání s P2P na datovou úsporu přenosu při odesílání médií.

Průměrné uživatelské hodnocení v tabulce č. 5 je průměrná hodnota ze všech hodnocení celkového dojmu zasílaných médií klienta, jež posuzovali ostatní tj. hodnoty sloupců. Do této hodnoty nejsou započítávány nepovedené pokusy o spojení. Číslo za lomítkem vyjadřuje celkový počet spojení s ostatními klienty těchto médií.

P2P	Klient 1	Klient 2	Klient 3	Klient 4
1. prům. hodnocení / počet klientů	3 / 1	4 / 1	2 / 1	4,3 / 3
2. prům. hodnocení / počet klientů	8 / 2	0 / 0	7,5 / 2	6 / 2
1. celkem odesláno (MB)	12,024877	11,892023	9,883686	7,310765
2. celkem odesláno (MB)	18,355107	0	15,369232	12,749101
1. celkem přijato (MB)	2,423078	2,296516	2,50178	33,87703
2. celkem přijato (MB)	15,671887	0	16,009095	15,018335
SFU				
1. prům .uživatelské hodnocení	7,3 / 3	7 / 3	7 / 3	5,6 / 3
2. prům .uživatelské hodnocení	7,6 / 3	8 / 3	7,6 / 3	5,6 / 3
1. celkem odesláno (MB)	11,094809	11,132961	11,510775	8,587178
2. celkem odesláno (MB)	11,759361	11,465178	11,408854	4,62554
1. celkem přijato (MB)	30,574059	30,522587	30,249064	34,085401
2. celkem přijato (MB)	26,610736	26,923855	27,077395	34,967644

Tabulka 5: *Konference 4 klientů – hodnocení a informace o přenosech*

9.4.3 Další testování čtveřice klientů

Dvojici testů provedených přes internetovou síť odpovídají tabulky č. 10, 11 a 12, 13 (v přílohách). Jednalo se o totožnou konfiguraci, jako v případě předešlém. Tato měření poukazují na několik faktů. Z tabulek 12 a 13 je patrné, že v rozmezí dané doby nemusí být P2P spojení mezi koncovými body prakticky možné, nicméně v jiném okamžiku jej lze zřídit. Současně ale vidíme, že v některých případech je P2P komunikace hodnocena i jako zcela dokonalá. V tabulce č. 11 vidíme, že některá spojení byť dosahují dobré kvality, jsou v průběhu z nenadání zcela přerušena.

9.4.4 Testování zátěže šesti klienty

Následoval test postupného připojování klientů do konference, jež reprezentují tabulky č. 14–17 (v přílohách) společně s tabulkou č. 6. Zde jsou při totožné konfiguraci testování jako v minulých případech využita zařízení Dell Inspiron 15 5000 Series, Lenovo V130-15IKB, HP 15-ba062nc, Lenovo B50-80 a dále Lenovo V110-15—AP a HP Pavilion 15 rt3290.

S každým příchodem bylo provedeno hodnocení. Nejprve byla provedena všechna hodnocení v P2P módu a posléze obdobně v SFU módu. Lze pozorovat, že mimo dvou konkrétních spojení byla v konferenci zřízena všechna vzájemná spojení mezi šesti připojenými klienty. V předchozím případě, kdy se propojovalo pouze pět klientů, byla situace s navázáním spojení horší. Spojení mezi úplně prvními klienty, tj. klienty jedna a dva, byla oběma hodnocena jako téměř excelentní tj. číslem osm. Když byli v konferenci i všichni ostatní, bylo toto jejich spojení hodnoceno v P2P stále pozitivně číslem sedm jako dobré. V tabulce č. 7 jsou vypočítány průměrné hodnoty hodnocení klientů ze všech tabulek pro hodnocení postupného připojování klientů.

Vzhledem k tomu, že kvalita při použití P2P techniky při společném streamování mezi klienty v měření, jež zaznamenává tabulka č. 6, bylo celkově slabší a neumožňovalo plynulou konferenci, bylo dále provedeno testování mezi klienty, jež by ověřilo P2P techniku a kvalitu spojení jednotlivě mezi daným účastníkem a klientem jedna pro porovnání nově naměřených výsledků s těmi, které zaznamenává tabulka č. 6 pro P2P prvního sloupce a řádku tj., jak se hodnotí klienti navzájem právě s klientem jedna. Toto měření zaznamenává tabulka č. 8. Každá buňka hodnocení v této tabulce obsahuje průměry hodnocení obou účastníků. Neboť oba účastníci hodnotili vždy stejnou hodnotou, nezískáváme hodnoty s desetinnými čísly.

P2P	Klient 1	Klient 2	Klient 3	Klient 4	Klient 5	Klient 6
Klient 1	x	7	7	0	6	7
Klient 2	7	x	4	4	6	5
Klient 3	4	6	x	3	0	1
Klient 4	0	7	4	x	5	5
Klient 5	6	4	0	6	x	3
Klient 6	7	5	1	5	4	x
SFU						
Klient 1	x	7	7	8	8	7
Klient 2	8	x	6	8	8	7
Klient 3	7	5	x	6	8	6
Klient 4	6	8	4	x	8	8
Klient 5	6	7	4	8	x	7
Klient 6	7	7	6	7	7	x

Tabulka 6: *Hodnocení 6 klientů v rámci konference*

Technika	2 klienti	3 klienti	4 klienti	5 klientů	6 klientů
P2P	8	6,7	5	5,7	5
SFU	8	7,6	6,9	7	6,9

Tabulka 7: *Průměrná hodnocení z tabulky č. 6 a tabulek 14–17*

P2P	Klient 1
Klient 1	x
Klient 2	9
Klient 3	8
Klient 4	0
Klient 5	8
Klient 6	8

Tabulka 8: *Následné konference pouze ve 2 – jednotliví klienti odpovídají tabulce č. 6*

10 Kombinované režimy přenosu a automatizace jejich přepínání

Zásadní změnou v přístupu k P2P je již implementovaný přechod z čistě P2P spojení na zapojení SFU serveru ne jako pouze náhradní řešení tj. přenosového serveru některých médií, ale jako směrovač všech médií. Doposud jsme ale k oběma přístupům přistupovali odděleně. Oba režimy přenosu, jak P2P tak i SFU, jsou funkční. Z předcházející kapitoly jasně vyplývá, že využití SFU přístupu představuje stabilitu a kvalitu. P2P přístup takto pozitivně hodnotit nelze, nicméně v komunikaci pouze dvou účastníků vykazuje taktéž vysokou kvalitu a šetří serverové zdroje. Řešením využití přístupů může být jejich kombinace.

Do konferenční místnosti se účastníci připojují postupně a v určitém okamžiku jsou připojeni právě dva a v mnohých případech je tento počet finální. Podle statistiky [81], kterou si dělala organizace Callstats, věnující se technologii WebRTC, je až kolem 50 % uskutečněných hovorů na jejich online platformě pouze dvou členných, což je nezanedbatelná hodnota.

Z těchto hledisek se jako optimální řešení nabízí kombinace obou režimů, které se budou automaticky přepínat na pozadí běžící online aplikace. Pro tuto variantu je třeba navrhnout a implementovat přepínací mechanizmus.

Cílem je, aby klienti byli změnou módu jen minimálně vyrušeni, což znamená, aby si v ideálním případě změny přepnutí módu mediální komunikace vůbec nevšimli, a aby byla zcela plynulá a nezjevná. V reálné implementaci se k tomuto ideálu snažíme maximálně přiblížit. Tímto také odpadá jedna z variant přepínání módu tj. ve vyhodnoceném okamžiku pro přepínání první komunikaci voláním příslušné metody jednoduše ukončit a v témže okamžiku komunikaci druhým přístupem spustit. Tím by právě došlo až na několika sekundový audio-vizuální výpadek.

Přepínací mechanizmus musí také reflektovat situaci, při níž dojde k selhání při zřizování P2P spojení, aby mohlo dojít k přepnutí módu nezávisle na implementovaných pravidlech pro automatickou změnu. V případě nemožnosti přímého spojení je možné pokus opakovat a pokud ani opakované spojení není možné, komunikace touto technikou musí být realizováno pomocí SFU. V neposlední řadě je vhodné disponovat také možností čistě manuálního nastavení módu pro potřeby ladění a testování či možné budoucí úpravy apod.

Jedním z klíčových parametrů je počet účastníků koference jakožto hranicí přepnutí módu. S ohledem mobilní zařízení, datovou zátěž a předchozím výsledkům budou hranicí právě dva účastníci v přímé P2P komunikaci.

Zde je také třeba vyjasnit otázku, jak bude přepnutí módu realizováno vizuálně. Nabízí se dvě varianty. Provést přepnutí módu každého jednotlivého klienta individuálně: při běhu současné komunikace, řekněme v P2P, se zřizuje komunikace druhého typu, SFU. A právě při jeho navázání přepnout zdroj zobrazovaného vysílání konkrétního klienta a původní komunikaci ukončit. Tím by ale mohlo dojít k rušivému a nejednotnému problikávání jednotlivých klientských video panelů, protože by změny byly nesynchronizované. Druhou a zvolenou variantou bude provést přepnutí médií všech klientů v místnosti v jedinný okamžik. V praxi by tak došlo obdobně jako v předcházející variantě ke zřizování druhé komunikace paralelně ke stávající a to skrytě. Po přepnutí všech klientských panelů se původní komunikace samozřejmě zruší. Uživatel by tak vnímal tuto změnu jako rychlé ”probliknutí”panelů všech účastníků. Na krátký čas dojde k vyššímu zatížení než běžně působí samotná konference a v rámci několika sekundového spojení dojde nevyhnutelně k duplicitě mediálních přenosů. Zajistit zcela nepozorovatelné přepnutí, byť se mediální data načítají na druhé straně z téhož zdrojového zařízení (webkamery a mikrofonu), se jeví jako nesnadné, neboť je potřeba modifikovat HTML a každý přenášený stream je z podstaty

samostatný. Jeden je navíc přenášen přes vzdálený server, kdežto druhá komunikace je zřizována přímo.

K určení okamžiku přepnutí módu může dojít buď ve chvíli načtení všech spojení, nebo po uplynutí standardní doby pro zřízení komunikace určitého počtu klientů. Byť tato varianta je svým způsobem nedokonalá, zjednodušuje implementaci. V první fázi nasazení P2P pro měření bylo využito tohoto řešení – módy se přepínaly čistě po uplynutí stanovené doby. Po zvážení bude přistoupeno k druhému způsobu. Mód bude přepnut až v okamžiku, kdy načte daný klient tolik mediálních stop resp. `Consumer` objektů, které mají být danému klientu zaslány od ostatních. Zde je třeba počítat s reálnou situací, kdy koncový bod bude například notifikován o určitém počtu stop k přijetí, ale mezi tím se jeden z koncových bodů odpojí, a proto nebude možné tento počet mediálních stop načíst. Je třeba ošetřit i tyto situace.

10.1 Automatická změna módu

Je třeba definovat, jaké jsou události iniciující přepínání módu vzhledem k počtu uživatelů v místnosti. Těmi jsou příchod klienta do místnosti a jeho opuštění. Jak vidíme v tabulce č. 9 hraniční jsou dva a tři klienti. Počet uživatelů však nestačí, jde i právě o onu událost, jak se konference dostala do tohoto počtu, neboť z toho vyplývá, v jakém nastavení byla předtím a co musí následovat. Současně je třeba uzpůsobit systém tak, že přechody mezi stavy trvají určitý čas a je nutno definovat chování systému i pro tyto situace a nově příchozí klienty v tyto momenty.

Původní návrh automatické změny módu počítal s P2P módem jako výchozím pro konferenci dvou účastníků. Taková konfigurace by jistě byla funkční a dosáhli bychom požadovaného. Jeho nevýhodou by ale byla situace, při které by nedošlo k propojení klientů. Vytvoření spojení trvá z pozorování minimálně dvě až tři či více sekund. Při navazování komunikace v případě nemožnosti jeho zřízení se doba může protáhnout až na patnáct vteřin než dojde k selhání vyhledávání. Nabízí se i možnost pokus o spojení opakovat. V takovém případě by se doba spojování mohla protáhnout na více než třicet sekund než by došlo k definitivnímu zavržení P2P. Každopádně se jedná o dobu příliš dlouhou na to, aby klient čekal na navázání spojení. Jako praktičtější se proto jeví zřídit SFU komunikaci, která bude v případě zřízení P2P přepnuta a nahradí SFU komunikaci. Pokud se spojení nepodaří vytvořit, komunikace bude nerušeně pokračovat v SFU spojení. Nevýhodou tohoto přístupu může být zvýšená (datová) zátěž zařízení při inicializaci, která je způsobena navazováním obou typů komunikace. Za zmínku také stojí, že ve stejném prostředí zřízení SFU komunikace se jeví obecně jako nepatrně delší než P2P. Patrně proto, že se vytváří spojení se serverem a poté mezi serverem a příjemci. Řádově o vteřinu a více.

Při přepínání módu v případě příchodu 2. klienta do konference v tabulce č. 9 jsou zřizovány oba typy komunikace a pokud je P2P funkční, je po určité době přepnuto z SFU módu do P2P. SFU komunikace je ukončena. V případě další události tj. opuštění třetího klienta z místnosti dojde k automatickému vyhodnocení přepínání komunikace z SFU módu do P2P. Pokud se toto přepnutí nepodaří realizovat, zůstává SFU komunikace. Příchod třetího iniciuje změnu do SFU módu. Byl-li již systém v SFU, zůstává beze změny. Poslední situací je odejití čtvrtého klienta. Tato situace je shodná se všemi, jež následují. Zde dochází ke zřízování komunikace SFU bez dalších vyjímek.

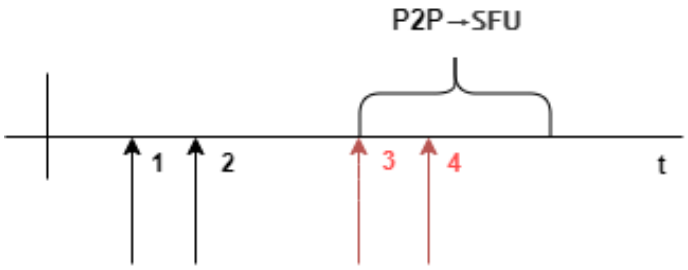
Pod pojmem zobrazovaný mód v této tabulce myslíme aktuálně využívaná a zobrazovaná přenášená média dostupných uživatelů připojených danou technikou. Jeden klient čeká na příchod dalších. Automaticky není nastaven na žádný mód. Ostatní sloupce tabulky jsou pro každou událost v posledním řádku rozděleny na dvě buňky.

První představuje mód, ve kterém se místnost nachází na počátku tj. při vzniku této události a druhá buňka, v jakém módu se nachází místnost po aplikování veškeré logiky aplikace – na konci všech změn. Tam, kde jsou lomítkem odděleny dva módy se jedná o jednu z variant podle toho, jestli se P2P mód podařilo zřídit či nikoli.

Je třeba zvážit také situaci, při které dochází k samotné změně módu do SFU. Tato operace trvá určitou dobu. Otázkou je, jak připojit média klienta respektive klientů, jež vstoupí právě v této situaci. Vyobrazena je na obrázku č. 25. Původní klienti změnu uvidí v daném okamžiku jako nenadálou. Nově příchozí klient čeká na připojení resp. přepnutí módu a zobrazení všech v místnosti.

počet klientů	1	2				3			
událost	příchod 1.	příchod 2.		opuštění 3.		příchod 3.		opuštění 4.	
přepínání	–	SFU→P2P		SFU→P2P		P2P→SFU / –		–	
zobrazený mód	–	SFU	P2P/SFU	SFU	P2P/SFU	P2P/SFU	SFU	SFU	SFU

Tabulka 9: *Automatická změna módu – události příchodů a opuštění místnosti klienty*



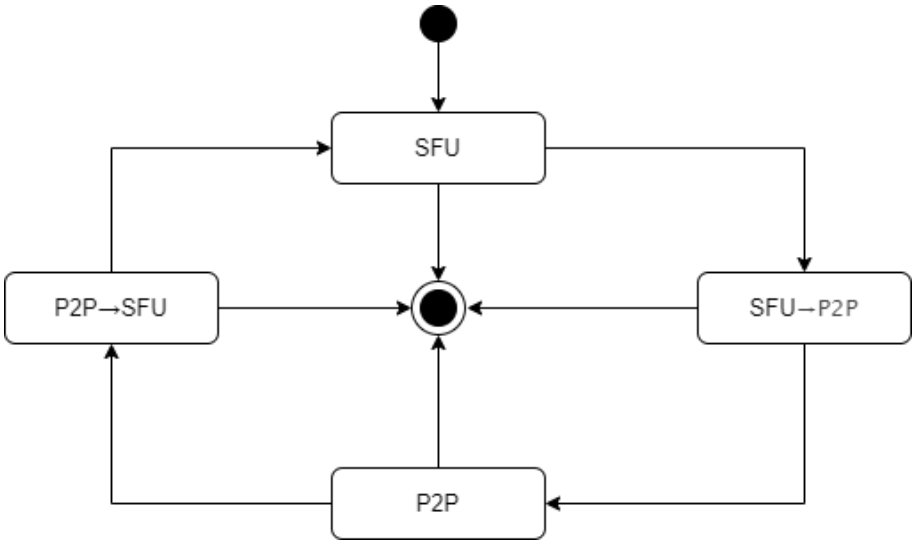
Obrázek 25: Připojení dalšího klienta právě v okamžiku přepínání módu z P2P do SFU

Schéma návrhu funkce automatické změny, kterou zajišťuje serverová část, je zobrazeno na obrázku č. 27 v obrázkových přílohách. Její implementace ponese jméno `automaticChangeSystemMode`. Bude volána právě při zřízení komunikačního websocketového spojení s klientem a naopak při jeho ukončení s jeho odchodem. Výčet informací, jež budou uchovávány v souvislosti s přepínáním módu:

- **Mód** – indikace módu (P2P/SFU)
- **Automatický profil** – indikace automatického / manuálního profilu přepínání módů.
- **Přepínání módu** – indikace, zda-li v daném okamžiku dochází k přepínání módu do SFU

Na straně klienta dojde k události, jež způsobí zaslání zprávy serveru o potřebě změny módu. Může se jednat o povel uživatele či automaticky generovanou událost nutné změny módu. V prvním případě je autoritou osoba, ve druhém implementovaná logika. Samotný povel ke změně módu je ale iniciován vždy až v běžící instanci serveru, která jej vyhodnotí a všem klientům konference zašle pokyn, jež ponese označení 'systemMode'.

Automatický mód zůstává, ikdyž se nepodaří zřídit P2P, zůstává SFU mód a celý proces síťového vyhledávání trasy je opakován. Pokud je tento pokus negativní, systém P2P instance zruší a komunikace pokračuje tak jak dosud.



Obrázek 26: Stavový diagram klientské části aplikace

Obrázek č. 26 zobrazuje stanový diagram, jež znázorňuje přepínání módů v klientské části aplikace z pohledu jejích prosesů.

10.1.1 Ošetření selhání spojení

Po dobu připojování se `RTCPeerConnection` nachází ve stavu `connecting`. To je fáze síťového připojování viz. kapitola č. 2.4 ICE – Interactive Connectivity Establishment. Při testování se objevila dvojice klientů, kteří se navzájem P2P technikou nepřipojili nikdy resp. při všechn pokusech i mimo zaznamenaná měření se spojení nenavázalo. Jedná se o klienta jedna a čtyři v tabulce č. 6. V tomto případě se spojení dostalo ze stavu `connecting` přímo do `failed`. Veškerá komunikace a výměna klíčových informací mezi oběma koncovými body proběhla v pořádku. Pokud nebylo logikou aplikace časově omezeno vyhledávání síťového spojení, fáze připojování trvala až 15 sekund. Uvážíme-li i následné opakování tohoto pokusu o připojení, musíme přidat dalších 15 sekund.

Kladně bude vyhodnoceno to spojení, které se dostane do stavu `connected`. Nebudou však kontrolovány příchozí stopy. Následně bude spuštěn časovač nastavený na dobu dvou sekund, aby došlo k ustálení a poté bude přepnut mód. Ve skutečnosti bude zaslána žádost serveru o přepnutí do P2P. Tu zasílá pouze iniciátor mediální komunikace. Server žádost přijme a vydá povel oběma klientům, aby mód přepnuli a zrušili SFU komunikaci.

V ukázce níže můžeme vidět registraci anonymní funkce s řídící strukturou switch, jež při změně stavu objektu `RTCPeerConnection` spouští daný případ. Metoda `tryReconnect()` objektu `P2PPoint` je volána právě když dojde k selhání spojení.

```
    this.pc.onconnectionstatechange = (event) => {
        switch(this.pc.connectionState) {
            case "connected":
                this._request('switchToP2P');
                break;
            case "failed":
                this.tryReconnect();
                break;
            case "closed":
                this._cancelPoint();
                break;
        }
}
```

Metoda `tryReconnect()` se pokusí o navázání P2P komunikace opakovaně. V momentě, kdy dojde k restartu ICE komponenty je také volána událost `onnegotiationneeded`, ve které iniciátor zasílá opětovně vygenerovanou nabídku vlastních médií.

```
    tryReconnect()
    {
        if (this.isInitiator) {
            return;
        }
        if (this.reConnected >= MAX_RECONNECT_ATTEMPTS) {
            this._cancelPoint();
            return;
        }
        this.reConnected++;
        this.pc.restartIce();
    }
```

11 Diskuse

V případě nasazení technologie WebRTC odpadá nutnost instalovat na klientských zařízeních jakékoli dodatečné aplikace či rozšíření. Stěžejní výhodou jsou také velmi široké možnosti její integrace v podobě vlastních řešení a to bez nutnosti platit licenční poplatky za její užívání. Také je třeba poukázat na nestálost API v prohlížečích a nutnost údržby systému.

11.1 Nastavení kodeku v mediální relaci

V případě kodeků lze zmínit nastupující nový video kodek AV1, jež díky svým kompresním možnostem přinese plošné snížení množství přenášených dat nejen v oblasti online konferencí ale streamování obecně. Tento kodek je již v současnosti v prohlížečích dostupný a lze jej využívat pro přehrávání videa na uzpůsobených webových portálech nicméně v rámci WebRTC není možné jej nyní aplikovat, neboť není podporován. V rámci mediálního spojení dvou koncových bodů je prohlížeči WebRTC technologií automaticky navržen optimální kodek podporovaný oběma stranami pomocí již zmiňovaných SDP popisů. Nastavení kodeku AV1 pro relaci bylo také testováno v Chrome 80.0 (na platformě Windows) a nebylo možné jej aplikovat.

Konfigurace kodeku P2P spojení je v současné době realizovatelné dvěma způsoby. Přímý způsob představuje manuální zásah do SDP popisu generovaného samotným `RTCPeerConnection` objektem tj. parsováním a úpravou textového řetězce a druhou možností je před zahájením mediální relace změnit seznam kodeků, jež jsou k dispozici pro nastavované spojení více v kapitole č. 9.2.3 Konfigurace kodeků pomocí RTCRtpTransceiver, nicméně je třeba dodat, že toto API není ve všech prohlížečích podporováno.

11.2 Nasazení přenosového serveru

V rámci práce bylo hledáno optimální řešení, jež by přemostilo nedostatky přímé P2P komunikace. Řešení bylo nalezeneno v podobě tzv. SFU serveru, který slouží jako směrovač mediální komunikace a principielně je použit jinak než klasický přenosový server dle koncepce WebRTC. K výběru vhodného serveru došlo na základě výsledků komparativní studie a posouzením dalších aspektů. Mezi třemi nejlépe hodnocenými servery (Janus, Mediasoup a Medooze) bylo zvoleno řešení Mediasoup, které současně poskytuje i klientskou aplikaci a samotný server je k dispozici jako snadno integrovatelný Node balíček.

11.3 Testování Mediasoup aplikace na dlouhé vzdálenosti

V rámci práce bylo testováno streamování pomocí původní Mediasoup aplikace na dlouhé vzdálenosti. Jak popisuje kapitola č. 7, server byl nasazen v Chicagu ve Spojených státech amerických a koncoví uživatelé se nacházeli v České republice. Cílem bylo ověřit, zda-li i v takovéto situaci bude řešení Mediasoup schopné poskytovat kvalitní přenos bez výrazného zpoždění přenášených médií. Bylo realizováno jednak uživatelské hodnocení tak i měření, jež streamovalo mezi dvěma záložkami téhož zařízení. Přenášen byl odpočet času stopkami. Přenášeným resp. odesílaným obrazem tak byla vizuální časová známka, kterou bylo možné odečítat oproti přijímanému obrazu.

Zde se může nabídnout otázka, jakou relevanci mají tato měření realizovaná se stopkami. Jisté však je, že celkové uživatelské hodnocení přenášených streamů bylo hodnoceno pozitivně hodnotami sedm a osm ACR škály nicméně zpoždění serveru nasazeného v Chicagu oproti serveru v Hluboké nad Vltavou je podle měření přibližně 4-5x větší (tj. 450-650 ms).

11.4 Porovnání technik

Hlavním otazníkem bylo, zda-li P2P komunikaci zcela vyřadit, což si žádalo porovnání obou typů přístupů. Bylo potřeba disponovat platformou pro testování obou technik a z tohoto důvodu buď mohly být realizovány oddělené systémy či například využít současné aplikace Mediasoup s již implementovanou SFU komunikací, k čemuž bylo přistoupeno. Aplikace byla analyzována a s využitím získaných znalostí o WebRTC technologii byla do systému implementována k SFU komunikaci také P2P komunikace mezi klienty v manuálně přepínatelném módu. Byla také zvažována možnost využít implementace Mediasoup tak, aby prostřednictvím jejích tříd byla realizována P2P komunikace. To bohužel není možné, neboť komunikace je uzpůsobena pro klient-server pojetí a serverovou část nelze jednoduše převést na jiného klienta.

Pro porovnání obou přístupů dále musely být zjištěny možnosti pro implementaci logiky, jež by zajistila měření daných spojení tak, aby bylo možné porovnat uskutečněné konference. Hodnocení kvality médií bylo určeno uživateli na základě ACR škály hodnocení. Na základě provedených měření nasazení SFU jednotky je prokazatelně jednak datově úspornější, což je výhodou zejména v oblasti mobilních zařízení, tak je i stabilnější – s jejím nasazením nebylo sledováno, že by docházelo k selhání zřízení spojení a kvalita médií se též ukázala při větším počtu účastníků jako kvalitnější oproti P2P. Kvalita konference byla i při šesti účastnících, jež využívali svých notebooků, znatelně lepší, byť i její kvalita se zhoršila v porovnání s nižším počtem účastníků konference.

Dále můžeme konstatovat, že P2P technika při větším počtu účastníků v kvalitě oproti SFU přístupu selhávala. Zatěžuje více klientská zařízení a datové přenosy a zřízení přímé komunikace není spolehlivé ve všech případech. Nicméně, jak ukazuje uživatelské hodnocení, stále dosahuje v některých případech srovnatelného resp. dobrého až excelentního spojení a to zejména, pokud se jedná o spojení pouze dvou účastníků, jak vyplynulo z testování se šesti účastníky v konferenci. Toto dokladují tabulky č. 7 a 8 ve vztahu k tabulce č. 6. Dále je třeba mít na mysli, že existují spojení, která navázat pomocí P2P nelze. Takovým případem je i spojení klientů jedna a čtyři tabulky č. 8. Spojení mezi těmito uživateli se nikdy předtím ani po tomto testování navázat nepodařilo. V rámci P2P spojení spíše ojediněle docházelo k nenadálému pádu spojení.

Během vývoje bylo pozorováno a to jak při nesčetném množství testů v rámci implementace systému a následného testování, tak měřeních jako takových, že charakteristickou chybou při streamování médií za použití Mediasoup SFU jednotky bylo krátké a občasné – ne příliš časté – zaseknutí videa či zadrhnutí audia. Naopak obecně sledovanou chybou P2P spojení je spíše než zaseknutí zpomalení či zrychlení přehrávání streamu. Spojení, jež byla hodnocena kolem hodnoty pět často buď přenášela obraz či zvuk trhaně, nebo měnila rozlišení videa – docházelo k jeho snižování a nebyla již plynulá.

Výsledky ukázaly v případě obou technik na podobnou kvalitu při dvou účastnících a celkově nižší kvalitu přenosu s nasazeným P2P módem oproti SFU zejména při vyšším počtu účastníků tj. tři a více. Dále byla měřena markantní datová úspora v mediálním přenosu úměrnou počtu připojených klientů při využití SFU módu. Stejně tak lze poukázat na úspory serverových zdrojů při P2P komunikaci.

11.5 Implementace kombinace režimů

Z důvodů uvedených v předešlé kapitole bylo navrženo jako řešení implementovat automatizační logiku pro přepínání obou zmíněných přístupů, jež by zkombinovala obě techniky. Práce tak také přináší poznatky z implementace této optimalizace. Klíčové jsou hraniční situace v systému. Aplikace před začleněních těchto úprav implementovala chod P2P a SFU režimů komunikace pro účely porovnání přístupů.

Dva účastníci jsou nejprve připojeni SFU módem. Současně dochází k vytváření P2P spojení, které pokud je úspěšně zřízeno, nahrazuje původní komunikaci. V případě připojení tří a více účastníků je systém opět přepnut do SFU. V případě odchodu třetího se opět systém pokusí o navázání P2P komunikace. Realizovaný systém tak neplýtvá zdroji serveru ani koncových klientů a k vyššímu zatížení dochází pouze na krátkou dobu přepínání režimů.

Realizovaná aplikace při samotném přepnutí překresluje vždy celý panel s uživatelem resp. reactovou komponentu, jejíž součástí je i HTML video element se streamem daného přístupu. Jak se ukázalo, toto řešení není ideální, neboť tak dochází ke krátkému ale viditelnému vizuálnímu výpadku panelu s videem při přepínání. Spíše by však mělo dojít ke změně samotného streamu video elementu v rámci komponenty, jež bude interně toto nastavování zajišťovat na základě aktuálně nastaveného módu komunikace.

11.6 Přenos skutečně požadovaného rozlišení videa

Nabytých poznatků z oblasti statistik WebRTC lze využít v rámci reálné implementace a to úpravou přenášeného rozlišení. To se hodí zejména u aplikací, které přenášejí HD rozlišení či větší obraz ve dvojici s mobilním zařízením nižšího rozlišení, jež rozbrazuje video poloviční rozměrů. Následkem může být zasílání videa v nadbytečném rozlišení. Řešením je na straně odesílatele načtení video stopy v maximálním rozlišení, které je odesíláno nezměněné či zmenšené. V případě příjemce jeden aspekt je rozlišení samotného zařízení a druhý je skutečně zobrazovaná velikost videa webovou aplikací – rám zobrazující video může být velikostně omezen. Tato aplikace je svým charakterem určena pro početnější konference, kde se nepočítá s přenosem videa ve vysokém rozlišení. Využíváno bude zejména konfigurace 640x480 pixelů. Další možností je v rámci WebRTC aplikace implementovat uživatelsky nastavitelné profily v několika škálách, jež budou ovlivňovat rozlišení přijímaného videa. Sám uživatel tak poté může snížit datovou náročnost přenosu. Tento přístup však lze aplikovat tam, kde je podporováno příslušné WebRTC API.

Odesílatel konfiguruje objekt `RTCRtpSender` (jež náleží danému `RTCPeerConnection` spojení) koncového bodu metodou `setParameters` a úpravou poměru odesílaného rozlišení k načtenému tzv. `scaleResolutionDownBy`. Tato akce vyžaduje na straně odesílatele znalost požadovaného (poměru) rozlišení příjemce. Zde je dobré připomenout, že podstatné je odesílat video v rámcově obdobném rozlišení. Přesných parametrů v reálném světě mnohdy není možné dosáhnout kvůli různorodosti celkové konfigurace. Poměr může být jednoduše upravován podle požadavku druhého klienta přičemž volbě nastavení poměru je třeba dát určité hranice.

12 Závěr

Cílem práce bylo optimalizovat nasazení WebRTC technologie v konferenčním systému, čehož bylo úspěšně dosaženo. Práce systematicky dokumentuje jednotlivé kroky, které byly vzhledem k optimalizaci komunikace uskutečněny a rozebírají jednotlivé oblasti pro možné optimalizace. Tento úkol zahrnoval prostudování potřebné problematiky, nasazení vybraného serverového řešení, implementaci P2P spojení, testování a měření a optimalizační úpravy. V průběhu práce v rámci aplikace Mediasoup představovala implementace P2P časově náročnou část realizace a to zejména kvůli analýze vnitřního fungování systému pro jeho úpravy, tak aby bylo možné implementovat potřebné úpravy.

Kromě lepší manipulace s video elementem, jež je popsána v předchozí kapitole se lze zabývat celkovou optimalizací samotného serverového řešení. Jedna běžící instance v závislosti na implementaci a optimalizaci je schopna přemostit určitý počet mediálních spojení. Takových instancí může být navíc spouštěno více a je třeba zajistit jejich škálování, případně rozmístění a regulovat zatíženost jednotlivých instancí. Další oblastí, jež může být vylepšována, jsou přenosy na dlouhé vzdálenosti za účelem určitého snížení prodlev způsobených přenosem přes server, byť v našem případě bylo streamování na dobré úrovni, mohou být i v tomto směru provedeny další pozitivní změny.

Pro přepínání obou režimů je třeba v určitém okamžiku mít spuštěné oba typy komunikace, aby jeden typ navázal spojení a ustálil jej, zatímco aktivní komunikace probíhá prostřednictvím druhého přístupu. Zřízení spojení může doprovázet celá řada problémů jako dlouho trvající zřizování spojení, nenadálé výpadky a kromě úspěšného i neúspěšné spojení. Proto konferenční systém musel implementovat logiku, která v závislosti na počtu klientů a stavech spojení realizuje přepínání režimů P2P a SFU. Podobně jako vytvořená aplikace řeší otázku přenosu médií i platforma Google Hangouts, která na svých webových stránkách uvádí [91], že dva klienti jsou spojeni P2P a při více účastnících komunikace probíhá přes server, byť neuvádí jak.

Strojové hodnocení kvality videa automatickým mechanizmem je velkým tématem a to zejména pokud hovoříme o posouzení kvality streamovaných médií. Otázkou je, zda-li je toto v současné době možné. Takové hodnocení by mohlo sloužit nejen pro testování aplikace samotné, ale například také pro hodnocení kvality spojení ještě předtím než se uživatel pokusí fakticky připojit do místnosti – například na stránce s možností nastavit média a své údaje před vstupem do konference. Podle kvality také může být zvolen P2P či SFU přístup. Některé existující metody pro hodnocení kvality nemusí jako výstupy uvádět hodnoty, jež by korespondovaly s uživatelským hodnocením a mohou být nepřesné. Na druhé straně strojové hodnocení má své nesporné výhody jako jednoduchost provedení samotného testu, neboť nejsou vyžadováni další lidé resp. hodnotitelé. Tato problematika by jistě mohla být také dalším námětem budoucích prací.

13 Literatura a použité zdroje

[1] Google release of WebRTC source code. *W3C Public Mailing List Archives* [online]. [cit. 2019-04-19]. Dostupné z: http://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html

[2] DUŠEK, Jiří. Online konferenční systém založený na HTML5. Č. Budějovice, 2016. bakalářská práce (Bc.). JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH. Pedagogická fakulta

[3] LEVENT-LEVI, Tsahi. 10 Massive Applications Using WebRTC. *BlogGeek.me* [online]. 18. prosince 2017 [cit. 2019-04-19]. Dostupné z: https://bloggeek.me/massive-applications-using-webrtc

[4] JONES, Brandon a Nell WALICZEK, ed. WebXR Device API: W3C First Public Working Draft, 5 February 2019. *W3C* [online]. 5. února 2018 [cit. 2019-04-19]. Dostupné z: https://www.w3.org/TR/2019/WD-webxr-20190205

[5] ABOBA, Bernard, ed. WebRTC Next Version Use Cases. *W3C* [online]. 11. prosince 2018 [cit. 2019-04-19]. Dostupné z: https://www.w3.org/TR/webrtc-nv-use-cases

[6] SIM, SHERWIN. Six Things You Need To Know About WebRTC SFUs. *Temasys* [online]. 2017, 5 December 2017 [cit. 2018-09-17]. Dostupné z: https://temasys.io/six-things-need-know-webrtc-sfus

[7] VISCARRA, Rafael. A Guide to: WebRTC Media Servers & Open Source Options. *WebRTC.ventures* [online]. 28 November 2017 [cit. 2018-09-17]. Dostupné z: https://webrtc.ventures/2017/11/a-guide-to-webrtc-media-servers-open-source-options/

[8] MediaStreamTrack. `MDN web docs` [online]. [cit. 2020-03-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/MediaStreamTrack

[9] NAGY, Marcin. How to integrate callstats.io REST API, Part 1: Signaling state changes. *Callstats.io* [online]. 12. 12. 2017 [cit. 2020-03-21]. Dostupné z: https://www.callstats.io/blog/2017/12/12/signaling-state-changes

[10] Free open source implementation of TURN and STUN Server: *Github* [online]. [cit. 2020-03-26]. Dostupné z: https://github.com/coturn/coturn

[11] XTurn - Xirsys TURN Server in Elixir: *Github* [online]. [cit. 2020-03-26]. Dostupné z: https://github.com/xirsys/xturn

[12] Restund [online]. [cit. 2020-03-26]. Dostupné z: http://www.creytiv.com/restund.html

[13] Pion TURN: *Github* [online]. [cit. 2020-03-26]. Dostupné z: https://github.com/pion/turn

[14] Traversal Using Relays around NAT (TURN) Extension for IPv6. *IETF Tools* [online]. [cit. 2020-03-26]. Dostupné z: https://tools.ietf.org/html/rfc6156

[15] Session Traversal Utilities for NAT (STUN) Extension for Third-Party Authorization. IETF Tools [online]. [cit. 2020-03-26]. Dostupné z: https://tools.ietf.org/html/rfc7635

[16] MACHI, Jim. WebRTC MCU Architecture - All For One And One For All. *TMCNET* [online]. 17 Januray 2017 [cit. 2018-09-19]. Dostupné z: http://blog.tmcnet.com/industry-insight/2017/01/webrtc-mcu-architecture—all-for-one-and-one-for-all.html

[17] ZELAYA, Hector. Multi-Party WebRTC Option 3: SFU. *WebRTC.ventures* [online]. 16 July 2018 [cit. 2018-09-19]. Dostupné z: https://webrtc.ventures/2018/07/multi-party-webrtc-option-3-sfu

[18] Session Traversal Utilities for NAT (STUN). IETF Tools [online]. [cit. 2019-12-09]. Dostupné z: https://tools.ietf.org/html/rfc5389

[19] LEVENT-LEVI, Tsahi. WebRTC Multiparty Video Alternatives, and Why SFU is the Winning Model. *BlogGeek.Me* [online]. 7 March 2016 [cit. 2018-09-17]. Dostupné z: https://bloggeek.me/webrtc-multiparty-video-alternatives

[20] BENHAMICHE, Samy. Scalability in video-conferencingm *Medium* (Part 1) [online]. [cit. 2020-03-27]. Dostupné z: https://medium.com/linagora-engineering/scalability-in-video-conferencing-part-1-276f52b4acac

[21] SFU WebRTC multiparty video architecture and its uses [online] Dostupné z: http://blog.enfintechnologies.com/sfu-webrtc-multiparty-video-architecture-and-its-uses

[22] MACHI, Jim. WebRTC SFU Architecture = Champion of Large Scale Video Conferences. *TMCNET* [online]. 24 January 2017 [cit. 2018-09-19]. Dostupné z: http://blog.tmcnet.com/industry-insight/2017/01/webrtc-sfu-architecture-champion-of-large-scale-video-conferences.html

[23] HO, Anthony T. S. a Shujun LI. *Handbook of Digital Forensics of Multimedia Data and Devices*. Wiley-IEEE Press, 2015. ISBN 978-1-118-75707-9.

[24] LIU, YU. AV1 beats x264 and libvpx-vp9 in practical use case. *Facebook Code* [online]. 10. duben 2018 [cit. 2018-11-20]. Dostupné z: https://code.fb.com/video-engineering/av1-beats-x264-and-libvpx-vp9-in-practical-use-case/

[25] Cisco Visual Networking Index: Forecast and Methodology, 2016–2021. *Cisco* [online]. 2017 [cit. 2018-11-17]. Dostupné z: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[26] The Alliance for Open Media Kickstarts Video Innovation Era with "AV1" Release. *Alliance for Open Media* [online]. 28. března 2018 [cit. 2018-11-24]. Dostupné z: https://aomedia.org/the-alliance-for-open-media-kickstarts-video-innovation-era-with-av1-release/

[27] MONTGOMERY, Chris. Next generation video: Introducing AV1. Xiph.Org Foundation [online]. 9. dubna 2018 [cit. 2018-11-24]. Dostupné z: https://people.xiph.org/ xiphmont/demo/av1/demo1.shtml

[28] They Developed It. They Benefit From It. They Stand Behind It. *Alliance for Open Media* [online]. 2018 [cit. 2018-11-20]. Dostupné z: https://aomedia.org/membership/members

[29] AV1 Features. In: *Alliance for open media* [online]. [cit. 2019-03-29]. Dostupné z: https://aomedia.org/av1-features/

[30] YouTube is testing usage of AV1 codec for videos [online]. In: . [cit. 2019-03-29]. Dostupné z: https://www.guru3d.com/news-story/youtube-is-testing-usage-of-av1-codec-for-videos.html

[31] Bitmovin & AV1. *Bitmovin: Software to Solve Complex Video Problems* [online]. [cit. 2019-04-08]. Dostupné z: https://bitmovin.com/av1

[32] BEAUFORT, François. Audio/Video Updates in Chrome 70. *Google Developers* [online]. 2018 [cit. 2018-11-20]. Dostupné z: https://developers.google.com/web/updates/2018/09/chrome-70-media-updatesav1-decoder

[33] GILES, Ralph a Martin SMOLE. DASH playback of AV1 video in Firefox. *Mozilla HACKS* [online]. 28. listopad 2017 [cit. 2018-11-24]. Dostupné z: https://hacks.mozilla.org/2017/11/dash-playback-of-av1-video

[34] MPEG-4/H.264 video format. *Can i use* [online]. [cit. 2019-04-10]. Dostupné z: https://caniuse.com/#search=h264

[35] HEVC/H.265 video format. *Can i use* [online]. [cit. 2019-04-10]. Dostupné z: https://caniuse.com/#search=h265

[36] Frequently Asked Questions. *WebM* [online]. [cit. 2019-04-10]. Dostupné z: https://www.webmproject.org/about/faq

[37] WebM video format. *Can i use* [online]. [cit. 2019-04-10]. Dostupné z: https://caniuse.com/#search=webm

[38] H.264/MPEG-4 AVC. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2019 [cit. 2019-04-13]. Dostupné z: https://en.wikipedia.org/wiki/H.264/MPEG-4_AVC#cite_note-1

[39] High Efficiency Video Coding. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2019 [cit. 2019-04-13]. Dostupné z: https://en.wikipedia.org/wiki/High_Efficiency_Video_Coding

[40] SHARABAYKO, Maxim P., Oleg G. PONOMAREV a Roman I. CHERNYAK. Intra Compression Efficiency in VP9 and HEVC. *ResearchGate* [online]. Listopad 2013 [cit. 2019-04-12]. Dostupné z: https://www.researchgate.net/publication/259955411_Intra_compression_efficiency_in_VP9_and_HEVC

[41] ŘEŘÁBEK, Martin, Philippe HANHART, Pavel KORSHUNOV a Touradj EBRAHIMI. Quality Evaluation of HEVC and VP9 Video Compression in Real-Time Applications. *Infoscience* [online]. 2015 [cit. 2019-04-12]. Dostupné z: https://infoscience.epfl.ch/record/207496/files/QoMEX2015_VP9.pdf

[42] SILLARS, Doug. Video Playback On The Web: The Current State Of Video (Part 1). *Smashing magazine* [online]. 24. říjen 2018 [cit. 2019-04-13]. Dostupné z: https://www.smashingmagazine.com/2018/10/video-playback-on-the-web-part-1

[43] PAVLIČ, Jani a Jernej BURKELJCA. FFmpeg based Coding Efciency Comparison ofH.264/AVC, H.265/HEVC and VP9 Video CodingStandards for Video Hosting Websites. *Academia* [online]. leden 2019 [cit. 2019-04-13]. Dostupné z: https://www.academia.edu/38173358/FFmpeg_based_Coding_Efficiency_Comparison_of_H.264_AVC_H.265_HEVC_and_VP9_Video_Coding_Standards_for_Video_Hosting_Websites

[44] AV1 video format. *Can i use* [online]. [cit. 2019-04-10]. Dostupné z: https://caniuse.com/#search=av1

[45] SITNIK, Andrey. Better web video with AV1 codec. *Extraterrestrial product development consultancy* [online]. [cit. 2020-03-27]. Dostupné z: https://evilmartians.com/chronicles/better-web-video-with-av1-codec

[46] ANDRÉ, Emmanuel, Nicolas LE BRETON, Augustin LEMESLE, Ludovic ROUX a Alexandre GOUAILLARD. Comparative Study of WebRTC Open Source SFUs for Video Conferencing. *CoSMo Software: Real-Time Communication Solutions, Tools and Services* [online]. říjen 2018 [cit. 2019-03-26]. Dostupné z: https://www.cosmosoftware.io/publications/andre2018-comparative-study-of-sfus-pdf

[47] GOUAILLARD, Alex. Breaking Point: WebRTC SFU Load Testing (Alex Gouaillard). *WebrtcH4cKS* [online]. 2018, 18. října 2018 [cit. 2019-03-26]. Dostupné z: https://webrtchacks.com/sfu-load-testing

[48] GOUAILLARDA, Alexandr. WebRTC Video Quality Assessment. *WebRTC by Dr Alex* [online]. [cit. 2019-03-27]. Dostupné z: http://webrtcbydralex.com/index.php/2018/10/11/webrtc-video-quality-assessment

[49] FELDMANN, Christian. Multi-Codec DASH Dataset: An Evaluation of AV1, AVC, HEVC and VP9. *BITMOVIN: Software to Solve Complex Video Problems* [online]. 2018, 20. březen 2018 [cit. 2018-11-20]. Dostupné z: https://bitmovin.com/av1-multi-codec-dash-dataset

[50] HANHART, Philippe a Touradj EBRAHIMI. Calculation of average coding efficiency based on subjective quality scores. *Infoscience* [online]. 2013 [cit. 2019-04-08]. Dostupné z: https://infoscience.epfl.ch/record/192802/files/Elsevier2013_preprint.pdf

[51] Detect Supported Audio Formats with JavaScript. *David Walsh Blog* [online]. 10. října 2017 [cit. 2019-01-10]. Dostupné z: https://davidwalsh.name/detect-supported-audio-formats-javascript

[52] HTMLMediaElement.canPlayType(). MDN Web docs [online]. [cit. 2019-01-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement/canPlayType

[53] Testing with Kite. *WebRTC* [online]. 1. září 2018 [cit. 2019-03-26]. Dostupné z: https://webrtc.org/testing/kite

[54] *CoSMo Software: Global Leader in WebRTC Technology & Implementation* [online]. [cit. 2019-11-14]. Dostupné z: https://www.cosmosoftware.io

[55] Janus: The general purpose WebRTC server. *Meetecho* [online]. 11. říjen 2018 [cit. 2019-03-27]. Dostupné z: https://janus.conf.meetecho.com

[56] VPS Server [online]. [cit. 2020-04-02]. Dostupné z: https://www.vpsserver.com

[57] World Map Clipart Black And White. *Clipart* [online]. [cit. 2020-04-03]. Dostupné z: https://www.clipart.email/clipart/world-map-clipart-black-and-white-22252.html

[58] Flashphoner [online]. [cit. 2020-04-03]. Dostupné z: https://flashphoner.com/oh-webcam-online-broadcasting-latency-thou-art-a-heartless-bitch/

[59] *Mediasoup* [online]. [cit. 2019-03-27]. Dostupné z: https://mediasoup.org

[60] Medooze: WebRTC Media Server [online]. [cit. 2019-03-27]. Dostupné z: https://github.com/medooze/media-server

[61] HART, Chad. Anatomy of a WebRTC SDP. *WebrtcHacks* [online]. [cit. 2019-03-28]. Dostupné z: https://webrtchacks.com/sdp-anatomy

[62] Introduction to WebRTC protocols. *MDN Web docs* [online]. [cit. 2019-04-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Protocols

[63] DUTTON, Sam. WebRTC in the real world: STUN, TURN and signaling. *HTML5 Rocks* [online]. 4. listopadu 2013 [cit. 2019-03-28]. Dostupné z: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure

[64] KRUKOVA, Alina. What is WebRTC. *TrueConf* [online]. 17. červen 2015 [cit. 2020-04-03]. Dostupné z: https://trueconf.com/blog/reviews-comparisons/webrtc-wiki.html

[65] BRUAROEY, Jan-Ivar, BERGKVIST, Adam, Daniel C BURNETT, Cullen JENNINGS, Anant NARAYANAN, Bernard ABOBA, Taylor BRANDSTETTER a Jan-Ivar, ed. WebRTC 1.0: Real-time Communication Between Browsers. *W3C* [online]. 27. září 2018 [cit. 2019-03-28]. Dostupné z: https://www.w3.org/TR/webrtc/

[66] Lifetime of a WebRTC session. *MDN Web Docs* [online]. [cit. 2019-03-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Session_lifetime

[67] Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). In: Request for Comments: 5766 [online]. Internet Engineering Task Force (IETF), Duben 2010, s. 67 [cit. 2019-03-28]. ISSN 2070-1721.

[68] WebRTC Peer-to-peer connections. *Can i use* [online]. [cit. 2020-04-18]. Dostupné z: https://caniuse.com/search=webrtc

[69] ROSENBERG, J. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. *Internet Engineering Task Force (IETF)* [online]. Duben 2010 [cit. 2019-03-29]. Dostupné z: https://tools.ietf.org/html/rfc5245

[70] RTCOfferOptions.iceRestart. *MDN Web Docs* [online]. [cit. 2019-03-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/RTCOfferOptions/iceRestart

[71] RTCPeerConnection.iceConnectionState. *MDN Web Docs* [online]. [cit. 2020-03-21]. Dostupné z: https://www.w3.org/TR/webrtc/#dom-rtcicetransport

[72] RTCPeerConnection.connectionState. *MDN web docs* [online]. [cit. 2020-03-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/connectionState

[73] GRIGORIK, Ilya. *High Performance Browser Networking: What Every Web Developer Should Know About Networking And Web Performance.* September 2013. United States: O'Reilly Media, 2013. ISBN 1449344763.

[74] Object RTC (ORTC) API for WebRTC. *Can i use* [online]. [cit. 2020-04-18]. Dostupné z: https://caniuse.com/feat=objectrtc

[75] Architecture. *ORTC* [online]. [cit. 2020-03-28]. Dostupné z: https://ortc.org/architecture/

[76] Handling WebRTC session disconnections. *BlogGeek.me* [online]. 8. 4. 2019 [cit. 2020-03-28]. Dostupné z: https://bloggeek.me/handling-session-disconnections-in-webrtc

[77] Electron Documentation: About Electron. *Electron* [online]. [cit. 2019-03-29]. Dostupné z: https://electronjs.org/docs/tutorial/about

[78] Electron: Apps. *Electron* [online]. [cit. 2019-03-29]. Dostupné z: https://electronjs.org/apps

[79] LEVENT-LEVI, Tsahi. WebRTC Electron Implementations are on fire. Electron [online]. 29. ledna 2018 [cit. 2019-03-29]. Dostupné z: https://bloggeek.me/webrtc-electron-implementations/

[80] Building and Scaling Video Conferencing in Slack. *Youtube* [online]. 28. října 2017 [cit. 2019-03-29]. Dostupné z: https://www.youtube.com/watch?v=VJj4ddWDTbs

[81] A CASE OF HIGH LATENCY: HAIRPINNING OR DISTANCE. *Callstats.io: Build better real-time communication products with WebRTC analytics* [online]. 23. leden 2019 [cit. 2019-03-31]. Dostupné z: https://www.callstats.io/white-papers/case-of-high-latency?

[82] MELLEN, Allie. Why is Delay an Important WebRTC Metric for Contact Centers?. *Callstats.io: Build better real-time communication products with WebRTC analytics* [online]. 23. leden 2019 [cit. 2019-03-31]. Dostupné z: https://www.callstats.io/blog/why-is-delay-an-important-webrtc-metric-for-contact-centers

[83] GROZEV, Boris. Improving Scale and Media Quality with Cascading SFUs (Boris Grozev). WebrtcHacks [online]. Listopad 2018 [cit. 2019-04-02]. Dostupné z: https://webrtchacks.com/sfu-cascading/

[84] P2P4121. Jitsi [online]. 17. červenec 2017 [cit. 2019-04-05]. Dostupné z: https://jitsi.org/news/p2p4121/

[85] ROSSHOLM, Andreas. Video Quality Assessment. *Medium* [online]. 9. April 2018 [cit. 2019-11-14]. Dostupné z: https://medium.com/@eyevinntechnology/video-quality-assessment-34abd35f96c0

[86] Subjective video quality assessment methods for multimedia applications: Audiovisual quality in multimedia services. *International Telecommunication Union* [online]. duben 2008 [cit. 2019-11-14]. Dostupné z: https://www.itu.int/rec/T-REC-P.910-200804-I/en

[87] RTCRtpSender. *Resources for developers, by developers.: MDN* [online]. [cit. 2019-11-17]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/RTCRtpSender

[88] RTCRtpReceiver. *Resources for developers, by developers.: MDN* [online]. [cit. 2019-11-17]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/RTCRtpReceiver

[89] RTCRtpTransceiver. *Resources for developers, by developers.: MDN* [online]. [cit. 2019-11-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/RTCRtpTransceiver

[90] Date. *Resources for developers, by developers.: MDN* [online]. [cit. 2019-11-22]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

[91] Peer-to-peer calling in classic Hangouts. *Google Help* [online]. [cit. 2020-05-12]. Dostupné z: https://support.google.com/hangouts/answer/6334301?hl=en

[92] *Node* [online]. [cit. 2019-03-22]. Dostupné z: https://nodejs.org

14 Seznam obrázků

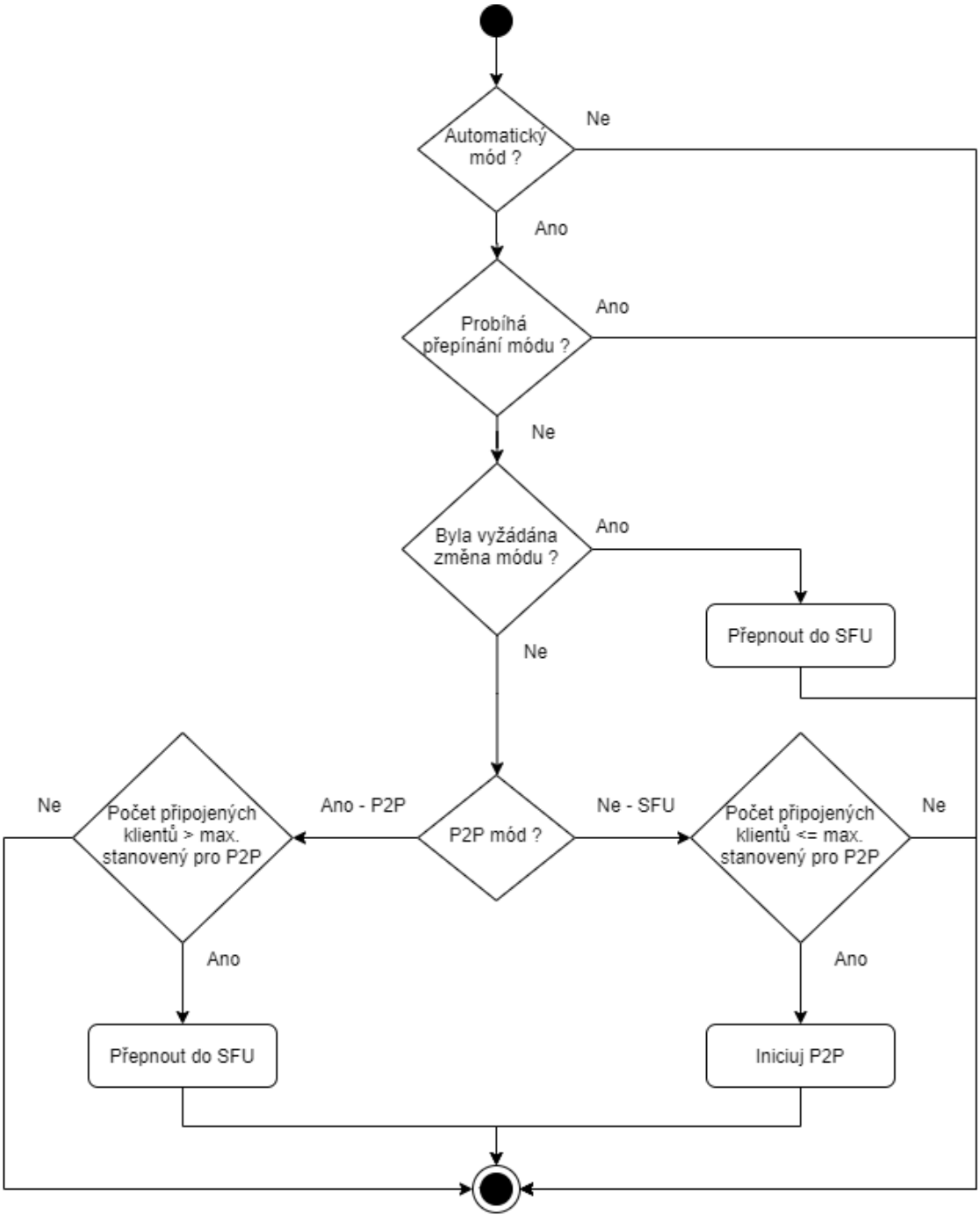
15 Seznam tabulek

16 Přílohy

16.1 Zdrojové soubory

Přílohou k práci jsou zdrojové soubory vytvořeného konferenčního systému na podkladě Mediasoup aplikace.

16.2 Obrázkové přílohy



Obrázek 27: Vyhodnocení automatické změny módu

Obrázek 28: Úspěšnost vyhledání síťového spojení pro WebRTC komunikaci pomocí ICE komponenty

16.3 Tabulky

P2P	Klient 1	Klient 2	Klient 3	Klient 4
Klient 1	x	7	7	7
Klient 2	4	x	7	6
Klient 3	6	7	x	7
Klient 4	8	8	7	x
SFU				
Klient 1	x	9	8	9
Klient 2	9	x	5	9
Klient 3	7	7	x	9
Klient 4	9	9	8	x

Tabulka 10: *Konference 4 účastníků*

P2P	Klient 1	Klient 2	Klient 3	Klient 4
Klient 1	x	9	7!	4
Klient 2	9	x	6!	8
Klient 3	9	9	x	9
Klient 4	9	9	3!	x
SFU				
Klient 1	x	9	8	9
Klient 2	7	x	8	8
Klient 3	9	9	x	9
Klient 4	9	9	8	x

Tabulka 11: *Opakování hodnocení – tbl č. 10*

P2P	Klient 1	Klient 2	Klient 3	Klient 4	Klient 5
Klient 1	x	0	6	0	0
Klient 2	0	x	0	0	0
Klient 3	6	0	x	0	0
Klient 4	0	0	0	x	3
Klient 5	1	0	0	6	x
SFU					
Klient 1	x	9	8	7	9
Klient 2	7	x	8	7	9
Klient 3	7	7	x	7	8
Klient 4	7	8	9	x	9
Klient 5	7	8	9	9	x

Tabulka 12: *Konference 5 účastníků*

P2P	Klient 1	Klient 2	Klient 3	Klient 4	Klient 5
Klient 1	x	0	0	8	0
Klient 2	0	x	0	0	0
Klient 3	0	0	x	0	0
Klient 4	9	0	0	x	9
Klient 5	9	0	0	9	x
SFU					
Klient 1	x	4	8	7	9
Klient 2	1	x	1	1	1
Klient 3	6	4	x	8	8
Klient 4	9	6	8	x	9
Klient 5	9	6	9	9	x

Tabulka 13: *Opakování hodnocení – tbl. č. 12*

P2P	Klient 1	Klient 2
Klient 1	x	8
Klient 2	8	x
SFU		
Klient 1	x	8
Klient 2	8	x

Tabulka 14: *Hodnocení 2 klientů konference*

P2P	Klient 1	Klient 2	Klient 3
Klient 1	x	8	6
Klient 2	8	x	5
Klient 3	7	6	x
SFU			
Klient 1	x	8	7
Klient 2	8	x	7
Klient 3	8	8	x

Tabulka 15: *Hodnocení 3 klientů konference – rozšíření případu tbl. č. 14*

P2P	Klient 1	Klient 2	Klient 3	Klient 4
Klient 1	x	8	5	0
Klient 2	8	x	5	2
Klient 3	5	2	x	4
Klient 4	0	7	4	x
SFU				
Klient 1	x	8	6	8
Klient 2	7	x	7	8
Klient 3	7	7	x	6
Klient 4	7	8	4	x

Tabulka 16: *Hodnocení 4 klientů konference – rozšíření případu tbl. č. 15*

P2P	Klient 1	Klient 2	Klient 3	Klient 4	Klient 5
Klient 1	x	8	6	0	0
Klient 2	8	x	5	3	0
Klient 3	7	4	x	6	0
Klient 4	0	8	4	x	4
Klient 5	6	5	0	0	x
SFU					
Klient 1	x	9	7	8	7
Klient 2	8	x	7	8	8
Klient 3	7	5	x	6	8
Klient 4	7	8	4	x	8
Klient 5	7	8	6	5	x

Tabulka 17: *Hodnocení 5 klientů konference – rozšíření případu tbl. č.* 16