УДК [004.773.5+621.391]:004.85

РАЗРАБОТКА WEBRTC-ПРИЛОЖЕНИЯ ДЛЯ ВИДЕОСВЯЗИ 1:1 С ТЕКСТОВЫМ ЧАТОМ И ОБЩЕЙ ГРАФИЧЕСКОЙ ДОСКОЙ НА БАЗЕ SPRING BOOT С ЭЛЕМЕНТАМИ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ И МАШИННОГО ОБУЧЕНИЯ

А.О. ШЕВЧЕНКО, М.И. ЗУБАРЕВ

Белгородский государственный технологический университет им. В.Г. Шухова
(г. Белгород, Россия)

E-mail: alinealina941@gmail.com

Аннотация. Статья посвящена разработке учебного прототипа веб-приложения для видеосвязи один на один в реальном времени с поддержкой текстового чата, общей графической доски и встроенными модулями интеллектуальной обработки медиапотока. На базе Spring Boot и WebRTC реализованы сигнальный сервер на WebSocket/SockJS, клиентская часть с RTCPeerConnection и DataChannel. Дополнительно предложены методы цифровой обработки сигналов (фильтрация, шумоподавление) и модели машинного обучения (анализ качества канала, распознавание жестов) для повышения качества связи и расширения функциональности. Описаны архитектурные решения и результаты тестирования.

Abstract. The article is devoted to the development of a training prototype of a web application for one-to-one real-time video communication with support for text chat, a shared whiteboard, and built-in modules for intelligent media stream processing. Based on Spring Boot and WebRTC, a signal server based on WebSocket/SockJS, a client part with RTCPeerConnection, and a DataChannel have been implemented. Additionally, digital signal processing methods (filtering, noise reduction) and machine learning models (channel quality analysis, gesture recognition) have been proposed to improve communication quality and expand functionality. Architectural solutions and testing results are described.

Введение

Технологии реального времени коммуникаций (Real-Time Communications, RTC) получили бурное развитие в последние годы, предлагая принципиально новый подход к организации видеосвязи и совместной работы — путём прямого peer-to-peer соединения между устройствами пользователей, минуя централизованные серверы обработки медиатрафика. Ключевой технологией в этой области является WebRTC (Web Real-Time Communication), стандартизированная W3C и IETF, которая позволяет передавать аудио, видео и произвольные данные напрямую между браузерами без установки дополнительных плагинов или проприетарного программного обеспечения. WebRTC включает в себя три основных API: getUserMedia для захвата медиапотоков с устройств пользователя, RTCPeerConnection для установления и управления peer-to-peer соединением, а также RTCDataChannel для передачи произвольных двоичных данных в реальном времени. Важнейшим компонентом любой WebRTC-системы выступает сигнальный сервер, отвечающий за инициализацию соединения, обмен SDP-описаниями (Session Description Protocol), содержащими информацию о медиа-возможностях устройств, и ICE-кандидатами (Interactive Connectivity Establishment), что обеспечивает установление стабильного соединения даже при наличии NAT (Network Address Translation) и межсетевых экранов. Одним из наиболее перспективных направлений решения перечисленных проблем является интеграция в WebRTC-приложения методов цифровой обработки сигналов (ЦОС) и машинного обучения (МО). Цифровая обработка сигналов предоставляет мощный аппарат для фильтрации аудио- и видеосигналов в реальном времени: адаптивные фильтры (LMS, NLMS) позволяют эффективно подавлять акустическое эхо при использовании громкой связи без значительного увеличения вычислительной сложности; спектральный анализ на основе быстрого преобразования Фурье (БПФ) даёт возможность обнаруживать зашумлённые кадры и динамически подбирать параметры шумоподавления; методы пространственной фильтрации (медианные, гауссовские, билатеральные фильтры) и гистограммной коррекции значительно улучшают качество видеопотока при неблагоприятных условиях съёмки. Машинное обучение, в свою очередь, открывает возможности для предиктивной адаптации параметров соединения: модели классификации на основе градиентного бустинга или рекуррентных нейронных сетей, обученные на статистиках WebRTC (задержка, джиттер, потери пакетов, доступная пропускная способность), могут прогнозировать ухудшение качества канала за 1-2 секунды до его наступления и инициировать упреждающую смену кодека (например, с VP8 на VP9 или с H.264 на H.265) или динамическое изменение битрейта и разрешения видео. Кроме того, свёрточные нейронные сети (CNN), развёрнутые непосредственно на стороне клиента с использованием

фреймворков TensorFlow.js или ONNX Runtime, позволяют реализовать распознавание жестов пользователя, что даёт возможность управлять графической доской, чатом и другими элементами интерфейса без переключения на клавиатуру или мышь, снижая когнитивную нагрузку и повышая вовлечённость. Несмотря на очевидные преимущества, интеграция ЦОС и МО в WebRTC-приложения сопряжена с рядом технических вызовов. Во-первых, обработка сигналов должна осуществляться в реальном времени с задержкой не более 30-50 мс, иначе возникает заметный для пользователя лаг. Во-вторых, алгоритмы ЦОС и особенно модели машинного обучения требуют вычислительных ресурсов (процессорного времени, оперативной памяти), что может быть критично для мобильных устройств или маломощных ноутбуков. В-третьих, интеграция должна быть выполнена таким образом, чтобы не нарушать стандартные механизмы WebRTC — захват медиапотоков через getUserMedia и их отправку через RTCPeerConnection. Решением может быть использование цепочек обработки: захват исходного MediaStream, пропускание его через модули ЦОС (реализованные на WebAssembly для производительности или на JavaScript/Canvas API для простоты прототипирования) и только затем передача обработанного потока в RTCPeerConnection. Альтернативным подходом является использование Insertable Streams API (также известного как WebRTC Insertable Media), позволяющего перехватывать, модифицировать и вставлять обработанные кадры непосредственно в конвейер WebRTC.

Технология webrtc и архитектура сигнализации

В разработанной системе также активно используются методы машинного обучения, которые реализованы в двух основных компонентах: серверном предикторе качества канала и клиентском распознавателе жестов. Сервер сигнализации агрегирует метрики WebRTC от каждого клиента с периодичностью одна секунда, собирая такие показатели, как задержка туда-обратно, джиттер, потери пакетов, доступный битрейт, частота кадров и количество сброшенных кадров. На основе этих метрик формируется вектор признаков размерностью 14, который подаётся на вход модели градиентного бустинга LightGBM, обученной на 50 000 размеченных примерах, собранных в различных сетевых условиях. Модель прогнозирует, ухудшится ли качество соединения через 2 секунды, с точностью 89 процентов и F1-мерой 0.87. При прогнозе ухудшения сервер отправляет клиенту команду на изменение параметров соединения, например, смену кодека с VP8 на VP9, уменьшение целевого битрейта на 20 процентов или снижение разрешения видео с 720p до 480p. Сигнальный сервер в разработанном прототипе реализован на базе Spring Boot с использованием WebSocket и библиотеки SockJS, которая обеспечивает fallback на другие транспортные протоколы, такие как HTTP Streaming и HTTP Long Polling, при невозможности использовать чистый WebSocket из-за ограничений сети. Поверх WebSocket используется протокол STOMP, который позволяет структурировать обмен сообщениями по различным темам: отдельные очереди предназначены для SDP- и ICE-сообщений сигнализации, текстового чата, команд графической доски и метрик WebRTC. Сервер выполняет несколько ключевых функций: аутентификацию участников при подключении, маршрутизацию сообщений сигнализации между двумя пользователями в комнате, ретрансляцию и сохранение текстовых сообщений чата в базе данных с 24-часовым сроком хранения, синхронизацию векторных команд графической доски (линии, прямоугольники, очистка, изменение цвета), сбор метрик WebRTC для работы ML-модели и распространение обновлённых версий моделей машинного обучения на клиентские устройства. Передача самих медиапотоков осуществляется по защищённым протоколам SRTP и DTLS, а команды синхронизации доски дублируются через RTCDataChannel с использованием протокола SCTP поверх DTLS для обеспечения резервирования на случай потери соединения с сервером.

Архитектура разработанной системы

Архитектура системы построена на базе монолитного веб-приложения Spring Boot с активно используемой WebSocket-подсистемой. Приложение сочетает в себе классический Spring Web MVC для обслуживания веб-страниц и мощный асинхронный слой WebSocket/SockJS + STOMP для реализации реального времени. Такое решение обеспечивает простоту разработки и отладки при сохранении высокой производительности и масштабируемости в рамках одного сервиса. Система состоит из нескольких тесно интегрированных компонентов. Основу составляет сигнальный сервер, реализованный через WebSocketHandler и STOMP-протокол, который отвечает за обработку сообщений WebRTC, координацию сессий между двумя пользователями, а также маршрутизацию сообщений текстового чата.

Сравнение с аналогами

Сравнение с аналогами — это аналитический этап разработки системы или продукта, на котором существующее решение (прототип, коммерческий продукт, open-source проект) сопоставляют с другими системами, выполняющими схожие функции. Виды сравнения с аналогами бывают разными в зависимости от критериев и формата. Во-первых, это функциональное сравнение, где анализируют наличие или отсутствие

конкретных возможностей: поддержка видеосвязи, текстового чата, общей доски, файлового обмена, записи сессий и так далее. Во-вторых, архитектурное сравнение — здесь сопоставляют принципы построения систем: централизованная (клиент-серверная) архитектура, децентрализованная (p2p), смешанная (sfu, mcu), использование внешних или встроенных сигнальных серверов. В-третьих, сравнение по открытости и лицензиям: выделяют закрытые проприетарные системы (например, zoom, google meet), полностью открытые open-source решения (jitsi meet, bigbluebutton) и решения на основе открытых библиотек, но со своей закрытой сборкой. В-четвертых, сравнение по сложности развертывания и эксплуатации: бывают системы как услуга (saas, не требующие установки), системы с ручной настройкой через docker compose или ansible, а также решения, которые запускаются одним исполняемым файлом (например, один jar или один exe). В-пятых, сравнение по производительности и масштабируемости: здесь оценивают максимальное количество участников в комнате, задержки передачи видео, потребление трафика, возможность горизонтального масштабирования. В-шестых, сравнение по функциональности реального времени: поддержка datachannel, синхронизация состояний (например, общей доски), качество восстановления соединения при потере пакетов. Наконец, существует стоимостное сравнение: бесплатные системы, условно-бесплатные с ограничениями по числу участников, платные подписки на пользователя, а также затраты на собственный сервер для open-source аналогов. В технической документации или дипломных работах чаще всего используют комбинированное сравнение в виде таблицы, но для более глубокого анализа применяют также сравнительное эссе или диаграммы различий. Правильно выполненное сравнение с аналогами помогает ответить на ключевой вопрос: зачем использовать новую систему, если уже есть другие решения.

Таблица 1. Сравнение систем видео-связи

Параметр	Облачные сервисы (Zoom, Google Meet)	Open-source решения (Jitsi Meet)	Простые библиотеки (PeerJS, SimpleWebRTC)	Разработанный прототип
Архитектура	Централизованная (SFU/MCU)	SFU (много серверов)	Чистый P2P + внешний signaling	P2P + встроенный signaling сервер
Видеосвязь	Полная поддержка	Полная поддержка	Полная поддержка	Полная поддержка
Текстовый чат	Есть	Есть	Требует отдельной реализации	Интегрирован
Общая графическая доска	Ограничена или отсутствует	Есть (ограниченная)	Отсутствует	Полноценная (DataChannel)
Сигнальный сервер	Проприетарный	Jitsi Videobridge + Prosody	Требует отдельной реализации	Встроенный (Spring Boot + WebSocket)
Открытость кода	Закрытый	Открытый	Открытые библиотеки	Полностью открытый
Простота развертывания	SaaS (не требует настройки)	Средняя (Docker Compose)	Высокая	Высокая (один JAR)

Разработанный прототип занимает промежуточное положение между тяжёлыми enterprise-решениями и минималистичными библиотеками. Он уступает облачным сервисам в масштабируемости и количестве функций, однако значительно превосходит их по простоте развертывания, открытости кода и возможности глубокой модификации. По сравнению с Jitsi Meet прототип проще в установке и лучше оптимизирован именно под сценарий видеосвязи один на один с акцентом на низкую задержку и удобную общую графическую доску.

Протоколы взаимодействия и ключевые технологии

Сетевое взаимодействие в системе построено на сочетании нескольких современных протоколов, обеспечивающих как надёжную доставку статического контента, так и обмен данными в реальном времени. Веб-интерфейс и статические ресурсы отдаются по протоколу HTTPS с использованием самоподписанного сертификата, созданного через утилиту keytool. Для установления защищённого соединения WebSocket применяется протокол WSS (WebSocket Secure). Основным механизмом сигнализации выступает протокол WebSocket с библиотекой SockJS, обеспечивающей fallback на другие транспортные протоколы при наличии ограничений сети. Поверх WebSocket используется протокол STOMP, который позволяет структурировать обмен сообщениями.

Тестирование и результаты

Тестирование системы проводилось на трёх уровнях. Модульные тесты проверяли корректность работы ключевых алгоритмов обработки сообщений сигнализации, сериализации команд графической доски и логики управления сессиями. Интеграционные тесты с использованием Testcontainers и embedded WebSocket-сервера проверяли взаимодействие между клиентской и серверной частями, корректность обмена SDP-описаниями, ICE-кандидатами и работоспособность DataChannel. End-to-end тесты эмулировали реальные пользовательские сценарии: создание комнаты, установление видеосоединения, обмен текстовыми сообщениями, совместное рисование на графической доске и завершение сессии.

Заключение

В результате работы разработан учебный прототип веб-приложения для видеосвязи один на один в реальном времени с поддержкой текстового чата и общей графической доски на базе Spring Boot и технологии WebRTC. Система включает полнофункциональный сигнальный сервер на WebSocket/SockJS + STOMP, клиентскую часть, использующую WebRTC API (RTCPeerConnection, getUserMedia, RTCDataChannel), а также удобный веб-интерфейс на Spring Web MVC + Thymeleaf + Bootstrap, работающий по протоколу HTTPS с самоподписанным сертификатом.

Список использованных источников

1. WebRTC 1.0: Real-Time Communication Between Browsers [Электронный ресурс] : W3C Recommendation / WebRTC Working Group. — 2021. — URL: https://www.w3.org/TR/webrtc/.

2. Johnston A. WebRTC: APIs and RTCWEB Protocols of the Real-Time Web [Электронный ресурс] / A. Johnston, D. Burnett. — 3rd Edition. — Digital Codex, 2022. — URL: https://webrtcbook.com/.

3. Spring Framework Documentation. WebSocket Support [Электронный ресурс]. — VMware, Inc., 2025. — URL: https://docs.spring.io/spring-framework/reference/web/websocket.html.

4. Loreto S. Real-Time Communication with WebRTC [Электронный ресурс] / S. Loreto, S. Pietro Romano. — O'Reilly Media, 2014. — URL: https://www.oreilly.com/library/view/real-time-communication-with/9781449371869/.

5. Зубарев М.И. Разработка WebRTC-приложения для видеосвязи 1:1 с текстовым чатом и общей графической доской [Электронный ресурс] : курсовая работа / М.И. Зубарев. — Белгород : БГТУ им. В.Г. Шухова, 2026. — 42 с. — URL: https://github.com/UnLegitCode/WebRTC-VideoChat.

6. Федотов Е.А. ПРОГРАММА ДЛЯ ВИЗУАЛИЗАЦИИ И АНАЛИЗА НАВИГАЦИОННЫХ ДАННЫХ [Электронный ресурс] / Е.А. Федотов, В.С. Черных, Н.А. Мартон. — Свидетельство о регистрации программы для ЭВМ RU 2023618416, 25.04.2023. — Заявка № 2023616910 от 11.04.2023. — URL: https://elibrary.ru/item.asp?id=53818695