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 и другие. Для максимальной производительности, приложения для видеоконференций обычно пишут на языке C++.

C++ - это мощный и универсальный 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.

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.

2 ОБЗОР ЛИТЕРАТУРЫ

2.1 Обзор методов и алгоритмов решения поставленной задачи

Приложение для P2P видеозвонков является оконным приложением на Qt (так же, как и, например, Zoom), что позволяет ему быть кросс-платформенным. 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. Интерфейсы были разработаны с помощью Qt Designer.

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`

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:
4. Получаем входящий фрейм (с таймаутом ожидания).
5. Извлекаем параметры: текущий фрейм, номер чанка и данные из фрейма.
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, извлеченным из нулевого чанка и удаляем 0 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 под размер окна и отправляем на обработку главному окну.
13. Очищаем полученный фрейм и переходим к шагу 3
15. 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:
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.

РЕЗУЛЬТАТЫ РАБОТЫ

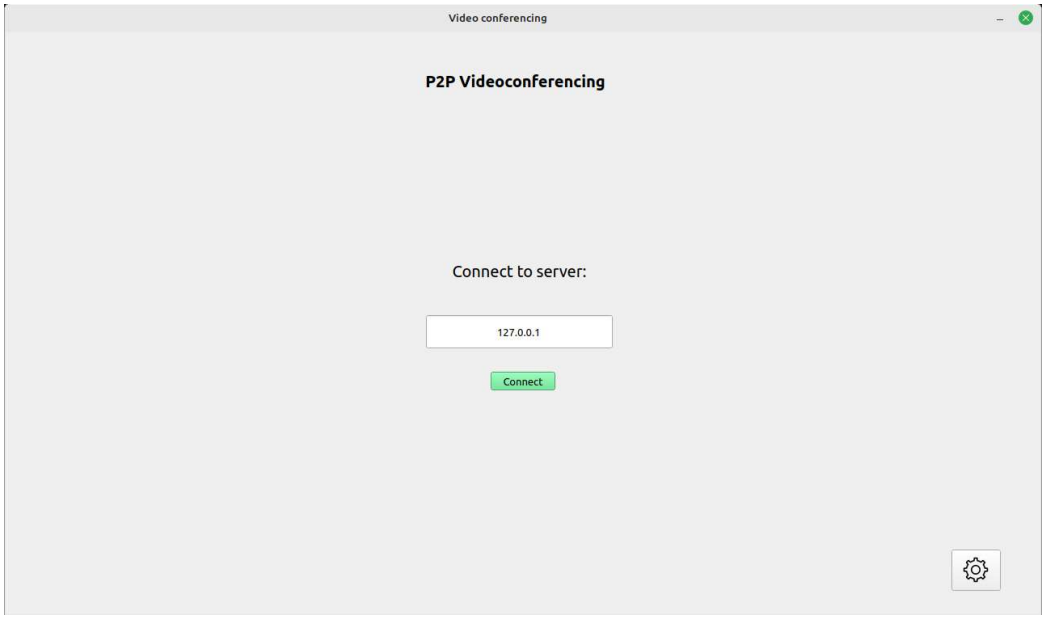

Рисунок 1 – Начальное окно

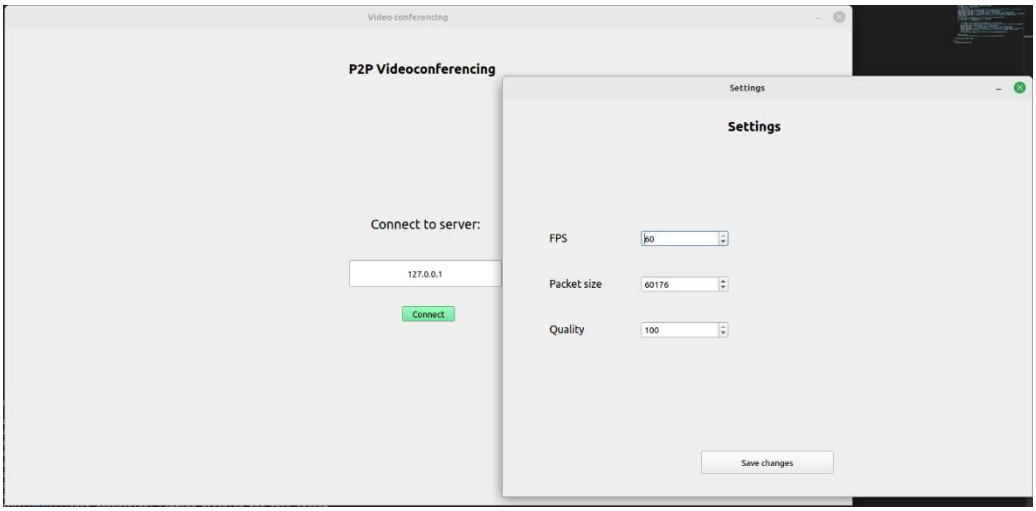

Рисунок 2 – Окно настроек

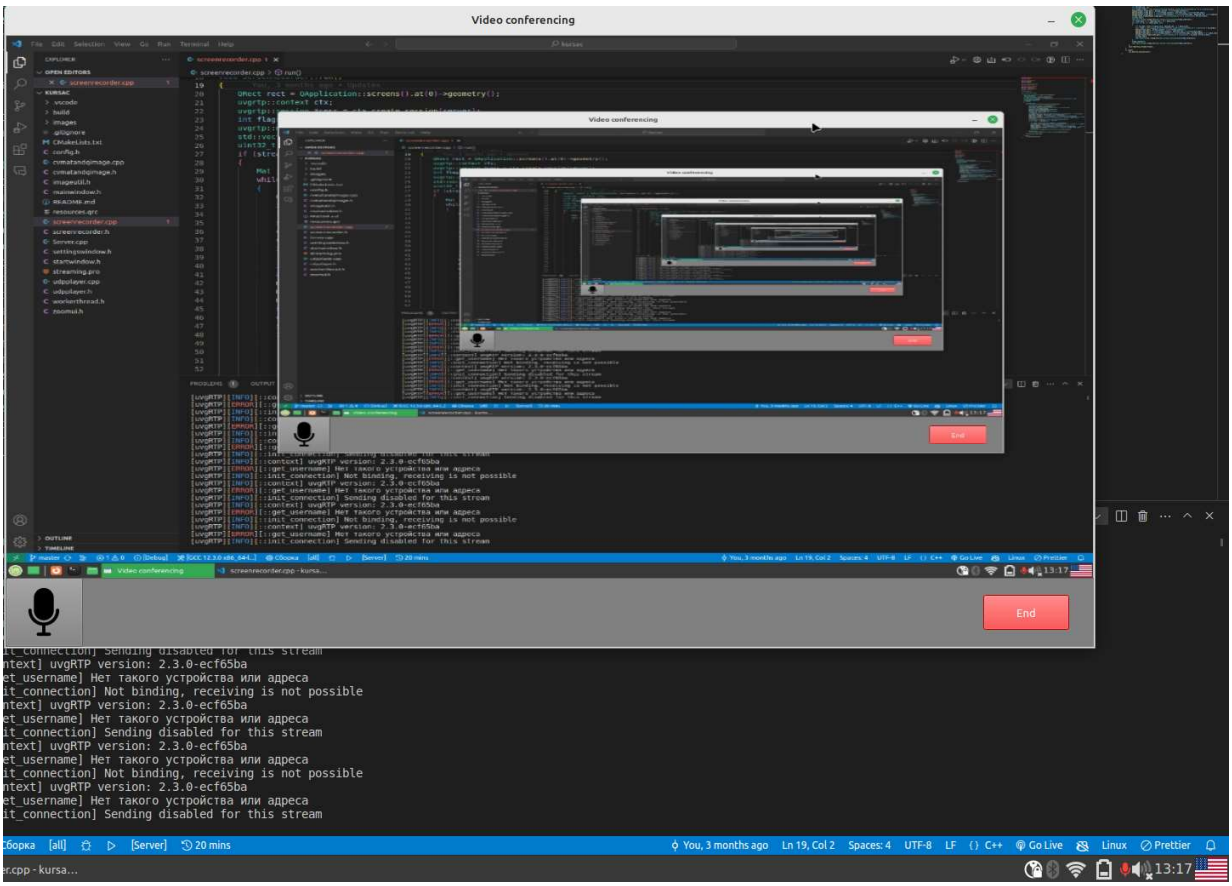

Рисунок 3 – Окно звонка (микрофон включен)

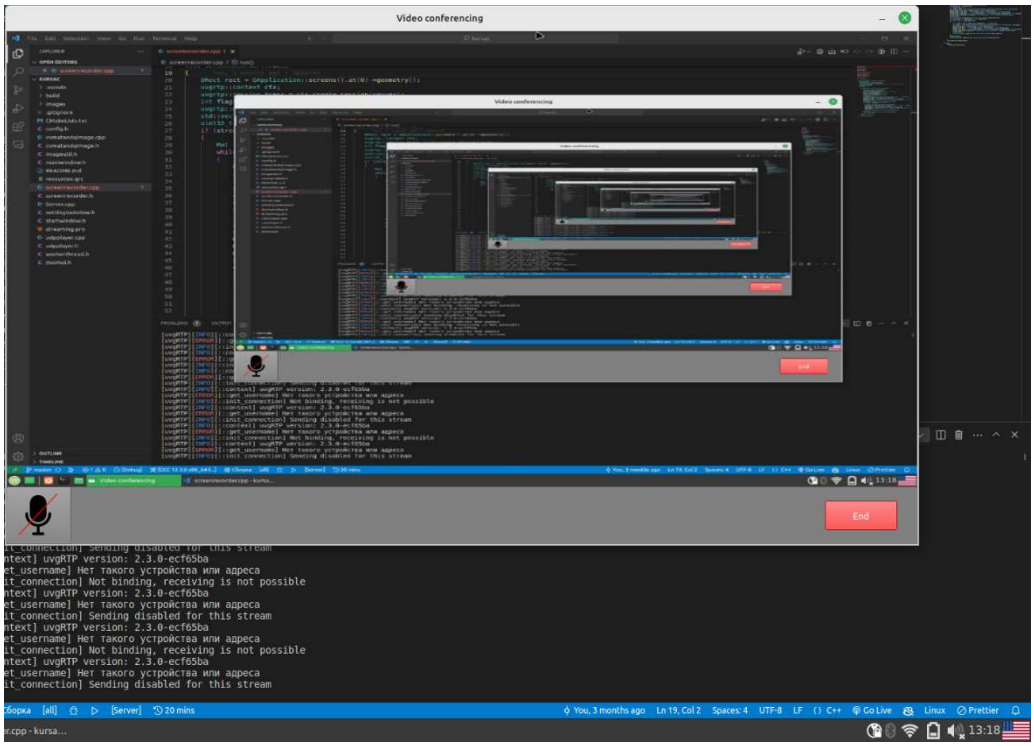

Рисунок 4– Окно звонка (микрофон выключен)

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.

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

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