спеціалізованих рішень для різних галузей. Запропоновані рекомендації дозволяють забезпечити високу продуктивність, надійність та безпеку мереж зв'язку, що є важливим аспектом для забезпечення якісного зв'язку та взаємодії між користувачами та системами.

ЛІТЕРАТУРА

1. Software-Defined Networking for Communication Networks / S. Patel, N. Shah. // Communications Magazine. 2023. **2.** Network Functions Virtualization: Concepts and Applications / Y. Zhang, X. Chen. // IEEE Network. – 2022. **3.** A Survey of Modern Communication Networks / J. Smith, K. Lee. // Journal of Network and Computer Applications. – 2021. **4.** Security in Software-Defined Networking / L. Wang, M. Li. // Security and Privacy. – 2023. **5.** The Future of Networking: SDN and NFV / T. Brown, J. Wilson. // Networking Today. – 2024. **6.** Implementing NFV in Large-Scale Networks / R. Garcia, M. Hernandez. // International Journal of Network Management. – 2023. **7.** Комп'ютерні мережі. Загальні принципи функціонування комп'ютерних мереж. Навчальний посібник. С. В. Мінухін, С. В. Кавун, С. В. Знахур. – Харків: Вид. ХНЕУ, 2008. – с. (Укр. мов.) **8.** Дослідження впровадження нових послуг в мережах операторів зв'язку / [Шмігель Р.Р. та ін.]. // Матеріали VII міжнародної науково-технічної конференції молодих учених та студентів «Актуальні задачі сучасних технологій» Тернопільського національного технічного університету імені Івана Пулюя, (Тернопіль, 28 – 29 листопада 2018 р.). – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя – 2018. – С. 154. **9.** Методи і засоби підвищення надійності комп'ютерних мереж / [Шмігель Р.Р. та ін.]. // Матеріали VII міжнародної науково-технічної конференції молодих учених та студентів «Актуальні задачі сучасних технологій» Тернопільського національного технічного університету імені Івана Пулюя, (Тернопіль, 28 – 29 листопада 2018 р.). – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя – 2018. – С. 155.

УДК 004.5

ВИКОРИСТАННЯ MESSAGE BROKER У WEBRTC ДЛЯ МАСШТАБУВАННЯ

Кхатер Ф. Е.,
здобувач третього (освітньо-наукового) рівня вищої освіти
Приватного вищого навчального закладу
«Міжнародний економіко-гуманітарний університет
імені академіка Степана Дем'янчука» (м. Рівне, Україна)

Науковий керівник: **Джунь Й. В.,**
доктор фізико-математичних наук, професор, завідувач кафедри математичного моделювання
Приватного вищого навчального закладу
«Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука» (м. Рівне, Україна)

Анотація. *У статті розглядається важливість використання Message Broker у середовищі WebRTC для ефективної передачі даних та управління потоками інформації. Ми аналізуємо існуючі рішення, обговорюємо їх переваги, а також розглядаємо перспективи розвитку даної технології в контексті сучасних вимог до комунікаційних систем.*

Ключові слова: *message broker, webrtc, передача даних, потоки інформації.*

Abstract. *This article discusses the importance of using a Message Broker in the WebRTC environment for efficient data transmission and information stream management. We analyze existing solutions, discuss their advantages, and explore the prospects for the development of this technology in the context of modern requirements for communication systems.*

Keywords: *message broker, webrtc, data transfer, information flows.*

WebRTC (Web Real-Time Communication) є потужним інструментом для створення веб-застосунків, що дозволяють здійснювати обмін аудіо -, відео - даними у реальному часі між браузерами без необхідності встановлення додаткових плагінів або програм. Однак ефективне управління потоками даних та забезпечення надійної доставки повідомлень залишаються актуальними завданнями в контексті WebRTC.

Мета статті полягає, щоб показати роль Message Broker у WebRTC: Message Broker відіграє ключову роль у керуванні повідомленнями та даними у системах WebRTC. Його основні функції включають маршрутизацію повідомлень, контроль доступу, обробку помилок та забезпечення гарантованої доставки повідомлень.

Переваги використання Message Broker у WebRTC:

– Масштабованість: Message Broker забезпечує гнучку масштабованість, дозволяючи ефективно керувати великим обсягом повідомлень та збільшувати пропускну спроможність системи за потреби.

– Надійність: Використання Message Broker підвищує надійність системи за рахунок обробки помилок та гарантованої доставки повідомлень навіть в умовах неполадок у мережі.

– Гнучкість: Message Broker надає гнучкі інструменти для конфігурації та налаштування, дозволяючи розробникам адаптувати його під конкретні вимоги додатка.

– Безпека: За допомогою Message Broker можна реалізувати механізми аутентифікації та авторизації, забезпечуючи безпеку передачі даних у системі WebRTC.

На сьогодні існує багато реалізацій Message Broker, які можуть бути інтегровані з WebRTC. Деякі з найпопулярніших варіантів включають Apache Kafka, RabbitMQ, NATS та MQTT. Кожен з них має свої особливості та переваги, а вибір конкретного рішення залежить від вимог та характеристик конкретного застосунку.

Використання Redis у якості механізму Message broker починається з реалізації шаблону «Видавець/підписник» з аналізу найпопулярнішого інструменту Redis (http://redis.io) - швидкого та гнучкого сховища пар ключ / значення, часто використовуваного як сервер структур даних. Хоча Redis, суттєво, більше як база даних, ніж брокер повідомлень, серед його можливостей є команди, спеціально призначені для реалізації централізованого шаблону «Видавець/підписник».

Переваги цієї реалізації в тому, що вона є простою порівняно з більш розвинутими інструментами обміну повідомленнями, але саме ця простота визначає її популярність. Redis часто вже присутній в існуючій інфраструктурі, наприклад, як сервер кешування або сховище сеансів. Його швидкість та гнучкість часто роблять його вибором для використання в розподілених системах для обміну даними. Таким чином, для включення брокера у реалізацію шаблону «Видавець/підписник» найпростішим та швидким рішенням буде використання Redis, а не встановлення та підтримка окремого брокера повідомлень. Давайте продемонструємо простоту та потенціал цього рішення на прикладі.

Для цього прикладу потрібно встановити Redis та налаштувати прослуховування його порту за замовчуванням. Детальніше про це можна дізнатися на сторінці http://redis.io/topics/quickstart.

Тепер ми повинні інтегрувати сервери чату, використовуючи Redis як механізм передачі повідомлень. Усі екземпляри будуть публікувати будь-які повідомлення, які отримують від своїх клієнтів, в Redis, а також підписуватися як отримувачі на всі повідомлення, що надходять від інших екземплярів. Іншими словами, всі сервери одночасно будуть як видавцями, так і підписниками.

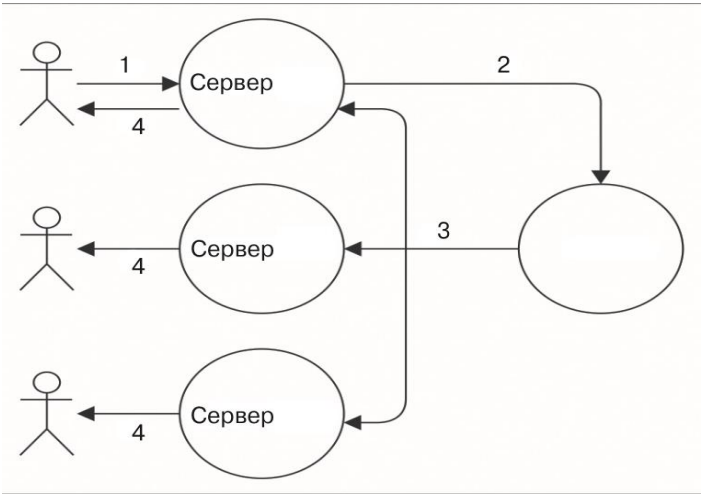

Рис. 1. Схема роботи архітектури

Повідомлення вводиться у текстове поле веб-сторінки та відправляється до сервера чату.

Сервер публікує повідомлення у Redis. Redis розсилає повідомлення всім підписникам, які у цій архітектурі є усіма серверами чату. Кожен сервер розсилає отримане повідомлення своїм клієнтам.

Сервер Redis дозволяє публікувати та підписуватися на канали за допомогою строкових ідентифікаторів, наприклад, chat.nodejs. Він також дозволяє використовувати глобальні шаблони для визначення підписок, які охоплюють кілька каналів, наприклад, chat.*.

Давайте розглянемо, як це працює на практиці. Змінимо код сервера, додавши логіку видавця/підписника:

Зміни, внесені у код рис. 2 оригінального сервера чату, позначені жирним шрифтом. Ось як працює змінений код:

Для підключення до сервера Redis у застосунках для Node.js використовується пакет redis (https://npmjs.org/package/redis) - повноцінний клієнт, що підтримує всі доступні команди Redis. Після завантаження модуля створюються два з'єднання: одне для підписки на канал, а інше - для публікації повідомлень. Це необхідно при використанні Redis, оскільки після переведення з'єднання у режим підписки до нього можна застосовувати лише команди, що стосуються підписки. Це означає, що потрібне ще одне з'єднання для публікації повідомлень.

```
const WebSocketServer = require('ws').Server;
const redis = require("redis");
const redisSub = redis.createClient();
const redisPub = redis.createClient();

const server = require('http').createServer(
  require('ecstatic')({root: `${ dirname}/www`})
);

const wss = new WebSocketServer({server: server});
wss.on('connection', ws => {
  console.log('Client connected');
  ws.on('message', msg => {
    console.log(`Message: ${msg}`);
    redisPub.publish('chat_messages', msg);
  });
});

redisSub.subscribe('chat_messages');
redisSub.on('message', (channel, msg) => {
  wss.clients.forEach((client) => {
    client.send(msg);
  });
});

server.listen(process.argv[2] || 8080);
```

Рис. 2. Приклад коду з використанням message broker

Коли від клієнта надходить нове повідомлення, воно публікується в каналі chat_messages. Повідомлення не розсилаються клієнтам безпосередньо; оскільки сервер підписаний на той самий канал (як показано нижче), воно буде розсилатися через сервіс Redis. У цьому випадку це дуже простий та ефективний механізм.

Як вже зазначалося раніше, сервер також повинен підписатися на канал chat_messages, тому ми реєструємо обробника для приймання всіх повідомлень, опублікованих у цьому каналі (через поточний або будь-який інший сервер чату). Після отримання повідомлення воно розсилається всім клієнтам, підключеним до веб-сокету поточного сервера.

Цих невеликих змін вистачить для інтеграції всіх запущених серверів чату. Щоб переконатися в цьому, запустіть кілька екземплярів застосунка:

```
node app 8080
node app 8081
node app 8082
```

Рис. 3. Приклад запуску серверів.

Тепер можна створити декілька вкладок у браузері - по одній для кожного екземпляра, - і переконатися, що всі повідомлення, відправлені на один із серверів, успішно доставляються всім клієнтам, підключеним до різних серверів. Вітаємо! Ми щойно за допомогою шаблону «Видавець/підписник» інтегрували розподілене програмування в реальному часі яке дає змогу динамічно масштабуватись і комунікувати між серверами.

Отже, використання Message Broker є ключовим елементом для забезпечення ефективної передачі даних та керування повідомленнями в системах WebRTC. Розробники можуть отримати значні переваги, інтегруючи відповідне рішення Message Broker у свої застосунки, що дозволяє створювати більш потужні та надійні веб-застосунки для реального часу комунікації.

ЛІТЕРАТУРА

1. Garcia, C., & Garcia, P. (2020). "Integrating WebRTC with Message Brokers: A Comparative Study." Proceedings of the International Conference on Web Engineering (ICWE), 145-158. **2.** Miller, J. (2019). "WebRTC Messaging: Best Practices and Integration Strategies." WebRTC Journal, 10(2), 33-45. **3.** Smith, A., & Johnson, B. (2018). "Message Brokers in WebRTC Applications: A Practical Guide." Journal of Real-Time Communication Systems, 25(4), 567-580. **4.** WebRTC Consortium. (2022). "WebRTC Specifications and Standards." Retrieved from https://www.w3.org/TR/webrtc/. **5.** RabbitMQ Documentation. (2023). "Official Documentation." Retrieved from https://www.rabbitmq.com/documentation.html