

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки
Факультет радіоелектроніки та телекомунікацій
(повне найменування інституту, факультету)

Кафедра радіотехніки та телекомунікацій
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)
бакалавра
(ступінь вищої освіти)

на тему **СЕРВЕР ДЛЯ ВІДЕОДЗВІНКІВ НА БАЗІ ТЕХНОЛОГІЇ WebRTC**

Виконав: студент 4 курсу, групи РТ-917
спеціальності 172 «Телекомунікації та
радіотехніка»
(код і найменування спеціальності)

Освітня програма (спеціалізація) _____
«Інформаційні мережі зв'язку»
Контуганський Павло Валерійович
(прізвище та ініціали)

Керівник Самойлик С.С.
(прізвище та ініціали)

Рецензент Кудерметов Р.К.
(прізвище та ініціали)

2021 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет Інститут інформатики та радіоелектроніки, ФРЕТ
Кафедра Радіотехніка та телекомунікації
Ступінь вищої освіти Бакалавр
Спеціальність 172 «Телекомунікації та радіотехніка»
(код і найменування)
Освітня програма (спеціалізація) «Інформаційні мережі зв'язку»
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри РТТ 
к.т.н., доц Морозов С.В.
« » червня 2021 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА (КИ)

Контуганського Павла Валерійовича
(прізвище, ім'я, по батькові)
1. Тема проєкту (роботи) Сервер для відеодзвінків на базі технології WebRTC

керівник проєкту (роботи) Самойлик Сергій Сергійович, к.ф.-м.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «30» квітня 2021 року № 164

2. Строк подання студентом проєкту (роботи) 25 травня 2021 року

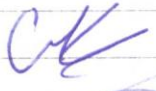
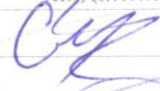
3. Вихідні дані до проєкту (роботи) Розробити додаток для надання послуг телемедицини на базі технології WebRTC

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Постановка задачі
2. Дослідження технології WebRTC
3. Розробка додатку для надання послуг телемедицини на базі технології WebRTC
4. Аналіз ефективності додатку

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація Microsoft PowerPoint з поясненням алгоритму проведення досліджень ,
фрагменти програмного коду додатку

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	Самойлик С.С., доцент		
нормоконтроль	Мороз Г.В., ст. викладач		

7. Дата видачі завдання «02» квітня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка задачі	09.04.2021	
2	Дослідження технології WebRTC	16.04.2021	
3	Дослідження відео- та аудіокодеків	23.04.2021	
4	Побудова серверної частини додатку	05.05.2021	
5	Побудова клієнтської частини додатку	12.05.2021	
6	Оформлення ІЗ та рецензії	19.05.2021	
7	Пр проходження нормо контролю, рецензування, антитіагіату, оформлення дублінського ядра	25.05.2021	

Студент(ка)


(підпис)

Костюганський П.В.

(прізвище та ініціал)

Керівник проєкту (роботи)


(підпис)

Самойлик С.С.

(прізвище та ініціал)

РЕФЕРАТ

ПЗ: 47 сторінок, 24 рисунків, 3 таблиць, 8 джерел

Мета роботи – розробка додатку для відеодзвінків з метою надання послуг телемедицини на базі технології WebRTC.

Об'єкт дослідження – технологія комунікації в реальному часі WebRTC.

Метод дослідження – теоретичний і практичний аналіз технології WebRTC.

Аналізуючи отримані у роботі теоретичні результати, було розроблено додаток для відеодзвінків. Результати аналізів та досліджень можуть бути використані у подальшому задля створення додатків для відеоконференцій для різних цілей, в тому числі для проведення освітніх заходів віддалено.

WEBRTC, ВІДЕОЗВ'ЯЗОК, БРАУЗЕР, ЧАТ, INTERNET

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ.....	7
ВСТУП	8
1 ПРОТОКОЛ ЗВ'ЯЗКУ ДЛЯ ОРГАНІЗАЦІЇ ВІДЕОКОНФЕРЕНЦІЇ.....	10
1.1 Порівняльний аналіз протоколів зв'язку	10
1.1.1 RTP.....	10
1.1.2 HLS.....	11
1.1.3 Flash	11
1.1.4 WebRTC.....	11
1.1.5 Саморобні протоколи.....	12
1.2 Історія WebRTC	12
1.3 Підтримка браузерами протоколу WebRTC	13
1.4 Сервіси що використовують WebRTC	14
2 ТЕХНОЛОГІЯ WEBRTC.....	16
2.1 Аудіокодек для додатку	19
2.1.1 Opus.....	19
2.1.2 G.711	20
2.1.3 G.722	20
2.1.4 iLBC	21
2.1.5 iSAC	21
2.2 Відеокодек для додатку.....	21
2.2.1 VP8.....	21
2.2.2 H.264	22
2.2.3 VP9.....	22
2.2.4 H.265	22
2.2.5 AV1	22
2.3 Архітектури зв'язку WebRTC	23
2.3.1 SFU.....	23
2.3.2 MESH.....	23

	6
2.3.3 MCU	24
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЕКТУ	26
3.1 Архітектура додатку WebRTC	26
3.2 Розробка серверної частини додатку	30
3.3 Розробка клієнтської частини додатку	33
3.4 Інтерфейс додатку WebRTC	37
3.5 Аналіз ефективності	37
ВИСНОВКИ.....	46
ПЕРЕЛІК ПОСИЛАНЬ	47

ПЕРЕЛІК СКОРОЧЕНЬ

ICE	—	інтерактивне встановлення з'єднання
STUN	—	назва мережевого протоколу
TURN	—	назва мережевого протоколу
HTML	—	мова розмітки гіпертексту
CSS	—	каскадні таблиці стилів
ПЗ	—	програмне забезпечення

ВСТУП

Медичні послуги стали більш інноваційними та орієнтованими на пацієнтів за допомогою інформаційних технологій.

Обмін інформацією між пацієнтами та лікарями став простішим завдяки комунікаційним технологіям що покращують послуги та технології охорони здоров'я. Пацієнти можуть легко звернутися до лікарів та сервісів охорони здоров'я послуг за допомогою засобів зв'язку. Більше того, можна обмінюватися медичними даними з/серед постачальників медичних послуг. В результаті цих послуг з'явилася нова галузь медицини, яка називається телемедицина, завдяки чому постачальники медичних послуг можуть віддалено діагностувати, контролювати та лікувати захворювання за допомогою засобів зв'язку. Телемедицина має на меті надати віддалену клінічну підтримку, подолати географічні бар'єри та поліпшити стан здоров'я, результати використання різних видів інформаційно-комунікаційних технологій. Телемедицина має багато засобів використання в медичній галузі та здебільшого називається телерадіологією, телепатологією, телетравматологією, телекардіологією, теледерматологією, телеофтальмологією, телепсихологією та телехірургією. Використання інформаційних технологій у медицині з кожним днем посилюється, а телемедицина – все більше стає необхідністю сучасних медичних послуг.

У галузі телемедицини доступно багато інструментів, і онлайн-консультації є одним із ключових елементів у цій галузі, що дозволяють зменшити витрати на лікарські послуги, подолати дефіцит медичної допомоги, щоб підтримувати кращу медичну допомогу вдома, а також покращувати охорону здоров'я, медичну освіту та пацієнта-лікаря та міжпрофесійні відносини. Веб-спілкування в реальному часі (WebRTC), проект з відкритим кодом, що забезпечує спілкування веб-браузерів у режимі реального часу, є одним із важливих інструментів у цьому відношенні. WebRTC пропонує безпечний одноранговий медіа та прямий обмін файлами

між браузерами без необхідності посередницьких серверов або сторонніх програм. Використовується інтерфейс програмування WebRTC (API) для розробки програм зв'язку з каналами передачі даних, аудіо та відео. API WebRTC має три основні елементи: (I) PeerConnection, (II) DataChannel та (III) MediaStream. PeerConnection встановлює безпосереднє спілкування між пристроями. DataChannel – це служба передачі даних між пристроями за допомогою двонаправленого однорангового зв'язку. MediaStream створює аудіо- та відеопотоки та керує їх вмістом.

WebRTC був використаний для онлайн-медичної консультації, що дозволяє проводити лікування та реабілітацію, а також для діагностики шляхом передачі рентгенологічних зображень. Наприклад, надсилання ультразвукових зображень органів через фреймворк WebRTC для віддаленої діагностики. У деяких випадках використовується WebRTC для моніторингу та обстеження людей похилого віку та пацієнтів з фізичними чи психічними вадами. Архітектура, заснована на WebRTC, була розроблена для віддаленого та постійного моніторингу людей похилого віку, дистанційного медичного обслуговування обстеження та екстрене втручання.

У роботі представлено програму онлайн-консультацій, засновану на власній технології WebRTC, і в цій роботі, відповідні протоколи зв'язку додатка детально пояснені. Висвітлений додаток протестовано для різних мереж, браузерів та мобільних систем шляхом оцінки якісних параметрів зв'язку. Таким чином, ефективність WebRTC у різних сценаріях використання була розкрита у перший раз. Більше того, вивчаються необхідні часи для встановлення та закінчення дзвінка, щоб виявити їх вплив на якість спілкування.

1 ПРОТОКОЛ ЗВ'ЯЗКУ ДЛЯ ОРГАНІЗАЦІЇ ВІДЕОКОНФЕРЕНЦІЇ

WebRTC (Web Real Time Communications) – це стандарт, який описує передачу поточкових аудіо, відео даних і контенту між браузерами (без установки плагінів необхідних розширень) або іншими підтримуючими його додатками в режимі реального часу (рис. 1.1). Дана технологія дозволяє перетворити браузер в кінцевий термінал відеоконференцзв'язку. Щоб почати спілкування, досить просто відкрити веб-сторінку конференції [1].

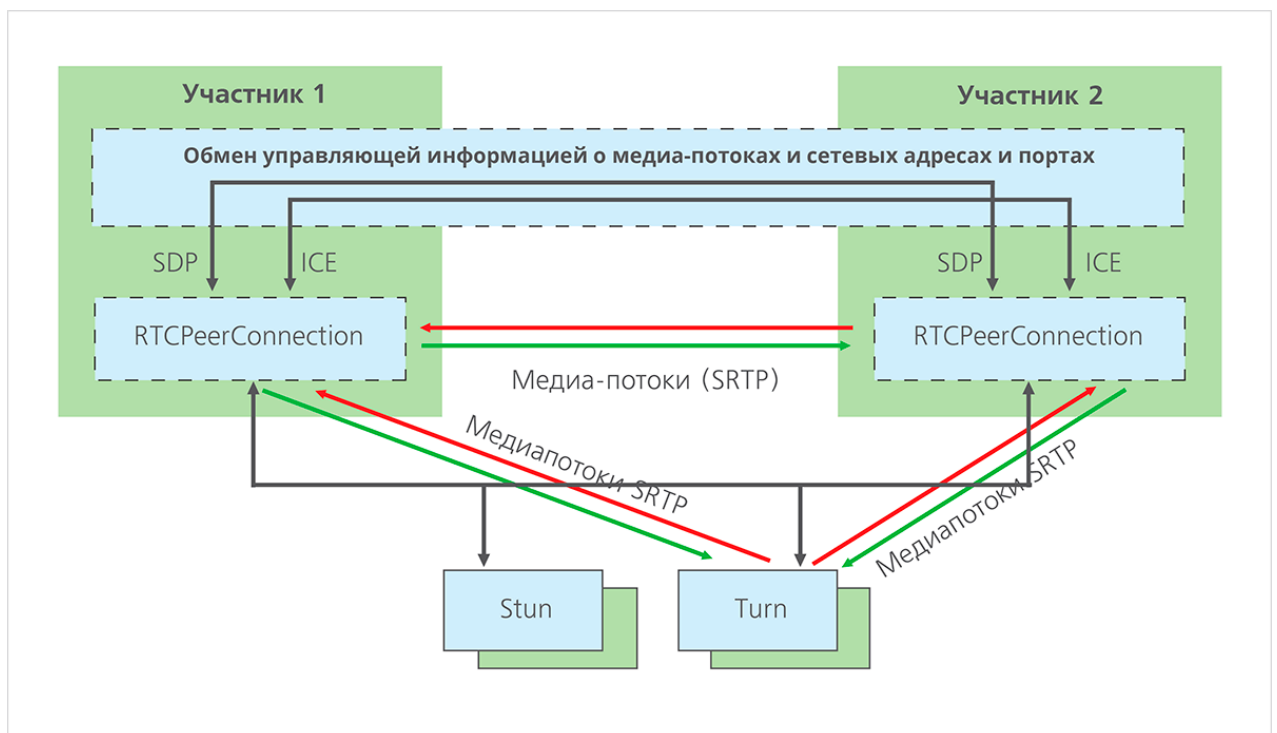


Рисунок 1.1 – Принцип роботи WebRTC

1.1 Порівняльний аналіз протоколів зв'язку

1.1.1 RTP

Браузери, як правило, не підтримують RTSP, тому потік буде конвертуватися для браузера через проміжний сервер. Можна було б використовувати RTMP який добре працює з Flash, але мета роботи –

розробити рішення для відеоконференцій без необхідності встановлення додаткових плагінів. Технологія RTSP і її похідні – добре працюють використовуючи Flash програвачі (RTMFP, RTMP). Так само – даний підхід має більші затримки на встановлення з'єднання ніж WebRTC.

1.1.2 HLS

Цей протокол має найбільшу затримку серед інших альтернатив. Оскільки запити використовують тільки стандартні транзакції HTTP, протокол дозволяє потоку долати міжмережеві екрани або проксі-сервера, пропускають HTTP-трафік, на відміну від протоколів на базі UDP, таких як RTP, а також доставляти до споживачів через існуючі CDN. Даний протокол підтримується багатьма браузера без необхідності встановлення додаткових плагінів, але його затримки значно вищі ніж у WebRTC.

1.1.3 Flash

Існує дуже багато рішень для організації відеоконференцій, де використовують Flash. Під всі браузери зроблені Flash розширення. Але це рішення вже поступово втрачає свою популярність так як Flash разом з легкістю імплементації тих або інших речей приніс в світ браузерів велику прогалину в безпеці. Необхідність установки Flash розширення, небезпечність самого Flash контенту і на додачу той факт що google chrome припинив повністю підтримувати Flash – зіграли проти цієї технології.

1.1.4 WebRTC

На відміну від попередніх технологій це рішення має низку плюсів:

а) технологія підтримується всіма основними браузера без установки будь-яких розширень;

б) забезпечує низьку затримку в порівнянні з іншими протоколами;

в) може використовуватися навіть без медіа серверів, забезпечуючи зв'язок р2р.

Основною проблемою цієї технології є різні фаєрволи і інші системи які можуть заблокувати отримання IP-адреси свого отримувача, а без цього неможливо налагодити зв'язок. Для вирішення цієї проблеми використовуються STUN/TURN сервери.

1.1.5 Саморобні протоколи

Саморобні протоколи дозволили б отримати рішення що володіє будь-якими характеристиками якими б ми захотіли, але дане рішення дуже часово витратне і для його правильної імплементації знадобилися б фахівці які можуть таке зробити.

1.2 Історія WebRTC

У травні 2010 року Google придбав Global IP Solutions або GIPS, компанію, що займається розробкою програмного забезпечення для VoIP та відеоконференцій, яка розробила багато компонентів, необхідних для RTC, таких як кодеки та методи ехоподавлення. Google відкрив джерело технологій GIPS та взаємодіяв із відповідними органами з питань стандартів IETF та W3C, щоб забезпечити галузевий консенсус. У травні 2011 року Google випустив проект з відкритим кодом для спілкування в режимі реального часу на основі браузера, відомий як WebRTC. Після цього продовжується робота з стандартизації відповідних протоколів в IETF та API браузера в W3C. У січні 2011 року компанія Ericsson Labs створила першу реалізацію WebRTC, використовуючи модифіковану бібліотеку WebKit. У жовтні 2011 року W3C опублікував свій перший проект специфікації. Важливими етапами WebRTC є перший відеодзвінок у різних браузерах

(лютий 2013 р.), перша передача даних між браузерами (лютий 2014 р.), а станом на липень 2014 р. Google Hangouts використовував WebRTC. Проект API W3C базувався на попередній роботі, виконаній в WHATWG. Його називали API ConnectionPeer, і в Ericsson Labs було створено концепцію попередніх стандартів. Робоча група WebRTC очікує, що ця специфікація суттєво зміниться на основі:

- результати поточних обмінів у супутній групі RTCWEB на IETF для визначення набору протоколів, які разом із цим документом визначають спілкування в реальному часі у веб-браузерах. Хоча жоден протокол передачі сигналів не передбачений, SIP через WebSockets (RFC 7118) часто використовується частково завдяки застосовності SIP до більшості передбачених сценаріїв зв'язку, а також наявності програмного забезпечення з відкритим кодом, такого як JsSIP;
- проблеми конфіденційності, що виникають при розкритті місцевих можливостей та місцевих потоків;
- технічні обговорення в групі, зокрема щодо впровадження каналів даних;
- досвід, отриманий в результаті раннього експериментування;
- відгуки інших груп та осіб.

У листопаді 2017 року специфікація WebRTC 1.0 перейшла з робочого проекту на рекомендацію кандидата. У січні 2021 року специфікація WebRTC 1.0 перейшла від Рекомендації кандидата до Рекомендації.

1.3 Підтримка браузерами протоколу WebRTC

WebRTC підтримується такими браузерами:

- настільний ПК:
 - a) Microsoft Edge 12+;
 - б) Google Chrome 28;
 - в) Mozilla Firefox 22+;

- г) Safari 11+;
- д) Opera 18+;
- е) Vivaldi 1.9;
- ж) Brave;
- Android:
 - а) Google Chrome 28+ (увімкнено за замовчуванням з 29);
 - б) Mozilla Firefox 24+;
 - в) Opera Mobile 12+;
 - г) ОС Chrome;
 - д) ОС Firefox;
 - е) BlackBerry 10;
- iOS:
 - а) MobileSafari / WebKit (iOS 11+);
 - б) Tizen 3.0.

1.4 Сервіси що використовують WebRTC

Google Meet – сервіс миттєвого обміну повідомленнями, а також проведення відео- та аудіодзвінків, випущений в 2017 році компанією Google. У браузерах, заснованих на Chromium (Google Chrome і ін.) використовується багато прихованих можливостей WebRTC, які не описані в документації і періодично з'являються першими в його рішеннях для Meet (як і в його попередника Hangouts). Так було із захопленням екрану, розмиванням фону, підтримкою апаратного кодування на деяких платформах.

Jitsi Meet – додаток з відкритим вихідним кодом, випущене компанією 8x8. Технологія Jitsi заснована на архітектурі Simulcast, що означає нестабільну роботу на слабких каналах зв'язку і високі вимоги до швидкості підключення на стороні сервера. Дозволяє проводити веб-конференції тільки в браузері і не має повноцінних клієнтських додатків для спільної роботи,

підтримані конференції з кількістю учасників не більше 75 (до 35 з високою якістю зв'язку). Для повноцінного використання Jitsi в корпоративному середовищі необхідна самостійна розробка і установка додаткового ПЗ.

BigBlueButton – це вільне програмне забезпечення для відеоконференцзв'язку. Особливий акцент розробники роблять на дистанційну освіту (присутні такі функції як інтерактивна дошка, показ контенту, підтримка опитувань і т. п.). Підтримує веб-конференції до 100 учасників [1].

2 ТЕХНОЛОГІЯ WEBRTC

Розглянемо роботу технології на прикладі дзвінка між двома абонентами через браузер:

а) користувач відкриває сторінку, яка містить WebRTC контент;

б) браузер запитує доступ до веб-камери і мікрофону, якщо це необхідно (рис. 2.1). Поки користувач не дозволить доступ до пристрою, воно не буде використовуватися. У випадках, коли це не є обов'язковим (наприклад, при перегляді трансляцій), ніяких додаткових дозволів не потрібно;

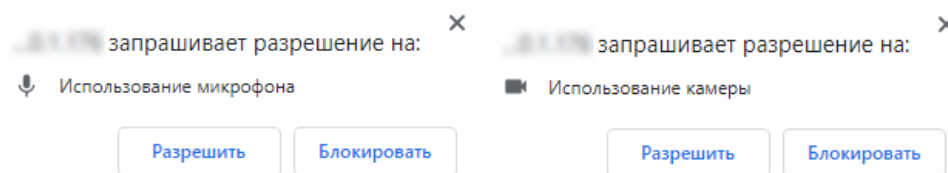


Рисунок 2.1 – Діалогові вікна запиту доступу до мікрофону та веб-камери

в) у браузері, ініціюванні з'єднання, формується SDP-пакет (рис. 2.2, 2.3). Це текстовий файл, що містить всю необхідну інформацію про параметри з'єднання. Наприклад, що буде передаватися (звук, відео, дані), за допомогою яких кодеків, які параметри підтримуються браузером і т. д. [5];

г) в залежності від реалізації технології, ініціатор з'єднання передає цей пакет іншим учасникам. Найчастіше для цього використовується сигнальний сервер і WebSocket протокол;

д) на приймаючій стороні браузер отримує SDP-пакет, а потім генерує подібний, але з урахуванням інформації з першого. Другий пакет відправляється назад, до ініціативної сторони. Тепер обидва клієнти вже мають мінімальне уявлення один про одного;

Global Lines
o=- 4611731400430051336 2 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE 0 1
a=msid-semantic: WMS lgsCFqt9kN2fVKw5wg3NKqGdATQoltEwOdMS
Audio Lines
m=audio 58779 UDP/TLS/RTP/SAVPF 111 103 104 9 0 8 106 105 13 126
c=IN IP4 217.130.243.155
a=rtcp:51472 IN IP4 217.130.243.155
ICE Candidates
a=candidate:1467250027 1 udp 2122260223 192.168.0.196 46243 typ host generation 0
a=candidate:1467250027 2 udp 2122260222 192.168.0.196 56280 typ host generation 0
a=candidate:435653019 1 tcp 1845501695 192.168.0.196 0 typ host tcptype active generation 0
a=candidate:435653019 2 tcp 1845501695 192.168.0.196 0 typ host tcptype active generation 0
a=candidate:1853887674 1 udp 1518280447 47.61.61.61 36768 typ srflx raddr 192.168.0.196 rport 36768 generation 0
a=candidate:1853887674 2 udp 1518280447 47.61.61.61 36768 typ srflx raddr 192.168.0.196 rport 36768 generation 0
a=candidate:750991856 2 udp 25108222 237.30.30.30 51472 typ relay raddr 47.61.61.61 rport 54763 generation 0
a=candidate:750991856 1 udp 25108223 237.30.30.30 58779 typ relay raddr 47.61.61.61 rport 54761 generation 0
ICE Parameters
a=ice-ufrag:Oyef7uvBlwafI3hT
a=ice-pwd:T0teqPLNQqOf+5W+Is+P2p16
DTLS Parameters
a=fingerprint:sha-256 49:66:12:17:0D:1C:91:AE:57:4C:C6:36:DD:D5:97:D2:7D:62:C9:9A:7F:B9:A3:F4:70:03:E7:43:91:73:23:5E
a=setup:actpass
a=mid:0
a=extmap:1 urn:ietf:params:rtp-hdrex:src-audio-level
a=extmap:3 http://www.webrtc.org/experiments/rtp-hdrex/abs-send-time
a=sendrecv
a=rtcp-mux
Codec Parameters
a=rtpmap:111 opus/48000/2
a=fmtp:111 minptime=10; useinbandfec=1
a=rtpmap:103 ISAC/16000
a=rtpmap:104 ISAC/32000
a=rtpmap:9 G722/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:106 CN/32000
a=rtpmap:105 CN/16000
a=rtpmap:13 CN/8000
a=rtpmap:126 telephone-event/8000
a=maxptime:60
SSRC Parameters
a=ssrc:3570614608 cname:4TOk42mSJXCKVla6
a=ssrc:3570614608 msid:lgsCFqt9kN2fVKw5wg3NKqGdATQoltEwOdMS 35429d94-5637-4686-9ecd-7d0622261ce8
a=ssrc:3570614608 mslabel:lgsCFqt9kN2fVKw5wg3NKqGdATQoltEwOdMS
a=ssrc:3570614608 label:35429d94-5637-4686-9ecd-7d0622261ce8

Рисунок 2.2 – Приклад SDP пакета ч. 1

Video Lines
m=video 60372 UDP/TLS/RTP/SAVPF 100 101 116 117 96
c=IN IP4 217.130.243.155
a=rtcp:64891 IN IP4 217.130.243.155
ICE Candidates
a=candidate:1467250027 1 udp 2122260223 192.168.0.196 56143 typ host generation 0
a=candidate:1467250027 2 udp 2122260222 192.168.0.196 58874 typ host generation 0
a=candidate:435653019 1 tcp 1518280447 192.168.0.196 0 typ host tcpType active generation 0
a=candidate:435653019 2 tcp 1518280446 192.168.0.196 0 typ host tcpType active generation 0
a=candidate:1853887674 1 udp 1518280447 47.61.61.61 36768 typ srflx raddr 192.168.0.196 rport 36768 generation 0
a=candidate:1853887674 1 udp 1518280447 47.61.61.61 36768 typ srflx raddr 192.168.0.196 rport 36768 generation 0
a=candidate:750991856 1 udp 25108223 237.30.30.30 60372 typ relay raddr 47.61.61.61 rport 54765 generation 0
a=candidate:750991856 2 udp 25108222 237.30.30.30 64891 typ relay raddr 47.61.61.61 rport 54767 generation 0
ICE Parameters
a=ice-ufrag:Oyef7uvBlwafI3hT
a=ice-pwd:T0teqPLNQqOf+5W+Is+P2p16
DTLS Parameters
a=fingerprint:sha-256 49:66:12:17:0D:1C:91:AE:57:4C:C6:36:DD:D5:97:D2:7D:62:C9:9A:7F:B9:A3:F4:70:03:E7:43:91:73:23:5E
a=setup:actpass
a=mid:1
a=extmap:2 urn:ietf:params:rtp-hdext:toffset
a=extmap:3 http://www.webrtc.org/experiments/rtp-hdext/abs-send-time
a=extmap:4 urn:3gpp:video-orientation
a=sendrecv
a=rtcp-mux
Codec Parameters
a=rtpmap:100 VP8/90000
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=rtcp-fb:100 goog-remb
a=rtpmap:101 VP9/90000
a=rtcp-fb:101 ccm fir
a=rtcp-fb:101 nack
a=rtcp-fb:101 nack pli
a=rtcp-fb:101 goog-remb
a=rtpmap:116 red/90000
a=rtpmap:117 ulpfec/90000
a=rtpmap:96 rtv/90000
a=fmtp:96 apt=100
SSRC Parameters
a=ssrc-group:FID 2231627014 632943048
a=ssrc:2231627014 cname:4TOk42mSjXCKVla6
a=ssrc:2231627014 msid:jgsCFqt9kN2fVkw5wg3NKqGdATQoItEwOdMS daed9400-d0dd-4db3-b949-422499e96e2d
a=ssrc:2231627014 mslabel:jgsCFqt9kN2fVkw5wg3NKqGdATQoItEwOdMS
a=ssrc:2231627014 label:daed9400-d0dd-4db3-b949-422499e96e2d
a=ssrc:632943048 cname:4TOk42mSjXCKVla6
a=ssrc:632943048 msid:jgsCFqt9kN2fVkw5wg3NKqGdATQoItEwOdMS daed9400-d0dd-4db3-b949-422499e96e2d

Рисунок 2.3 – Пример SDP пакета ч. 2

е) в залежності від реалізації, паралельно з попередніми кроками відбувається аналіз стану підключення до мережі. Клієнтам передається адреса STUN-сервера, який використовується, щоб дізнатися зовнішній IP-адресу пристрою. Він порівнюється з внутрішнім IP-адресою для того, щоб визначити чи використовується NAT в даному підключенні і, якщо так, то як маршрутизуються UDP-пакети. У більш складних випадках (наприклад, коли використовується подвійний NAT), застосовуються TURN-сервера. Вони, по суті, є ретрансляторами, перетворюючи з'єднання клієнт-клієнт (P2P) в клієнт-сервер-клієнт [5];

ж) якщо всі кроки пройдені успішно, то з'єднання встановлюється. Періодично викликається подія onicescandidate, яке передає інформацію про IP-адреси, налаштуваннях NAT, спроби підключення між клієнтами [1].

2.1 Аудіокодек для додатку

Кодеки WebRTC можна розділити на обов'язкові (браузери, які реалізують цю технологію повинні їх підтримувати) і додаткові (не включені в стандарт, але додані деякими браузерами).

Для стиснення аудіотрафіка в WebRTC використовуються обов'язкові кодеки (Opus і G.711) і додаткові (G.722, iLBC, iSAC) [8].

2.1.1 Opus

Opus – це аудіокодек з низькою затримкою кодування (від 2.5 мс до 60 мс), підтримкою змінного бітрейта і високим рівнем стиснення, що ідеально підходить для передачі потокового аудіо сигналу в мережах зі змінною пропускнуною спроможністю. Є основним аудіокодеком для WebRTC. Opus – гібридне рішення, що поєднує в собі кращі характеристики кодеків SILK (компресія голосу, усунення спотворень людської мови) і CELT (кодування звукової інформації). Кодек знаходиться у вільному доступі,

розробникам, які його використовують, не потрібно платити відрахування правовласникам. У порівнянні з іншими аудіокодеками, Opus, безсумнівно, виграє по безлічі показників. По ряду параметрів він перевершує досить популярні кодеки з низьким бітрейтом, такі, як MP3, Vorbis, AAC LC. Opus відновлює найбільш наближену до оригіналу "картину" звуку, ніж AMR-WB і Speex.

2.1.2 G.711

G.711 – застарілий голосовий кодек з високим бітрейтом (64 kbps), який найчастіше застосовується в системах традиційної телефонії. Основною перевагою є мінімальна обчислювальна навантаження через використання легких алгоритмів стиснення. Кодек відрізняється низьким рівнем компресії голосових сигналів і не вносить додаткової затримки звуку під час спілкування між користувачами.

G.711 підтримується великою кількістю пристроїв. Системи, в яких використовується цей кодек, легші в застосуванні, ніж ті, які засновані на інших аудіокодеках (G.723, G.726, G.728 і т. д.). За якістю G.711 отримав оцінку 4.2 в тестуванні MOS (оцінка в межах 4-5 є найвищою і означає гарну якість, аналогічне якості передачі голосового трафіку в ISDN і навіть вище) [7].

2.1.3 G.722

G.722 – є стандартом ITU-T, прийнятий в 1988 році, в даний час є безкоштовним. Може працювати зі швидкістю 48, 56 і 64 кбіт/с, забезпечуючи якість звуку на рівні G.711. І аналогічно G.711 є застарілим. Підтримується в Chrome, Safari і Firefox.

2.1.4 iLBC

iLBC (internet Low Bitrate Codec) – вузькосмуговий мовний кодек з відкритим вихідним кодом. Можна користуватися в Chrome і Safari. Через високий стиснення потоку при використанні даного кодека зростає навантаження на процесор.

2.1.5 iSAC

iSAC (internet Speech Audio Codec) – широкосмуговий мовний аудіокодек, раніше пропріетарний, який в даний час є частиною проекту WebRTC, проте не обов'язковий для використання. Підтримується в Chrome і Safari. У реалізації для WebRTC використовується адаптивний бітрейт від 10 до 52 кбіт/с з частотою дискретизації 32 kHz [1].

2.2 Відеокодек для додатку

Питання вибору відеокодека для WebRTC зайняло у розробників кілька років, в результаті в стандарт увійшли VP8 і H.264. Також існують реалізації необов'язкових відеокодеків (H.265, VP9, AV1).

2.2.1 VP8

VP8 – вільний відеокодек з відкритою ліцензією, відрізняється високою швидкістю декодування відеопотоку і підвищеною стійкістю до втрати кадрів. Кодек універсальний, його легко впровадити в апаратні платформи, тому дуже часто розробники систем відеоконференцзв'язку використовують його в своїх продуктах. Сумісний з браузерами Chrome, Edge, Firefox і Safari (12.1+).

2.2.2 H.264

Платний відеокодек H.264 став відомий набагато раніше свого побратима. Це кодек з високим ступенем стиснення відеопотоку при збереженні високої якості відео. Широка поширеність цього кодека серед апаратних систем відеоконференцзв'язку передбачає його використання в стандарті WebRTC. Сумісний з браузерами Chrome (52+), Edge, Firefox (у версіях 68+ для Android підтримка була припинена) і Safari [6].

2.2.3 VP9

VP9 – відкритий і безкоштовний стандарт стиснення відео, розроблений в 2012 році компанією Google. Є розвитком ідей, закладених в VP8 і в подальшому було розширено в рамках AV1. Сумісний з браузерами Chrome (48+) і Firefox.

2.2.4 H.265

H.265 – платний відеокодек, що є наступником H.264, що забезпечує таке ж візуальне якість при вдвічі меншому бітрейті. Це досягається за допомогою більш ефективних алгоритмів стиснення. В даний час цей кодек конкурує з безкоштовним AV1.

2.2.5 AV1

AV1 – відкритий кодек для стиснення відео, розроблений спеціально для передачі відео по мережі Інтернет. Підтримується в Chrome (70+) і Firefox (67+) [1].

2.3 Архітектури зв'язку WebRTC

Для здійснення зв'язку між декількома користувачами існує 3 архітектури (рис. 2.4).

2.3.1 SFU

Одним з учасників може виступити сервер, який налагодить спілкування «один на один» з першим учасником, отримає від нього дані, потім налагодить спілкування, теж в режимі «Один на один» з іншими учасниками і перешле їм ці дані. Тобто сервер тримає безліч peer-to-peer каналів зв'язку і просто копіює дані в усі ці канали. У термінології WebRTC такий сервер служить як Selective Forwarding Unit (SFU).

2.3.2 MESH

Однак, комунікація в групі можлива не тільки за допомогою SFU. Ви можете запитати – чому кожен не може посилати дані кожному, без будь-якого сервера, і будете праві. Це називається MESH-комунікація.

Тут є два ключових моменти:

- за MESH схемою кожен з учасників посилає і отримує $N-1$ потоків даних, де N -розмір групи. Тобто вимоги на upload speed для кожного учасника в MESH схемою збільшуються з ростом групи. Тоді як в SFU схемою кожен учасник завжди посилає тільки один потік. Не у кожного учасника швидкість з'єднання з мережею потягне посилку $N-1$ потоків;

- на жаль, за MESH схемою браузер робить аудіо-відео стиснення на кожен посилається потік. А це, як ми знаємо, найбільш ресурсно-витратний процес. За MESH схемою, наведеної вище, браузер кожного учасника буде стискати відео VP8/VP9/H264 кодеком 4 рази. Це викликано вимогою WebRTC до адаптації якості даних до якості каналу – якщо зв'язок не дуже,

то компресія стане сильніше. Що є добре, але спричиняє необхідність стискати незалежно на кожен канал (PeerConnection). На жаль, на звичайній машині браузер при стисненні 720p відео вже займає 30% процесора, тобто більше трьох каналів не потягне [8].

2.3.3 MCU

Кожен учасник посилає свої аудіо/відео потоки на сервер, звідки вони потрапляють до всіх інших, але у вигляді одного потоку. MCU об'єднує потоки від багатьох учасників в один вихідний потік і розсилає його іншим учасникам. Так само даний вид архітектури оптимізувати для інтеграції різних аудіо/відео ефектів, перетворень так як всі ці перетворення будуть застосовуватися лише до одного вихідного потоку. Відмінність від MESH полягає в тому що сервер додатково бере на себе повноваження по об'єднанню аудіо/відео контенту (дані операції ресурсовитратні), але з іншого боку – кожен клієнт не отримуватиме окремо потоки кожного учасника, а отримає один потік з аудіо/відео контентом всіх учасників. Якщо наприклад у нас спілкуються 3 людини то кожен з них буде посилати на сервер свої аудіо і відео потоки і отримуватиме тільки 1 потік який вже буде містити аудіо/відео всіх інших учасників [5].

Отже, при великій кількості учасників комунікації оптимальним варіантом є сервер архітектури SFU. Але так як ми розроблюємо додаток для дзвінків 1 на 1 оптимальною архітектурою буде MESH.

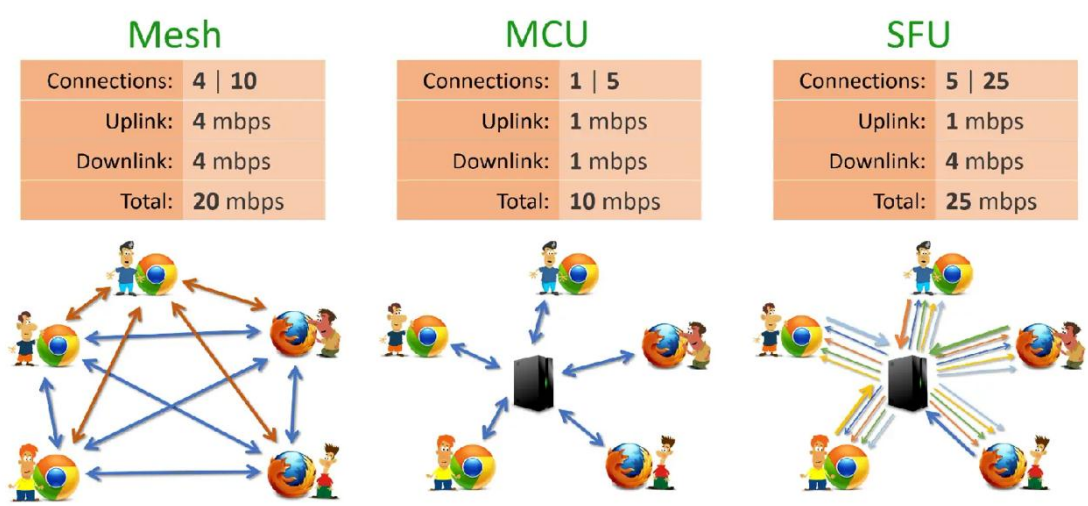


Рисунок 2.4 – Схеми архітектур зв'язку WebRTC

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Архітектура додатку WebRTC

Архітектура зв'язку WebRTC, що використовується в нашому додатку, показана на рисунку 3.1. У цій архітектурі обом пристроям необхідно надіслати необхідну інформацію про протокол через сервер socket.io для встановлення з'єднання.

Кандидати в Інтерактивне встановлення зв'язку (ICE) ініціюються та ідентифікуються через протокол для знаходження клієнтів. Для цього спочатку STUN (Session Traversal Utilities for Network Address Translator) сервер використовується для ідентифікації адрес клієнтських пристроїв. Якщо сервер STUN успішно визначає адреси, прямий носій інформації встановлюється зв'язок між однолітками. У деяких випадках сервер STUN не може забезпечити пряме підключення до пристрою. Тоді обхід за допомогою сервера перетворювача мережевих адрес (TURN) дозволяється отримати публічну адресу пристроїв та передача інформації між клієнтами.

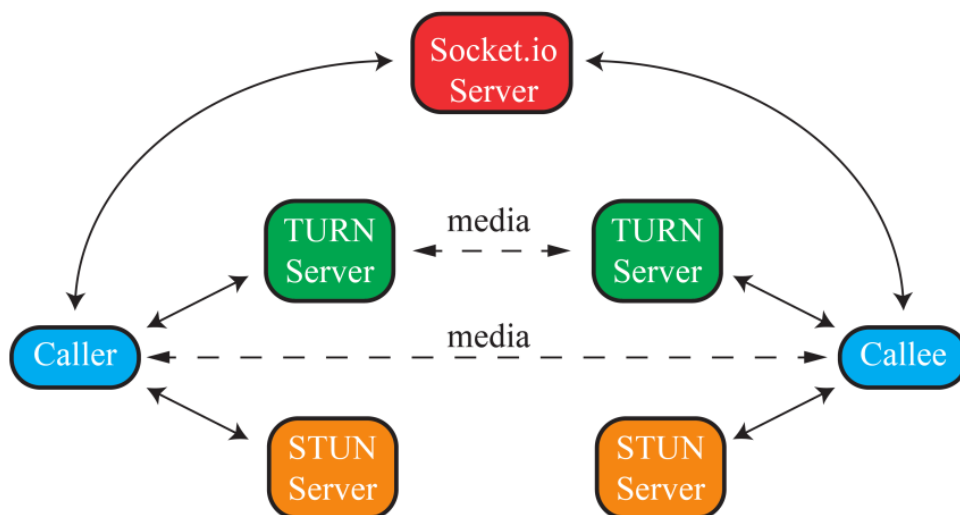


Рисунок 3.1 – Спілкування між клієнтами (викликаючим та викликаним) за допомогою серверів TURN, STUN та Socket.io

На рисунку 3.2 показано протокол зв'язку між клієнтами для встановлення процедури виклику, що складається з етапів ініціації, хендшейку, ICE переговорів та відбою. На етапі ініціації перші клієнти входять в додаток, і вони присвоюються різним кімнатам сервером `socket.io`. Ці кімнати використовуються для відправки комунікаційного статусу клієнтів. Таким чином, користувачі перебувають у режимі очікування, де вони можуть приймати або ініціювати дзвінки. Для цього в процесі виклику, одноранговий абонент починає дзвінок і реєструється в кімнаті другого однорангового абонента, що отримує дзвінок. Якщо кімната абонента заповнена (тобто підключено більше двох клієнтів) або порожня, процес виклику припиняється. Якщо ні, то абонент отримує вхідний дзвінок. Якщо абонент приймає дзвінок, процедура зв'язку WebRTC буде розпочато.

Фреймворк WebRTC використовується для надсилання/отримання медіа даних між пристроями. По-перше, клієнти запускають свої медіапотоки (відео та/або аудіо). Коли потоки готові, абонент створює однорангове з'єднання, додає свій локальний потік, створює пропозицію та надсилає цю пропозицію адресату. Для спілкування в чаті запропонована програма буде дотримуватись тієї ж процедури, приймаючи медіапотоки користувачів у готовому стані, не запускаючи медіа, програма запустить канал даних для передачі текстових повідомлень. Коли абонент отримує цю пропозицію, абонент також створює однорангове з'єднання, встановлює віддалений опис, додає свій локальний потік, створює відповідь та надсилає цю відповідь абоненту. Після отримання відповіді виклик встановлює віддалені описи. Поки встановлюється зв'язок клієнтів, ініціюються кандидати у ICE. Якщо створюється локальний кандидат ICE, протокол перевіряє створену пропозицію та додає віддаленого кандидата ICE. Якщо є один із цих перевірених пунктів, локальний ICE кандидат надсилається віддаленому користувачеві, і віддалений користувач додає цього кандидата як віддаленого кандидата ICE. Коли віддалений та локальний потоки готові, стан з'єднання ICE є контрольованим. Якщо стан ICE – підключений або завершений,

дзвінок буде встановлено, а екрани зв'язку для чату/аудіо/відео дзвінків будуть показані клієнтами. Коли ICE-з'єднання перебуває в перевірюваному або новому стані, протокол буде чекати і перевірити стан з'єднання ICE ще раз. Якщо з'єднання ICE знаходиться в іншому стані (відключено, не вдалося, або закрито), підключення буде перезапущено, закривши однорангове з'єднання. Якщо абонент або абонент завершить виклик під час дзвінку, однорангове з'єднання та канал передачі даних будуть закриті та медіа потоки будуть припинені. Крім того, абонент вийде з кімнати абонента та зареєструється у власній кімнаті. Отже, процедура завершення виклику буде завершена, і клієнти будуть готові прийняти або ініціювати новий дзвінок.

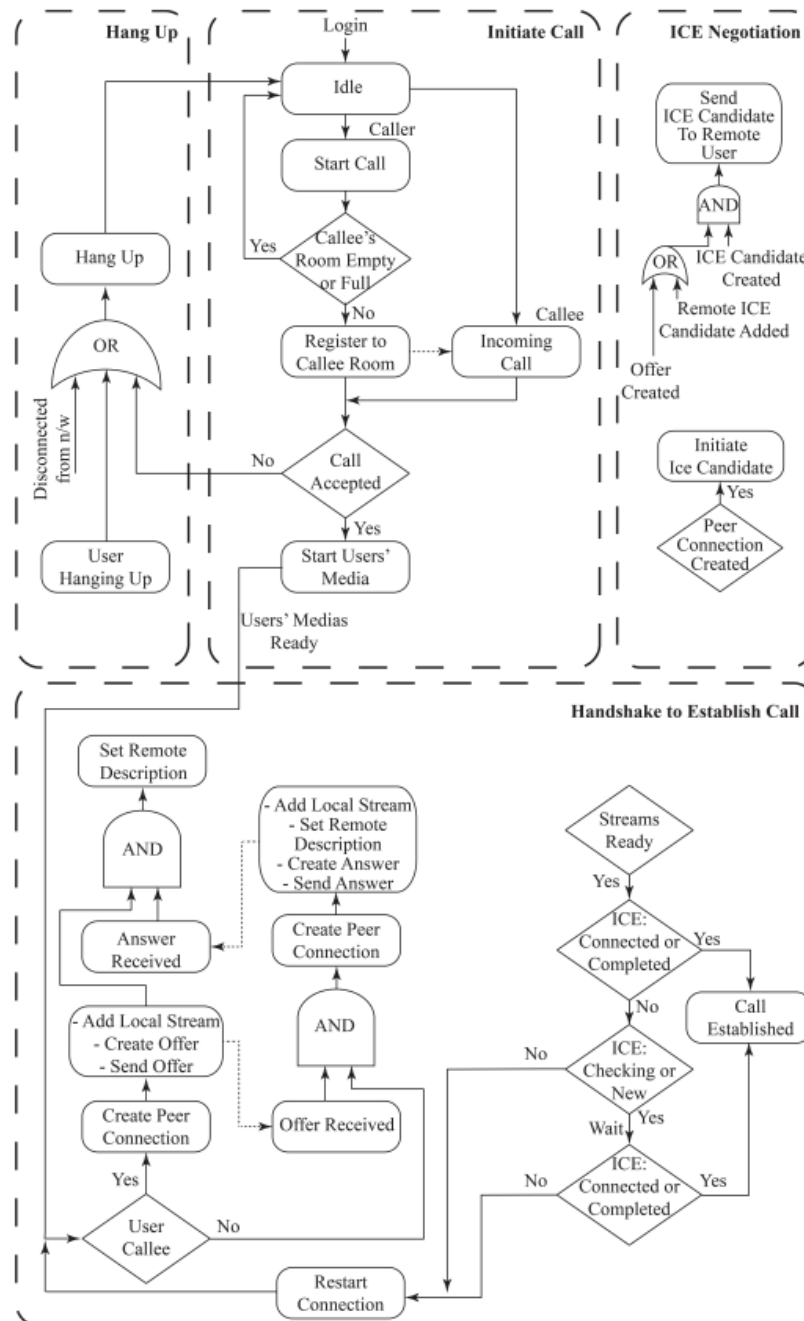


Рисунок 3.2 – Протокол спілкування між клієнтами

Схема машини станів надається для процедури складеної із етапів ініціації виклику, хендшейку, переговорів ICE та відбою.

3.2 Розробка серверної частини додатку

Для розробки сигнального серверу оберемо технологію Node.js [2]. Ця технологія стійка до великої кількості клієнтів а також дозволяє мати ізоморфний програмний код для клієнту та серверу. Розглянемо структуру серверного додатку (рис. 3.3).

```
node_modules
package.json
package-lock.json
server.js
```

Рисунок 3.3 – Структура проекту

Тут представлено 3 найголовніші елементи:

- директорія `node_modules` – містить залежності додатку, в нашому випадку `Socket.io` та його залежності;
- файл `package.json` – метадані проекту, перелік залежностей, сценарії запуску проекту (рис. 3.4);
- файл `server.js` – містить усю логіку серверного додатку.

```

{
  "name": "webrtc-biz",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/super0kontik/webrtc-biz.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/super0kontik/webrtc-biz/issues"
  },
  "homepage": "https://github.com/super0kontik/webrtc-biz#readme",
  "dependencies": {
    "socket.io": "4.1.2"
  }
}

```

Рисунок 3.4 – Файл package.json

Файл server.js складається з декількох основних функцій що реалізують логіку додатку (рис. 3.5-3.7).

```

function isUsernameUnique(name) {
  let isUnique = true;
  let i;

  for (i=0; i<connectionArray.length; i++) {
    if (connectionArray[i].username === name) {
      isUnique = false;
      break;
    }
  }

  return isUnique;
}

```

Рисунок 3.5 – Функція “isUsernameUnique”, що реалізовує логіку валідації імені користувача на унікальність

```
function sendToOneUser(target, msgString) {
    let i;

    for (i=0; i<connectionArray.length; i++) {
        if (connectionArray[i].username === target) {
            connectionArray[i].send(msgString);
            break;
        }
    }
}
```

Рисунок 3.6 – Функція “sendToOneUser”, що реалізовує відправлення приватного повідомлення єдиному користувачеві

```
IOserver.on( ev: 'connect', listener: function(socket : Socket<DefaultEventsMap, DefaultEventsMap, DefaultEventsMap> ) {
    if (!originIsAllowed(request.origin)) {
        socket.disconnect();
        log( text: "Connection from " + request.origin + " rejected.");
        return;
    }

    // Add the new connection to our list of connections.

    log( text: "Connection accepted from " + connection.remoteAddress + ".");
    connectionArray.push(socket);

    socket.clientID = nextID;
    nextID++;

    // Send the new client its token; it send back a "username" message to
    // tell us what username they want to use.

    const msg = {
        type: "id",
        id: socket.clientID
    };
    socket.send( args: 'connected', msg);
```

Рисунок 3.7 – Вхідна точка для клієнтів що підключаються

В якості TURN серверу пропонується обрати один з проектів з відкритим вихідним кодом, що добре оптимізовані та відлагоджені, наприклад coturn [3].

3.3 Розробка клієнтської частини додатку

Клієнтська частина додатку реалізована за допомогою HTML5, CSS, JavaScript. Ці технології є нативними тому розмір додатку є мінімальним.

Розглянемо структуру проекту (рис. 3.8):

```
adapter.js  
chat.css  
chatclient.js  
index.html  
socket.io.min.js
```

Рисунок 3.8 – Структура проекту

Тут представлено наступні файли:

- adapter.js – бібліотека для покращення сумісності платформ з технологією WebRTC;
- chat.css – опис стилів інтерфейсу додатку;
- chatclient.js – файл що реалізує логіку додатку;
- index.html – містить розмітку інтерфейсу додатку (рис. 3.9);
- socket.io.min.js – клієнтська бібліотека для з'єднання з сервером Socket.io.

```

<!doctype html>
<html>
<head>
  <title>WebRTC Calling App</title>
  <meta charset="utf-8">
  <link href="chat.css" rel="stylesheet">
  <script type="text/javascript" src="chatclient.js"></script>
  <script src="adapter.js"></script>
  <script src="socket.io.min.js"></script>
</head>
<body>
<div class="container">
  <p>Enter a username:
    <input id="name" type="text" maxlength="12" required
      autocomplete="username" inputmode="verbatim" placeholder="Username">
    <input type="button" name="login" value="Log in" onclick="connect()"></p>
  <p>Click a username in the user list to ask them to enter a one-on-one video chat with you.</p>
</div>
<ul class="userlistbox"></ul>
<div class="chatbox"></div>
<div class="camerabox">
  <video id="received_video" autoplay></video>
  <video id="local_video" autoplay muted></video>
  <button id="hangup-button" onclick="hangUpCall();" role="button" disabled>
    Hang Up
  </button>
</div>
<div class="empty-container"></div>
<div class="chat-controls">
  Chat:<br/>
  <input id="text" type="text" name="text" size="100" maxlength="256"
    placeholder="Say something meaningful..." autocomplete="off"
    onkeyup="handleKey(event)" disabled>
  <input type="button" id="send" name="send" value="Send" onclick="handleSendButton()" disabled>
</div>
</body>
</html>

```

Рисунок 3.9 – Вміст файлу index.html

Файл chatclient.js складається з блоків, необхідних для використання WebRTC та взаємодії з сигнальним сервером. Нижче представлено деякі з них (рис. 3.10-3.14).

```
function setUsername() {
    myUsername = document.getElementById( elementId: "name").value;

    sendToServer( msg: {
        name: myUsername,
        date: Date.now(),
        id: clientID,
        type: "username"
    });
}
```

Рисунок 3.10 – Функція “setUsername”, реалізує реєстрацію користувача під певним іменем

```
function handleSendButton() {
    const msg = {
        text: document.getElementById( elementId: "text").value,
        type: "message",
        id: clientID,
        date: Date.now()
    };
    sendToServer(msg);
    document.getElementById( elementId: "text").value = "";
}
```

Рисунок 3.11 – Функція “handleSendButton”, реалізує відправку повідомлення іншому користувачеві

```
async function createPeerConnection() {
    log( text: "Setting up a connection...");

    // Create an RTCPeerConnection which knows to use our chosen
    // STUN server.

    myPeerConnection = new RTCPeerConnection( configuration: {
        iceServers: [
            {
                urls: "turn:" + myHostname, // A TURN server
                username: "webrtc",
                credential: "turnserver"
            }
        ]
    });

    // Set up event handlers for the ICE negotiation process.

    myPeerConnection.onicecandidate = handleICECandidateEvent;
    myPeerConnection.oniceconnectionstatechange = handleICEConnectionStateChangeEvent;
    myPeerConnection.onicegatheringstatechange = handleICEGatheringStateChangeEvent;
    myPeerConnection.onsignalingstatechange = handleSignalingStateChangeEvent;
    myPeerConnection.onnegotiationneeded = handleNegotiationNeededEvent;
    myPeerConnection.ontrack = handleTrackEvent;
}
```

Рисунок 3.12 – Функція “createPeerConnection”, викликається для створення WebRTC з’єднання

```

function handleUserlistMsg(msg) {
    const listElem = document.querySelector( selectors: ".userlistbox");

    // Remove all current list members. We could do this smarter,
    // by adding and updating users instead of rebuilding from
    // scratch but this will do for this sample.

    while (listElem.firstChild) {
        listElem.removeChild(listElem.firstChild);
    }

    // Add member names from the received list.

    msg.users.forEach(function(username) {
        const item = document.createElement( tagName: "li");
        item.appendChild(document.createTextNode(username));
        item.addEventListener( type: "click", invite, options: false);
        listElem.appendChild(item);
    });
}

```

Рисунок 3.13 – Функція “handleUserlistMsg”, реалізовує зміну переліку користувачів при реєстрації нового або від’єднання існуючого користувача

```

function closeVideoCall() {
    const localVideo = document.getElementById( elementId: "local_video");
    log( text: "Closing the call");

    if (myPeerConnection) {
        log( text: "--> Closing the peer connection");

        myPeerConnection.ontrack = null;
        myPeerConnection.onnicecandidate = null;
        myPeerConnection.oniceconnectionstatechange = null;
        myPeerConnection.onsignalingstatechange = null;
        myPeerConnection.onicegatheringstatechange = null;
        myPeerConnection.onnotificationneeded = null;

        myPeerConnection.getTransceivers().forEach(transceiver => {
            transceiver.stop();
        });

        if (localVideo.srcObject) {
            localVideo.pause();
            localVideo.srcObject.getTracks().forEach(track => {
                track.stop();
            });
        }

        myPeerConnection.close();
        myPeerConnection = null;
        webcamStream = null;
    }

    document.getElementById( elementId: "hangup-button").disabled = true;
    targetUsername = null;
}

```

Рисунок 3.14 – Функція “closeVideoCall”, містить логіку завершення дзвінку

3.4 Інтерфейс додатку WebRTC

Інтерфейс користувача нашого додатку наведено на рис. 3.15 Після входу користувачів до програми вони можуть бачити користувачів для здійснення чатів, аудіо- та відеодзвінків.

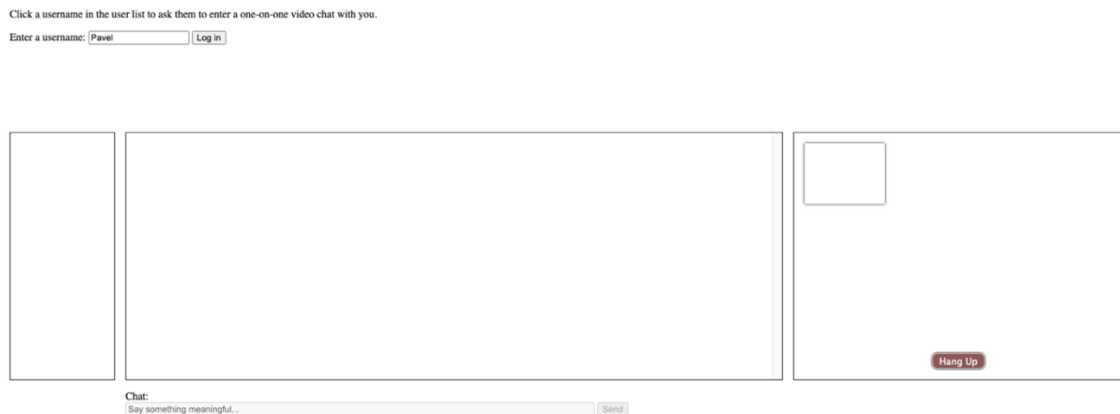


Рисунок 3.15 – Зовнішній вигляд програми

Зліва відображено перелік користувачів, по центру – вікно для відображення повідомлень та форма для відправлення нових повідомлень. Праворуч знаходиться місце для відображення відео співбесідника та відео з власної веб-камери.

Користувачі можуть закінчити спілкування натисканням червоної кнопки відключення.

3.5 Аналіз ефективності

Спілкування у відео, аудіо та чату в режимі реального часу на основі WebRTC було протестовано в локальній мережі з вимкненим троттлінгом імітуючим 3G, 4G, DSL з'єднання (профілі троттлінгу наведено у табл. 3.1) та без нього та для різних браузерів (версія Chrome 59.0.3071.86, Firefox версія 53.0.3).

Таблиця 3.1 – Параметри імітацій різних типів підключення

Профіль	Шв. завантаження, кб/с	Шв. вивантаження, кб/с	Затримка, мс
3G	1000	750	40
4G	4000	3000	20
DSL	2000	1000	5

Дані про ефективність зв'язку були отримані через GetStats API, який надає доступ до даних WebRTC у браузерах Chrome. Використовуючи цей API, було проаналізовано (I) час зворотного рейсу (RTT), який є вимірюваним часом, що минув у мілісекундах від часу надсилання запиту браузера до часу прийому відповіді сервера браузером; (II) отримані пакети, тобто загальна кількість пакетів, отриманих для цього з'єднання через мережу; (III) втрачені пакети, тобто загальна кількість втрачених пакетів для цього з'єднання; та (IV) частота кадрів (кадрів в секунду), лише для відеозв'язку. Ми також проаналізували втрату пакетів, яка є відсотком втрачених пакетів відносно загальної кількості отриманих пакетів. Більше того, виміряли загальний час підключення і зависання. Загальний час підключення вказує на час, що минув між початком користувацького медіа та станом встановленого дзвінка (рис. 3.16-3.18). Час зависання – це час завершення етапу відбою.

Загальний час, с

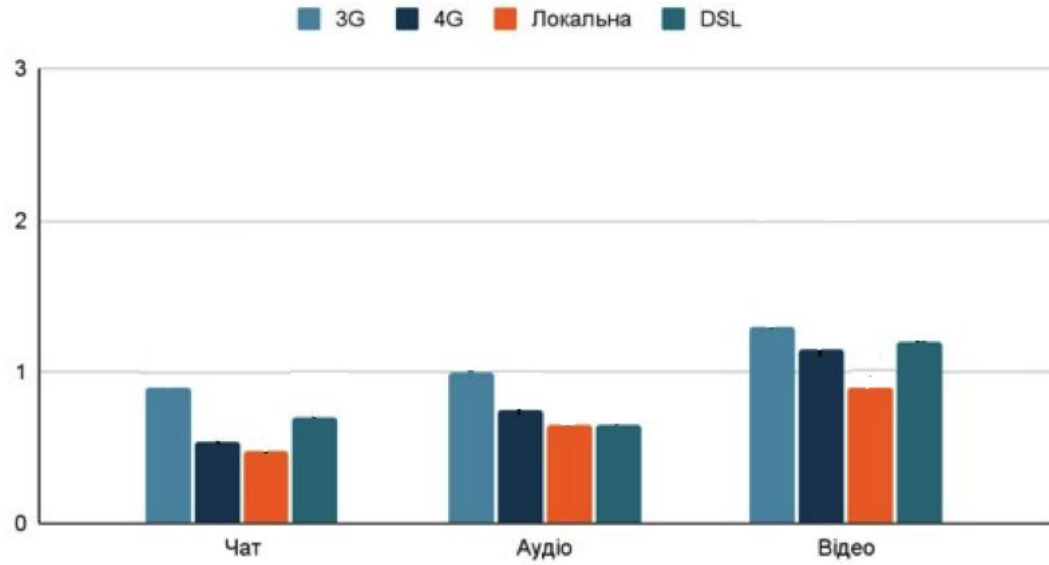


Рисунок 3.16 – Загальний час з'єднання з'єднань
для різних типів дзвінків у 3G, 4G, локальному та DSL

Час зависання, с

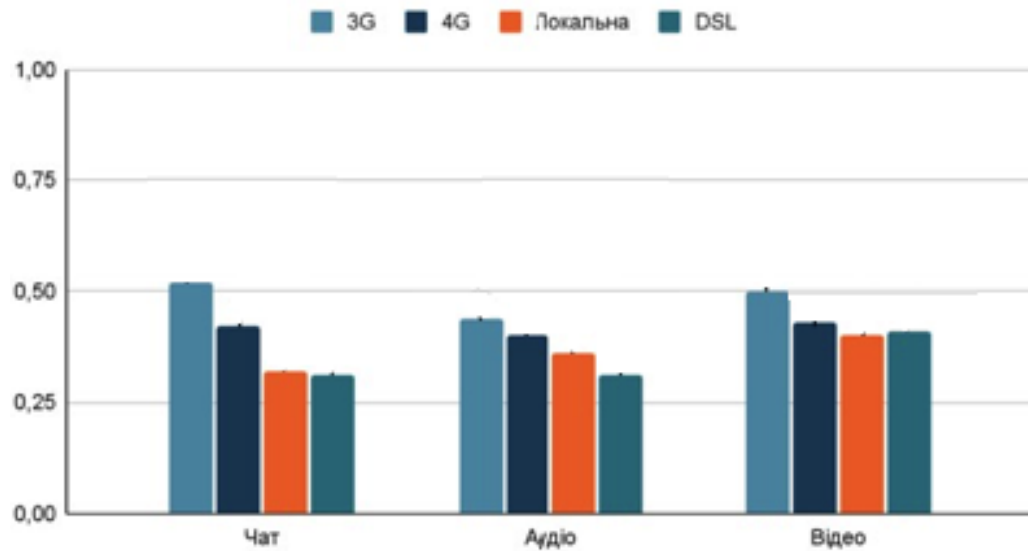


Рисунок 3.17 – Час зависання з'єднань
для різних типів дзвінків у 3G, 4G, локальному та DSL

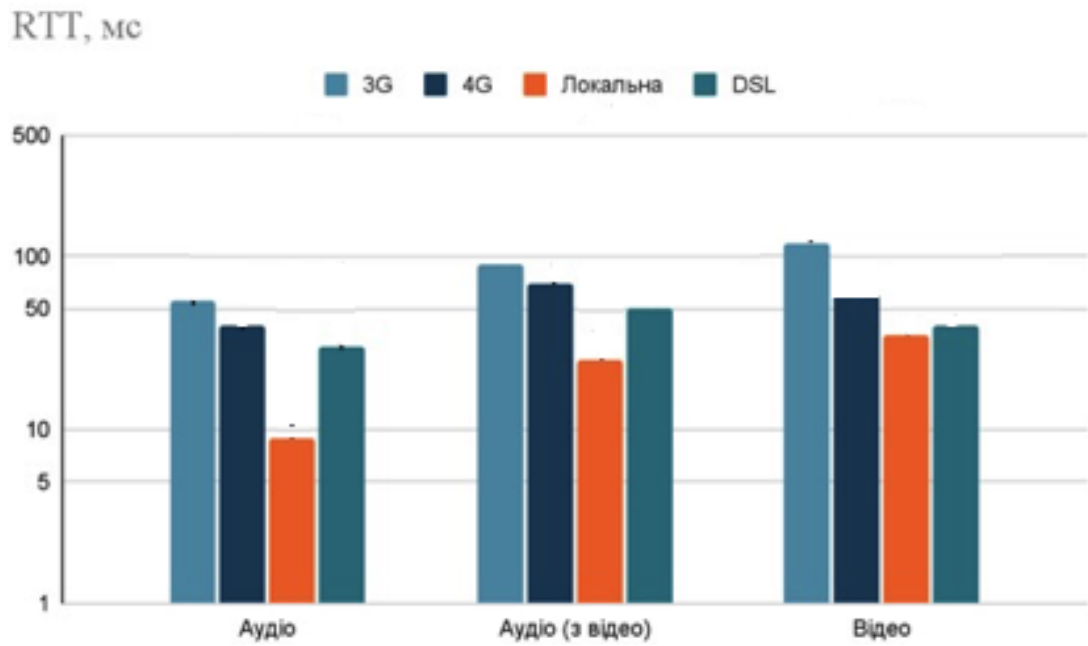


Рисунок 3.18 – RTT з'єднань
для різних типів дзвінків у 3G, 4G, локальному та DSL

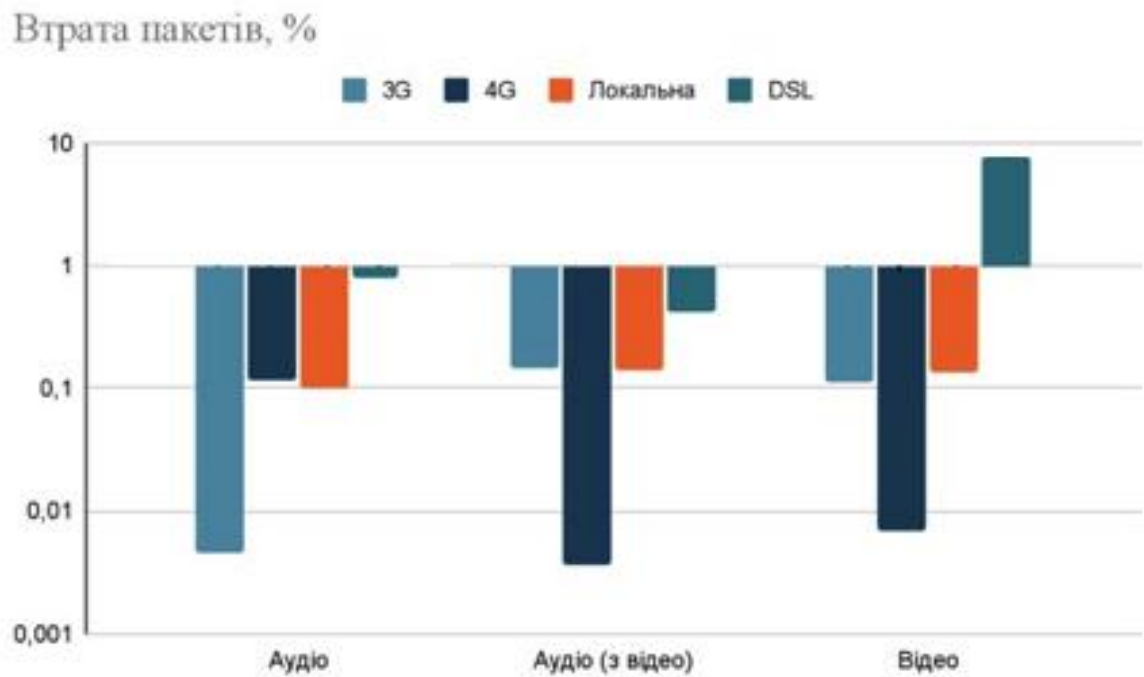


Рисунок 3.19 – Втрати пакетів з'єднань
для різних типів дзвінків у 3G, 4G, локальному та DSL

Було протестовано роботу різних типів спілкування (чат, аудіо, відео) за допомогою шести різних дзвінків. Кожен дзвінок тривав 5 хв. Під час дзвінка дані про ефективність дзвінка збиралися з інтервалом у 5 с. Для дзвінків у чаті користувачі щосекунди надсилали повідомлення довжиною 20 символів. Для відеодзвінків – відео розміром 640×480 у форматі 30 fps було встановлено для потокової передачі. Через 5 хв дзвінок було автоматично завершено. Використовувався браузер Chrome для аналізу різних характеристик підключення нашого додатку. Дані абонента використовувались для всіх тестів. Усі дані щодо ефективності наведені в таблицях і на рисунках.

Дані про ефективність програми були проаналізовані для різних типів з'єднань, таких як 3G, 4G, DSL, та локальні. Технологія 4G нещодавно набрала обертів і почала замінювати технологію 3G. У 2018 році 29% та 45% абонентів мобільного зв'язку використовують мережі 3G та 4G відповідно. Хоча використання технології 4G збільшується і навіть впроваджується технологія 5G, прогнозується, що до 2024 року 3G технологія все ще становитиме 17% абонентів мобільного зв'язку у всьому світі. Тим часом очікується, що 4G досягне 56% всіх абонентів мобільного зв'язку. Кожен перевірений тип підключення має різні особливості архітектури мережі, такі як термінологія швидкості передачі даних, безпеки, доступу та передачі даних. Більше того, мережа радіодоступу може змінюватися для різних архітектур мережі. Наприклад, 3G та 4G – це мобільні мережі, які базуються на UMTS та LTE, відповідно.

Отже, 3G та 4G мають різні діапазони частот, пропускну здатність каналів, протоколи, типи каналів тощо.

Крім того, різні мережі мають різну швидкість підключення до Інтернету, яку вимірювали як $8,97 \pm 1,59$ Мбіт/с, $26,07 \pm 3,45$ Мбіт/с та $13,18 \pm 2,55$ Мбіт/с для з'єднань 3G, 4G та DSL відповідно. Так як локальні тести підключення проводились в одній і тій же локальній мережі, швидкість з'єднання, як очікувалося, була вищою.

3G призвів до збільшення тривалості загального часу з'єднання, його збільшення значно відрізнялося від 4G та локальних мереж для чату та відеодзвінків. Однак для аудіодзвінків загальна сума час з'єднання не суттєво відрізнявся для різних типів з'єднань. Досягнуто загального часу підключення $1,38 \pm 0,24$ с для відеодзвінків у 3G. З іншого боку, цей час підключення був зменшений до $0,97 \pm 0,12$ с протягом локального зв'язку. Оскільки у відеодзвінках був величезний потік відеомедіа, ці дзвінки мали більший показник часу з'єднання, ніж чат та аудіо дзвінки. Хоча 3G показував найдовший час відбою для кожного типу дзвінка, час зависання залишався менше 0,53 с для різних типів дзвінків. Значення часу відбою 3G були суттєво відмінні від інших мереж для чату та звукових дзвінків, і вони також суттєво відрізняються від локальної та DSL-мережі для відеодзвінків. DSL та локальні мережі мали менше часу відбою, ніж мережі 3G та 4G. RTT значення мали великі відмінності у всіх мережах для всіх типів викликів. Під час звукових дзвінків мережа 3G досягла значення RTT $78,42 \pm 5,76$ мс, тоді як локальна мережа залишалася на рівні $11,86 \pm 2,48$ мс. Для аудіо-пакетів відеодзвінків, мережа 3G досягла $55,04 \pm 6,41$ мс RTT, а локальна мережа залишилася на рівні $5,82 \pm 2,54$ мс.

Що стосується відеопакетів відеодзвінків, мережа 3G досягла $77,98 \pm 37,99$ мс RTT-значень а локальна мережа залишалася на рівні $18,31 \pm 22,42$ мс. Тим часом локальна мережа показала найменші значення RTT для пакетів аудіовикликів, аудіопакетів відеодзвінків та відеопакетів відеодзвінків відповідно. RTT мав максимальні значення для 3G-з'єднань і становили $77,98 \pm 38,00$ мс для відеопакетів відеодзвінків. Це значення значно нижче 100 мс, що є показником для оптимальної якості медіа.

Якість зв'язку можна проаналізувати за допомогою RTT та втрат пакетів . Визначено високий, середній, і низька якість зв'язку, як показано нижче: (I) RTT менше 50 мс, а втрата пакетів менше 0,5%; (II) або RTT перевищує 50 мс, а втрата пакетів – менше 0,5%, або RTT – менше 50 мс та втрата пакетів перевищує 0,5%; (III) RTT перевищує 50 мс, а втрата пакетів

перевищує 0,5%. Щодо цього, 29% та 63,5% від загальної кількості дзвінків мають високу та середню якість відповідно для відеозв'язку, тоді як в аудіозв'язку 83,5% та 15% загальних дзвінків мають високу та середню якість відповідно. Загальний час і значення часу зависання досліджували щодо якості відеозв'язку. Коли загальний час був менше 1 с, було досягнуто високоякісного зв'язку для всіх дзвінків (табл. 3.2). Більшість дзвінків (98%) було принаймні середньої якості зв'язку, коли загальний час був від 1 до 2 с. За цей 1-2 с загальний інтервал, 26%, 72% та 2% дзвінків були відповідно високою, середньою та низькою якістю. Поза цим інтервалом, відсоток низькоякісних дзвінків (33,5%) суттєво збільшився, і лише 16,5% дзвінків мали високу якість. Для значень часу зависання менше 0,3 с, 25% та 75% дзвінків були високою та середньою якістю, відповідно (таблиця 3.3).

Таблиця 3.2 – Якість відеодзвінків відносно відсотка дзвінків через різні загальні інтервали часу

Загальний час	0-1 с	1-2 с	>2 с
Висока якість *	100%	26%	16.5%
Середня якість **	0	72%	50%
Низька якість ***	0	2%	33.5%

Примітки:

* Втрата пакетів <0,5% і RTT <50 мс

** (Втрата пакетів <0,5% і RTT > 50 мс) або (Втрата пакетів > 0,5% і RTT <50 мс)

*** Втрата пакетів > 0,5% і RTT > 50 мс.

Таблиця 3.3 – Якість відеодзвінків відносно відсотка дзвінків через різні інтервали часу зависання

Час зависання	0–0.3 с	0.3–0.6 с	>0.6 с
Висока якість *	25%	28%	37.5%
Середня якість **	75%	66%	37.5%
Низька якість ***	0	6%	25%

Примітки:

* Втрата пакетів <0,5% і RTT <50 мс

** (Втрата пакетів <0,5% і RTT > 50 мс) або (Втрата пакетів > 0,5% і RTT <50 мс)

*** Втрата пакетів > 0,5% і RTT > 50 мс.

Співвідношення високої та середньої якості було трохи знижено до 94% (тобто 28% та 66% від загальної кількості дзвінків високої та середньої якості), коли час зависання становив від 0,3 до 0,6 с. Після цього інтервалу часу співвідношення низької якості зв'язку було збільшено до 25%. Для часу зависання не було знайдено чіткого інтервалу часу для розрізнення високої і середньої якості спілкування. На основі цих результатів (таблиця 3.2 та таблиця 3.3) загальний час може дати кращу ймовірність прогнозування якості зв'язку для певних інтервалів часу, ніж час зависання.

Наприклад, низький загальний час (<1 с) свідчив про високу якість зв'язку у всіх тестах та помірний загальний час (1-2 с) вказував на високу або середню якість спілкування у більшості тестів (98%). Таким чином, якість спілкування корелюється із загальним часом, і його можна просто проаналізувати, використовуючи параметр загального часу без постійного моніторингу параметрів продуктивності WebRTC, отриманих з API GetStats.

Додаток було протестовано за допомогою 84 різних неавтоматизованих та некерованих відеодзвінків. У цих викликах середні

значення загального часу підключення та часу зависання були визначені як 6,03 с та 0,55 с відповідно.

У цих тестах 6% дзвінків мали загальний час менше 1 с, що свідчить про високу якість зв'язку (табл. 3.2), тоді як 47,6% дзвінків мали загальний час менше 2 с. Отже, очікувалося що майже половина відеодзвінків матиме високу ймовірність досягти щонайменше середньої якості зв'язку на основі результатів у таблиці 3.2, що свідчить про те, що наш додаток має достатню якість для онлайн-консультацій.

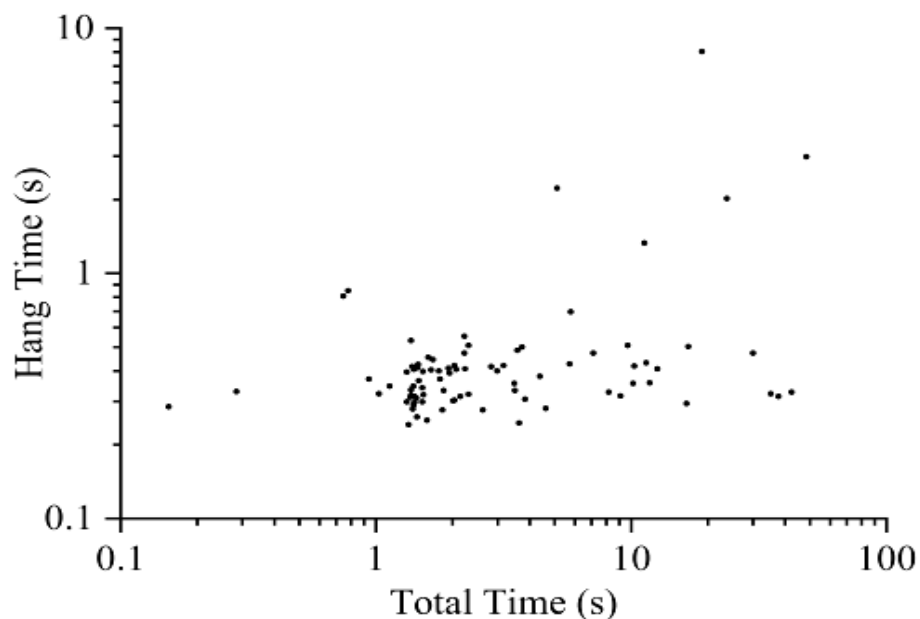


Рисунок 3.20 – Відповідний загальний час з'єднання та час зависання 84 різних відеодзвінків, здійснених за допомогою додатку

ВИСНОВКИ

Була представлена онлайн-програма консультацій на базі WebRTC, яка була протестована в різних мережах, браузерах та мобільних операційних системах вперше, щоб показати свою ефективність у різних сценаріях використання.

Звукові дзвінки показали хорошу якість зв'язку у всіх тестах. З іншого боку, відеодзвінки з Android, Chrome, iOS, Opera і Safari в мережі DSL мали погіршення параметрів зв'язку, такі як RTT і втрата пакетів. Це вказує на проблеми спілкування, які погіршують якість зв'язку.

Наприклад, проблеми з заморожуванням відео можна спостерігати в iOS з високим RTT (> 100 мс) і втратою пакетів ($> 1\%$) значення. Більше того, загальні значення часу показали хороший зв'язок із якістю зв'язку, що може виключити постійний моніторинг RTT і параметрів втрати пакетів. Для телемедичних послуг хороша трансляція є необхідною умовою для безперебійного обслуговування. Результати тестів показали хороші умови спілкування для типу зв'язку та між браузерами. Отже, ця представлена програма може бути використана різним способом в різних мережах та браузерах. Мережеві технології 5G демонструють значний приріст продуктивності з точки зору надійності з'єднання, спектральної ефективності, пропускної здатності системи та діапазону передачі. При розгортанні мережі 5G, додаток можна перевірити в мережі 5G, щоб виявити вплив 5G на якість спілкування. Як майбутню роботу запропоновану заявку можна протестувати з пацієнтами та охороною здоров'я постачальників, щоб розкрити його придатність для онлайн-медичної консультації. Більше того, зображення надвисокої роздільної здатності є необхідним для дерматологічної діагностики. Очевидно, що WebRTC може відкрити нові шляхи в галузі телемедицини, наприклад, розробку безпечних медичних програм для конкретних захворювань можливість телеконсультацій.

ПЕРЕЛІК ПОСИЛАНЬ

1. WebRTC – Что это и как работает? WebRTC сервер своими руками. [Електронний ресурс]: Режим доступу: <https://trueconf.ru/webrtc.html>.
2. Node.js. [Електронний ресурс]: Режим доступу: <https://nodejs.org/uk/>.
3. coturn/coturn: coturn TURN server project. [Електронний ресурс]: Режим доступу: <https://github.com/coturn/coturn>.
4. Тукі Дж. Порівняння окремих засобів при аналізі дисперсії [Текст] / Дж. Тукі // Біометрія – Вашингтон, 1949. – Вип. 5. – С. 97-114.
5. Johnston A. B. WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web [Text] / A. B. Johnston, D.C. Burnett – Publisher «Digital Codex LLC», 2012. – 170 p.
6. Sergiienko A. B. WebRTC Blueprints [Text] / A. B. Sergiienko – Publisher «Packt», 2014. – 176 p.
7. Roy R. R. Handbook of SDP for Multimedia Session Negotiations: SIP and WebRTC IP Telephony [Text] / R. R. Roy – Publisher «CRC Press», 2018. – 924 p.
8. Loreto S. Real-Time Communication with WebRTC: Peer-to-Peer in the Browser / S. Loreto – Publisher «O'Reilly Media», 2014. – 164 p.