Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
Факультет інформаційних технологій
Кафедра безпеки інформації та телекомунікацій

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеню бакалавра

студента *Паляничка Ігор Віталійович*

академічної групи *172-17ск-1*

спеціальності *172 Телекомунікації та радіотехніка*

спеціалізації[1]

за освітньо-професійною програмою *Телекомунікації та радіотехніка*

на тему *Дослідження сервісів потокового мовлення*

Керівники	Прізвище, ініціали	Оцінка за шкалою		Під-пис
		рейтинговою	інституційною	
Кваліфікаційної роботи	к.ф.-м..н., доц. Магро В.І.			
розділів:				
спеціальний	к.ф.-м..н., доц. Магро В.І.			
економічний	к.е.н., доц. Романюк Н.М.			
Рецензент				
Нормоконтролер	к.ф.-м.н., проф. Гусєв О.Ю.			

Дніпро
2020

ЗАТВЕРДЖЕНО:
завідувач кафедри
безпеки інформації та телекомунікацій
_____ д.т.н., проф. Корнієнко В.І.

«_____»_____20___року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня бакалавра

студенту ***Паляничка Ігор Віталійович*** **академічної групи** ***172-17ск-1***
(прізвище ім'я по-батькові) (шифр)

спеціальності *172 Телекомунікації та радіотехніка*
(код і назва спеціальності)

на тему ***Дослідження сервісів потокового мовлення***

затверджену наказом ректора НТУ «Дніпровська політехніка» від _____ № _____

Розділ	Зміст	Термін виконання
Розділ 1	Розглянути загальні характеристики медіасервісних платформ та технології які забезпечує мінімальну затримку доставки контенту	
Розділ 2	Дослідити медіасервісні платформи та технологію яка забезпечує мінімальну затримку доставки контенту	
Розділ 3	Провести розрахунок вартості методики організації потокового мовлення за допомогою медіасервісної платформи	

Завдання видано _____ **Магро В.І.**
(підпис керівника) (прізвище, ініціали)

Дата видачі: _____

Дата подання до екзаменаційної комісії: _____

Прийнято до виконання _____ **Паляничка І.В.**
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 60 с., 9 рис., 6 табл., 1додаток, 18 джерел.

Об’єкт дослідження – медіа платформа потокового мовлення.

Предмет дослідження – методики мінімізації затримки в сервісах потокового мовлення за допомогою програмного забезпечення.

Метою роботи є зменшення затримки доставки контента в сервісах потокового мовлення шляхом вибору оптимального програмного забезпечення.

Методи дослідження: математичного моделювання, імітаційного моделювання, а також методи обчислювальної математики та статистичного моделювання.

В першому розділі кваліфікаційної роботи наведена загальна характеристика медіасервісних платформ та технології яка забезпечує мінімальну затримку.

В спеціальній частині виконано дослідження медіасервісних платформ та технології яка забезпечує мінімальну затримку.

В економічній частині проведено розрахунок вартості методики організації потокового мовлення за допомогою медіасервісної платформи.

Технічним результатом отриманим в кваліфікаційній роботі є мінімізація затримки доставки медіаконтента за допомогою медіасервера Kurento та технології WEBRTC.

WEBRTC, МЕДІАСЕРВІСИ ПОТОКОВОГО МОВЛЕННЯ, ЗАТРИМКА, KURENTO, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

ABSTRACT

Explanatory note: 60 pp., 9 fig., 6 tab., 1 additional deposit, 18 sources.

The object of study – is a media platform of streaming.

The subject of research – methods of minimizing delays in streaming services using software.

The aim of the work is to reduce the delay in streaming services by choosing the optimal software.

Research methods: mathematical modeling, simulation modeling, as well as methods of computational mathematics and statistical modeling.

The first section of the qualification work provides a general description of the media service platform and technology that provides minimal delay.

In a special part, a study of the media service platform and technology that provides minimal delay.

In the economic part, the calculation of the cost of the method of organizing streaming broadcasting using a media service platform.

The technical result obtained in the qualification work is to minimize the delay of streaming broadcast using Kurento media server and WEBRTC technology.

WEBRTC, BROADCASTING MEDIA SERVICES, DELAY, KURENTO, SOFTWARE.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

W3C – World Wide Web Consortium

API – Application programming interface

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

HTTP – Hyper Text Transfer Protocol

WebRTC – Web Real-Time Communications

ПЗ – Програмне Забезпечення

VoIP – Voice over Internet Protocol

PSTN – Public Switched Telephone Network

SIP – Session Initiation Protocol

WebGL – Web-based Graphics Library

NAT – Network Address Translation

ЗМІСТ

С.

ВСТУП

Останнім часом люди по всьому світу стали частіше дивитись відео на цифрових пристроях чим по телевізору. Це стало переломним моментом і ця тенденція продовжується й сьогодні. Цифрове відео є найбільш домінуючою послугою серед медіа контенту, тому зростає запит на сервіси потокового мовлення.

Актуальність теми. Останнім часом постає необхідність у використанні медіасервісів потокового мовлення в реальному часі в першу чергу для нормального функціонування робочого та навчального процесу, та навіть для звичайної зустрічі з родичами та друзями. Що до робочого процесу, коли необхідно приймати рішення як найшвидше постає питання у вибору медіасервісу з мінімальною затримкою. Наразі серед існуючих технологій які забезпечують мінімальну затримку передачі відео та аудіо можна виділити стандарт WebRTC. WebRTC – це новий етап у розвитку інтернет-комунікацій. Даний стандарт дозволяє досягти мінімальної затримки передачі відео та аудіо що зараз є суттєвою вимогою при виборі медіасервісів, а також дає змогу перетворити браузер в кінцевий термінал відеоконференцзв'язку, що дає перевагу перед іншими технологіями за рахунок того, що спілкування відбувається в реальному часі без установки плагінів і інших розширень. Також технологія WebRTC дає шанс розробникам трансформувати зв'язок, надаючи надійні та безпечні комунікації корпоративного класу. Це відкриває можливості для організації вебінарів, онлайн-нарад, відеоконференцій та інших заходів. Всі перераховані обставини зумовлюють актуальність дипломного проекту.

1 СТАН ПИТАННЯ. ПОСТАНОВКА ЗАДАЧІ

1.1 Дослідження сервісів потокового мовлення на базі технології WebRTC

На сучасному етапі розвитку медіасервісних платформ виникає необхідність запровадження технології яка дозволить зменшити затримку транслювання медіапотоку. Серед існуючих технологій значну перевагу має технологія WebRTC.

WebRTC (Web Real-Time Communication) – це стандарт, який дозволяє веб-додаткам встановлювати пряме з'єднання між двома браузерами без необхідності передавати дані через веб-сервер. Даний стандарт складається з публічного API і набору протоколів для передачі даних між браузерами. Стандарт WebRTC дозволяє комунікацію між браузерами в реальному часі, використовуючи протокол UDP в парі з протоколом TCP для звичайних HTTP-запитів.

Технічно WebRTC це вирішення трьох основних завдань:

1. Забезпечення доступу браузера до веб-камери і мікрофону кінцевого користувача виключно засобами HTML5 і JavaScript без додаткових інстальованих модулів або Flash. Для цього в вихідний код браузера його розробник вбудовує підтримку певних компонентів.

2. Створення з'єднання для передачі аудіо-, відео потоку, наприклад, в режимі звичайного дзвінка або відеоконференції через Інтернет (в тому числі peer-to-peer).

3. Забезпечення передачі потоку даних. Для вирішення цих завдань використовуються три основних API, що входять в WebRTC: MediaStream, RTCPeerConnection і RTCDataChannel.

1.2 Основні API-технології WebRTC

Доставка високоякісних RTC-додатків, таких як аудіо- і відеоконференції і обмін даними між однорангових мережами, вимагає багато нових функціональних можливостей в браузері: можливості обробки аудіо та відео, нові API-інтерфейси додатків і підтримка декількох нових мережевих протоколів. На щастя, в більшості браузерів ця абстракція пояснюється трьома основними API.

MediaStream забезпечує отримання аудіо-, відео потоку, який взагальномупадку може включати кілька синхронізованих «доріжок», наприклад, один потік відео і дві аудіодоріжки для лівого і правого каналів стереомікрофона. Треба відзначити, що джерелом відео може бути не тільки веб-камера, а й робочий стіл користувача (що дає можливість демонструвати екран).

RTCPeerConnection забезпечує з'єднання «точка-точка» для передачі отриманих медіапотоків. На його «совісті» лежить обробка сигналу (зокрема, очищення картинки від шумів, гучності мікрофона), контроль використовуваних кодеків (перекодування на льоту, якщо відео з камери закодовано нестандартним чином), забезпечення безпосередньо самого з'єднання через мережу, а також шифрування і управління пропускною спроможністю, тобто підстроювання параметрів відеосигналу під наявний канал.

RTCDataChannel забезпечує двосторонню передачу даних через встановлений з'єднання. Як і медіапотоків, передані дані шифруються.

Однак перераховані API також є лише верхівкою айсберга: сигналізація, виявлення тимчасових вузлів, узгодження з'єднань, безпеку і цілі рівні нових протоколів – це всього лише кілька компонентів, необхідних для їх об'єднання.

Архітектура і протоколи, що забезпечують управління WebRTC, також визначають його робочі характеристики: затримка установки з'єднання, накладні витрати протоколу і семантику передачі. Фактично, на відміну від всіх інших комунікацій браузера, WebRTC передає свої дані по UDP. Однак

UDP - це всього лише відправна точка. Для того, щоб комунікація в браузері стала реальністю, потрібно набагато більше, ніж просто UDP.

1.3 Стандарти та розробка WebRTC

Включення взаємодії в режимі реального часу в браузері - це, можливо, одна з найбільш значних доповнень до веб-платформи з початку її існування. WebRTC відривається від звичної моделі взаємодії клієнт-сервер, що призводить до повного перевизначення мережевого рівня в браузері, а також приносить абсолютно новий медіа-стек, який необхідний для ефективної обробки аудіо та відео в режимі реального часу.

В результаті архітектура WebRTC складається з більш ніж десятка різних стандартів, що охоплюють API-інтерфейси додатків і браузерів, а також безліч різних протоколів і формати даних, необхідних для його роботи:

1. Web Real-Time Communications (WebRTC) – робоча група W3C, яка відповідає за визначення API-інтерфейсів браузера.

2. Real-Time Communication in Web-browsers (RTCWeb) - робоча група IETF, що відповідає за визначення протоколів, форматів даних, безпеку і всі інші необхідні аспекти, які забезпечують одноранговий зв'язок в браузері.

WebRTC не є певним стандартом. Основна мета - забезпечення зв'язку в режимі реального часу між браузерами. Але WebRTC розроблений таким чином, що він може бути інтегрований в існуючі системи зв'язку: передача голосу по IP (VOIP), різні SIP-клієнти і навіть телефонні мережі загального користування (PSTN). Стандарти WebRTC не визначають будь-які конкретні вимоги до сумісності, але вони намагаються використовувати ті ж концепції та протоколи всюди, де це можливо.

1.4 Безпека технології WebRTC

Шифрування і безпека є вбудованими стандартними засобами. Більш того, на більшості серверів WebRTC надає наскрізне шифрування між вузлами, тим самим забезпечуючи безпечну передачу даних в режимі реального часу.

При взаємодії з сайтом по протоколу HTTPS веб-браузер гарантує, що користувач звертається саме до вибраного сайту, а також захищений від будь-яких мережевих атак. Сервісів обміну повідомленнями, електронної пошти та інших, які використовують сайти як посередника, для захисту потрібні додаткові кошти, такі як WebRTC.

Коли браузер з'єднується по протоколу HTTPS, присутня гарантія того, що користувач відвідав конкретний веб-сайт, який вказаний в адресному рядку, і отримує дані, які відображені в браузері. Дана схема підходить для двосторонніх взаємодій клієнт-сервер, але не для більш складних систем, таких сервісів, як електронна пошта або соціальна мережа, в яких сервер є посередником між користувачами. У подібних випадках користувачеві доводиться вірити, що сервер не змінює повідомлення, що не передає дані третім особам і не дістає ніякої особистої інформації з повідомлень. Ця довіра може бути обгрунтованим, але упевнитися в цьому немає можливості.

Схожа ситуація і з даними, які передаються за допомогою технології WebRTC. Так як дані передаються через сервер, користувачам доводиться довіряти сайту обробки. Як і з соціальними мережами, у більшості випадків це прийнято, тому що користувач сам звертається до сайту або звернення не вимагає високої конфіденційності, щоб не довіряти сайту. Однак буває так, що все ж необхідна конфіденційність, так як користувач хоче бути засвідченим, що присутній захист від сайту-посередника. У браузері повинні бути присутні засоби безпеки, які відповідають за функції, що забезпечуються самими сайтами: доступ до медіаконтенту і аутентифікація викликів. На даний момент сайти можуть отримати доступ до аудіо і відео користувача за допомогою інтерфейсів технології WebRTC, тому необхідно

реалізувати механізми обмеження доступу в API. У розробках WebRTC вже є певні кошти для збільшення безпеки, і вони вже з'являються у веб-браузерах.

У більшості веб-додатків безпеку надається на рівні двухточечного з'єднання. Користувач хоче провести з'єднання з сайтом https://example.com, браузер починає TLS-сеанс з сервером, який оголошує себе example.com, і за допомогою отриманих ним даних засвідчується в тому, що з'єднання правильне. У той час, як TLS забезпечує цілісність даних і конфіденційність, а браузер виконує аутентифікацію, звіряючи пароль, за допомогою якого провайдер ідентифікації надає підтвердження про зв'язок між відкритим ключем сервера і його «особистістю». Упізнана «особистість» вважається логічним джерелом даних і необхідного для його ініціалізації коду. Те й інше передається або безпосередньо від впізнаної «особистості», або з довірених джерел.

У W3C, в якій підтримується одноранговий зв'язок завдяки WebRTC, веб-сторінки, які відвідують користувачі, служать уповноваженими джерелами для виходу на адресатів, з якими ті хочуть зробити зв'язок. Посередник необхідний для забезпечення зв'язку в режимі реального часу незважаючи на можливість перебоїв в мережі, однак присутня ймовірність зв'язку не з тією людиною. Фактично, ця проблема є і в інших системах. Наприклад, сервери соціальних мереж дозволяють приймати повідомлення при перебоях зі зв'язком, однак не існує способу переконатися, що повідомлення буде передано необхідному адресату.

Додатки зв'язку в режимі реального часу загальноприйнято представляти у вигляді трикутника (рис. 1.1). Дві його вершини - «Комп'ютер 1» і «Комп'ютер 2», а третя – сервіс викликів (наприклад, Skype або працює по протоколу SIP сервер-посередник). Дана модель описує типовий випадок застосування WebRTC, коли два користувача виробляють спілкування один з одним, а сервіс викликів забезпечує сайт з сигналізацією по протоколу HTTPS.

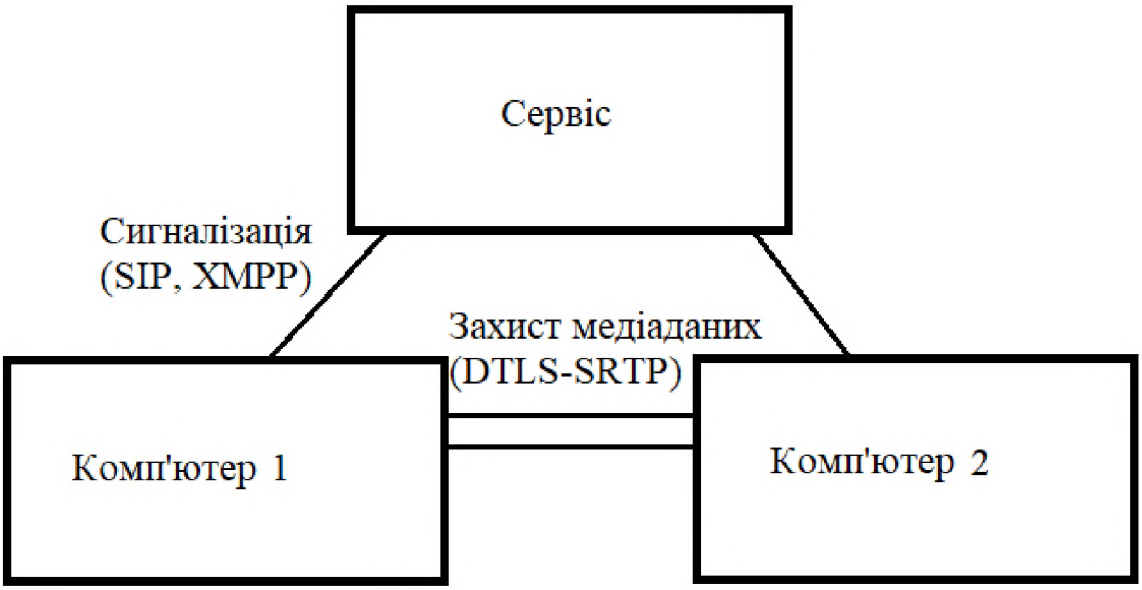
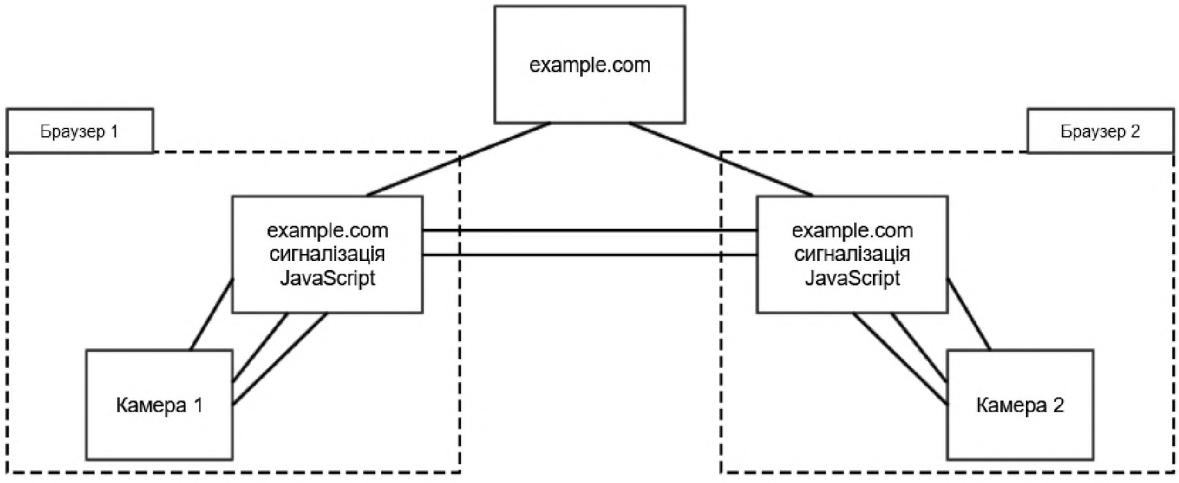


Рисунок 1.1 – Схема сеансу зв'язку реального часу

WebRTC завдяки коду, написаного на мові програмування JavaScript, який надається сайтом викликів, формує інтерфейс, а також управляє медіапотоком, в той час як є шлюзи для передачі відеоданих і сигналізації



(рис. 1.2).

Рисунок 1.2 – Підтримка комунікацій в реальному часі за допомогою коду JavaScript

Браузери 1 і 2, використовуючи додаток WebRTC, завантажують з сервера код JavaScript і HTML. Код на JavaScript, користуючись API WebRTC, виробляє з'єднання з іншими учасниками і забезпечує прийом і відтворення мультимедійних файлів. Також зазвичай сервер виконує функцію сигналізації, допомагаючи клієнтським програмам 1 і 2 зв'язатися один з одним. Присутня ймовірність, що від сервера буде переданий шкідливий код в браузер користувача. Однак в моделі WebRTC браузер – це довірена ланка. За допомогою засобів безпеки він може обмежувати можливості шкідливого коду.

Використання для сигналізації протоколу HTTPS дозволяє захиститися від мережевих атак між двома браузерами і сервером. Протокол DTLSSRTP (Datagram Transport Layer Security-Secure Real-Time Transport Protocol) дозволяє створити захищений канал, а сайт викликів підтверджує сутність браузерів.

Використовуючи тільки вбудовані в існуючі браузери засоби захисту, можна надати високий рівень безпеки. Веб-додатки, засновані на технології WebRTC, можна порівняти із захистом від атак з сервісами електронної пошти і соціальних мереж. Для деяких додатків WebRTC це найвищий рівень, на який можна розраховувати.

Для забезпечення наскрізної безпеки без вимог довіри сайту дзвінків, знадобляться нові механізми для управління аутентифікацією і доступу до мультимедійних файлів. Для аутентифікації потрібні провайдери ідентифікації, які грають ту ж роль, що і в системах клієнт-сервер, але оскільки потрібна симетрична аутентифікація, то таких провайдерів потрібно буде два. При управлінні доступом до мультимедійних файлів необхідно передбачити обмеження в API, що використовуються за допомогою коду, написаного на JavaScript, на сайті викликів для управління медіаданими.

Подібним чином, моделі загроз WebRTC потрібна наявність обов'язкових засобів, які реалізують такі функції безпеки:

1. Захист сигнального трафіку (між сервером і кожним з браузе рів).
2. Захист медіапотоків між браузерами.
3. Наскрізна аутентифікація для спілкуються браузерів.
4. Захист мультимедійних файлів від коду JavaScript.

Сервіс WebRTC вимагає від користувача дозволити доступ до мікрофона і камери. В результаті цього, користувач попереджений про включення мікрофона і камери на його ПК. Після дозволу користувача про включення, на вкладці браузера відображається червона точка, яка вказує на те, що доступ до пристроїв дозволений.

WebRTC використовує протокол датаграм безпеки транспортного рівня для передачі даних в режимі реального часу - DTLS (Datagram Transport Layer Security). Даний протокол вмонтований в усі браузери, які підтримують технологію WebRTC, такі як Chrome, Firefox і Opera. У зв'язку, який зашифрований за допомогою DTLS, виключається підробка інформації і підслуховування.

Крім DTLS, технологія WebRTC застосовує для шифрування мультимедійних файлів протокол передачі даних SRTP (Secure Real-Time Protocol). Даний протокол виключає перегляд або прослуховування IP-зв'язку третіми особами.

1.5 Аудіо- та відеоінструменти в WebRTC.

Багатий досвід проведення телеконференцій в браузері вимагає, щоб браузер мав доступ до системного обладнання для захоплення аудіо і відео. Ніяких сторонніх плагінів або настроювання драйверів, тільки простий і послідовний API. Однак сирі аудіо- і відеопотоки також недостатні самі по собі: кожен потік повинен оброблятися для підвищення якості, синхронізації, а вихідний бітрейт повинен регулюватися смугою пропускання що безперервно змінюється і затримкою між клієнтами.

На приймаючій стороні процес змінюється. Клієнт повинен декодувати потоки в режимі реального часу і мати можливість налаштовуватися на затримки мережі. Коротше кажучи, захоплення і обробка аудіо та відео - складна проблема. Однак доброю новиною є те, що WebRTC привносить в браузер повнофункціональні аудіо- і відеосистеми (рис. 1.3), які піклуються про всю обробку сигналів.

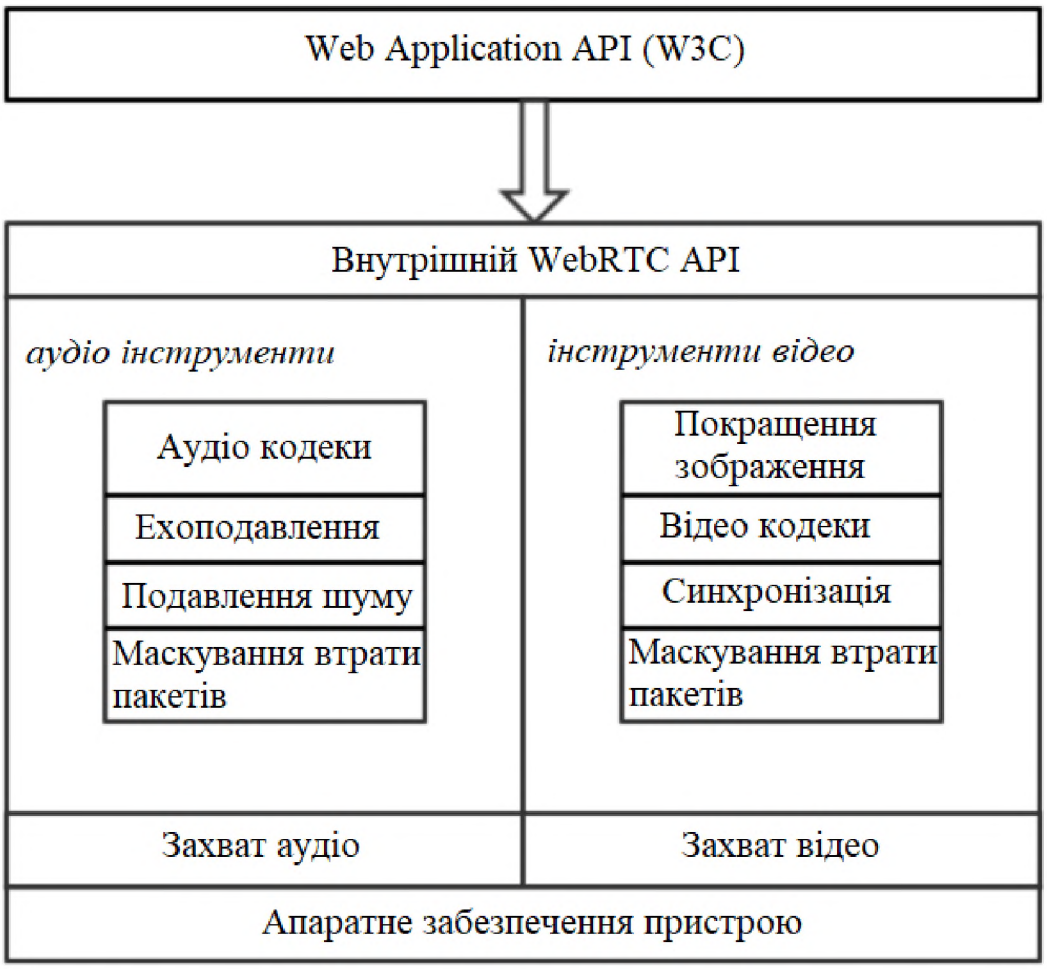


Рисунок 1.3 – Функціональність аудіо- і відеосистем

Отриманий аудіопотік обробляється для зменшення шуму і ехоподавлення, потім автоматично кодується одним з оптимізованих вузькосмугових або широкосмугових аудіокодеків. Нарешті, спеціальний алгоритм приховування помилок використовується, щоб приховати негативні наслідки втрати пакетів. Відеопроцесор виконує аналогічну обробку, оптимізуючи якість зображення, підбираючи оптимальні параметри

стиснення і налаштування кодека, застосовуючи маскування втрати пакетів і багато іншого.

Вся обробка виконується безпосередньо браузером, і що ще більш важливо, браузер динамічно коригує свій конвеєр обробки, щоб враховувати параметри аудіо- та відеопотоків і мережевих умов що постійно змінюються. Після того як вся ця робота буде завершена, вебдодаток отримує оптимізований потік, який потім може виводитися на локальний екран і динаміки.

1.5.1 Отримання аудіо і відео за допомогою getUserMedia

Специфікація Media Capture і Streams W3C визначає набір нових API-інтерфейсів JavaScript, які дозволяють додатку запитувати аудіо- і відеопотоки з платформи, а також набір API-інтерфейсів для управління і обробки отриманих медіапотоків. Об'єкт MediaStream (рис. 1.4) є основним інтерфейсом, який дозволяє використовувати всі ці функції.

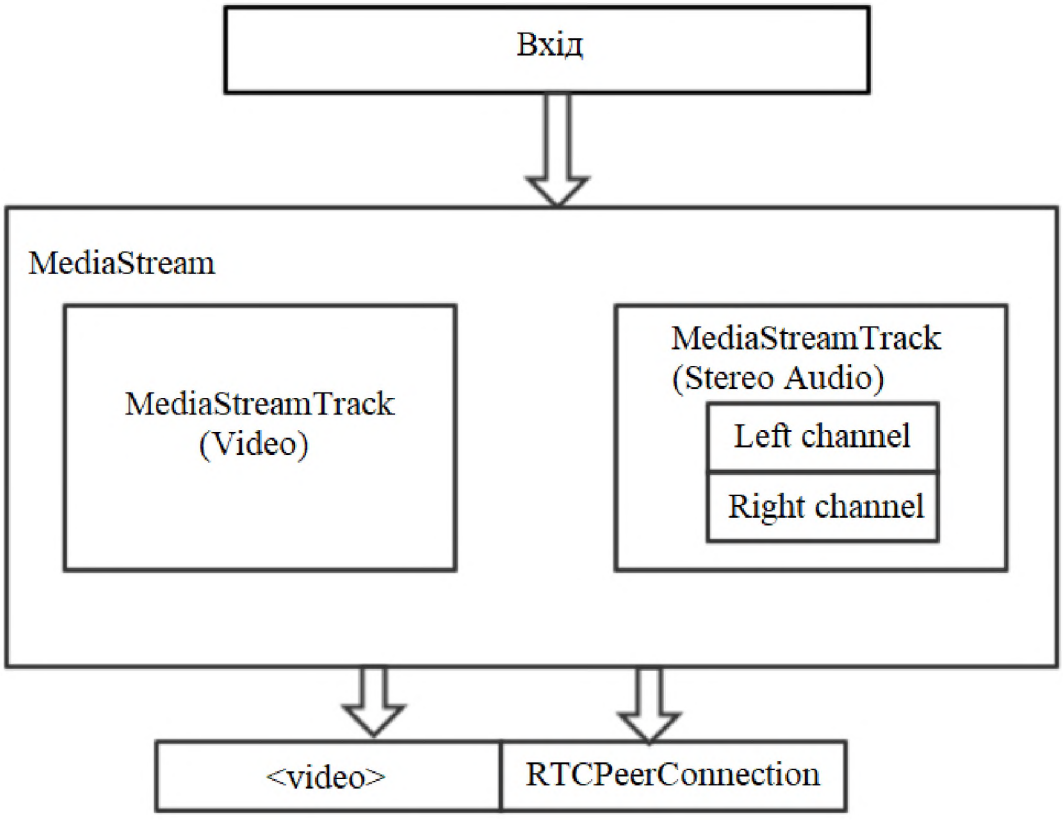


Рисунок 1.4 — Опис об'єкта MediaStream

1. Об'єкт MediaStream складається з однієї або декількох окремих доріжок (MediaStreamTrack).

2. Доріжки всередині об'єкта MediaStream синхронізуються один з одним.

3. Джерелом введення може бути фізичний пристрій, такий як мікрофон, веб-камера або локальний або віддалений файл з жорсткого диска користувача або віддаленого мережевого партнера.

4. Висновок MediaStream може бути відправлений в один або кілька пунктів призначення: локальний відео або аудіо елемент, код JavaScript для подальшої обробки або віддалений спеціальний робочий вузол.

Об'єкт MediaStream представляє потік відеоданих в реальному часі і дозволяє програмного коду отримувати дані, управляти окремими доріжками і визначати вихідні дані. Всі аудіо та відео обробки, такі як подавлення шумів, вирівнювання, поліпшення зображення і багато іншого, автоматично обробляються аудіо- та відеосистемами.

Однак особливості отриманого медіапотоку обмежені можливостями вхідного джерела: мікрофон може видавати тільки аудіопотік, а деякі веб-камери можуть відтворювати потокове відео з більш високою роздільною здатністю, ніж інші. В результаті при запиті медіапотоків в браузері API getUserMedia () дозволяє нам вказати список обов'язкових і необов'язкових обмежень для відповідності потребам додатку. API getUserMedia () відповідає за запит доступу до мікрофону і камери від користувача, а також отримання потоків, відповідних зазначеним обмеженням.

Надані API також дозволяють додатку управляти окремими доріжками, клонувати їх, змінювати обмеження і багато іншого:

1. Web Audio API дозволяє обробляти аудіо в браузері.

2. Canvas API дозволяє здійснювати захоплення і подальшу обробку окремих відеокадрів.

3. API CSS3 і WebGL API можуть застосовувати різні 2D / 3D-ефекти для вихідного потоку.

Після того, як потік буде отримано, його можна використовувати в інших браузерах.

getUserMedia () - простий API для отримання аудіо-та відеопотоків. Дані автоматично оптимізуються, кодуються і декодуються за допомогою аудіо- та відеопристроїв WebRTC, а потім направляються на один або кілька виходів.

1.5.2 Бітрейт аудіо та відео

При запиті аудіо і відео з браузера необхідно уважно стежити за розміром і якістю потоків. У той час як вхідні пристрої можуть захоплювати потоки високої якості, комп'ютер і пропускна здатність повинні бути в змозі обробити дані. В існуючих реалізаціях WebRTC використовуються кодеки Opus і VP8:

1. Кодек Opus використовується для аудіо і підтримує постійне і змінне кодування бітрейта. Вимагає 6-510 Кбіт / с пропускної здатності. Перевага полягає в тому, що кодек може легко перемикатися і адаптуватися до змінної пропускної здатності.

2. Кодер-декодер VP8, який використовується для кодування відео, також вимагає 100-2000 Кбіт / с пропускної здатності, а бітрейт залежить від якості потоків:

а) 720p при 30 FPS: 1,0 ~ 2,0 Мбіт / с

б) 360p при 30 FPS: 0,5 ~ 1,0 Мбіт / с

в) 180p при 30 FPS: 0,1 ~ 0,5 Мбіт / с

В результаті односторонній виклик високої чіткості може зажадати до 2,5+ Мбіт / с пропускної здатності мережі. При додаванні ще декількох тимчасових вузлів якість знизиться з урахуванням додаткової пропускної здатності, вимог до процесора, графічного процесора і пам'яті.

1.6 Підтримка технології WebRTC мобільними браузерами.

Одним з головних переваг технології WebRTC є те, що мобільні браузери підтримують дану технологію. Це дозволяє глядачам не бути прив'язаними до свого ПК і мати можливість в будь-який момент підключитися до трансляції через мобільний телефон.

1.7 Підтримувані браузери.

Підтримувані браузери і їх можливості для ПК (табл. 1.1):

1) Google Chrome (17+) і всі браузери на основі движка Chromium.

2) Mozilla FireFox (18+).

3) Opera (18+).

Таблиця 1.1 — Підтримувані браузери і їх можливості

Можливості	Google Chrome	Mozilla FireFox	Opera
PeerConections	+	+	+
getUserMedia	+	+	+
mediaConstraints	+	+	+
TURN support	+	+	+
MediaStream API	+	+	+
WebAudio Integrations	+	+	+
Reliable dataChanenls	+	+	+
Screen Sharing		-	-
Stream re-broadcasting	-	-	-
Echo cancellation	-	-	-
Solid	+	+	+

interoperability			

Підтримувані мобільні браузери і їх можливості для Android (табл. 1.2):

1) Google Chrome (28+);

2) Mozilla Firefox (24+);

3) Opera Mobile (12+).

Таблиця 1.2 — Підтримувані мобільні браузери і їх можливості

Можливості	Google Chrome	Mozilla FireFox	Opera
PeerConections	+	+	+
getUserMedia	+	+	+
mediaConstraints	+	+	+
TURN support	+	+	+
MediaStream API	+	+	+
Simulcast	+	-	-
dataChanenls	+	+	+
Screen Sharing	+	+	-
Stream re-broadcasting	+	+	+
Echo cancellation	+	+	+
H.264 video	-	-	+
VP8 video	+	+	+

1.8 Переваги та недоліки технології WebRTC

1.8.1 Переваги

Таким чином:

1. Не потрібна додаткова установка ПО.

2. Висока якість за рахунок:

а) використання сучасних аудіо-(Opus) і відеокодеків (VP8);

б) автоматичного налаштування якості потоку під умови з'єднання;

в) вбудованої системи шумо- і ехоподавлення;

г) автоматичного регулювання рівня чутливості мікрофонів учасників;

3. Високий рівень безпеки, так як всі з'єднання зашифровані і захищені.

4. Захоплення та демонстрація робочого столу.

5. Реалізація різних інтерфейсів управління на основі WebSockets і HTML5.

6. Можливість впровадити в свій сервіс або продукт, так як вихідний код проекту знаходиться у відкритому доступі.

7. Кросплатформеність. Якщо браузер підтримує технологію WebRTC, то один і той же WebRTC-додаток буде працювати однаково добре на будь-якій операційній системі. Це дозволяє значно заощадити ресурси для розробки ПО.

1.8.2 Недоліки

В свою чергу наявні наступні недоліки:

1. API змінюється, так як проект знаходиться в розробці. Тому інколи доводиться вносити зміни в свій код.

2. Кросбраузерність. Немає підтримки технології WebRTC в таких браузерах, як Internet Explorer від Microsoft і Safari від Apple.

1.9 Порівняння технології WebRTC с технологією Adobe Flash

1.9.1 Відеозв'язок в Adobe Flash

Adobe Flash – це платформа, створена компанією Adobe Systems, для мультимедійних презентацій або веб-додатків. Широко використовується для створення рекламних банерів, ігор, анімацій, а також відтворення на веб-сторінках аудіо- і відеозаписів.

Популярна колись технологія Adobe Flash не мала конкурентів і аналогів, так як дозволяла не тільки програвати мультимедійні файли, а також здійснювати відеодзвінки прямо з браузерів.

Сьогодні більшість онлайн-сервісів, що надають відеозв'язок через браузер, використовує дану технологію. Для здійснення дзвінків з браузера, користувачеві спочатку необхідно завантажити та встановити на ПК під управлінням Windows Adobe Flash Player.

Передача аудіо- і відеопотоків здійснюється від одного плеєра до іншого через сервер завдяки використанню транспортного протоколу RTMP, який працює поверх TCP. Завдання протоколу TCP полягає в збереженні цілісності переданих даних.

Основним недоліком flash-додатків є надмірне навантаження на центральний процесор, Це пов'язано з неефективністю віртуальної машини Flash Player. Але варто відзначити, що іноді має місце той факт, що flash-додатки недостатньо оптимізовані їх розробниками через використання «генераторів» flash-додатків.

Інший важливий недолік полягає в тому, що не завжди є можливість запустити Flash-додаток, або вона пов'язана з деякими труднощами (наприклад, необхідно встановити плагін або обновити його до останньої версії). Деякі користувачі (або системні адміністратори в рамках цілої мережі) відключають у налаштуваннях браузера можливість завантажувати контент, оброблюваний плагінами або, що завантажується у фреймах з метою інформаційної безпеки (у зв'язку з можливою загрозою з боку контенту, наприклад, перехоплення буфера обміну), економії системних ресурсів, або для порятунку від обридлої реклами.

Це робить технологію в цілому ненадійною також для розробників, яким ніхто не гарантує, що веб-додаток на основі Flash буде взагалі відтворено. Тому Flash, в основному, використовується для написання ігор, невеликих напівінтерактивних анімацій і для красиво оформленої реклами, тобто в сфері розваг і дизайну. Для серйозних веб-додатків, де взаємодія з

користувачем повинна бути без шкоди красі, звичайно використовується Javascript, або взагалі не використовуються ніякі технології крім тих, що 100 % працюють (HTML, CGI).

В Інтернеті можна знайти сайти, повністю оформлені у вигляді Flash-додатка (увесь контент, а також елементи навігації). Зазвичай це сайти, що присвячені іграм, дизайнерські студії, особисті сторінки та інші сайти, метою яких є вразити відвідувача красою й незвичайністю реалізації. Великі портали й інформаційні ресурси намагаються уникати використання Flash (за винятком вставки рекламних баннеров, неможливість виводу яких не викликає незручностей для користувачів).

Використання Flash для розміщення текстової інформації перешкоджає її індексуванню пошуковими системами. Однак існує безліч способів розв'язати цю проблему. Одним зі способів розв'язку даної проблеми є використання тексту у форматі HTML, у футері сторінки.

У реалізаціях Adobe Flash час від часу знаходять «діри», що дозволяють зловмисникам проводити різноманітні дії з системою. Так, наприклад, в жовтні 2008 року була знайдена уразливість, що дозволяє дистанційно керувати веб-камерою і мікрофоном.

У жовтні 2015 року в плагіні Adobe Flash Player була знайдена уразливість, за допомогою якої шкідливе програмне забезпечення здатне вбудовуватися в протоколи програвача і проникати на комп'ютери жертв. Також була знайдена уразливість, через яку Firefox і Chrome на час відключили Flash, хоча його можна було включити на певному веб-сайті, але через кілька днів Flash включили.

Google з 2016 року планує відключити підтримку Flash в своєму браузері Chrome. Починаючи з версія 55, flash відключений за замовчуванням для всіх сайтів. Однак користувачі ще можуть вручну включити на конкретному сайті Flash плеєр.

Відносно підтримуваних ОС, Flash Player підтримується тільки на Windows. Компанія Adobe не бажає підтримувати і розвивати технологію Flash для Linux, MacOS, а також мобільних ОС.

Переваги:

1. використання шифрування AES забезпечує захист медіатрафіка від взлому і прослуховування;

2. AEC (ехоподавлення) і джиттер-буфер забезпечують хорошу якість відтворення звуку;

3. підтримка великої кількості браузерів.

Недоліки:

1. відсутність AGC (автоматичного регулювання посилення);

2. затримка трафіку через використання протоколу RTMP;

3. обов'язкова наявність проміжного сервера;

4. не підтримується в мобільних операційних системах, таких як iOS і Windows Mobile;

5. закриті засоби розробки.

1.9.2 Відеозв'язок на базі технології WebRTC

На зміну технології Flash приходить WebRTC, яка передбачає ініціалізацію і управління медіапотоків, а також здійснює їх передачу через інтернет і програвання в браузері.

На відміну від технології Flash, яка позиціонувалася для створення веб-додатків і мультимедійних презентацій, WebRTC спочатку розроблялася як повноцінний клієнт для передачі відеопотоків між веб-браузерами. Тому розробники передбачили всі нюанси для того, щоб задовольнити потреби будь-якого користувача. А саме: підтримка всіх популярних операційних систем, використання сучасних кодеків і можливість передачі відеоданих з мінімальною затримкою. Завдяки цьому стало можливим не тільки приймати медіадані, а також і передавати їх під час онлайн-конференції з великою кількістю учасників.

Переваги:

1. передача трафіку без затримок;
2. підтримка AEC, AGC, джиттер-буфер;
3. використання сучасних відкритих кодеків Opus і VP8;
4. кросплатформеність: доступність всіх ОС;
5. повноцінне VoIP в браузері Chrome.

Недоліки:

1. несумісність з традиційним VoIP обладнанням.

1.9.3 Загальне порівняння технологій WebRTC і Adobe Flash

Якість відеосигналу залежить від декількох факторів:

1. Дозвіл відео;
2. Бітрейт;
3. Кодек і його параметри.

При порівнянні якості технології WebRTC з технологією Adobe Flash було виявлено, що у WebRTC якість краще, ніж у Adobe Flash. Можливості технологій WebRTC і Adobe Flash наведені в табл. 1.3:

Таблиця – 1.3. Порівняння технології WebRTC з платформою Flash

Можливості	Flash	WebRTC
AEC (Acoustic Echo Cancellation)	+	+
AGC (Automatic Gain Control)	-	+
Адаптивний джиттер-буфер	+	+
Підтримка SIP	-	-
Використання аудіокодека Speex	+	-
Використання	-	+

аудіокодека Opus		
Використання відеокодека VP8	-	+

Продовження таблиці 1.3

Робота в ОС Windows	+	+
Робота в ОС Linux, Android	-	+
Робота без плагінів	-	+

1.10 Мережеве з’єднання в WebRTC

Як оговорювалося раніше WebRTC – це технологія, орієнтована на браузери, яка дозволяє з'єднати два клієнта для відео-передачи даних. Основні особливості – внутрішня підтримка браузерами (не потрібні сторонні впроваджуються технології типу adobe flash) і здатність з'єднувати клієнтів без використання додаткових серверів - з'єднання peer-to-peer (p2p).

Встановити з'єднання p2p – досить важке завдання, так як комп'ютери не завжди мають публічний IP адрес, тобто адрес в інтернеті. Через невелику кількість IPv4 адрес (і для цілей безпеки) був розроблений механізм NAT, який дозволяє створювати приватні мережі, наприклад, для домашнього використання. Багато домашніх роутерів зараз підтримують NAT і завдяки цьому всі домашні пристрої мають вихід в інтернет, хоча провайдери інтернету зазвичай надають один IP адрес. Публічні IP адреси - унікальні в інтернеті, а приватні ні. Тому з'єднатися p2p – важко.

WebRTC успішно справляється з цією проблемою, використовуючи протокол ICE, який, правда, вимагає використання додаткових серверів (STUN, TURN).

1.10.1 Фази WebRTC

Щоб з'єднати два вузла через протокол WebRTC (або просто RTC) необхідно провести деякі попередні дії для встановлення з'єднання. Це, перша фаза – установка з’єднання. Друга фаза - передача відео-даних.

Хоч технологія WebRTC в своїй роботі використовує безліч різних способів комунікації (TCP і UDP) і має гнучке переключення між ними, ця технологія не має протоколу для передачі даних про з'єднання. Тому необхідно мати певний додатковий спосіб передачі даних, ніяк не пов'язаний з WebRTC. Це може бути сокетна передача, протокол HTTP, це може бути навіть протокол SMTP . Цей механізм передачі початкових даних називається сигнальним. Передавати потрібно не так багато інформації. Всі дані передаються у вигляді тексту і діляться на два типи - SDP і Ice Candidate. Перший тип використовується для встановлення логічного з'єднання, а другий для фізичного. Як тільки ми передамо всю потрібну інформацію, вузли зможуть з'єднатися і більше наша допомога потрібна не буде. Таким чином, сигнальний механізм, який ми повинні реалізувати окремо, буде використовуватися тільки при підключенні, а при передачі відео-даних використовуватися не буде.

Перша фаза — фаза встановлення з’єднання:

- Ініціатор
    1. Отримання локального (свого) медіа потоку і установка його для передачі (getUserMediaStream).
    2. Пропозиція почати відео-передачу даних (createOffer).
    3. Отримання свого SDP об'єкта і передача його через сигнальний механізм (SDP).
    4. Отримання своїх Ice candidate об'єктів і передача їх через сигнальний механізм (Ice candidate).
    5. Отримання віддаленого (чужого) медіа потоку і відображення його на екрані (onAddStream).
- Очикуючий дзвінка

1. Отримання локального (свого) медіа потоку і установка його для передачі (getUserMediaStream).
2. Отримання пропозиції почати відео-передачу даних і створення відповіді (createAnswer).
3. Отримання свого SDP об'єкта і передача його через сигнальний механізм (SDP).
4. Отримання своїх Ice candidate об'єктів і передача їх через сигнальний механізм (Ice candidate).
5. Отримання віддаленого (чужого) медіа потоку і відображення його на екрані (onAddStream).

Незважаючи на гадану заплутаність кроків тут їх насправді три: відправка свого медіа потоку (п.1), установка параметрів з'єднання (пп.2-4), отримання чужого медіа потоку (п.5). Найскладніший – другий крок, тому що він складається з двох частин: встановлення фізичного та логічного з'єднання. Перша вказує шлях, по якому повинні йти пакети, щоб дійти від одного вузла мережі до іншого. Друга вказує параметри відео / аудіо - яку використовувати якість, які використовувати кодеки.

1.10.2 Дескриптор сесії (SDP)

У різних комп'ютерів завжди будуть різні камери, мікрофони, відеокарти та інше обладнання. Існує безліч параметрів, якими вони володіють. Все це необхідно скоординувати для медіа передачі даних між двома вузлами мережі. WebRTC робить це автоматично і створює спеціальний об'єкт - дескриптор сесії SDP. Передаючи цей об'єкт іншому вузлу, можна передавати медіа дані. Тільки зв'язку з іншим вузлом поки немає.

Для цього використовується будь-який сигнальний механізм. SDP можна передати і через сокети, і людиною (повідомити його іншим сайтом по телефону). Дається вже готовий SDP і його потрібно переслати. А при

отриманні на іншій стороні - передати у відомство WebRTC. Дескриптор сесії зберігається у вигляді тексту і його можна змінити в своїх додатках, але, як правило, це не потрібно. Як приклад, при з'єднанні компьютера з телефоном іноді потрібно примусово вибирати потрібний аудіо кодек.

Зазвичай під час активного з'єднання необхідно вказувати якусь адресу, наприклад URL. Тут в цьому немає необхідності, так як через сигнальний механізм користувач сам відправе дані за призначенням. Щоб вказати WebRTC, що потрібно встановити p2p з'єднання, необхідно викликати функцію createOffer. Після виклику цієї функції і вказівки їй спеціального callback'а буде створено SDP об'єкт і переданий в цей же callback. Все, що потрібно від користувача - передати цей об'єкт по мережі іншому вузлу (співрозмовнику). Після цього на іншому кінці через сигнальний механізм прийдуть дані, а саме цей SDP об'єкт. Цей дескриптор сесії для цього вузла чужий і тому несе корисну інформацію. Отримання цього об'єкта - сигнал до початку з'єднання. Тому користувач повинен погодитися на це і викликати функцію createAnswer. Функція createAnswer - повний аналог createOffer. Знову в callback користувача передадуть локальний дескриптор сесії і його необхідно передати по сигнальному механізму назад.

Варто відзначити, що викликати функцію createAnswer можна тільки після отримання чужого SDP об'єкта. Тому що локальний SDP об'єкт, який буде генеруватися при виклику createAnswer, повинен спиратися на віддалений SDP об'єкт. Тільки в такому випадку можливо скоординувати свої настройки відео з настройками співрозмовника. Також не варто викликати createAnswer і createOffer до отримання локального медіа потоку - їм буде нічого писати в SDP об'єкт.

Так як в WebRTC є можливість редагувати SDP об'єкт, то після отримання локального дескриптора його потрібно встановити. При отриманні дескриптора його необхідно теж встановити. Тому користувач повинен на одному вузлі встановити два дескриптора - свій і чужий (тобто локальний і віддалений).

Після такого “рукостискання” вузли знають що необхідно один одному. Наприклад, якщо вузол 1 підтримує кодеки A і B, а вузол 2 підтримує кодеки B і C, то, так як кожен вузол знає свій і чужий дескриптори, обидва вузла виберуть кодек B. Встановлена логіка з'єднання, можна передавати медіа потоки, але є інша проблема - вузли і раніше пов'язані лише сигнальним механізмом.

1.10.3 Кандидати (ICE candidate)

Технологія WebRTC заплутана через свою нову методологію. При установці з'єднання не вказується адреса того вузла, з яким потрібно з'єднатися. Встановлюється спочатку логічне з'єднання, а не фізичне, хоча завжди робилося навпаки.

З'єднання вже встановлено (логічне з'єднання), але немає шляху, по якому вузли мережі можуть передавати дані. Нехай вузли знаходяться в одній приватній мережі. Вони можуть легко з'єднуватися один з одним за своїми внутрішніми IP адресами (або можливо, з якихось інших, якщо використовується не TCP / IP).

Через деякі callback'и WebRTC повідомляє користувачу Ice candidate об'єкти. Вони теж приходять в текстовій формі і також, як з дескрипторами сесії, вони пересилаються через сигнальний механізм. Якщо дескриптор сесії містив інформацію про установки користувача на рівні камери і мікрофона, то кандидати містять інформацію про розташуванні в мережі користувача. Якщо користувач передасть їх іншим сайтом, то той зможе фізично з'єднатися з користувачем, а так як у нього вже є і дескриптор сесії, то і логічно зможе з'єднатися і дані «потечуть». Якщо він не забуде відправити користувачеві і свій об'єкт кандидата, тобто інформацію про те, де знаходиться він сам в мережі, то і користувач зможе з ним з'єднатися. Зауважу, тут ще одна відмінність від класичного клієнт-серверної взаємодії. Спілкування з HTTP сервером відбувається за схемою запит-відповідь, клієнт відправляє дані на сервер, той обробляє їх і шле за адресою, вказаною в

пакеті запиту. В WebRTC необхідно знати дві адреси і з'єднувати їх з двох сторін.

Відмінність від дескрипторів сесії полягає в тому, що встановлювати потрібно тільки віддалених кандидатів. Редагування тут заборонено і не може принести ніякої користі. У деяких реалізаціях WebRTC кандидатів необхідно встановлювати тільки після установки дескрипторів сесії.

Виникає питання, чому дескриптор сесії був один, а кандидатів може бути багато? Тому що розташування в мережі може визначатися не тільки своїм внутрішнім IP адресою, але також і зовнішньою адресою маршрутизатора, і не обов'язково одного, а також адресами TURN серверів.

Два вузла знаходяться в одній мережі. Їх ідентифікують за допомогою IP адрес. Більше ніяк. Правда, ще можна використовувати різні транспортні протоколи (TCP і UDP) і різні порти. Це і є та інформація, яка міститься в об'єкті кандидата – IP, PORT, TRANSPORT і якась інша. Для прикладу, використовується UDP транспорт і 531 порт.

Користувач знаходиться в вузлі p1, WebRTC передасть користувачеві такий об'єкт кандидата - [10.50.200.5, 531, udp]. Якщо користувач в вузлі p2, то кандидат такий - [10.50.150.3, 531, udp]. Через сигнальний механізм p1 отримає кандидата p2 (тобто розташування вузла p2, а саме його IP і PORT). Після чого p1 може з'єднатися з p2 безпосередньо. Більш правильно, p1 буде посилати дані на адресу 10.50.150.3:531 в надії, що вони дійдуть до p2. Не важливо, чи належить ця адреса вузлу p2 або якомусь посередникові. Важливо лише те, що через цю адресу дані будуть надсилатися і можуть досягти p2.

Поки вузли знаходяться одній мережі то це не викликає додаткових зусиль. Кожен вузол має тільки один об'єкт кандидата (завжди мається на увазі свій, тобто своє розташування в мережі). Але кандидатів стане набагато більше, коли вузли будуть знаходиться в різних мережах.

Більш складний випадок: один вузол знаходиться за роутером (точніше, за NAT), а другий вузол знаходиться в одній мережі з цим роутером (в інтернеті).

Цей випадок має приватне рішення проблеми. Домашній роутер зазвичай містить таблицю NAT. Це спеціальних механізм, розроблений для того, щоб вузли усередині приватної мережі роутера змогли звертатися, наприклад, до веб-сайту.

Веб-сервер з'єднаний з інтернетом безпосередньо, має публічну IP адресу. Нехай це буде вузол p2. Вузол p1 (веб-клієнт) шле запит на адресу 10.50.200.10. Спочатку дані потрапляють на роутер r1, точніше на його внутрішній інтерфейс 192.168.0.1. Після чого, роутер запам'ятовує адресу джерела (адреса p1) і заносить його в спеціальну таблицю NAT, потім змінює адресу джерела на свій (з p1 на r1). Далі, за своїм зовнішнім інтерфейсом роутер пересилає дані безпосередньо на веб-сервер p2. Веб-сервер обробляє дані, генерує відповідь і відправляє назад. Відправляє роутеру r1, так як саме він стоїть в зворотній адресі (роутер підмінив адресу на свій). Роутер отримує дані, дивиться в таблицю NAT і пересилає дані вузлу p1. Роутер виступає тут як посередник.

Виникає наступне питання: що якщо кілька вузлів з внутрішньої мережі одночасно звертаються до зовнішньої мережі? Як роутер зрозуміє кому відправляти відповідь назад? Дана проблема вирішується за допомогою портів. Коли роутер підміняє адресу вузла на свій, він також підміняє і порт. Якщо два вузла звертаються до інтернету, то роутер підміняє порти їх джерел на різні. Тоді, коли пакет з веб-сервера прийде назад до роутера, то роутер зрозуміє по порту, кому призначений даний пакет.

Вузол p2 має одного кандидата (своє розташування в мережі - 10.50.200.10), а вузол p1, який знаходиться за роутером з NAT, матиме двох кандидатів - локального (192.168.0.200) і кандидата роутера (10.50.200.5). Перший не знадобиться, але він генерується, так як WebRTC нічого не знає про віддалений хост – він може перебувати в тій же мережі, а може і ні.

Запис в таблиці NAT генерується тільки коли дані виходять з внутрішньої мережі. Тому вузол p1 повинен першим передати дані і тільки після цього дані вузла p2 зможуть дістатися до вузла p1.

На практиці обидва вузла будуть знаходитися за NAT. Щоб створити запис в таблиці NAT кожного роутера, вузли повинні щось відправити до віддаленого вузла, але на цей раз ні перший не може дістатися до другого, ні навпаки. Це пов'язано з тим, що вузли не знають своїх зовнішніх IP адрес, а відправляти дані на внутрішні адреси не має сенсу.

Однак, якщо зовнішні адреси відомі, то з'єднання буде легко встановлено. Якщо перший вузол відішле дані на роутер другого вузла, то роутер їх проігнорує, так як його таблиця NAT поки порожня. Однак в роутері першого вузла в таблиці NAT з'явився потрібний запис. Якщо тепер другий вузол відправить дані на роутер першого вузла, то роутер їх успішно передасть першому вузлу. Тепер і таблиця NAT другого роутера має потрібні дані.

Проблема в тому, що, щоб дізнатися свій зовнішній IP адрес, потрібен вузол який знаходиться в загальній мережі. Для вирішення такої проблеми використовуються додаткові сервери, які підключені до інтернету безпосередньо. З їх допомогою також створюються записи в таблиці NAT.

1.11 Огляд медіасервера Kurento

Медіасервер WebRTC – це свого роду «мультимедійне проміжне програмне забезпечення» (воно знаходиться в середині комунікаційних вузлів), де медіатрафік проходить при переході від джерела до адресата.

Kurento – архітектура, яка підтримує WebRTC, орієнтована для потокової обробки даних в реальному часі. В основі архітектури Kurento лежить медіасервер під назвою Kurento Media Server (KMS). Kurento Media Server заснований на сумісних засобах обробки мультимедійних файлів, що означає, що будь-яка з його функцій, є модуль що підключається, який

можна активувати або деактивувати. Крім того, можна легко створювати додаткові модулі, що розширюють Kurento Media Server, з новими функціональними можливостями, які можуть бути підключені динамічно.

Kurento Media Server забезпечує групові комунікації, мікшування, транскодування, запис і відтворення. Крім того, він також надає розширені модулі для обробки медіа, включаючи комп'ютерний зір, доповнену реальність і багато іншого. Крім того, Kurento Media Server здатний приймати відеопотоки від більшості типів IP-камер, що підтримують протоколи RTSP або HTTP. Kurento Media Server може організувати збереження відеозаписів на сервер.

1.12 Висновки до першого розділу

Перший розділ присвячено розгляду загальних характеристик медіасервісної платформи Kurento та технології WebRTC яка забезпечує мінімальну затримку доставки медіаконтенту.

1. Розглянуто технологію WebRTC та визначено властивості які дозволяють забезпечити мінімальну затримку. Це важливо при створенні та плануванні кроків імплементації медіасервісної платформи на базі технології WebRTC.

2. Виконано порівняльну характеристику технології WebRTC та технології Adobe Flash. Виявлено що технологія WebRTC має перевагу у мінімальній затримці передачі медіаданих за рахунок протоколу UDP та кросплатформеності. А також недоліки у вигляді відсутності шифрування медіатрафіка та невеликої кількості підключень.

1.13 Постановка задачі до другого розділу

Метою дипломного проекту є зменшення затримки доставки контенту в сервісах потокового мовлення шляхом вибору оптимального програмного забезпечення.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

1. Розгорнути та налаштувати медіасервісну платформу Kurento.
2. Дослідити затримку доставки медіаконтенту в медіасервісній платформі Kurento на базі технології WebRTC.
3. Порівняти медіасервісну платформу Kurento з платформами Twitch та YouTube.

2 СПЕЦІАЛЬНА ЧАСТИНА

2.1 Розгляд технології WEBRTC та Kurento

Звертаючи на обставини розглянуті в першому розділі в дипломному проекті значну увагу приділено розгляду проблеми мінімізації затримки медіапотоків в медіасервісних платформах. Відомо що, серед існуючих технологій значну перевагу перед іншими технологіями має технологія WebRTC.

Відомо що, технологія WebRTC (Web Real-Time Communications) - це технологія, яка дозволяє Web-додаткам і сайтам захоплювати і вибірково передавати аудіо і / або відео медіа-потоки, а також обмінюватися довільними даними між браузерами, без обов'язкового використання посередників. Набір стандартів, які включає в себе технологія WebRTC, дозволяє обмінюватися даними і проводити пірингові телеконференції, без необхідності користувачеві встановлювати плагіни або будь-яке інше додаткове програмне забезпечення. WebRTC складається з декількох взаємопов'язаних програмних інтерфейсів (API) і протоколів, які працюють разом.

WebRTC є багатоцільовим і разом з Media Capture and Streams API, надають потужні мультимедійні можливості для Web, включаючи підтримку аудіо та відео конференцій, обмін файлами, захоплення екрану, управління ідентифікацією та взаємодія з застарілими телефонними системами, включаючи підтримку передачі сигналів тонового набору DTMF. З'єднання між вузлами можуть створюватися без використання спеціальних драйверів або плагінів, і часто без проміжних сервісів.

З'єднання між двома вузлами представлено як об'єкт інтерфейсу RTCPeerConnection. Як тільки з'єднання встановлено і відкрито, використовуючи об'єкт RTCPeerConnection, медіапотоки (MediaStreams) і / або канали даних (RTCDataChannels) можуть бути додані до з'єднання.

Медіа потоки можуть складатися з будь-якої кількості треків (доріжок) медіаінформації. Ці треки, представлені об'єктами інтерфейсу MediaStreamTrack, і можуть містити один або кілька типів мультимедійних файлів, включаючи аудіо, відео, текст (такі як субтитри або назва глав). Більшість потоків складаються, як мінімум, тільки з одного аудіо треку (однієї аудіо доріжки), або відео доріжки, і можуть бути відправлені й отримані, як потоки (медіа-дані в справжнім часу) або збережені в файл.

Так само, можна використовувати з'єднання між двома вузлами для обміну довільними даними, використовуючи об'єкт інтерфейсу RTCDataChannel, що може бути використано для передачі службової інформації, біржових даних, пакетів ігрових статусів, передача файлів або закритих каналів передачі даних.

2.1.1 Дослідження принципу роботи WEBRTC

На сам перед WebRTC – це проект, який дозволяє отримувати медіаданні (аудіо та відео) через браузер і встановлювати Peer-to-Peer з'єднання між двома клієнтами, через яке можуть передаватися звичайні дані і медіапотоків (рис. 2.1).

По суті WEBRTC являє собою:

1. Media Streams (getUserMedia).

2. Peer Connection.

Media Streams – це API, що дозволяє отримати доступ до камери і мікрофону через браузер без будь-яких плагінів і флеша.

Peer Connection – це API, який дозволяє встановити Peer-to-Peer з'єднання між браузерами.

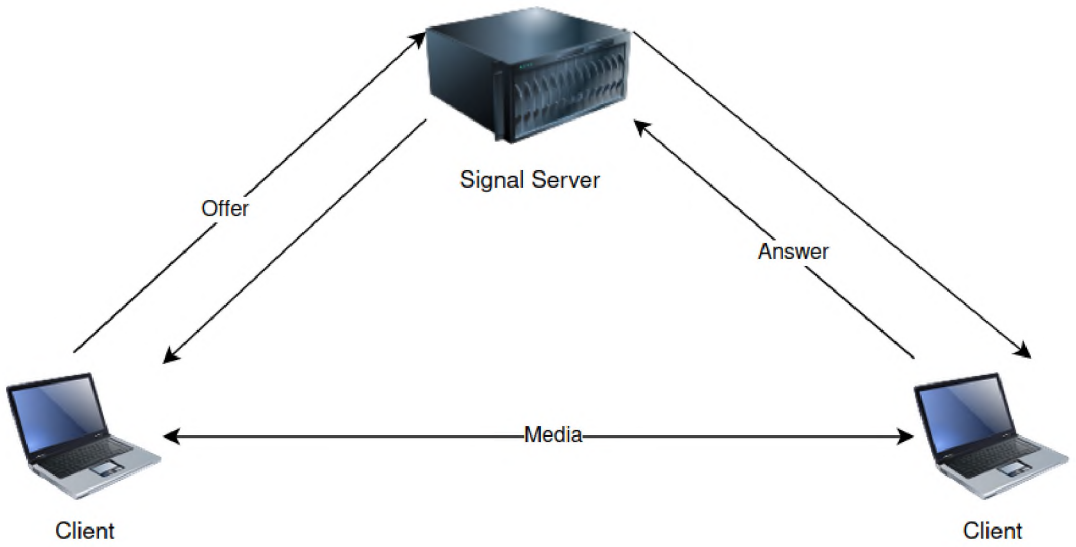


Рисунок 2.1 – З’єднання між клієнтами

1. Перший клієнт відправляє так званий Offer другому клієнту через сервер (Signal Server).

2. Другий клієнт відправляє через сервер відповідь першому клієнту.

3. Встановлюється P2P з’єднання між клієнтами.

Примітно, що в подальшому для роботи такого з'єднання сервер стає опціональним. Тобто після його виключення дані все також будуть передаватися. Подальша участь Signal Server’а потрібно для правильного закриття з'єднання, додавання учасників в потік і т.д.

Варто відзначити що метод з'єднання представлений вище працює тільки для двох клієнтів які з’єднуються між собою, у випадку якщо потрібно організувати з'єднання типу вебінар або відеоконференції то буде необхідний WEBRTC медіа сервер (рис. 2.2).

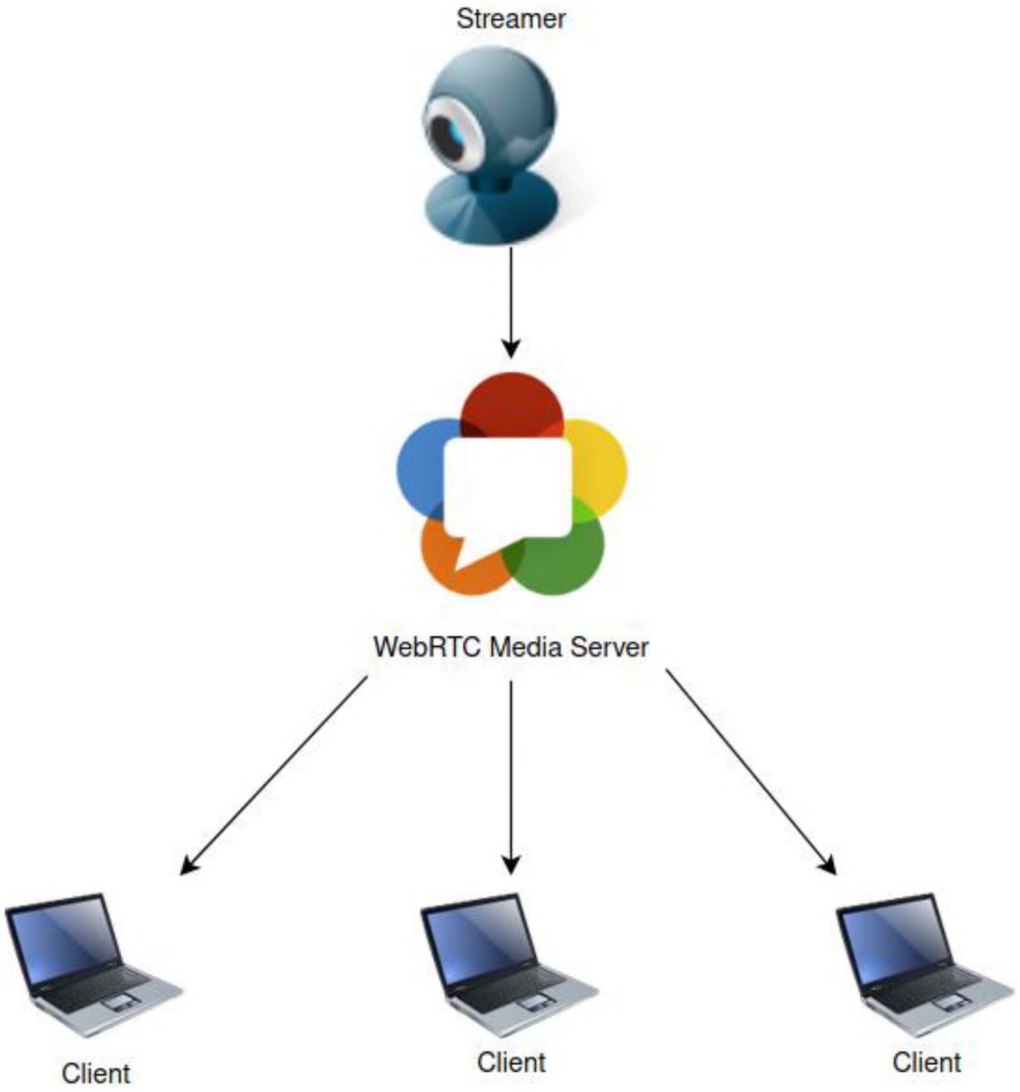


Рисунок 2.2 – З’єднання через WEBRTC Media Server

2.1.2 Дослідження STUN та TURN серверів

При ініціалізації WebRTC необхідно вказати доступні STUN і TURN сервери. Якщо серверу не будуть вказані, то з'єднатися зможуть тільки вузли в одній мережі (підключених до неї без NAT). Відразу варто відзначити, що для 3g-мереж обов'язково використання TURN серверів.

STUN сервер – це просто сервер в інтернеті, який повертає зворотну адресу, тобто адресу вузла відправника. Вузол, що знаходиться за роутером, звертається до STUN серверу, щоб пройти через NAT. Пакет, що прийшов до STUN серверу, містить адресу джерела – адреса роутера, тобто зовнішню адресу нашого вузла. Цю адресу STUN сервер і відправляє назад. Таким

чином, вузол отримує свій зовнішній IP адрес і порт, через який він доступний з мережі. Далі, WebRTC за допомогою цієї адреси створює додаткового кандидата (зовнішній адресу роутера і порт). Тепер в таблиці NAT роутера є запис, який пропускає до нашого вузла пакети, відправлені на роутер за потрібною порту.

Між вузлом і сервером може стояти не один роутер, а декілька. В такому випадку вузол отримає адресу того роутера, який першим виходить в ту ж мережу, що і сервер. Іншими словами, отримаємо адресу роутера, підключеного до STUN сервера. Для p2p комунікації це якраз те, що нам потрібно, якщо не забути той факт, що в кожному роутері додасться необхідний нам рядок в таблицю NAT. Тому зворотний шлях буде знову так само гладкий.

TURN сервер – це покращений STUN сервер. Звідси відразу слід витягти, що будь-який TURN сервер може працювати і як STUN сервер. Однак, є і переваги. Якщо p2p комунікація неможлива (як наприклад, в 3g мережах), то сервер переходить в режим ретранслятора (relay), тобто працює як посередник. Зрозуміло, ні про яке p2p тоді мова не йде, але за рамками механізму ICE вузли думають, що спілкуються безпосередньо.

Може виникнути питання, для чого потрібний TURN сервер якщо вистачає STUN сервера? Справа в тому, що існує декілька різновидностей NAT. Вони однаково підміняють IP адреса і порт, однак у деяких з них вбудований додатковий захист від "фальсифікації". Наприклад, в симетричній таблиці NAT зберігаються ще 2 параметра – IP і порт віддаленого вузла. Пакет із зовнішньої мережі проходить через NAT у внутрішню мережу тільки в тому випадку, якщо адреса і порт джерела співпадають з записаними в таблиці. Тому фокус зі STUN сервером не вдається – таблиця NAT зберігає адресу і порт STUN сервера і, коли роутер отримує пакет від WebRTC співрозмовника, він його відкидає, так як він "фальсифікований". Він прийшов не від STUN сервера.

Таким чином, TURN сервер потрібен в тому випадку, коли обидва співрозмовники знаходяться за симетричним NAT (кожен за своїм).

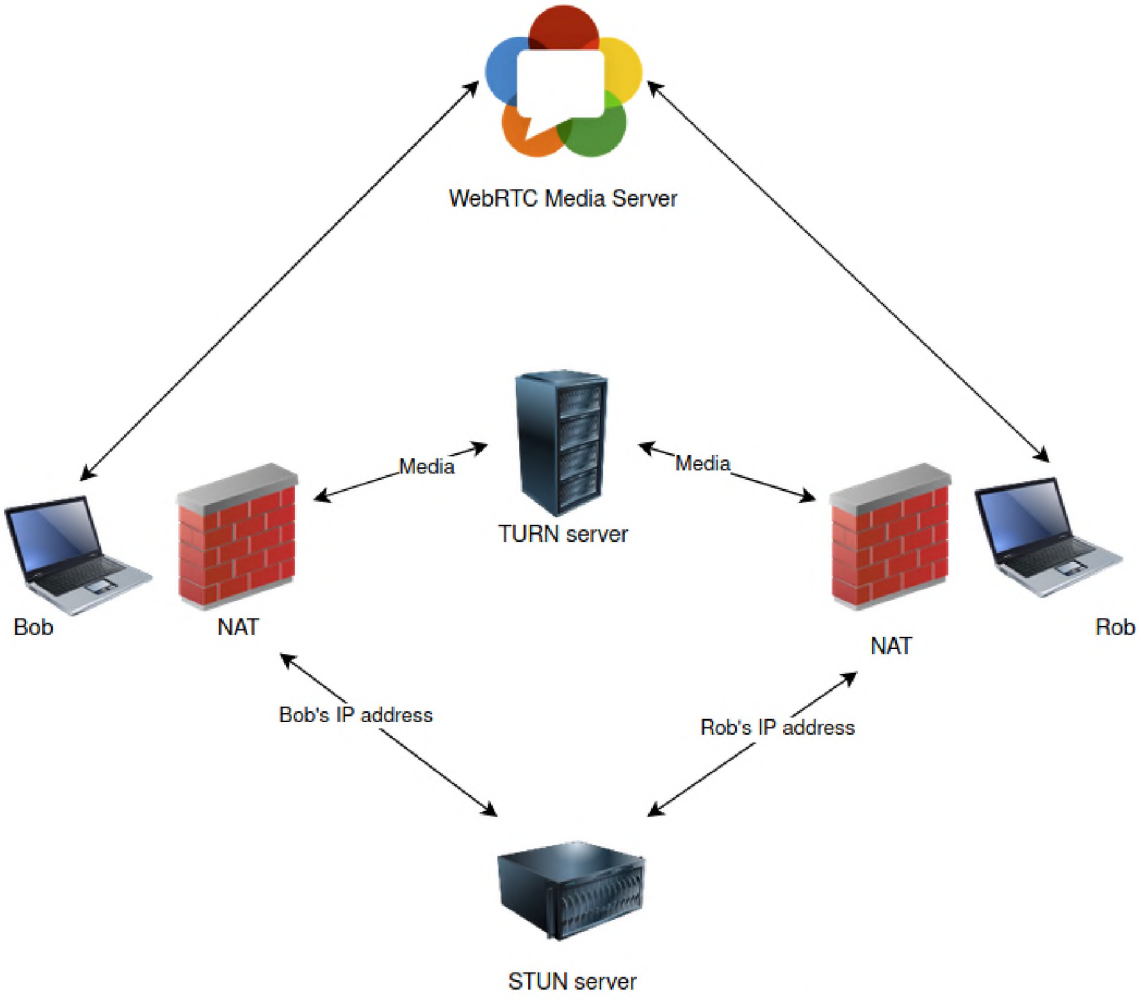


Рисунок 2.3 — STUN та TURN сервери

2.2 Вибір медіасервісної платформи

Серед існуючих медіа серверів які підтримують технологію WEBRTC, можно виділити наступні:

Wowza Media Server – Wowza Media Server з 2004 року використовується разом з Flash. Передача медіапотоків здійснюється по RTMP протоколу. Влітку 2016 року для Wowza Media Server вийшло оновлення з підтримкою WebRTC. Нова версія знаходиться в стадії beta-тестування. Для використання Wowza Media Server без обмежень необхідна платна ліцензія.

Ant Media Server – Ant Media дозволяє розробникам транслювати відео в прямому ефірі зі свого браузера за допомогою WebRTC, а прямий потік можна поширювати за допомогою RTMP та HLS завдяки WebRTC Adapter. Іншими словами, користувачі можуть транслювати відео в прямому ефірі з браузерів, як це робиться з флеш-плагіном, на щастя, цього разу немає необхідності в жодному стороннім плагіні. Але на жаль в безкоштовній версії Ant Media Server не можливо досягти мінімальної затримки під час трансляції.

Jitsi Videobridge – Jitsi Videobridge є частиною великого проекту Jitsi, який почав розроблятися ще в 2005 році, як SIP-комутатор. У 2014 розробники додали підтримку WebRTC.

Jitsi Videobridge – це маршрутизатор WebRTC потоків. Він перенаправляє медіа потоки між учасниками чату. Jitsi Videobridge володіє меншою функціональністю в порівнянні з медіасерверами. Рішення на його основі вимагають менше системних ресурсів. На відміну від медіасерверів Jitsi Videobridge не вміє записувати, транскодіровать або змішувати медіапотоків разом.

Kurento – це медіа-сервер WebRTC та набір клієнтських API, що спрощує розробку сучасних відео-додатків для комп'ютерів та смартфонів. Особливості Kurento Media Server включають групові комунікації, перекодування, запис, змішування, трансляцію та маршрутизацію аудіовізуальних потоків.

В якості диференціальної функції Kurento Media Server також надає розширені можливості обробки медіа, включаючи індексацію відео, розширену реальність та аналіз мовлення. Модульна архітектура Kurento спрощує інтеграцію алгоритмів обробки медіа сторонніх розробників (тобто розпізнавання мовлення, аналіз настроїв, розпізнавання облич тощо), які можна прозоро використовувати розробниками додатків як решта вбудованих функцій Kurento.

Для дослідження WEBRTC вирішено використовувати Kurento Media Server, так як Wowza Media Server та Ant Media Server вимагають додаткових грошових внесень, а Jitsi Videobridge володіє меньшим фукціоналом ніж перераховані вище медіа сервери, а також Jitsi Videobridge – це маршрутизатор потоків WEBRTC і не є медіа сервером.

2.3 Установка медіасервера Kurento

Для реалізації мети дипломного проекту на хмарному обчислювальному комплексі Google Cloud Platform було встановлено медіасервісну платформу Kurento.

Даний медіасервер підтримується операційною системою Linux.

Для початку потрібно установити серверну частину цього проекту, а саме Kurento Media Server (KMS). Це можна зробити двома способами:

1) Налаштувати локальну установку з yum/apt-get install. Цей метод дозволяє мати повний контроль під час процесу установки.

2) Використовуючи Kurento Docker images. Docker дозволяє запускати Kurento на будь-якій «машині», так, наприклад, можна запускати KMS в системі з операційною системою Fedora або CentOS. Теоретично це можливо навіть під керуванням Windows, але поки команда Kurento не досліджувала таку можливість.

Установка локально.

Для установки останньої версії медіасервера Kurento необхідно ввести наступні команди в терміналі:

- sudo apt-get update && sudo apt-get install --no-install-recommends --yes \ gnupg
- sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 5AFA7A83
- source /etc/upstream-release/lsb-release 2>/dev/null || source /etc/lsb-release
- sudo tee "/etc/apt/sources.list.d/kurento.list" >/dev/null <<EOF

deb [arch=amd64] http://ubuntu.openvidu.io/6.13.2 $DISTRIB_CODENAME kms6
EOF

- sudo apt-get update && sudo apt-get install --no-install-recommends --yes \
  kurento-media-server

Після установки медіасервера існує дві команди для управління:
sudo service kurento-media-server start – для запуска медіасервера.
sudo service kurento-media-server stop – для зупинки медіасервера.
Установка використовуючи Kurento Docker images

Для установки останньої версії медіасервера Kurento необхідно ввести наступні команди в терміналі:

- docker pull kurento/kurento-media-server:latest
- docker run --name kms -d -p 8888:8888 \
  kurento/kurento-media-server:latest

2.3.1 Установка клієнтської частини Kurento

Для того щоб упевнитись що Kurento Media Server був установлений без помилок та працює нормально необхідно встановити Kurento Magic Mirror за допомогою наступних команд:

- curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash –
- sudo apt-get install -y nodejs
- sudo npm install -g bower
- git clone https://github.com/Kurento/kurento-tutorial-node.git
- cd kurento-tutorial-node/kurento-magic-mirror
- git checkout master
- npm install
- npm start

2.4 Установка TURN сервера

Для того що би повністю реалізувати мету дипломного проекту під час виконання роботи також було розгорнуто та налаштовано TURN сервер.

TURN (Traversal Using Relay NAT) – це протокол, який дозволяє вузлу за NAT або брандмауером отримувати вхідні дані через TCP або UDP з'єднання. Така можливість особливо актуальна для вузлів позаду симетричних NAT, або брандмауерів, які збираються стати приймаючою стороною в з'єднанні з одним конкретним вузлом. TURN не призначений для перекидання портів сервера через NAT, він підтримує з'єднання точка-точка між вузлами розташованими за NAT (як в IP-телефонії). У цьому плані він зберігає функції безпеки, забезпечені симетричним NAT і брандмауерами, але змінює таблиці трансляції так, щоб вузол на внутрішній стороні міг стати приймаючою стороною з'єднання.

Coturn – це TURN сервер з відкритим вихідним кодом. Для його установки необхідно ввести наступні команди:

- sudo apt-get -y update
- sudo apt-get -y install Coturn

Для того щоб Coturn запускався при старті або перезапуску сервера потрібно ввести:

- sudo nano /etc/default/Coturn

Та необхідно розкоментувати (тобто видалити знак «#») строку:
TURNSERVER_ENABLED=1

Для конфігурування Coturn необхідно відкрити або створити файл /etc/turnserver.conf і потрібно задати свої данні:

<YOUR_USERNAME>, <YOUR_PASSWORD>, <INTERNAL_IP> та <YOUR_PUBLIC_IP_ADDRESS>

Для перезапуску Coturn ввести наступну команду:

- sudo service coturn restart

2.5 Порівняння сервісу онлайн-конференцій на базі технології WebRTC з популярними сервісами для трансляції в реальному часі Twitch і YouTube

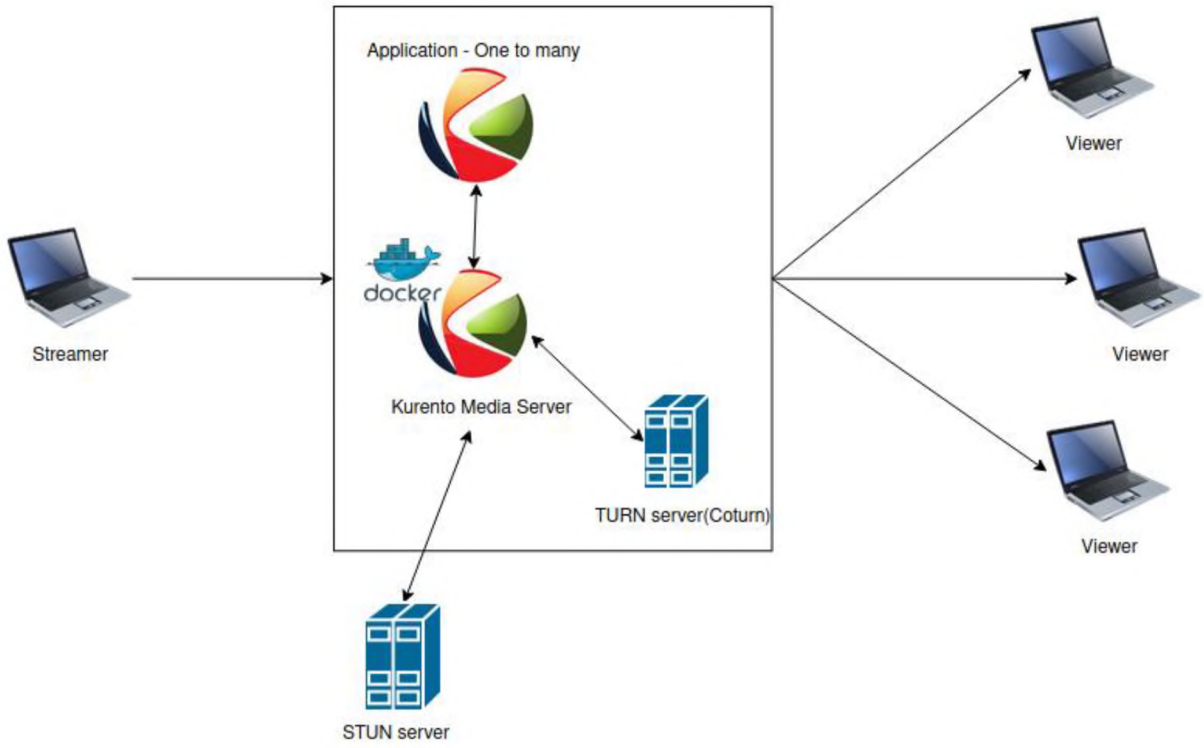


Рисунок 2.4 – Схема обробки даних медіасервісом Kurento

Для того щоб показати переваги технології WebRTC під час виконання дипломного проекту було прийнято рішення порівняти цю технологію з існуючими популярними сервісами Twitch та YouTube. Схема обробки даних медіасервісом Kurento наведена на рис. 2.4.

Одним з найважливішим фактором під час дослідження медіасервісу є затримка та якість відеопотоку. Але нас найбільше цікавить саме затримка.

При дослідженні Kurento Media Server виявлено що KMS працює з аудіокодеком OPUS та відеокодеком VP8.

OPUS – це кодек 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 пропускною спроможністю. Так же OPUS — це 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.

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.

Під час виконання дипломного проекту було встановлено що загальна затримка відправки та отримання відеопотоку між браузерами складає 2 - 3 секунди, за умови що браузери знаходяться в одній мережі. У випадку якщо браузери знаходяться в різних мережах (що для нас є більш необхідним) затримка складає 3 - 5 секунд. Необхідно зазначити що передача відеопотоку відбувалася при якості (роздільній здатності) 720 пікселів та фреймрейтом (частота кадрів) 30 FPS.

Відомо що однією з властивостей технології WEBRTC є те що, задля забезпечення мінімальної затримки в технології WEBRTC нехтують якістю зображення. Тому якщо у одного з користувачів виникнуть проблеми з якістю мережі, то якість зображення в нього значно погіршиться.

Також було встановлено що при використанні додатку типу вебінар (one to many), тобто з одного джерела транслюється відео на багато різних пристроїв, було виявлено що окрім проблеми з якістю мережі через яку

знижується якість, наявна проблема множинних підключень, тобто при великій кількості підключень до одного джерела може спостерігатись погіршення якості зображення. Але цю проблему можна вирішити збільшенням технічних характеристик серверу та збільшенням пропускної здатності мережі на стороні джерела.

2.5.1 Розгляд платформ Twitch та YouTube

В дипломному проекті розглянемо основні відомі платформи Twitch та YouTube (рис 2.5).

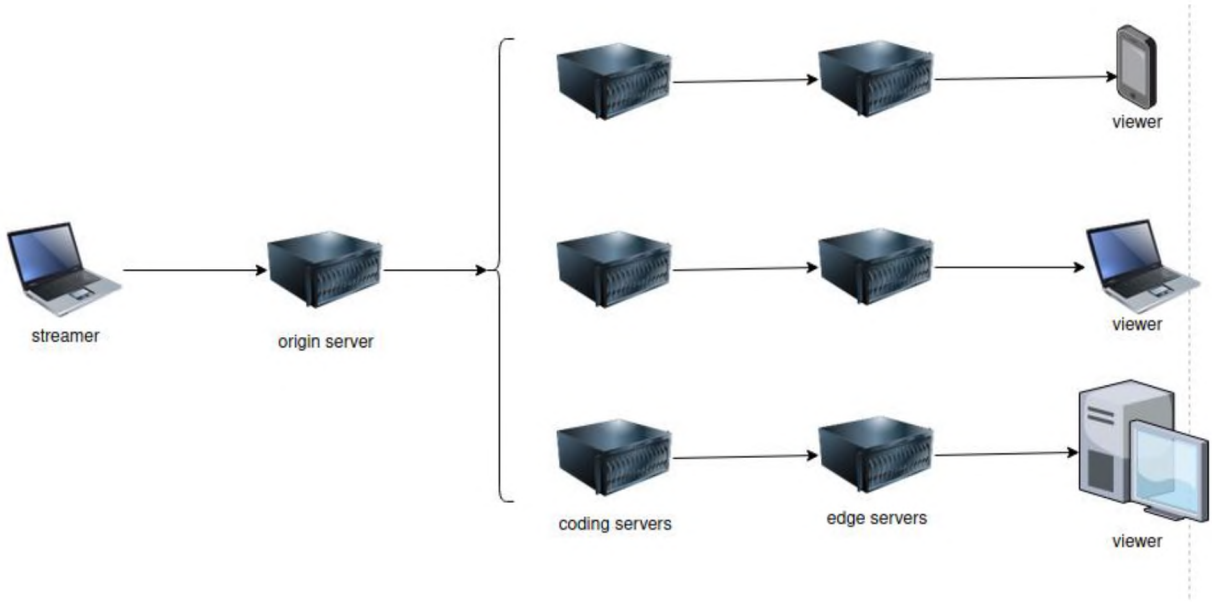


Рисунок 2.5 – Схема обробки даних медіасервісами Twitch та YouTube

Є декілька серверів(origin server) які отримують трансляцію від стримера, з них відео забирають сервери(coding server) для перекодування в різні роздільні здатності, а вже з них забирають відеопотік сервери(edge servers) які віддають відео глядачам, яких може бути багато.

З рис 2.5 видно що з усього ланцюга транслювання відео більшість часу витрачається на перекодування та на відправку відео на сервери, щоб вони відправили відео глядачам. З одного боку, це збільшує затримку, а з іншого якість відео залишається на досить високому рівні.

Під час виконання дипломного проекту виконано порівняння зазначених технологій (Twitch, YouTube, WebRTC). Результати порівняння наведені в табл 2.1.

Таблиця 2.1 – Порівняння сервісу на базі технології WebRTC з Twitch і YouTube

Критерії	WebRTC	YouTube	Twitch
Затримка	3-5 сек.	20-30 сек.	10-15 сек.
Установка доп. ПЗ	Не потрібно	Потрібно	Потрібно
Максимальна якість	720p	1080p	1080p
Максимальна частота кадрів	30 FPS	60 FPS	60 FPS
Аудіокодек	OPUS	AAC-LC	H.264
Відеокодек	VP8	AVC	AVC

З табл 2.1 видно, що YouTube і Twitch роблять упор на якість трансляції, в той час як затримка сягає великих значень. Через те що YouTube і Twitch є комерційними проектами в основу яких лежить пропрієтарне програмне забезпечення не представляється можливим виконати виміри затримки, але з іншого боку ці платформи в своїй документації наводять інформацію про те що в платформі YouTube стандартний час затримки складає 20-30 секунд, а в Twitch 10-15 секунд. З цього можна зробити висновки що YouTube і Twitch роблять упор на якість трансляції, в той час як затримка за рахунок обробки інформації в цих медіасервісах може сягати великих значень. З метою досягнення мінімальної затримки в технології WebRTC виконується зниження якості до 720 пікселів, в той час як YouTube і Twitch можуть дозволити собі трансляцію в якості (роздільній здатності) 1080 пікселів.

Ще одною основною перевагою сервісу на базі технології WebRTC є те, що для трансляції не потрібна установка додаткового ПЗ. Виходячи з результатів порівняння, сервіс на базі технології WebRTC не поступається сервісам YouTube і Twitch.

Підводячи підсумок можна сказати що мадіасервіс на базі технології WebRTC не поступається медіасервісам YouTube та Twitch.

2.6 Висновки до другого розділу

В даному розділі виконано дослідження медіа платформи Kurento на базі технології WebRTC. Проведено порівняння платформи Kurento з іншими платформами потокового мовлення, зокрема Twitch та YouTube.

Розглянуто принцип роботи медіасервісних платформ Kurento, YouTube та Twitch.

На основі хмарних технологій виконано розгортання медіасервісної платформи Kurento. Встановлено, що для підключення користувачів які знаходяться за NAT (Network Address Translation) необхідно додатково розгорнути та налаштувати TURN сервер.

Досліджено медіасервісну платформу Kurento на базі технології WebRTC і проведено порівняльний аналіз з медіасервісними платформами Twitch та YouTube.

3 ЕКОНОМІЧНА ЧАСТИНА

3.1 Визначення трудомісткості дослідження сервісів потокового мовлення.

Трудомісткість – показник, який характеризує витрати робочого часу на виробництво будь-якої споживчої вартості або на виконання конкретної технологічної операції. Трудомісткість дослідження сервісів потокового мовлення можливо розрахувати за формулою (3.1):

$$t = to + tдо + tдн + tроз + tп \quad (3.1)$$

де to – витрати праці на підготовку і опис поставленого завдання;

tдо – витрати праці на дослідження технології WebRTC;

tдн – витрати праці на дослідження API-технології WebRTC;

tроз – витрати праці на тестування технології WebRTC;

tп – витрати праці на підготовку документації та детальний опис запропонованого технічного рішення.

У таблиці 3.1 зведені данні тривалості процесів, що мали місце при розробці методики оцінки якості функціонування мережі NGN.

Отже, загальна трудомісткість за формулою 3.1:

$t = 5 + 45 + 50 + 30 + 38 = 168$ годин

Таблиця 3.1 – Тривалість робочих процесів

Назва робочого процесу	Тривалість, год.
Витрати праці на підготовку і опис поставленого завдання	5
Витрати праці на дослідження технології WebRTC	45
Витрати праці на дослідження API-технології WebRTC	50
Витрати праці на р тестування технології WebRTC	30
Витрати праці на підготовку документації та детальний опис запропонованого технічного рішення	35

3.2 Розрахунок витрат на дослідження сервісів потокового мовлення

Витрати на включають витрати на заробітну плату інженера телекомунікацій і вартість машинного часу.

Заробітна плата – це винагорода, яку за трудовим договором власник або уповноважений ним орган виплачує працівнику за виконану роботу. Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та господарської діяльності підприємства в цілому. Заробітна плата інженера телекомунікацій визначається за формулою 3.2:

$$З_{зп} = t \cdot С_{пр} \quad \text{грн} \quad (3.2)$$

де t – загальна трудомісткість розробки, яка розраховується за формулою 3.1, годин;

$C_{пр}$ – середня годинна заробітна плата інженера телекомунікацій (основна і додаткова) з урахуванням єдиного соціального внеску, грн/год.

Середня заробітна плата інженера телекомунікацій на 01.06.2020р. Складає 13 000 грн. Отже, заробітна плата інженера телекомунікацій з урахуванням премій (25%) і можливих надбавок (15%) складає 18 688 грн. Таким чином, річний фонд заробітної плати – 224 256 грн. Єдиний соціальний внесок складає 36%, тобто 80 732,16грн.

Річний фонд заробітної плати включає: фонд денної зарплати; оплату відпусток; оплату часу, витраченого на виконання держобов'язків; виплати відрядженим на інші підприємства; оплату за вислугу років та ін. Разом, річний фонд заробітної плати з урахуванням відрахувань на соціальні потреби склав 304 988,16 грн.

Номінальний річний фонд часу роботи одного робітника визначається відніманням з повного календарного фонду часу за рік неробочих (вихідних і святкових) днів, відпустки. Він є максимально можливим часом, протягом якого могла б вироблятися робота при встановленому режимі, якби не було жодних втрат робочого часу.

Визначимо номінальний річний фонд робочого часу за формулою 3.3, при цьому прийнявши середню тривалість робочого дня рівної 8 годинам:

$$F_н = (T_к - T_{св} - T_{вих} - T_{від}) \cdot 8 \qquad го$$

де $T_к$ – кількість календарних днів у році, $T_к = 365$ днів;

$T_{св}$ – кількість святкових днів у році, $T_{св} = 10$ днів;

$T_{вих}$ – кількість вихідних днів у році, $T_{вих} = 104$ дня;

$T_{від}$ – календарна тривалість відпустки, Твід = 25 днів.

Отже, річний фонд часу за формулою 3.3 дорівнює:

Fн = (365 – 10 – 104 - 25)8 = 1808 годин

Середня годинна заробітна плата інженера телекомунікацій визначається співвідношенням 3.4, яка має вигляд:

$$C_{зп} = \frac{ФЗП_{сп}}{F_н} \quad грн/г \quad (4)$$

де $ФЗП_{сп}$ – річний фонд заробітної плати з урахуванням відрахувань на соціальні потреби;

$F_н$ – річний фонд робочого часу.

Отже середня годинна заробітна плата інженера телекомунікацій за формулою 3.4 дорівнює:

$С_{ЗП}$ = 304 988,16 / 1808 = 168,69

Таким чином, витрати на оплату праці дослідника складають з урахуванням формули 3.2 отримаємо:

$З_{ЗП}$ = 168 × 168,69 = 28 339,92 грн

Розрахунок вартості машинного часу, необхідного для дослідження на ЕОМ включає витрати на програмне та апаратне забезпечення і витрати за електроенергію, здійснюється по формулі 3.5:

$$З_{мч} = C_o + C_{ел} \quad (3.5)$$

де C_o – витрати на обладнання, грн;

$C_{ел}$ – витрати на електроенергію, грн.

Для розрахунку вартості машино-часу необхідно знати вартість ЕОМ та ПЗ на момент їх придбання і введення в експлуатацію, і вартість споживаної електроенергії. Відповідні дані представлені в таблиці 3.2.

Витрати на електроенергію залежать від часу роботи на ЕОМ та собівартості машино-години роботи ЕОМ і розраховується за формулою:

$$C_{ел} = C_{мч} \cdot t \quad (3.6)$$

$$C_{мч} = W \cdot Ц_{ел} \quad (3.7)$$

де W – потужність ЕОМ, W = 0,8 кВт/год;

$Ц_{ел}$ – вартість 1$кВт \cdot год$ електроенергії.

Таблиця 3.2 – Вартість необхідного програмного та апаратного

Найменування	Вартість, грн
Персональний комп’ютер	16 500,00
Монітор	4 200,00
Комплект Клавіатура+миша Logitech Desktop	862,00
Операційна система Windows 10 Pro	7 500,00
Matlab 6.5	4500,00
Разом:	33 562

забезпечення

З 1.04.19 за обсяг, спожитий понад 150 кВт·год до 600 кВт·год електроенергії на місяць (включно) складає 2,1 грн.

$С_{ел} = 0,8 \times 2,1 \times 168 = 282,24$ грн

Враховуючи вартість програмного й апаратного забезпечення та витрати на електроенергію отримаємо вартість машинного часу:

$З_{МЧ} = 282,24 + 33\ 562 = 33\ 844,24$ грн

Отже, витрати на дослідження сервісів потокового мовлення:

$В_{РОЗ} = 33\ 844,24 + 28\ 339,92 = 62\ 184,16$ грн

3.3 Висновок

В економічному розділі виконаний розрахунок трудомісткості дослідження сервісів потокового мовлення, заробітної плати інженера телекомунікацій, капітальні затрати. Загальні капітальні витрати становлять 62 184,16 грн. трудомісткість дослідження сервісів потокового мовлення – 168 годин.

ВИСНОВКИ

В дипломному проекті виконано дослідження медіасервісної платформи Kurento з використанням технології WebRTC. Під час виконання дипломного проекту встановлено що використання технології WebRTC дозволяє забезпечити мінімальну затримку доставки контенту. Виконані порівняльні характеристики технології WebRTC з технологією Adobe Flash. Встановлено, що технологія WebRTC забезпечує мінімальну затримку передачі медіаданих за рахунок протоколу UDP та за рахунок кросплатформеності.

Під час виконання дипломного проекту виконано розгортання та налаштування медіасервісної платформи Kurento за допомогою хмарних технологій. Досліджена затримка доставки медіаконтенту в медіасервісній платформі Kurento з використанням технології WebRTC. Вимірювання затримки медіаконтенту в медіасервісній платформі склав 3-5 секунд.

Виконано порівняння ефективності роботи медіасервісної платформи Kurento з відомими медіасервісними платформами Twitch та YouTube. Встановлено, що технологія WebRTC забезпечує мінімальну затримку доставки медіаконтенту, проте це забезпечується за рахунок зниження роздільної здатності. При максимальному завантаженні медіасервісної платформи роздільна здатність відеоконтенту не перевищує 720 пікселів. В той час як платформи Twitch та YouTube не знижують роздільну здатність нижче чим 1080 пікселів. Про те в цих платформах (Twitch та YouTube) затримка може складати 20-30 та 10-15 секунд відповідно.

ПЕРЕЛІК ПОСИЛАНЬ

1. Red5 [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Red5

2. Installation and operation with Red5 Server [Електронний ресурс]. – Режим доступу: https://www.pvsm.ru/flash/7197

3. Roc 0.1, Ant 1.7, and Red5 streaming server release [Електронний ресурс]. – Режим доступу: https://www.opennet.ru/opennews/art.shtml?num=50788

4. Ant Media [Електронний ресурс]. – Режим доступу: https://antmedia.io/

5. Ant Media Server [Електронний ресурс]. – Режим доступу: https://github.com/ant-media/Ant-Media-Server/wiki

6. Nimble Streamer [Електронний ресурс]. – Режим доступу: http://wikireality.ru/wiki/Nimble_Streamer

7. Streaming Support in Nimble Streamer [Електронний ресурс]. – Режим доступу: https://ru.wmspanel.com/nimble/live_streaming

8. Softvelum Low Delay Protocol [Електронний ресурс]. – Режим доступу: https://ru.wmspanel.com/nimble/sldp

9. Broadcast online video with minimal delay [Електронний ресурс]. – Режим доступу: https://itnan.ru/post.php?c=1&p=265675

10. Java – One to many video call [Електронний ресурс]. – Режим доступу: https://doc-kurento.readthedocs.io/en/6.10.0/tutorials/java/tutorial-one2many.html

11. Java - One to one video call [Електронний ресурс]. – Режим доступу: https://doc-kurento.readthedocs.io/en/6.13.0/tutorials/java/tutorial-one2one.html

12. WebRTC API [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/ru/docs/Web/API/WebRTC_API

13. 5 mistakes when developing WebRTC calls [Електронний ресурс]. – Режим доступу: https://habr.com/ru/company/Voximplant/blog/351234/

14. Trickle ICE [Електронний ресурс]. – Режим доступу: https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/

15. WebRTC: Configure Your Own TURN / STUN Server [Електронний ресурс]. – Режим доступу: https://www.red5pro.com/docs/server/turnstun.html

16. WebRTC [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/WebRTC

17. Setup STUN / TURN server for NAT traversal [Електронний ресурс]. – Режим доступу: https://www.nomachine.com/AR07N00894

18. NAT [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/NAT

ДОДАТОК А. Перелік документів КВАЛІФІКАЦІЙНОЇ РОБОТИ

1 Пояснювальна записка на 60 сторінках.

2 Матеріали кваліфікаційної роботи на оптичному носії:

- Пояснювальна записка Паляничка І.В.docx
- Презентація.pptx

ДОДАТОК Б. Відгук керівника економічного розділу

Керівник розділу _____ Романюк Н.М.

(підпис) (прізвище, ініціали)

ДОДАТОК В. ВІДГУК КЕРІВНИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

ВІДГУК

на кваліфікаційну роботу ступеня бакалавр студента групи 172-17ск-1
Паляничка І.В. на тему:
«Дослідження сервісів потокового мовлення»

Мета кваліфікаційної роботи зменшення затримки в сервісах потокового мовлення шляхом вибору оптимального програмного забезпечення.

Обрана тема є актуальною через те що останнім часом постає необхідність у використанні медіасервісів потокового мовлення в реальному часі в першу чергу для нормального функціонування робочого та навчального процесу, та навіть для звичайної зустрічі з родичами та друзями. Що до робочого процесу, коли необхідно приймати рішення як найшвидше постає питання у вибору медіасервісу з мінімальною затримкою. Наразі серед існуючих технологій які забезпечують мінімальну затримку передачі відео та аудіо виділяють стандарт WebRTC.

В кваліфікаційній роботі розглянуто технічну розробку, що відноситься до процесу передачі медіаданих від постачальника послуг до споживача з допомогою медіасервісних платформ.

Технічним результатом отриманим в кваліфікаційній роботі є підтвердження можливості застосування медіасервісної платформи Kurento сумісно з технологією WebRTC для доставки медіаконтенту користувачам з мінімальною затримкою.

Оформлення кваліфікаційної роботи виконано на відповідному рівні і відповідає вимогам, що пред'являються до робіт даної кваліфікації. В цілому кваліфікаційна робота ступеня бакалавра повністю задовольняє вимогам, що пред'являються і заслуговує оцінки «добре», а її автор, Паляничка І.В. присвоєння кваліфікації технічного фахівця в галузі електроніки та телекомунікацій.

Керівник кваліфікаційної роботи, доц. Магро В.І.

ДОДАТОК Г. Рецензія

РЕЦЕНЗІЯ

на кваліфікаційну роботу ступеня бакалавр студента групи 172-17ск-1
Паляничка І.В. на тему:
«Дослідження сервісів потокового мовлення»

Представлена на рецензію кваліфікаційна робота виконана на 71 машинописній сторінці і складається з вступу, трьох розділів і висновку. Кваліфікаційна робота студента групи 172-17ск-1 Паляничка І.В. виконана в повному обсязі відповідно до завдання.

WebRTC – це новий етап у розвитку інтернет-комунікацій. Даний стандарт дозволяє досягти мінімальної затримки передачі відео та аудіо що зараз є суттєвою вимогою при виборі медіасервісів, а також дає змогу перетворити браузер в кінцевий термінал відеоконференцзв'язку, що дає перевагу перед іншими технологіями за рахунок того, що спілкування відбувається в реальному часі без установки плагінів і інших розширень. Також технологія WebRTC дає шанс розробникам трансформувати зв'язок, надаючи надійні та безпечні комунікації корпоративного класу. Це відкриває можливості для організації вебінарів, онлайн-нарад, відеоконференцій та інших заходів. Тому дослідження даної теми є актуальним завданням.

Позитивні сторони:

1. Зміст кваліфікаційної роботи відповідає завданню. Робота, яка виконана Паляничка І.В. , демонструє високий рівень знань і ступінь підготовленості його до майбутньої роботи з фаху.
2. Розглянуто загальну концепцію покращення часу доставки контенту.
3. Текст викладений грамотно, ясно, послідовно. Графічний матеріал оформлений якісно. Широко використовується науково-технічна література.

Недоліки:

1. Оглядова частина роботи присвячена тільки медіа платформі Kurento, а інформація про інші платформи відсутня.
2. Слід було б оцінити ефективність інших платформ.

В цілому кваліфікаційна робота ступеня бакалавр повністю задовольняє вимогам, що пред'являються і заслуговує оцінки «добре», а її автор, Паляничка І.В. , присвоєння кваліфікації технічного фахівця в галузі електроніки та телекомунікацій.

Рецензент,

_____
(підпис)