Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ПРИЛОЖЕНИЕ ДЛЯ P2P ВИДЕОЗВОНКОВ

БГУИР КП 1-40 02 01 209 ПЗ

Студент: гр. 250502 Дроздов А.И.

Руководитель: Богдан Е. В.

МИНСК 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последние годы в связи с пандемией и переходом всего мира на удаленную работу стал очень актуален вопрос удаленных звонков по сети. Многие компании работают полностью удаленно, даже не имея офиса. Это привело к росту популярности таких сервисов для видеоконференций, как Zoom, Google Meet, Microsoft Teams и другие. Для максимальной производительности, приложения для видеоконференций обычно пишут на языке C++.

C++ - это мощный [1] и универсальный [2] язык программирования, который широко используется при разработке программных приложений, системного программного обеспечения, драйверов устройств и встроенного микропрограммного обеспечения.

С тех пор этот язык эволюционировал и стал одним из самых популярных и широко используемых языков программирования в мире. Его популярность обусловлена его эффективностью, гибкостью и широким спектром применений, для которых он может быть использован. C++ известен своей высокой производительностью, поскольку позволяет выполнять низкоуровневые манипуляции с оборудованием и памятью, что делает его подходящим для разработки ресурсоемких приложений.

Для отправки видео и аудио по сети обычно используют протокол UDP либо RTP, так как он помогает отправлять постоянные потоки данных эффективно, безопасно и с минимальными потерями качества.

Также существует протокол SRTP – протокол RTP, который выполняется по защищенному соединению. А протокол RTCP добавляет возможность контролировать трафик и качество его получения. Он отправляет дополнительные метаданные о качестве и количестве полученных пакетов.

1 ПОСТАНОВКА ЗАДАЧИ

Исследовать принцип работы протокола RTP, UDP и реализацию приложений для видеоконференций. Реализовать протокол взаимодействия и графический интерфейс пользователя с возможностью настраивать параметры соединения.

Программа для видеозвонков должна содержать классы, отражающие основной функционал: запись видео, запись аудио, вывод аудио и видео.

Для обеспечения максимальной производительности и для того, чтобы избежать задержки при использовании графического интерфейса, все взаимодействие между клиентом и сервером должно выполняться в отдельных потоках. Обычно для каждой операции создается отдельный поток, так как нельзя, например, одновременно слушать входящие сообщения на порту и отправлять сообщения на другой сервер последовательно.

Необходимо добавить возможность отключать микрофон и проводить настройку параметров соединения. Обычно как минимум необходимо регулировать число кадров в секунду (как часто сохранять и отправлять изображения экрана), качество изображения (для экономии трафика изображения должны сжиматься в JPG), а также размер одного пакета данных.

Размер пакета данных должен быть кратным 2 (для выравнивания), а максимальный теоретический размер пакета ограничен 65535 байтами. На практике все зависит от значения MTU соединения – оно отличается у Ethernet и Wi-Fi сетей, а при использовании VPN или иных средств ещё меньше данных возможно поместить в один фрейм данных. Необходимо выбрать подходящие значения по умолчанию.

При отключении клиента необходимо в разумный промежуток времени это проверить и вернуться к главному экрану.

Качество передаваемых данных должно сохраняться при разных условиях соединения и даже на расстоянии. Необходимо поддерживать возможность подключиться сам к себе (ip 127.0.0.1)

2 ОБЗОР ЛИТЕРАТУРЫ

2.1 Анализ существующих аналогов

В настоящее время существует огромное множество приложений для видеоконференций. Все началось с приложения Skype, которое в дальнейшем было выкуплено Microsoft. Оно использовалось как для команд поддержки клиента в различных компаниях и командных звонках, так и для звонков между родственниками. На рисунке 2.1 можно наблюдать окно группового звонка по Skype. Оно включает в себя список участников, изображения с их камер, имена и панель управления.

Рисунок 2.1 – Приложение Skype

Также Skype совмещал чаты, группы и даже звонки по обычной телефонной линии.

С развитием технологий Skype стал устаревать. Сам Microsoft выпустили отдельный продукт для звонков для команд – Microsoft Teams. Он часто используется в основном в компаниях, например, в крупнейших юридических компаниях, таких как Orrick law firm. Microsoft щедро выделяет дополнительные ресурсы начинающим стартапам через программу Microsoft for Startups.

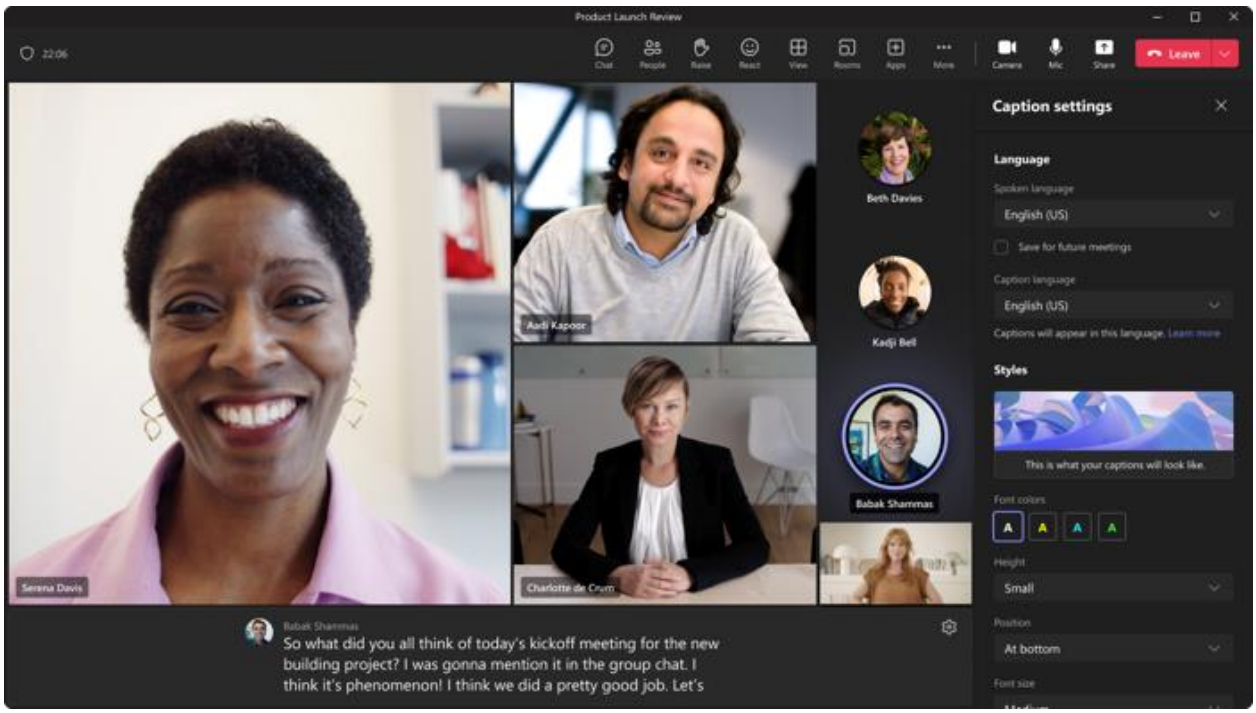

Рисунок 2.2 – Microsoft Teams

На рисунке 2.2 можно наблюдать интерфейс звонка в Microsoft Teams.

Уже можно заметить, что интерфейс звонка схож с интерфейсом Skype, но он больше подходит для команд (можно наблюдать командный чат, который доступен даже не во время звонка). Суть всегда примерно одинакова: есть нижняя или верхняя панель, где можно менять состояние микрофона, камеры и другой функционал: например, часто есть возможность создать доску для совместного рисования (удобно, например, для уроков в школах), оставлять реакции на выступления (для массовых звонков), поднимать руку (для секций Q&A) и не только.

Но Teams не только приложение для видеоконференций: это скорее целая система управления для команд. В чем-то это похоже на Slack, но под эгидой Microsoft.

Есть календарь будущих звонков, спринтов и не только. Диалоги часто состоят из threads – веток обсуждений, что помогает избежать сложности при навигации по диалогам. На рисунке 2.3 можно наблюдать чат в Microsoft Teams, что очень напоминает сервера в Slack.

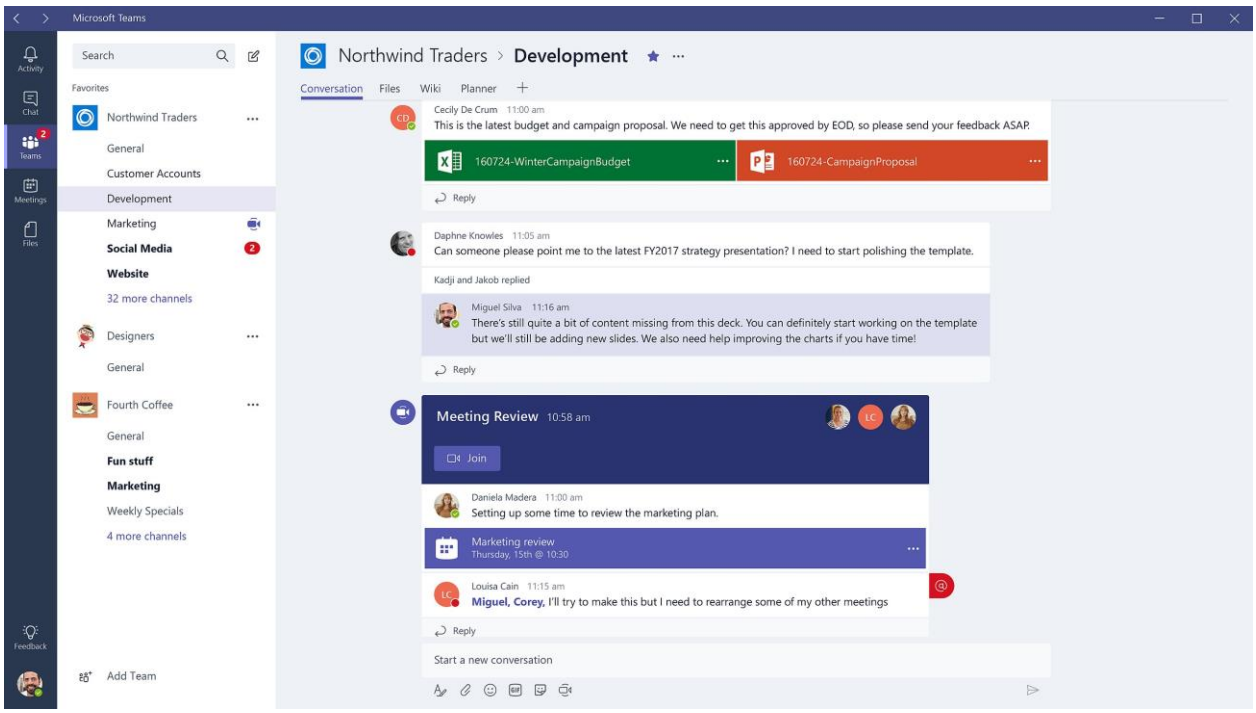

Рисунок 2.3 – чат для общения в Microsoft Teams

С началом пандемии и закрытием офисов многие компании перешли на удаленную работу. Причем даже сейчас многие компании остались на таком режиме работы, так как часто это более эффективно (нет затрат и времени на транспортацию). Также это помогает, например, нанимать людей из других стран, что часто бывает дешевле для компаний. Школы и университеты перешли на дистанционное обучение, и для обеспечения непрерывного процесса обучения технологии видеосвязи помогли соединить людей даже в трудные времена.

С пандемией сильно возросла популярность сервиса для видеоконференций Zoom, который являлся основным примером для данной курсовой работы.

Особенностью Zoom является то, что его интерфейс более понятен для обычных пользователей, а также он предоставляет бесплатные 40 минут на звонок. Когда время закончилось, всегда можно начать новый звонок.

Zoom изначально появился в Китае, когда для того, чтобы встречаться с коллегами, основателю приходилось ехать почти 10 часов на поезде каждый день. Так пришла идея сократить время, потраченное на транспорт и заменить его полезным мозговым штурмом, совершенным удаленно.

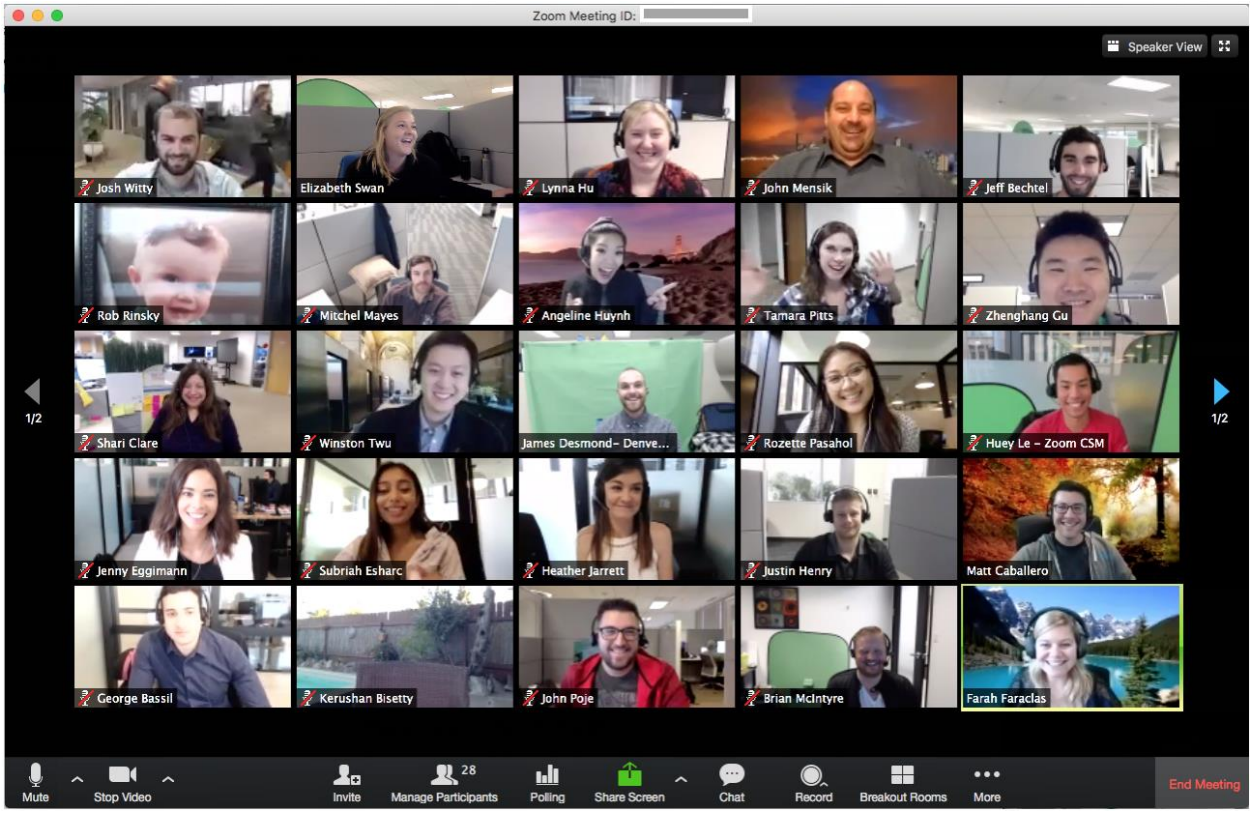

Рисунок 2.4 – Zoom

Интерфейс Zoom максимально прост (как показан на рисунке 2.4), и в то же время функционален. Он все ещё активно используется для дистанционных занятий, звонков команд и личных встреч. Он активно используется компаниями, такими как Yandex и Norton Rose Fulbright.

Ещё одной альтернативой является Google Meet, аналогичный продукт Zoom, но от Google. На рисунке 2.5 можно наблюдать интерфейс Google Meet. Он характеризуется минималистичностью, сделанный в классическом для Google стиле material design.

Обычно Google славится историей “мертвых” проектов, и существует целое кладбище закрытых Google проектов. Но не в данном случае. Инженеры Google и других лидирующих IT компаний активно используют Google Meet для звонков.

Ранее существовал Google Hangouts, который имел схожий функционал, но он также совмещал множество ненужного функционала и был закрыт Google.

Рисунок 2.5 – Google Meet

На самом деле даже популярные мессенджеры добавили функционал видеозвонков уже давно.

Рассмотрим их самые популярные реализации.

На самом деле реализация в мессенджерах наиболее похожа на то, как необходимо реализовать в нашем случае.

Обычно можно переключаться между звонками через централизованный сервер и P2P звонками.

Довольно часто звонки по P2P отключены в мессенджерах по умолчанию, так как это позволяет собеседнику узнать IP адрес другой стороны, а это не всегда является желаемым.

Например, в мессенджере Telegram по умолчанию любой пользователь может звонить любым другим. Из-за огромного количества спам ботов (также называемых юзерботами) рекомендуется запретить звонки от неконтактов.

Обычно на домашних компьютерах используется динамический ip адрес, это значит, что он меняется при перезагрузке роутера и при других условиях. Но при использовании статического ip адреса теоретически можно узнать локацию пользователя и запустить сканирование уязвимостей роутера.

Рисунок 2.6 – звонки в Telegram

В Telegram возможны как звонки между 2 участниками приложения, так и групповые звонки. Особенностью звонков в личных сообщениях является отображение эмодзи. Если они совпадают на обоих устройствах, значит соединение защищено. Пример звонков в личных сообщениях на рисунке 2.6 – можно наблюдать одновременно и свою камеру, и камеру собеседника на одном экране.

Особенностью групповых звонков является то, что есть возможность поднять руку. Это полезно на Q&A секциях, интервью с популярными личностями и не только. Пример группового звонка в Telegram можно наблюдать на рисунке 2.7. Всегда можно просмотреть список участников, а камеры обычно включают лишь панелисты – те, кто ведут сессию Q&A и подобных.

Несмотря на то, что Telegram основан на исходном коде VK, который является не самым оптимальным, после многих обновлений инфраструктура стала готова к таким нагрузкам.

Рисунок 2.7 – групповые звонки в Telegram

В настоящее время много команд в маленьких стартапах используют Telegram для всех своих задач, и рабочих, и нет.

Групповые видеозвонки есть и в Discord.

Их особенностью является то, что Discord часто используется в игровых комьюнити, и Discord имеет нативную интеграцию с многими играми (Rich Presence). Так что обычная трансляция экрана обычно совмещена с данными, например о текущей игре.

Доказательством того, что Discord является максимально популярным среди игровых комьюнити является то, что сервера по самым популярным играм, таким как Minecraft, являются самыми крупными серверами на всем Discord. Причем на серверах действуют ограничения, так как там находятся одновременно миллионы пользователей.

Также Discord для видеозвонков часто используют комьюнити рабочих групп по искусственному интеллекту, особенно в сфере Generative Pre-Trained Transformers с открытым исходным кодом.

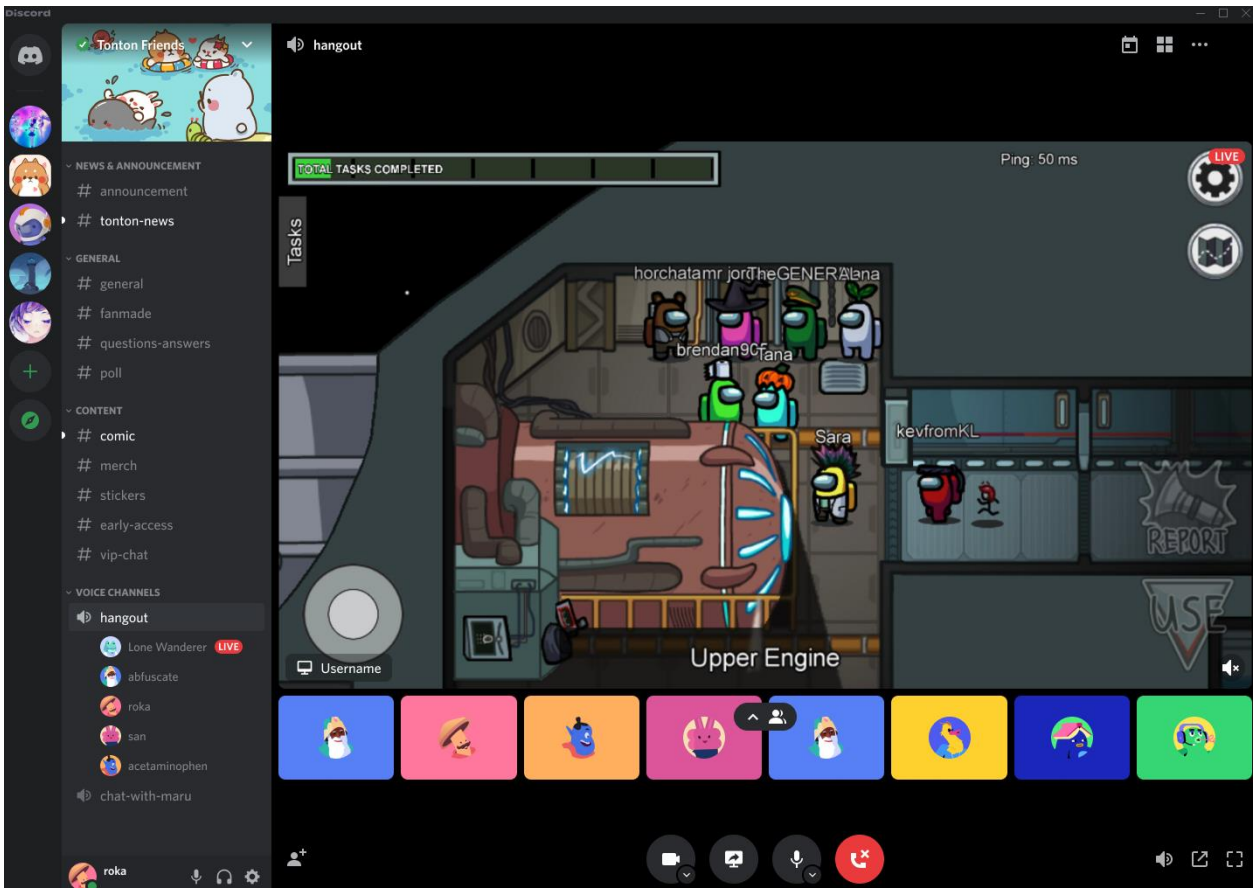

Рисунок 2.8 – групповые звонки в Discord

В Discord также возможно созваниваться либо вдвоем, либо группами. Особенностью групповых звонков является то, что можно либо создать группу из нескольких личных чатов, либо созваниваться на сервере. На серверах могут быть миллионы человек и Discord поддерживает огромную нагрузку без проблем. Пример звонка на сервере Discord можно наблюдать на рисунке 2.8. Можно параллельно писать сообщения как в каналах, привязанных к звонку, так и в любом другом канале сервера. Сервера аудио распределены по всему миру, и по умолчанию выбирается самый близкий к текущему пользователю, только если он не перегружен. Также используется технология Crisp с особыми кодеками для шумоподавления.

Количество альтернатив бесчисленное множество, но для реализации поставленной задачи был выбран лишь основной функционал, а именно: передача видео и аудио. Весь остальной функционал это лишь надстройки над базовыми примитивами. Например, групповые звонки просто требуют большего числа потоков.

Или, например, запись с веб-камеры — это просто очередной источник получения фреймов, а протокол отправки будет тот же.

2.2 Обзор методов и алгоритмов решения поставленной задачи

Приложение для P2P видеозвонков является оконным приложением на Qt [3] (так же, как и, например, Zoom), что позволяет ему быть кроссплатформенным. Для упрощения концепта, приложению не нужен централизованный сервер – каждый клиент является одновременно и клиентом, и сервером. Приложение работает в локальной сети (теоретически можно подключится и к удаленному серверу, но это нетривиально показать и могут быть потери пакетов).

Для передачи изображений был использован протокол RTP, а аудио передается по UDP. Сам по себе протокол RTP не гарантирует упорядоченности полученных данных и качества, но помогает в удобном формате отправлять изображения. Так как пакеты могут приходить в произвольном порядке, протокол был расширен – каждый пакет данных содержит текущий номер фрейма (1 фрейм = 1 скриншот экрана), и текущий номер sequence (так как каждый фрейм может быть потенциально большим, он разбивается на чанки и отправляется по частям, восстановить изображение можно лишь в правильном порядке). Для аудио был выбран протокол UDP вместо TCP для улучшения качества звука, т.к. звук — это непрерывный поток байт, а не что-то, имеющее четкую структуру. Для реконструкции изображения и получения скриншотов экрана (для получения новых фреймов) была использована библиотека OpenCV [4]. Интерфейсы были разработаны с помощью Qt Designer.

В современных приложениях обычно используется множество протоколов – сначала на самом нижнем уровне есть physical и data link layer – либо LAN, либо WAN, которое предоставляет доступ к глобальному интернету. Здесь и далее описывается модель OSI. Наблюдать стек, используемый в подобных приложениях можно на рисунке 2.9.

Далее идет network layer – IP (internet protocol) – используется для адресации и локации компьютеров в сети.

Самое главное происходит на transport layer – базово всегда используется протокол UDP, так как он быстрее и лучше для приложений с непрерывными потоками данных.

Но использовать сырой протокол UDP обычно нецелесообразно. Были созданы особые протоколы, направленные именно на отправку медиа.

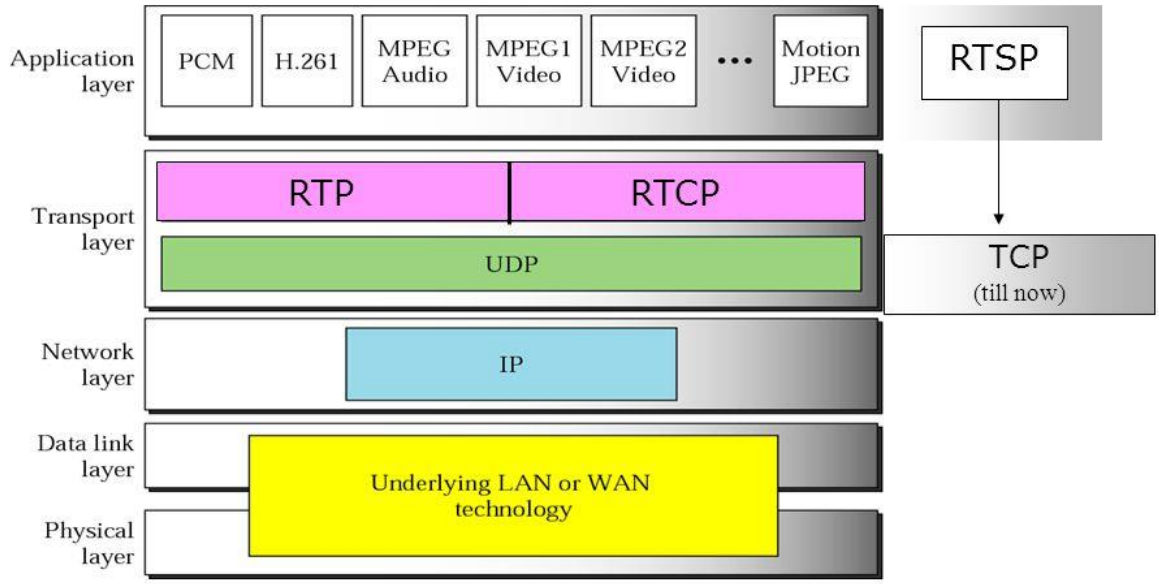

Рисунок 2.9 – стек, используемый в подобных приложениях

Протокол RTP задает формат данных для отправки медиа. Существует множество форматов данных, в данной работе использовался формат Generic для произвольных данных. Для лучшей производительности в реальных проектах используются протокол H264 и подобные. Они работают за счет небольшой нагрузки на процессор в момент закодирования фрейма, зато по сети они отправляются гораздо более эффективно.

Сами по себе фреймы не отправляются в нужном порядке. Для контроля качества используется RTCP – расширение протокола RTP. В данной работе используется свой протокол взаимодействия.

На application layer уже добавляется дополнительный функционал, например шумоподавление.

Для шумоподавления часто используются различные методы цифровой обработки сигналов и изображений. Например, часто применяют быстрое преобразование Фурье для приведения к ряду Фурье, а далее гораздо проще анализировать диапазоны частот.

Также можно реализовывать шумоподавление с помощью наложения специально сгенерированного звука. Но часто просто можно избежать рекурсивного воспроизведения собственного микрофона через кольцевой буфер.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описываются входные и выходные данные программы, диаграмма классов, а также приводится описание используемых классов и их методов.

3.1 Структура входных и выходных данных

Таблица 3.1 – файл с настройками соединения config.ini.

FPS	Размер одного чанка данных	Качество изображения
60	60416	90

Пример для таблицы 3.1:
[General]
fps=60
pack_size=60416
quality=90

3.2 Разработка диаграммы классов

Диаграмма классов для курсового проекта приведена в **Приложении А**.

3.3 Описание классов

3.3.1 Класс менеджера сессии

Класс `SessionManager` представляет текущий звонок и хранит все состояние для возможности завершения звонка и его повторного запуска.

Поля класса:

- `UDPPlayer *player` – проигрыватель аудио по UDP.
- `ScreenRecorder *recorder` – запись экрана текущего устройства и отправка другому клиенту.
- `MyThread *listen_thread` – получение изображений от клиента, реконструкция и отрисовка на окне (отдельный поток).
- `MainWindow *window` – окно звонка. После завершения удаляется и создается новым на каждый звонок.
- `StartWindow &start_window` – окно подключения к клиенту, которое отображается до или после звонка.
- `SettingsWindow *settings_window` – окно настроек, создается лишь на время его вызова через кнопку.
- `std::string ConnectServer` – сервер, к которому необходимо подключится.

Методы:

- `SessionManager(StartWindow &start_window)` – принимает объект начального окна, и настраивает обработку окна настроек.
- `connectButtonClicked()` – вызывается Qt при нажатии кнопки соединения, он обновляет поля ConnectServer
- `start()` – начинает звонок – загружает настройки из файла, запускает запись экрана и звука, запускает фоновые потоки получения и воспроизведения картинки и звука.
- `stop()` – ждет завершения потоков воспроизведение и очищает память (вызывается по кнопке End).

Класс `SessionManager` позволяет совершать неограниченное число звонков во время сессии нашей программы.

3.3.2 Класс записи экрана

Класс `ScreenRecorder` является потоком (`QThread`), который `FPS` раз в секунду делает снимки экрана и отправляет по сети (то есть передает видео с экрана)

Поля класса:

- `char* server` – сервер, на который отправлять видео.
- `int pack_size` – размер одного пакета.
- `int frame_interval` – промежуток, который необходимо ждать до отправки следующего фрейма.
- `int quality` – качество передаваемого изображения.

Методы:

- `ScreenRecorder(char* server, int pack_size, int frame_interval, int quality)` – конструктор инициализирует поля класса.
- `run()` – работает внутри другого потока и вызывается Qt. Он получает текущее окно, рисует курсор на скриншоте, преобразует `QPixMap` в `cv::Mat` и сжимает изображение для отправки, разбивает его на чанки и отправляет по сети. Метод работает до остановки (при окончании звонка) с промежутком `frame_interval`

Класс `ScreenRecorder` работает до тех пор, пока не запрошена остановка потока. С помощью сигнала Qt о завершении программы устанавливается специальный флаг, что позволяет потоку завершить работу без неожиданной остановки. Поток выполняет свою последнюю итерацию цикла и выходит из него и из функции `run()`. Тем временем сигнал после метода `requestInterruption()` вызывает метод `wait()`, что позволяет дождаться завершения потока.

3.3.3 Структуры данных фреймов

Структуры `FrameData` и `FrameChunk` помогают реконструировать изображения после получения по сети, вне зависимости от порядка присланных пакетов.

Структура `FrameData` имеет следующие поля:

- `int frame_num` – текущий номер фрейма (для отображение картинок в нужном порядке)
- `int buffer_size` – размер буфера, который нужно выделить под хранение одного изображения

Конструкторы `FrameData()` и `FrameData(int frame_num, int buffer_size)` инициализируют поля структуры значениями по умолчанию и заданными значениями соответственно.

Структура `FrameChunk` имеет следующие поля:

- `int seq` – номер чанка (чанки от 0 до N, объединенные по возрастанию seq дадут изображение)
- `int size` – размер чанка (все чанки равного размера кроме последнего)
- `uint8_t *data` – данные

Методы:

- `FrameChunk()` и `FrameChunk(int seq, int size, uint8_t *data)` – конструкторы инициализируют поля структуры значениями по умолчанию и заданными значениями соответственно.
- `FrameChunk(const FrameChunk &other)` – конструктор копирования позволяет избежать ошибочного освобождения памяти при передачи чанков в функцию
- `~FrameChunk()` – декструктор освобождает данные
- `operator<` – сравнивает два чанка по sequence (так как нам необходимо восстанавливать изображение по возрастанию)

3.3.4 Класс воспроизведения звука

Класс `UDPPlayer` получает аудио по UDP и воспроизводит в системный вывод звука

Он содержит следующие поля:

- `QAudioOutput *output` – системный вывод звука;
- `QUdpSocket *socket` – подключение по UDP для получения звука;
- `QIODevice *device` – абстракция Qt для соединения QAudioOutput и QUdpSocket.

Методы:

- `UDPPlayer()` – конструктор запускает сокет на порту, где мы ждем получения аудио, инициализируем аудиовыход и подключает функцию обработки данных.

- `playData()` – читает UDP датаграму и записывает на аудио устройство

3.3.5 Класс обработки изображений

Класс `MyThread` получает фреймы по чанкам и отображает фреймы на экран

Методы:

- `run()` – работает в другом потоке. Он получает данные и сохраняет их по чанкам. Чанк 0 содержит длину буфера, а дальше идет сам буфер. Он пытается восстановить весь фрейм размера `buffer_size`. Если получилось, вызывается сигнал `signalGUI`, который отправляет полученное изображение на экран. Метод работает до завершения звонка.
- `terminateThread()` – запрашивает остановку потока и ждет, пока поток сам себя завершит.

3.3.6 Классы главных оконных интерфейсов

Класс `MainWindow` содержит окно звонка

Поля класса:

- `QPixmap mainimg` – текущее изображение, полученное от клиента
- `QAudioInput *audio_input` – ввод с микрофона пользователя
- `QUdpSocket *audio_socket` – сокет для отправки аудио по UDP
- `bool mic_enabled` – показывает, включен ли микрофон

Методы:

- `MainWindow(QWidget *parent=nullptr)` – конструктор соединяет нажатие кнопки микрофона с методом toggleMic
- `init_audio_input(char *server)` – подключается к серверу и подсоединяет ввод с микрофона к отправке аудио по UDP.
- `start_audio()` – начинает запись с микрофона.
- `stop_audio()` – приостанавливает запись с микрофона. Используется для кнопки включения и выключения микрофона.
- `deinit_audio_input()` – останавливает запись и очищает память и исходящие соединения. Используется при завершении звонка
- `processImage(const QImage &img)` – устанавливает новое изображение, полученное по сети
- `toggleMic()` – меняет иконку кнопки и обновляет состояние записи с микрофона

Класс `StartWindow` содержит графический интерфейс для ввода ip адреса, подключения и кнопку открытия настроек

Класс `SettingsWindow` содержит окно для редактирования настроек в файле настроек config.ini

Методы:

- `SettingsWindow(QWidget *parent=nullptr)` – конструктор загружает настройки из файла и устанавливает начальные значения
- `saveSettings()` – сохраняет новые введенные значения в файл

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Разработка схем алгоритмов

void MyThread::run() – функция предназначена для получения и отображения потока изображений на экран.

Алгоритм по шагам:

1. Начало.
2. Запускаем слушатель пакетов на определенном порту.
3. Пока не запрошена остановка потока, переходим к шагу 4, иначе к шагу 14.
4. Получаем входящий фрейм (с таймаутом ожидания).
5. Извлекаем параметры из фрейма: текущий фрейм, номер чанка и данные.
6. Сохраняем новый чанк в std::map чанков для текущего фрейма.
7. Если есть чанк с номером 0, перейти к пункту 8. Иначе к пункту 9.
8. Выделяем буфер размера, извлеченного из нулевого чанка, и удаляем нулевой чанк.
9. Если буфер для текущего фрейма выделен, то переходим к пункту 10. Иначе к пункту 13
10. Проходим по массиву чанков для текущего фрейма и копируем во временный буфер.
11. Если размер временного буфера совпадает с ожидаемым, то переходим к пункту 12. Иначе к пункту 13.
12. Создаем изображение из буфера, меняем его под размер окна и отправляем на обработку главному окну.
13. Очищаем полученный фрейм и переходим к шагу 3
14. Конец.

void ScreenRecorder::run() – функция предназначена для записи экрана и отправки клиенту.

Алгоритм по шагам:

1. Начало.
2. Создаем соединение с клиентом для отправки пакетов.
3. Обнуляем счетчик фреймов.
4. Пока не запрошена остановка потока, переходим к шагу 5, иначе к шагу 12.
5. Получаем скриншот экрана.
6. Уменьшаем расширение скриншота и сжимаем изображение
7. Отправляем фрейм с данными о длине буфера (чанк 0)
8. Ожидаем frame_interval миллисекунд
9. Делим данные на чанки и отправляем их каждые frame_interval миллисекунд
10. Увеличиваем счетчик фреймов
11. Ждем frame_interval миллисекунд и переходим к шагу 4

12. Конец.

4.2 Разработка алгоритмов

Схема алгоритма метода MyThread::run() приведена в приложении Б. – эта функция является основной частью кода изменения изображения. Так как количество данных, что можно передать за раз ограничено протоколом UDP, мы сжимаем изображение и отправляем его по частям. Для обеспечения надежности доставки пакетов полученные чанки сортируются по своему номеру.

Схема алгоритма метода ScreenRecorder::run() приведена в приложении В. – эта функция использует платформо-зависимую функцию получения скриншота экрана, а далее использует OpenCV для его преобразования перед отправкой. Важно отправить начальный чанк с длиной буфера для избежания ошибок переполнения буфера.

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа разрабатывалась и запускалась на OS Linux. Также проводились тесты на MacOS. На OS Windows с небольшими изменениями программа скмпилировалась, но фаерволл или другие политики защиты Windows мешали принимать входящие данные

Разработка производилась в редакторе Visual Studio Code с расширением CMake Tools. Это позволяло удобно, быстро и эффективно собирать и разрабатывать сложный проект. На рисунке 5.1 показано, как по нажатию одной кнопки запуска автоматически генерируются зависимости проекты, собираются нужные файлы и линкуются в один бинарный файл.

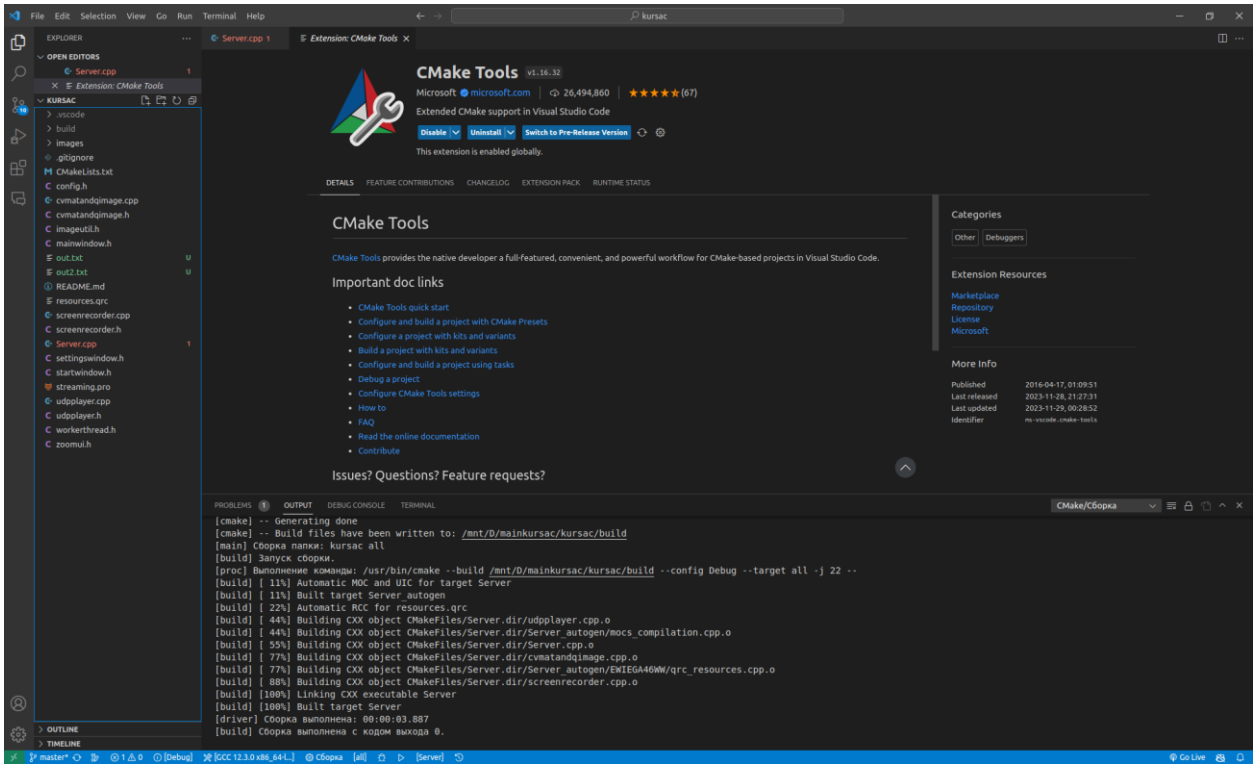

Рисунок 5.1 – Сборка в одну кнопку в VS Code

Проект можно собрать как с помощью cmake, так и qmake. Начиная с версии Qt 6 предпочтительней использовать CMake из-за его универсальности и нативной интеграции со всем функционалом Qt.

В данном проекте использовалась Qt 5 за счет большей доступности документации и самих версий Qt.

Сначала необходимо установить используемые библиотеки. Подразумевается, что Qt уже установлен с официального сайта. Здесь и далее инструкции для Ubuntu 22.04

```
sudo apt update
sudo apt install ccache libopencv-dev python3-opencv cmake
git libx11-dev libxfixes-dev libgl-dev libxext-dev
git clone https://github.com/ultravideo/uvgRTP
cd uvgRTP
mkdir build && cd build
cmake -DUVGRTP_DISABLE_CRYPTO=1 ..
make
sudo make install
```

Данные команды установят OpenCV, средство сборки CMake, а также некоторые дополнительные библиотеки, которые помогают взять скриншот экрана. Эта функция платформо-зависимая, так как Qt не предоставляет возможности получить скриншот экрана (это есть в Qt 6, но курсор мыши не виден)

Библиотека uvgrtp [5] используется для упрощения работы с протоколом RTP. Поверх этого строится свой протокол взаимодействия. На рисунке 5.2 можно наблюдать успешный процесс сборки библиотеки под OS Linux.

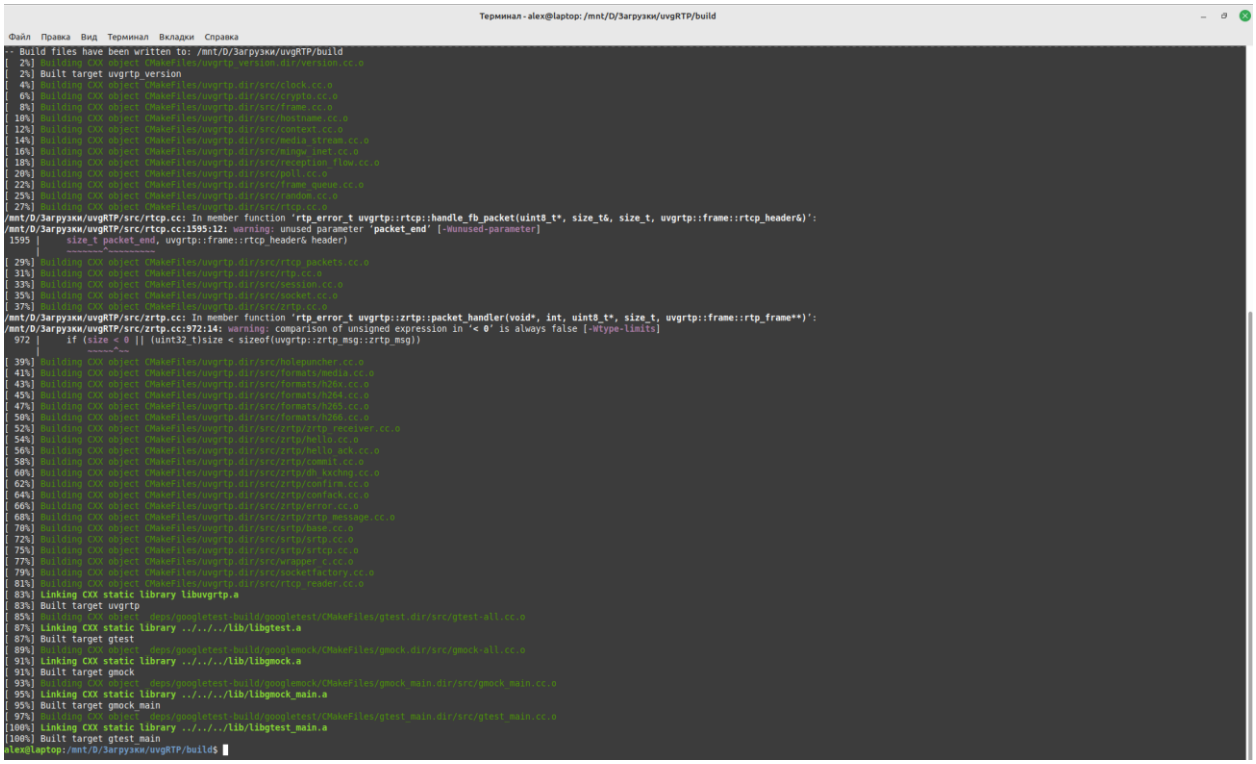

Рисунок 5.2 – сборка библиотеки uvgrtp через CMake

Далее сборка производиться через команды `qmake && make`, или `cmake . && make`

6 РЕЗУЛЬТАТЫ РАБОТЫ

При запуске программы нас приветствует окно подключения к серверу (рисунок 6.1). Можно ввести адрес 127.0.0.1 для подключения к самому себе (очень похоже на обычную запись экрана через программу OBS). При нажатии Connect происходит переход на окно звонка. Перед звонком можно настроить параметры соединения.

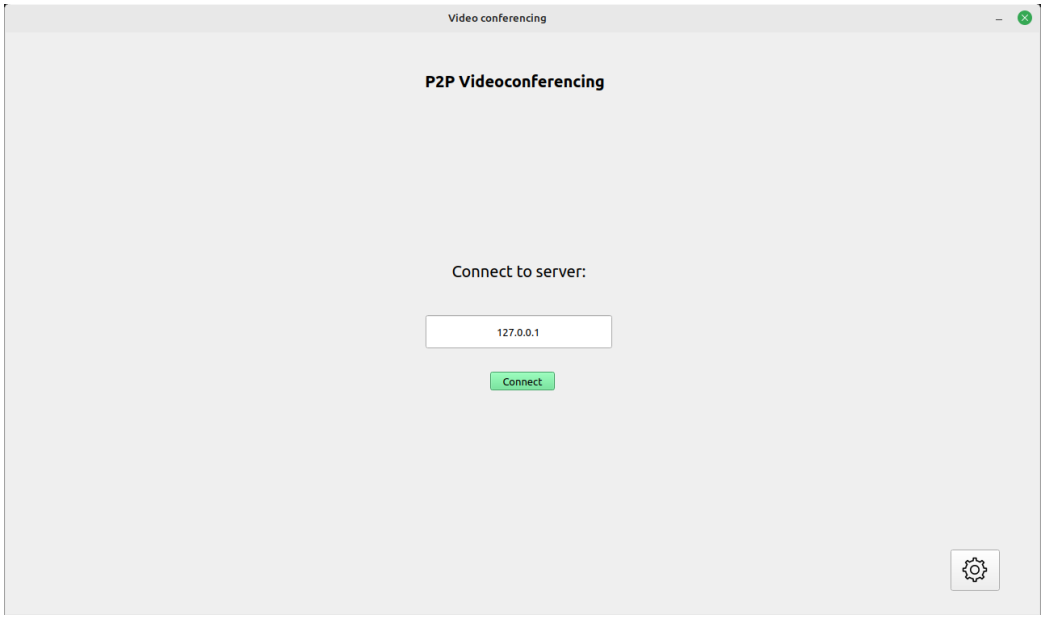

Рисунок 6.1 – Стартовое окно

Как показано на рисунке 6.2, окно настроек позволяет настроить основные параметры соединения: fps, размер пакета и качество изображения.

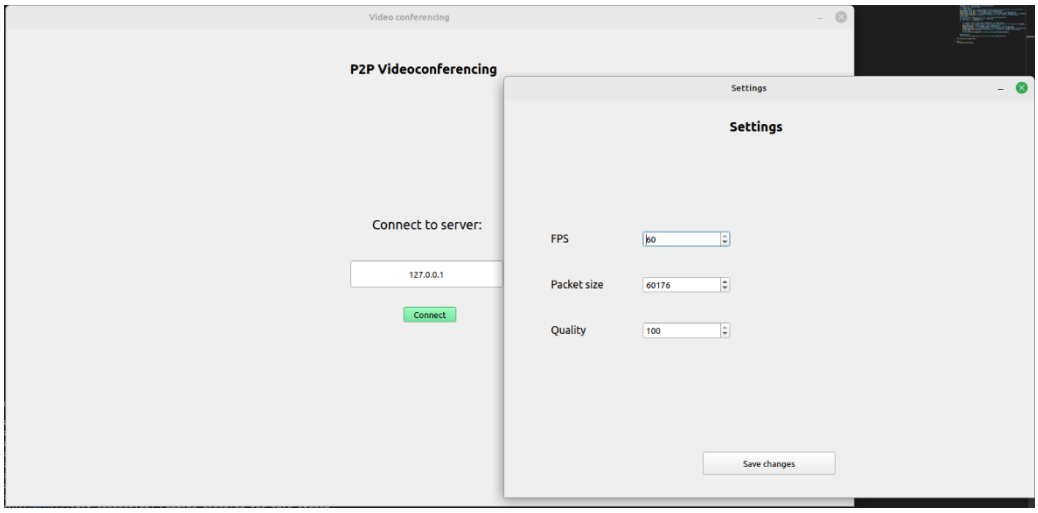

Рисунок 6.2 – Окно настроек

Настройки сохраняются в файл config.ini. Перед открытием диалога загружаются актуальные настройки. .ini файл был выбрал, так как это удобный читабельный формат и для пользователя, и для компьютера.

При подключении к серверу пользователь увидит окно, отображенное на рисунке 6.3.

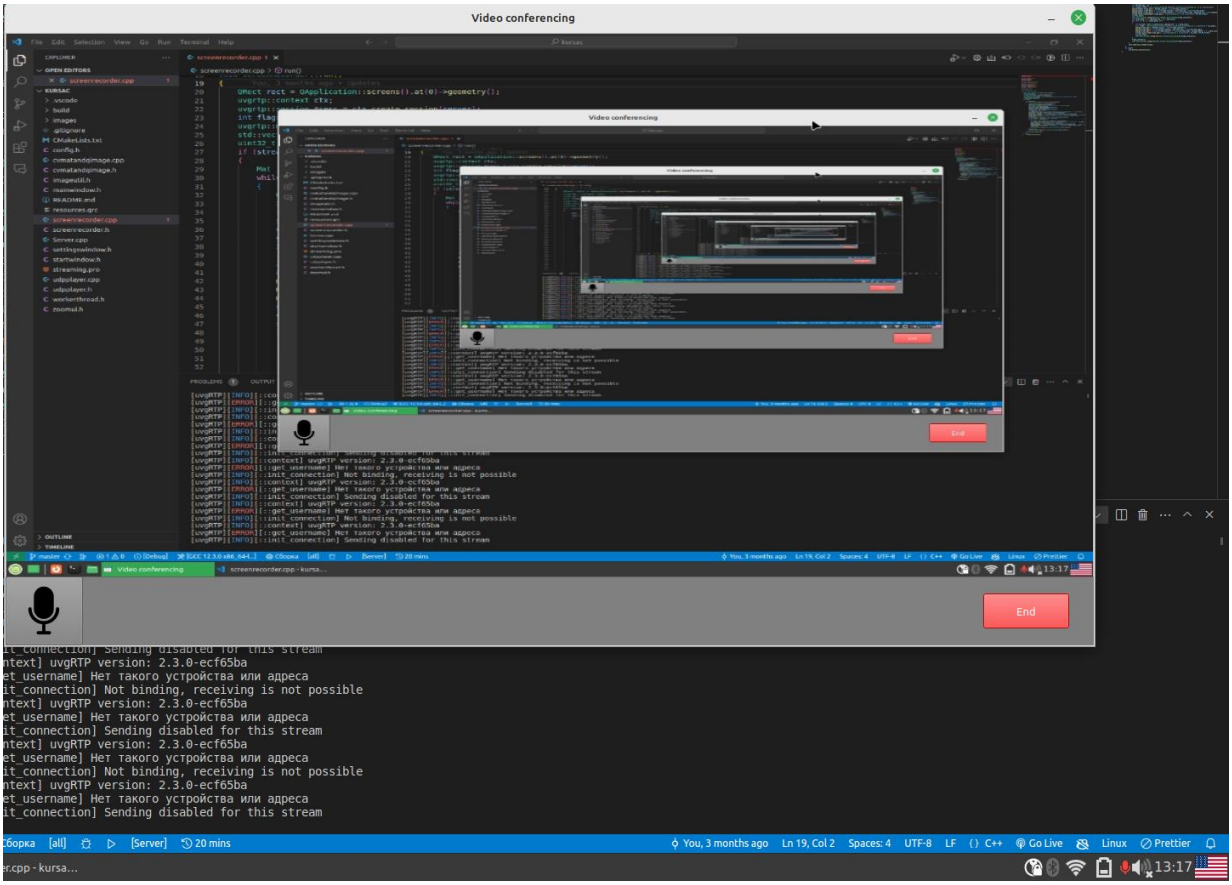

Рисунок 6.3 – Окно звонка (микрофон включен)

Интерфейс похож на интерфейс Zoom при звонке с 2 участниками: вместо сетки с видео участников, видно только видео противоположного участника. На панели управления можно завершить диалог и менять состояние микрофона. Как можно наблюдать в нижнем правом углу экрана, операционная система успешно отображает то, что происходит запись с микрофона. Так как в примере мы подключились сами к себе, можно наблюдать знаменитое рекурсивное отображение текущего экрана.

Аналогичную картинку можно было было бы наблюдать, если бы мы запустили запись в OBS. Это никак не влияет на работоспособность программы, но можно наблюдать за тем, как при движении мышки постепенно это отображается во всех окнах. Рекурсия бесконечная.

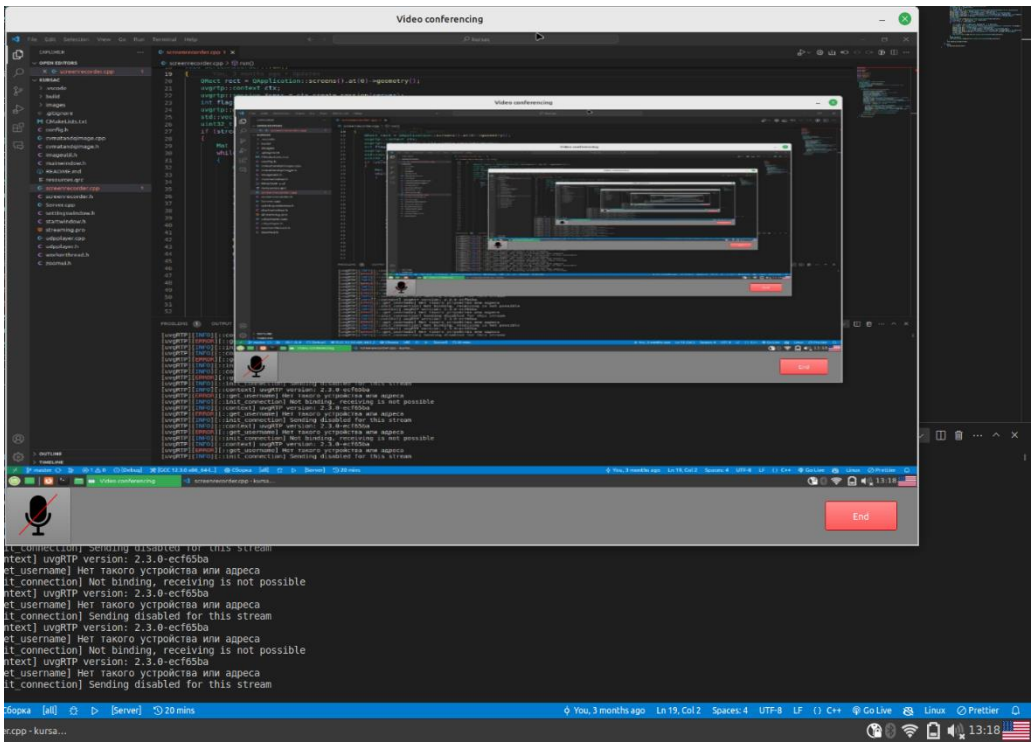

Рисунок 6.4 – Окно звонка (микрофон выключен)

Как можно наблюдать на рисунке 6.4, при выключении микрофона успешно обновляется иконка в приложении, и операционная система больше не отображает то, что какое-то приложение записывает микрофон.

Приложение было успешно протестировано на нескольких вариантах локальных сетей. Для тестирования из меню роутера узнавался ip адрес клиентов и вводился в поле ввода. Даже на расстоянии 15-20 метров между клиентами (при силе сигнала всего в 26% на одном из клиентов) с небольшими задержками можно было продолжать звонок. С экрана можно было успешно прочитать текст.

Приложение ограничено возможностями роутера по передачи пакетов. Из-за того, что абсолютно все сырые данные о фреймах должны быть переданы, происходит довольно сильная нагрузка на сеть. Это один из минусов P2P архитектуры. Но зато можно запустить звонок, не имея никаких серверов, а имея только 2 клиента, которые хотят созвонится.

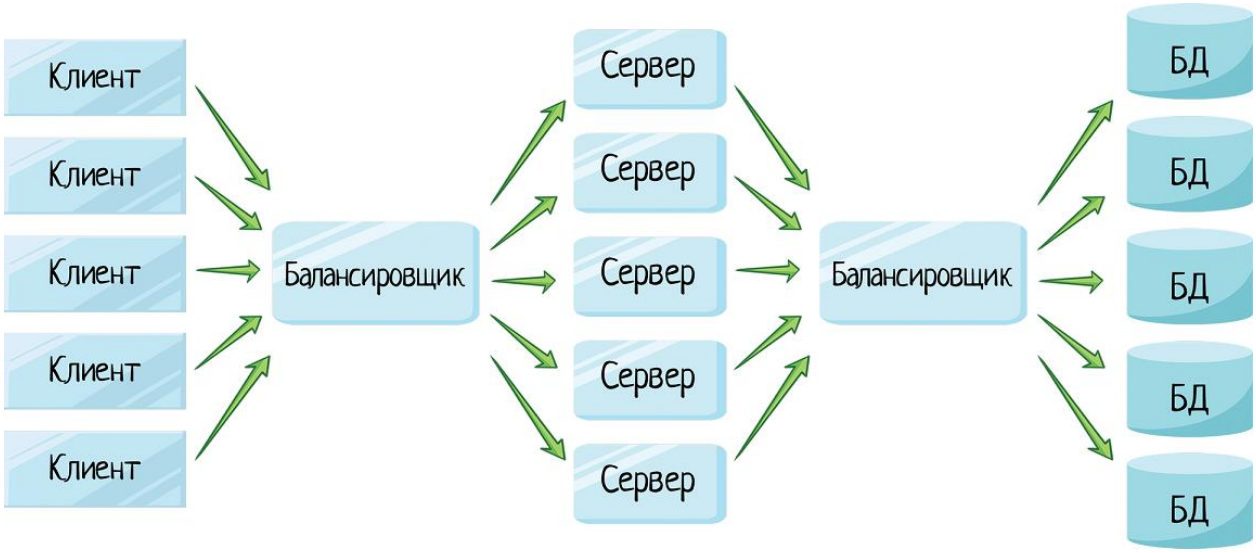

Рисунок 6.5 – Клиент-серверная архитектура

Теоретически можно было бы установить приложение на сервер как 1 клиент, а вторым клиентом было бы обычное потребительское устройство. Но на серверах нет микрофона и графического сервера в принципе. Его конечно можно установить, но обычно это не рекомендуется. Тем более что запись с микрофона в принципе невозможна, так как сервера в облаке обычно виртуальные и запущены вместе с тысячами себе подобных. Если бы это было бы возможно, то это являлось бы дырой безопасности в средствах виртуализации, которые используются на серверах (обычно QEMU или KVM)

Обычно в приложениях для видеозвонков, которые выдерживают большую нагрузку используется архитектура такая же, как указана на рисунке 6.5.

Множество клиентов подключается к балансировщику, который обычно 1 (обычно на 1 порте может работать лишь 1 сервис, а клиентам нужен фиксированный адрес сервера для подключения). После этого в зависимости от региона пользователя, загруженности серверов и других факторов запрос направляется на нужный сервер. Каждый сервер является полной копией всех остальных по реализации.

Причем часто доступ к базе данных тоже помещен за балансировщик, и база данных имеет много реплик.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было создано приложение для видеозвонков. Для этого использовалось Qt для графического интерфейса, OpenCV для обработки изображений и протоколы RTP и UDP.

В результате приложение предоставляет возможность созваниваться между 2 машинами в локальной сети. Качество изображения и звука сохранялось на удовлетворительном уровне даже через несколько стен и других преград для сигнала. В будущем возможно развить приложение до возможности созвона между несколькими компьютерами в локальной сети. Для этого необходимо создавать по 4 потока (воспроизведение аудио и изображения, запись аудио и изображения) для каждого нового клиента в звонке.

Приложения для видеозвонков являются ключевым компонентов в процессах работы современных команд. Это позволило создавать полностью удаленные компании без офисов и расширило возможности для глобальной кооперации.

Данная работа показала, что написать собственное приложение для видеоконференций не так сложно, как может показаться на первый взгляд. Самыми сложными частями проекта было разбитие фреймов на чанки для обеспечения их доставки в нужном порядке (иначе данные перетирались и буфер переполнялся), а также реализовать быструю реконструкцию изображения и отправку его в пользовательский интерфейс. Для этого пришлось преобразовывать типы Qt QPixmap в типы OpenCV cv::Mat и наоборот.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Объектно-ориентированное программирование на языке C++: учеб. пособие / Ю. А. Луцик, В. Н. Комличенко. – Минск : БГУИР, 2008.
2. The C++ Programming Language, – Bjarne Stroustrup, 1985
3. Qt 5 Documentation [Электронный ресурс]. -Электронные данные. -Режим доступа: https://doc.qt.io/qt5 - Дата доступа: 28.11.2023
4. OpenCV Documentation [Электронный ресурс]. -Электронные данные. -Режим доступа: https://docs.opencv.org/4.x - Дата доступа: 28.11.2023
5. uvgRTP Documentation [Электронный ресурс]. -Электронные данные. -Режим доступа: https://github.com/ultravideo/uvgRTP - Дата доступа: 28.11.2023

ПРИЛОЖЕНИЕ А
(обязательное)

Диаграмма классов

ПРИЛОЖЕНИЕ Б
(обязательное)

Схема метода MyThread::run()

ПРИЛОЖЕНИЕ В
(обязательное)

Схема метода ScreenRecorder::run()

ПРИЛОЖЕНИЕ Г
(обязательное)

Полный код программы

Файл config.h:

```
#ifndef CONFIG_H
#define CONFIG_H
#define FRAME_HEIGHT 720  // for transfer
#define FRAME_WIDTH 1080  // for transfer
#define FPS 60            // fps
#define PACK_SIZE 60176   // < max UDP packet size
#define ENCODE_QUALITY 90 // larger=more quality but large packet sizes
#define IMAGE_UDP_PORT 50000
#define AUDIO_UDP_PORT 55455
#define SETTINGS_FILE "config.ini"
#endif
```

Файл imageutil.h:

```
#include <QPainter>
#include <QCursor>
#ifdef Q_OS_LINUX
#include <X11/X.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/extensions/Xfixes.h>
#else // Q_OS_WINDOWs
#include <Windows.h>
#include <wingdi.h>
#pragma comment(lib, "User32.lib")
#pragma comment(lib, "Gdi32.lib")
#include <QtWinExtras>
#endif
namespace imageutil {
#ifdef Q_OS_LINUX
  /* WebCore/plugins/qt/QtX11ImageConversion.cpp */
  QImage qimageFromXImage(XImage *xi) {
    QImage::Format format = QImage::Format_ARGB32_Premultiplied;
    if (xi->depth == 24)
```

```
            format = QImage::Format_RGB32;
          else if (xi->depth == 16)
            format = QImage::Format_RGB16;
          QImage image = QImage(reinterpret_cast<uchar *>(xi->data), xi->width, xi->height, xi->bytes_per_line, format).copy();
          // we may have to swap the byte order
          if ((QSysInfo::ByteOrder == QSysInfo::LittleEndian && xi->byte_order == MSBFirst) || (QSysInfo::ByteOrder == QSysInfo::BigEndian && xi->byte_order == LSBFirst)) {
            for (int i = 0; i < image.height(); i++) {
              if (xi->depth == 16) {
                ushort *p = reinterpret_cast<ushort *>(image.scanLine(i));
                ushort *end = p + image.width();
                while (p < end) {
                  *p = ((*p << 8) & 0xff00) | ((*p >> 8) & 0x00ff);
                  p++; { {
              else {
                uint *p = reinterpret_cast<uint *>(image.scanLine(i));
                uint *end = p + image.width();
                while (p < end) {
                  *p = ((*p << 24) & 0xff000000) | ((*p << 8) & 0x00ff0000) | ((*p >> 8) & 0x0000ff00) | ((*p >> 24) & 0x000000ff);
                  p++; { { { {
          // fix-up alpha channel
          if (format == QImage::Format_RGB32) {
            QRgb *p = reinterpret_cast<QRgb *>(image.bits());
            for (int y = 0; y < xi->height; ++y) {
              for (int x = 0; x < xi->width; ++x)
                p[x] |= 0xff000000;
              p += xi->bytes_per_line / 4; { {
          return image; {
      #endif // Q_OS_LINUX
        QPixmap takeScreenShot(const QRect &area) {
          QRect screen;  /* interested display area */
          QImage qimage; /* result image */
      #ifdef Q_OS_LINUX
          QPoint cursorPos;
          Display *display = XOpenDisplay(nullptr);
```

```
            Window root = DefaultRootWindow(display);
            XWindowAttributes gwa;
            XGetWindowAttributes(display, root, &gwa);
            const auto goodArea = QRect(0, 0, gwa.width, gwa.height).contains(area);
            if (!goodArea) {
              screen = QRect(0, 0, gwa.width, gwa.height);
              cursorPos = QCursor::pos(); {
            else {
              screen = area;
              cursorPos = QCursor::pos() - screen.topLeft(); {
            XImage *image = XGetImage(display, root, screen.x(), screen.y(),
screen.width(), screen.height(), AllPlanes, ZPixmap);
            assert(nullptr != image);
            qimage = qimageFromXImage(image);
            /* draw mouse cursor into QImage
             *   https://msnkambule.wordpress.com/2010/04/09/capturing-a-screenshot-
showing-mouse-cursor-in-kde/
             *                                https://github.com/rprichard/x11-canvas-
screencast/blob/master/CursorX11.cpp#L31
             * */ {
              XFixesCursorImage *cursor = XFixesGetCursorImage(display);
              cursorPos -= QPoint(cursor->xhot, cursor->yhot);
              std::vector<uint32_t> pixels(cursor->width * cursor->height);
              for (size_t i = 0; i < pixels.size(); ++i)
                pixels[i] = cursor->pixels[i];
              QImage cursorImage((uchar *)(pixels.data()), cursor->width, cursor-
>height, QImage::Format_ARGB32_Premultiplied);
              QPainter painter(&qimage);
              painter.drawImage(cursorPos, cursorImage);
              XFree(cursor); {
            XDestroyImage(image);
            XDestroyWindow(display, root);
            XCloseDisplay(display);
          #elif defined(Q_OS_WINDOWS)
            HWND hwnd = GetDesktopWindow();
            HDC hdc = GetWindowDC(hwnd);
            HDC hdcMem = CreateCompatibleDC(hdc);
            RECT rect = {0, 0, GetDeviceCaps(hdc, HORZRES), GetDeviceCaps(hdc,
VERTRES)};
```

```
          const auto goodArea = QRect(rect.left, rect.top, rect.right,
rect.bottom).contains(area);
          if (!goodArea) {
            screen = QRect(rect.left, rect.top, rect.right, rect.bottom); {
          else {
            screen = area; {
          HBITMAP hbitmap(nullptr);
          hbitmap = CreateCompatibleBitmap(hdc, screen.width(), screen.height());
          SelectObject(hdcMem, hbitmap);
          BitBlt(hdcMem, 0, 0, screen.width(), screen.height(), hdc, screen.x(),
screen.y(), SRCCOPY);
          /* draw mouse cursor into DC
           * https://stackoverflow.com/a/48925443/5446734
           * */
          CURSORINFO cursor = {sizeof(cursor)};
          if (GetCursorInfo(&cursor) && cursor.flags == CURSOR_SHOWING) {
            RECT rect;
            GetWindowRect(hwnd, &rect);
            ICONINFO info = {sizeof(info)};
            GetIconInfo(cursor.hCursor, &info);
            const int x = (cursor.ptScreenPos.x - rect.left - rect.left -
info.xHotspot) - screen.left();
            const int y = (cursor.ptScreenPos.y - rect.left - rect.left -
info.yHotspot) - screen.top();
            BITMAP bmpCursor = {0};
            GetObject(info.hbmColor, sizeof(bmpCursor), &bmpCursor);
            DrawIconEx(hdcMem, x, y, cursor.hCursor, bmpCursor.bmWidth,
bmpCursor.bmHeight,
                       0, nullptr, DI_NORMAL); {
          qimage = QtWin::imageFromHBITMAP(hdc, hbitmap, screen.width(),
screen.height());
      #endif // Q_OS_LINUX
          return QPixmap::fromImage(qimage); {
      } // namespace imageutil


      Файл mainwindow.h:
      #ifndef MAINWINDOW_H
      #define MAINWINDOW_H
      #include <QMainWindow>
```

```
#include <QPixmap>
#include <QAudioInput>
#include <QUdpSocket>
#include "zoomui.h"
#include "udpplayer.h"
class MainWindow : public QMainWindow, public Ui::MainWindow {
  Q_OBJECT
private:
  QPixmap mainimg;
  QAudioInput *audio_input;
  QUdpSocket *audio_socket;
  bool mic_enabled;
public:
  MainWindow(QWidget *parent = nullptr)
      : QMainWindow(parent), mic_enabled(true) {
    setupUi(this);
    connect(micButton, SIGNAL(clicked()), this, SLOT(toggleMic())); {
  void init_audio_input(char *server) {
    QAudioFormat format = getAudioFormat();
    audio_input = new QAudioInput(format);
    audio_socket = new QUdpSocket();
    audio_socket->connectToHost(server, AUDIO_UDP_PORT);
    audio_socket->waitForConnected();
    start_audio(); {
  void start_audio() {
    audio_input->start(audio_socket); {
  void stop_audio() {
    audio_input->stop(); {
  void deinit_audio_input() {
    stop_audio();
    audio_socket->deleteLater();
    audio_input->deleteLater(); {
public slots:
  void processImage(const QImage &img) {
    imgpix = QPixmap::fromImage(img);
    pixmap->setPixmap(imgpix); {
  void toggleMic() {
```

```
        mic_enabled = !mic_enabled;
        QImage img(mic_enabled == true ? ":/mic-on.png" : ":/mic-off.png");
        if (mic_enabled)
          start_audio();
        else
          stop_audio();
        micButton->setIcon(QPixmap::fromImage(img)); {
      void beforeStopAll() {
        emit stopAll(); {
    signals:
      void stopAll();
    };
    #endif
```

Файл screenrecorder.cpp:

```
    #include <string>
    #include <vector>
    #include <QApplication>
    #include <QScreen>
    #include <QDebug>
    #include "opencv2/opencv.hpp"
    #include <uvgrtp/lib.hh>
    #include "screenrecorder.h"
    #include "imageutil.h"
    #include "cvmatandqimage.h"
    #include "config.h"
    using namespace cv;
    void ScreenRecorder::run() {
      QRect rect = QApplication::screens().at(0)->geometry();
      uvgrtp::context ctx;
      uvgrtp::session *sess = ctx.create_session(server);
      int flags = RCE_FRAGMENT_GENERIC | RCE_SEND_ONLY;
      uvgrtp::media_stream    *stream    =    sess->create_stream(IMAGE_UDP_PORT,
RTP_FORMAT_GENERIC, flags);
      std::vector<uint8_t> encoded;
      uint32_t frame_counter = 0;
      if (stream) {
```

```
          Mat image, send;
          while (!QThread::currentThread()->isInterruptionRequested()) {
            QPixmap pixmap = imageutil::takeScreenShot(rect);
            image = QtOcv::image2Mat(pixmap.toImage());
            resize(image, send, Size(FRAME_WIDTH, FRAME_HEIGHT), 0, 0,
INTER_LINEAR);
            std::vector<int> compression_params;
            compression_params.push_back(IMWRITE_JPEG_QUALITY);
            compression_params.push_back(quality);
            imencode(".jpg", send, encoded, compression_params);
            int payload_len = encoded.size();
            int current_seq = 0;
            auto header_frame = std::unique_ptr<uint8_t[]>(new
uint8_t[sizeof(uint32_t) + 2 * sizeof(int)]);
            memcpy(header_frame.get(), &frame_counter, sizeof(frame_counter));
            memcpy(header_frame.get() + sizeof(frame_counter), &current_seq,
sizeof(current_seq));
            memcpy(header_frame.get() + sizeof(frame_counter) +
sizeof(current_seq), &payload_len, sizeof(payload_len));
            stream->push_frame(header_frame.get(), sizeof(uint32_t) + 2 *
sizeof(int), RTP_NO_FLAGS);
            current_seq++;

std::this_thread::sleep_for(std::chrono::milliseconds(frame_interval));
            int total_pack = 1 + (payload_len - 1) / pack_size;
            for (int i = 0; i < total_pack; i++) {
              int to_send = min<int>(pack_size, payload_len - i * pack_size);
              auto frame = std::unique_ptr<uint8_t[]>(new uint8_t[sizeof(uint32_t)
+ sizeof(int) + to_send]);
              memcpy(frame.get(), &frame_counter, sizeof(frame_counter));
              memcpy(frame.get() + sizeof(frame_counter), &current_seq,
sizeof(current_seq));
              memcpy(frame.get() + sizeof(frame_counter) + sizeof(current_seq),
encoded.data() + i * pack_size, to_send);
              stream->push_frame(frame.get(), sizeof(uint32_t) + sizeof(int) +
to_send, RTP_NO_FLAGS);
              current_seq++;

std::this_thread::sleep_for(std::chrono::milliseconds(frame_interval)); {
            frame_counter++;

std::this_thread::sleep_for(std::chrono::milliseconds(frame_interval)); {
```

```
    sess->destroy_stream(stream); {
  if (sess)
    ctx.destroy_session(sess); {
```

Файл screenrecorder.h:

```
#ifndef SCREENRECORDER_H
#define SCREENRECORDER_H
#include <QThread>
#include <QMutex>
class ScreenRecorder : public QThread {
  Q_OBJECT
private:
  char *server;
  int pack_size;
  int frame_interval;
  int quality;
public:
  ScreenRecorder(char *server, int pack_size, int frame_interval, int
quality)
      : server(server), pack_size(pack_size),
frame_interval(frame_interval), quality(quality) {}
protected:
  virtual void run();
public slots:
  void terminateThread() {
    if (isRunning()) {
      requestInterruption();
      wait(); { {
};
#endif
```

Файл Server.cpp:

```
#include <iostream>
#include <cstdlib>
#include <map>
#include <set>
#include <QApplication>
```

```
#include <QMainWindow>
#include <QThread>
#include <QMutex>
#include <QDebug>
#include <QFile>
#include <QGraphicsPixmapItem>
#include <QMessageBox>
#include <QSettings>
#include <uvgrtp/lib.hh>
#include "opencv2/opencv.hpp"
using namespace cv;
#include "udpplayer.h"
#include "screenrecorder.h"
#include "config.h"
#include "zoomui.h"
#include "cvmatandqimage.h"
#include "workerthread.h"
#include "mainwindow.h"
#include "startwindow.h"
#include "settingswindow.h"
constexpr int RECEIVER_WAIT_TIME_MS = 10 * 1000;
struct FrameData {
  int frame_num;
  int buffer_size;
  FrameData() : frame_num(-1), buffer_size(0) {}
  FrameData(int frame_num, int buffer_size) : frame_num(frame_num),
buffer_size(buffer_size) {}
};
struct FrameChunk {
  int seq;
  int size;
  uint8_t *data;
  FrameChunk() : seq(-1), size(0), data(nullptr) {}
  FrameChunk(int seq, int size, uint8_t *data) : seq(seq), size(size),
data(data) {}
  bool operator<(const FrameChunk &other) const {
    return seq < other.seq; {
```

```
        FrameChunk(const FrameChunk &other) : seq(other.seq), size(other.size) {
          data = new uint8_t[size];
          memcpy(data, other.data, size); {
        ~FrameChunk() {
          delete[] data; {
      };
      void MyThread::run() {
        uvgrtp::context ctx;
        uvgrtp::session *sess = ctx.create_session("0.0.0.0");
        int flags = RCE_FRAGMENT_GENERIC | RCE_RECEIVE_ONLY;
        uvgrtp::media_stream *receiver = sess->create_stream(IMAGE_UDP_PORT,
RTP_FORMAT_GENERIC, flags);
        if (receiver) {
          std::map<uint32_t, FrameData> frames;
          std::map<uint32_t, std::set<FrameChunk>> chunks;
          while (!QThread::currentThread()->isInterruptionRequested()) {
            uvgrtp::frame::rtp_frame *frame = receiver-
>pull_frame(RECEIVER_WAIT_TIME_MS);
            if (!frame)
              break;
            uint32_t current_frame;
            int current_seq;
            memcpy(&current_frame, frame->payload, sizeof(uint32_t));
            memcpy(&current_seq, frame->payload + sizeof(uint32_t), sizeof(int));
            size_t real_len = frame->payload_len - sizeof(uint32_t) - sizeof(int);
            uint8_t *data = new uint8_t[real_len];
            memcpy(data, frame->payload + sizeof(uint32_t) + sizeof(int),
real_len);
            chunks[current_frame].insert(FrameChunk(current_seq, real_len, data));
            if (chunks[current_frame].begin()->seq == 0) // received header {
              int buffer_size;
              memcpy(&buffer_size, chunks[current_frame].begin()->data,
sizeof(int));
              frames[current_frame] = FrameData(current_frame, buffer_size);
              chunks[current_frame].erase(chunks[current_frame].begin()); {
            if (frames.count(current_frame)) {
              int offset = 0;
              int buffer_size = frames[current_frame].buffer_size;
```

```
            uint8_t *buffer = new uint8_t[buffer_size];
            for (auto it = chunks[current_frame].begin(); it != chunks[current_frame].end(); it++) {
              memcpy(buffer + offset, it->data, it->size);
              offset += it->size; {
            if (offset == buffer_size) {
              Mat rawData = Mat(1, buffer_size, CV_8UC1, buffer);
              Mat cvimg = imdecode(rawData, IMREAD_COLOR);
              if (cvimg.size().width == 0) {
                std::cerr << "decode failure!" << std::endl;
                continue; {
              resize(cvimg, cvimg, Size(1278, 638), 0, 0, INTER_LINEAR);
              QImage image = QtOcv::mat2Image(cvimg);
              emit signalGUI(image);
              frames.erase(current_frame);
              chunks.erase(current_frame); {
            delete[] buffer; {
          (void)uvgrtp::frame::dealloc_frame(frame); {
        sess->destroy_stream(receiver); {
      if (sess)
        ctx.destroy_session(sess); {
    class SessionManager {
    private:
      UDPPlayer *player;
      ScreenRecorder *recorder;
      MyThread *listen_thread;
      MainWindow *window;
      StartWindow &startWindow;
      SettingsWindow *settingsWindow;
      std::string ConnectServer;
    public:
      SessionManager(StartWindow &startWindow) : startWindow(startWindow) {
        QObject::connect(startWindow.settingsButton, &QPushButton::clicked, [&]() {
            settingsWindow = new SettingsWindow();
            settingsWindow->setFixedSize(settingsWindow->width(), settingsWindow->height());
```

```
                    QObject::connect(settingsWindow->saveButton,
&QPushButton::clicked, [&]() {
                        settingsWindow->saveSettings();
                        settingsWindow->close();
                        settingsWindow->deleteLater(); });
                    settingsWindow->show(); }); {
        void start() {
          startWindow.hide();
          player = new UDPPlayer();
          QSettings settings(SETTINGS_FILE, QSettings::IniFormat);
          int pack_size = settings.value("pack_size", PACK_SIZE).toInt();
          int frame_interval = (1000 / settings.value("fps", FPS).toInt());
          int quality = settings.value("quality", ENCODE_QUALITY).toInt();
          recorder = new ScreenRecorder((char *)ConnectServer.c_str(), pack_size,
frame_interval, quality);
          recorder->start();
          window = new MainWindow();
          window->setFixedSize(window->width(), window->height());
          QObject::connect(window->endButton, &QPushButton::clicked, [&](bool) {
                  stop();
                  startWindow.show(); });
          window->init_audio_input((char *)ConnectServer.c_str());
          window->show();
          listen_thread = new MyThread();
          QObject::connect(listen_thread,   SIGNAL(signalGUI(const   QImage   &)),
window, SLOT(processImage(const QImage &)));
          QObject::connect(listen_thread,       &QThread::finished,       window,
&MainWindow::beforeStopAll);
          QObject::connect(window, &MainWindow::stopAll, [&]()
                          { stop();
                  startWindow.show(); });
          listen_thread->start();
          QObject::connect(QApplication::instance(),       SIGNAL(aboutToQuit()),
listen_thread, SLOT(terminateThread()));
          QObject::connect(QApplication::instance(),       SIGNAL(aboutToQuit()),
recorder, SLOT(terminateThread())); {
        void stop() {
          listen_thread->terminateThread();
          recorder->terminateThread();
```

```
            window->deinit_audio_input();
            listen_thread->deleteLater();
            recorder->deleteLater();
            player->deleteLater();
            window->deleteLater(); {
          void connectButtonClicked() {
            QString ip = startWindow.ipLabel->text();
            if (ip.isEmpty() || QHostAddress(ip).isNull()) {
              QMessageBox::warning(&startWindow,  "Error",  "Please  enter  an  IP
address");
              return; {
            ConnectServer = ip.toLocal8Bit().data();
            start(); {
        };
        int main(int argc, char **argv) {
          QApplication app(argc, argv);
          StartWindow startWindow;
          startWindow.setFixedSize(startWindow.width(), startWindow.height());
          SessionManager manager(startWindow);
          QObject::connect(startWindow.connectButton, &QPushButton::clicked, [&]()
                           { manager.connectButtonClicked(); });
          startWindow.show();
          return app.exec(); {
```

Файл settingswindow.h:

```
#ifndef UI_SETTINGSWINDOW_H
#define UI_SETTINGSWINDOW_H
#include <QtCore/QVariant>
#include <QtCore/QSettings>
#include <QtWidgets/QApplication>
#include <QtWidgets/QHBoxLayout>
#include <QtWidgets/QLabel>
#include <QtWidgets/QPushButton>
#include <QtWidgets/QSpacerItem>
#include <QtWidgets/QSpinBox>
#include <QtWidgets/QVBoxLayout>
#include <QtWidgets/QWidget>
```

```
#include <QtWidgets/QLineEdit>
#include "config.h"
class CustomSpinbox : public QSpinBox {
  Q_OBJECT
public:
  CustomSpinbox(QWidget *parent = nullptr)
      : QSpinBox(parent) {
    lineEdit()->setReadOnly(true); {
};
class Ui_SettingsWindow {
public:
  QLabel *label;
  QWidget *verticalLayoutWidget;
  QVBoxLayout *verticalLayout;
  QHBoxLayout *horizontalLayout;
  QLabel *label_2;
  QSpinBox *fpsVal;
  QHBoxLayout *horizontalLayout_2;
  QLabel *label_3;
  CustomSpinbox *packetVal;
  QHBoxLayout *horizontalLayout_3;
  QLabel *label_4;
  QSpinBox *qualityVal;
  QWidget *horizontalLayoutWidget_4;
  QHBoxLayout *horizontalLayout_4;
  QSpacerItem *horizontalSpacer;
  QPushButton *saveButton;
  QSpacerItem *horizontalSpacer_2;
  void setupUi(QWidget *SettingsWindow) {
    if (SettingsWindow->objectName().isEmpty())
      SettingsWindow->setObjectName(QString::fromUtf8("SettingsWindow"));
    SettingsWindow->resize(800, 600);
    label = new QLabel(SettingsWindow);
    label->setObjectName(QString::fromUtf8("label"));
    label->setGeometry(QRect(330, 20, 101, 41));
    QFont font;
    font.setPointSize(15);
```

```
            font.setBold(true);
            label->setFont(font);
            label->setAlignment(Qt::AlignCenter);
            verticalLayoutWidget = new QWidget(SettingsWindow);
            verticalLayoutWidget-
>setObjectName(QString::fromUtf8("verticalLayoutWidget"));
            verticalLayoutWidget->setGeometry(QRect(70, 160, 271, 241));
            verticalLayout = new QVBoxLayout(verticalLayoutWidget);
            verticalLayout->setObjectName(QString::fromUtf8("verticalLayout"));
            verticalLayout->setContentsMargins(0, 0, 0, 0);
            horizontalLayout = new QHBoxLayout();
            horizontalLayout->setObjectName(QString::fromUtf8("horizontalLayout"));
            label_2 = new QLabel(verticalLayoutWidget);
            label_2->setObjectName(QString::fromUtf8("label_2"));
            QFont font1;
            font1.setPointSize(12);
            label_2->setFont(font1);
            horizontalLayout->addWidget(label_2);
            fpsVal = new QSpinBox(verticalLayoutWidget);
            fpsVal->setObjectName(QString::fromUtf8("fpsVal"));
            fpsVal->setMinimum(1);
            fpsVal->setMaximum(60);
            horizontalLayout->addWidget(fpsVal);
            verticalLayout->addLayout(horizontalLayout);
            horizontalLayout_2 = new QHBoxLayout();
            horizontalLayout_2-
>setObjectName(QString::fromUtf8("horizontalLayout_2"));
            label_3 = new QLabel(verticalLayoutWidget);
            label_3->setObjectName(QString::fromUtf8("label_3"));
            label_3->setFont(font1);
            horizontalLayout_2->addWidget(label_3);
            packetVal = new CustomSpinbox(verticalLayoutWidget);
            packetVal->setObjectName(QString::fromUtf8("packetVal"));
            packetVal->setMinimum(1024);
            packetVal->setMaximum(60416);
            packetVal->setSingleStep(1024);
            horizontalLayout_2->addWidget(packetVal);
```

```
            verticalLayout->addLayout(horizontalLayout_2);

            horizontalLayout_3 = new QHBoxLayout();

            horizontalLayout_3-
>setObjectName(QString::fromUtf8("horizontalLayout_3"));

            label_4 = new QLabel(verticalLayoutWidget);

            label_4->setObjectName(QString::fromUtf8("label_4"));

            label_4->setFont(font1);

            horizontalLayout_3->addWidget(label_4);

            qualityVal = new QSpinBox(verticalLayoutWidget);

            qualityVal->setObjectName(QString::fromUtf8("qualityVal"));

            qualityVal->setMinimum(1);

            qualityVal->setMaximum(100);

            horizontalLayout_3->addWidget(qualityVal);

            verticalLayout->addLayout(horizontalLayout_3);

            horizontalLayoutWidget_4 = new QWidget(SettingsWindow);

            horizontalLayoutWidget_4-
>setObjectName(QString::fromUtf8("horizontalLayoutWidget_4"));

            horizontalLayoutWidget_4->setGeometry(QRect(-1, 510, 801, 80));

            horizontalLayout_4 = new QHBoxLayout(horizontalLayoutWidget_4);

            horizontalLayout_4-
>setObjectName(QString::fromUtf8("horizontalLayout_4"));

            horizontalLayout_4->setContentsMargins(0, 0, 0, 0);

            horizontalSpacer = new QSpacerItem(40, 20, QSizePolicy::Expanding,
QSizePolicy::Minimum);

            horizontalLayout_4->addItem(horizontalSpacer);

            saveButton = new QPushButton(horizontalLayoutWidget_4);

            saveButton->setObjectName(QString::fromUtf8("saveButton"));

            QSizePolicy sizePolicy(QSizePolicy::Minimum, QSizePolicy::Fixed);

            sizePolicy.setHorizontalStretch(0);

            sizePolicy.setVerticalStretch(0);

            sizePolicy.setHeightForWidth(saveButton-
>sizePolicy().hasHeightForWidth());

            saveButton->setSizePolicy(sizePolicy);

            saveButton->setMinimumSize(QSize(200, 35));

            horizontalLayout_4->addWidget(saveButton);

            horizontalSpacer_2 = new QSpacerItem(40, 20, QSizePolicy::Expanding,
QSizePolicy::Minimum);

            horizontalLayout_4->addItem(horizontalSpacer_2);

            retranslateUi(SettingsWindow);
```

```
          QMetaObject::connectSlotsByName(SettingsWindow);
        } // setupUi
        void retranslateUi(QWidget *SettingsWindow) {
          SettingsWindow->setWindowTitle(QCoreApplication::translate("SettingsWindow", "Settings", nullptr));
          label->setText(QCoreApplication::translate("SettingsWindow", "Settings", nullptr));
          label_2->setText(QCoreApplication::translate("SettingsWindow", "FPS", nullptr));
          label_3->setText(QCoreApplication::translate("SettingsWindow", "Packet size", nullptr));
          label_4->setText(QCoreApplication::translate("SettingsWindow", "Quality", nullptr));
          saveButton->setText(QCoreApplication::translate("SettingsWindow", "Save changes", nullptr));
        } // retranslateUi
      };
      class SettingsWindow : public QWidget, public Ui_SettingsWindow {
        Q_OBJECT
      public:
        SettingsWindow(QWidget *parent = nullptr)
            : QWidget(parent) {
          setupUi(this);
          QSettings settings(SETTINGS_FILE, QSettings::IniFormat);
          fpsVal->setValue(settings.value("fps", FPS).toInt());
          packetVal->setValue(settings.value("packet", PACK_SIZE).toInt());
          qualityVal->setValue(settings.value("quality", ENCODE_QUALITY).toInt());
{
        void saveSettings() {
          QSettings settings(SETTINGS_FILE, QSettings::IniFormat);
          settings.setValue("fps", fpsVal->value());
          settings.setValue("packet", packetVal->value());
          settings.setValue("quality", qualityVal->value()); {
      };
      #endif
```

Файл startwindow.h:

```
#ifndef STARTWINDOW_H
#define STARTWINDOW_H
```

```
#include <QWidget>
#include <QtCore/QVariant>
#include <QtGui/QIcon>
#include <QtWidgets/QApplication>
#include <QtWidgets/QLabel>
#include <QtWidgets/QLineEdit>
#include <QtWidgets/QPushButton>
#include <QtWidgets/QWidget>
class Ui_StartWindow {
public:
  QLabel *label;
  QLabel *label_2;
  QLineEdit *ipLabel;
  QPushButton *connectButton;
  QPushButton *settingsButton;
  void setupUi(QWidget *StartWindow) {
    if (StartWindow->objectName().isEmpty())
      StartWindow->setObjectName(QString::fromUtf8("StartWindow"));
    StartWindow->resize(1280, 720);
    label = new QLabel(StartWindow);
    label->setObjectName(QString::fromUtf8("label"));
    label->setGeometry(QRect(510, 10, 241, 101));
    QFont font;
    font.setPointSize(15);
    font.setBold(true);
    label->setFont(font);
    label->setTextFormat(Qt::AutoText);
    label->setAlignment(Qt::AlignCenter);
    label_2 = new QLabel(StartWindow);
    label_2->setObjectName(QString::fromUtf8("label_2"));
    label_2->setGeometry(QRect(510, 270, 251, 51));
    QFont font1;
    font1.setPointSize(15);
    label_2->setFont(font1);
    label_2->setAlignment(Qt::AlignCenter);
    ipLabel = new QLineEdit(StartWindow);
    ipLabel->setObjectName(QString::fromUtf8("ipLabel"));
```

```
          ipLabel->setGeometry(QRect(520, 350, 231, 41));
          ipLabel->setAlignment(Qt::AlignCenter);
          connectButton = new QPushButton(StartWindow);
          connectButton->setObjectName(QString::fromUtf8("connectButton"));
          connectButton->setGeometry(QRect(600, 420, 80, 23));
          connectButton->setStyleSheet(QString::fromUtf8("background-color:
rgb(87, 227, 137);"));
          settingsButton = new QPushButton(StartWindow);
          settingsButton->setObjectName(QString::fromUtf8("settingsButton"));
          settingsButton->setGeometry(QRect(1170, 640, 61, 51));
          QIcon icon;
          icon.addFile(QString::fromUtf8(":/config.png"), QSize(), QIcon::Normal,
QIcon::Off);
          settingsButton->setIcon(icon);
          settingsButton->setIconSize(QSize(32, 32));
          retranslateUi(StartWindow);
          QMetaObject::connectSlotsByName(StartWindow);
        } // setupUi
        void retranslateUi(QWidget *StartWindow) {
          StartWindow->setWindowTitle(QCoreApplication::translate("StartWindow",
"Video conferencing", nullptr));
          label->setText(QCoreApplication::translate("StartWindow",          "P2P
Videoconferencing", nullptr));
          label_2->setText(QCoreApplication::translate("StartWindow", "Connect to
server:", nullptr));
          ipLabel->setPlaceholderText(QCoreApplication::translate("StartWindow",
"IP address", nullptr));
          connectButton->setText(QCoreApplication::translate("StartWindow",
"Connect", nullptr));
          settingsButton->setText(QString());
        } // retranslateUi
      };
      class StartWindow : public QWidget, public Ui_StartWindow {
        Q_OBJECT
      public:
        StartWindow(QWidget *parent = nullptr)
            : QWidget(parent) {
          setupUi(this); {
      };
```

```
#endif
```

Файл udpplayer.cpp:

```
#include "udpplayer.h"
UDPPlayer::UDPPlayer(QObject *parent) : QObject(parent) {
  socket = new QUdpSocket();
  socket->bind(AUDIO_UDP_PORT);
  QAudioFormat format = getAudioFormat();
  QAudioDeviceInfo info(QAudioDeviceInfo::defaultOutputDevice());
  if (!info.isFormatSupported(format))
    format = info.nearestFormat(format);
  output = new QAudioOutput(format);
  device = output->start();
  connect(socket, &QUdpSocket::readyRead, this, &UDPPlayer::playData); {
void UDPPlayer::playData() {
  while (socket->hasPendingDatagrams()) {
    QByteArray data;
    data.resize(socket->pendingDatagramSize());
    socket->readDatagram(data.data(), data.size());
    device->write(data.data(), data.size()); { {
QAudioFormat getAudioFormat() {
  QAudioFormat format;
  format.setSampleRate(8000);
  format.setChannelCount(1);
  format.setSampleSize(16);
  format.setByteOrder(QAudioFormat::LittleEndian);
  format.setSampleType(QAudioFormat::SignedInt);
  format.setCodec("audio/pcm");
  QAudioDeviceInfo info(QAudioDeviceInfo::defaultInputDevice());
  if (!info.isFormatSupported(format))
    format = info.nearestFormat(format);
  return format; {
```

Файл udpplayer.h:

```
#ifndef UDPPLAYER_H
#define UDPPLAYER_H
#include <QObject>
```

```
#include <QtMultimedia/QAudioOutput>
#include <QtMultimedia/QAudioInput>
#include <QtMultimedia/QAudioFormat>
#include <QUdpSocket>
#include "config.h"
class UDPPlayer : public QObject {
  Q_OBJECT
public:
  explicit UDPPlayer(QObject *parent = 0);
  ~UDPPlayer() {
    socket->deleteLater();
    output->deleteLater(); {
private slots:
  void playData();
private:
  QAudioOutput *output;
  QUdpSocket *socket;
  QIODevice *device;
};
QAudioFormat getAudioFormat();
#endif
```

Файл workerthread.h:

```
#ifndef WORKERTHREAD_H
#define WORKERTHREAD_H
#include <QThread>
#include <QMutex>
#include <QImage>
class MyThread : public QThread {
  Q_OBJECT
protected:
  virtual void run();
signals:
  void signalGUI(QImage);
public slots:
  void terminateThread() {
    if (isRunning()) {
```

```
            requestInterruption();
            wait(); { {
      };
      #endif
```

Файл zoomui.h:

```
      /********************************************************************************
      ** Form generated from reading UI file 'zoomui.ui'
      **
      ** Created by: Qt User Interface Compiler version 5.15.2
      **
      ** WARNING! All changes made in this file will be lost when recompiling UI file!
      ********************************************************************************/
      #ifndef UI_ZOOMUI_H
      #define UI_ZOOMUI_H
      #include <QtCore/QVariant>
      #include <QtGui/QIcon>
      #include <QtWidgets/QApplication>
      #include <QtWidgets/QFrame>
      #include <QtWidgets/QGraphicsView>
      #include <QtWidgets/QMainWindow>
      #include <QtWidgets/QPushButton>
      #include <QtWidgets/QWidget>
      #include <QtWidgets/QGraphicsPixmapItem>
      QT_BEGIN_NAMESPACE
      class Ui_MainWindow {
      public:
        QWidget *centralwidget;
        QGraphicsScene *graphicsScene;
        QGraphicsView *graphicsView;
        QGraphicsPixmapItem *pixmap;
        QPixmap imgpix;
        QFrame *frame;
        QPushButton *micButton;
        QPushButton *endButton;
```

```
        void setupUi(QMainWindow *MainWindow) {
          if (MainWindow->objectName().isEmpty())
            MainWindow->setObjectName(QString::fromUtf8("MainWindow"));
          MainWindow->resize(1280, 720);
          centralwidget = new QWidget(MainWindow);
          centralwidget->setObjectName(QString::fromUtf8("centralwidget"));
          graphicsScene = new QGraphicsScene;
          pixmap = new QGraphicsPixmapItem;
          graphicsScene->addItem(pixmap);
          graphicsView = new QGraphicsView(centralwidget);
          graphicsView->setObjectName(QString::fromUtf8("graphicsView"));
          graphicsView->setGeometry(QRect(0, 0, 1280, 640));
          graphicsView->setScene(graphicsScene);
          frame = new QFrame(centralwidget);
          frame->setObjectName(QString::fromUtf8("frame"));
          frame->setGeometry(QRect(0, 639, 1281, 102));
          frame->setStyleSheet(QString::fromUtf8("background-color: gray"));
          frame->setFrameShape(QFrame::StyledPanel);
          frame->setFrameShadow(QFrame::Raised);
          micButton = new QPushButton(frame);
          micButton->setObjectName(QString::fromUtf8("pushButton"));
          micButton->setGeometry(QRect(0, 0, 91, 81));
          micButton->setAutoFillBackground(false);
          micButton->setIcon(QPixmap::fromImage(QImage(":/mic-on.png")));
          micButton->setIconSize(QSize(64, 64));
          endButton = new QPushButton(frame);
          endButton->setObjectName(QString::fromUtf8("endButton"));
          endButton->setGeometry(QRect(1150, 20, 101, 41));
          endButton->setStyleSheet(QString::fromUtf8("background-color: red;\n"
                                                     "color:     rgb(255,     255,
255);"));
          MainWindow->setCentralWidget(centralwidget);
          retranslateUi(MainWindow);
          QMetaObject::connectSlotsByName(MainWindow);
        } // setupUi
        void retranslateUi(QMainWindow *MainWindow) {
```

```
          MainWindow->setWindowTitle(QCoreApplication::translate("MainWindow",
"Video conferencing", nullptr));

          micButton->setText(QString());

          endButton->setText(QCoreApplication::translate("MainWindow",      "End",
nullptr));

        } // retranslateUi

      };

      namespace Ui {

        class MainWindow : public Ui_MainWindow {

        };

      } // namespace Ui

      QT_END_NAMESPACE

      #endif // UI_ZOOMUI_H
```

ПРИЛОЖЕНИЕ Д
(обязательное)

Ведомость документов