МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ ПЕРЕДАЧИ ВИДЕОПОТОКА С КАМЕРЫ МОБИЛЬНОГО УСТРОЙСТВА В ПРИЛОЖЕНИЯ WINDOWS

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Голышева Юрия Олеговича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванова

Заведующий кафедрой
доцент, к. ф.-м. н. _____ С. В. Миронов

Саратов 2026

СОДЕРЖАНИЕ

1 Анализ предметной области

Качество встроенных камер ноутбуков и недорогих USB-камер заметно уступает камерам смартфонов даже среднего ценового сегмента. Существующие решения для использования камеры смартфона в роли веб-камеры (DroidCam, Iriun Webcam, iVCam) имеют ряд общих ограничений:

— бесплатные версии искусственно ограничены разрешением, водяными знаками или рекламой;
— протоколы передачи закрыты, что не позволяет встраивать решения в собственные проекты;
— для пользователей из России покупка платных версий затруднена.

Целью работы является разработка программного комплекса, который обеспечит передачу видеопотока с камеры мобильного устройства в приложения на Windows через Wi-Fi. В ходе работы поставлены задачи:

1. провести обзор существующих решений и используемых в работе технологий;
2. разработать мобильное приложение для захвата и передачи видеопотока с камеры Android-устройства в форматах JPEG, H.264 и H.265;
3. разработать серверное приложение для Windows, принимающее видеопоток, отображающее его и передающее в виртуальную камеру;
4. разработать DirectShow-фильтр виртуальной камеры, предоставляющий видеопоток сторонним приложениям;
5. обеспечить взаимодействие компонентов программного комплекса и провести его тестирование.

1.1 Используемые технологии

Для разработки мобильного приложения выбрана платформа .NET MAUI — кроссплатформенная инфраструктура для создания нативных приложений на C# и XAML. Для разработки серверного приложения выбрана связка .NET и WPF. Сочетание MAUI и WPF даёт единый язык C# и единую среду исполнения для всех частей программного комплекса.

Захват кадров с камеры Android-устройства выполняется через два программных интерфейса: `Camera2 API` и `CameraX`. Захват для сжатия в JPEG реализован через сценарий `ImageAnalysis` библиотеки `CameraX`. Захват для аппаратного кодирования в H.264 и H.265 реализован напрямую через `Camera2`

API с подключением входной поверхности (InputSurface) аппаратного кодера MediaCodec в качестве приёмника кадров. В этом режиме передача кадров с матрицы в кодер выполняется средствами графического процессора устройства, без участия центрального процессора.

H.264 и H.265 — стандарты сжатия видео, принятые ITU-T. Оба используют блочное межкадровое кодирование, сжатый поток разделён на единицы NAL разных типов: IDR (ключевой), P (предсказанный), SPS, PPS и VPS (параметрические наборы). B-кадры отключены настройкой кодера, так как они вносят задержку. Параметрические наборы дописываются перед каждым IDR опцией prepend-sps-pps-to-idr-frames. Современные видеокодеки работают в YUV, где отдельно хранится яркость Y и две компоненты цветности U и V. NV12 — раскладка YUV 4:2:0, в которой сначала идёт вся плоскость Y, а затем чередующиеся байты U и V. Для преобразования между RGB и YUV используется матрица коэффициентов из стандарта ITU-R BT.601.

DirectShow — мультимедийная подсистема Windows, построенная вокруг графа фильтров, соединённых через пины. Виртуальная камера реализуется как фильтр-источник — COM-сервер в виде DLL, регистрируемой утилитой regsvr32. При регистрации фильтр попадает в категорию с идентификатором CLSID_VideoInputDeviceCategory и становится виден в стандартном списке устройств видеозахвата.

Передача данных между мобильным и серверным приложениями выполняется по TCP. UDP теоретически дал бы меньшую задержку, но передача NAL-единиц по UDP без надстроек приводит к видимым артефактам при любой потере пакета, а JPEG-кадры пришлось бы фрагментировать вручную. Декодирование принятых кадров на серверной стороне выполняется библиотекой libavcodec через обёртку Sdcb.FFmpeg. Обмен данными между серверным приложением и виртуальной камерой реализован через именованную разделяемую память.

2 Разработка мобильного приложения

Мобильное приложение является источником видеопотока во всем программном комплексе. Оно захватывает кадры с камеры Android-устройства, кодирует их в JPEG, H.264 или H.265, а затем передаёт по локальной сети серверному приложению. Исходный код разделён на кросс-платформенный уровень (общий код на C# и разметка XAML: главная страница, страница настроек, абстракция камеры, сетевой клиент) и платформенный уровень для Android (обработчик абстракции камеры, отдельный класс для H.264 и H.265, класс настроек, активность). В приложении есть два способа захвата кадров. Первый работает с библиотекой `CameraX`. Второй работает напрямую с `Camera2 API` и аппаратным кодером `MediaCodec`. Формат потока определяется одним байтом, отправляемым сразу после подключения.

2.1 Способы захвата кадров

В способе захвата через `CameraX` создаётся компонент `ImageAnalysis` с форматом `YUV_420_888`. Обработка одного кадра (копирование данных в NV21 и сжатие в JPEG) на одном потоке процессора не успевала бы. Поэтому реализован диспетчер `DispatchingAnalyzer`, владеющий двумя экземплярами обработчика, кадры идут в них по очереди. Так как обработчики работают параллельно, JPEG-кадры могут оказаться готовы не в порядке номеров. Для его сохранения реализован класс `OrderedDelivery`, отдающий кадр только тогда, когда подошёл его порядковый номер. Сжатие в JPEG выполняется методом стандартного класса `YuvImage`, принимающего массив, размещённый в куче Java. Управляемые массивы C# выделяются в куче .NET, напрямую передать их в Java-метод нельзя, поэтому один раз при первом кадре выделяется Java-массив нужного размера, а при последующих кадрах содержимое NV21 копируется в этот же массив. Иначе на каждом кадре создавалось бы около 3 МБ мусора.

Способ захвата через `Camera2 API` и `MediaCodec` реализован в классе `H264EncoderPipeline`. У `MediaCodec` настраиваются тип кодирования, цветовой формат `Surface`, разрешение, частота кадров, битрейт, интервал ключевых кадров, профиль кодирования, запрет двунаправленных кадров, приоритет малой задержки и опция `prepend-sps-pps-to-idr-frames`. После настройки у кодера запрашивается поверхность ввода `InputSurface`, которая подключается к сессии захвата в качестве приёмника кадров. Затем запускается поток вывода,

в цикле берущий у кодера готовые NAL-блоки и передающий их во внешний обратный вызов. Битрейт вычисляется по формуле $B = W \times H \times F \times bpp$, где bpp — количество бит на пиксель (по умолчанию 0.083), и ограничивается значениями из настроек (по умолчанию 2 и 25 Мбит/с).

2.2 Сетевое взаимодействие и адаптивный битрейт

В классе `CameraStreamingService` реализована передача закодированных кадров. Каждый кадр упаковывается в двоичный пакет: 4 байта маркера начала, 4 байта длины полезной нагрузки в little-endian, полезная нагрузка и 4 байта маркера конца. Маркеры нужны, чтобы серверное приложение могло выделить границы каждого кадра в непрерывном TCP-потоке. Сам пакет собирается в буфере из `ArrayPool<byte>.Shared`. Готовый пакет кладётся в ограниченную очередь между обработчиком кадров и отдельным потоком отправки. Для JPEG размер очереди равен 1 и применяется вытеснение старого новым: каждый JPEG-кадр самодостаточен. Для H.264 и H.265 размер очереди равен 8 и при заполнении запись ожидает освобождения места до 500 миллисекунд: потери NAL-блоков стоит избегать, так как каждый зависит от предыдущих.

Очередь отправки является источником обратной связи о состоянии сети: раз в окно (по умолчанию две секунды) подсчитывается доля заблокированных записей. Если она не меньше 5%, битрейт уменьшается умножением на 0.85, если меньше 1% на протяжении трёх окон подряд — увеличивается умножением на 1.10. Изменение применяется через `MediaCodec.SetParameters`, сессию захвата при этом перезапускать не нужно.

2.3 Сканирование QR-кода и интерфейс

Для удобства подключения добавлен сканер QR-кода. Серверное приложение генерирует QR со строкой вида «`cam://192.168.0.106:8888`». На время сканирования происходит переключение на способ через `CameraX`. Каждые 200 миллисекунд один из двух обработчиков пытается распознать QR библиотекой `ZXing.Net`. Главный экран занимает всю площадь экрана устройства, в качестве фона — превью с камеры. В `Microsoft.Maui.Storage.Preferences` сохраняются все настройки, доступ инкапсулирован в классе `CameraSettings`. Помимо тёмной темы, приложение поддерживает светлую: цвета хранятся в классе `ThemeManager`, в XAML задаются через имена ресурсов, при переключении `ThemeManager` подставляет в эти ресурсы значения нужной палитры.

3 Разработка серверного приложения

Серверное приложение работает на стороне компьютера с Windows и выполняет в программном комплексе центральную роль: принимает видеопоток от мобильного приложения по локальной сети, при необходимости декодирует его, отображает в окне для пользователя и передаёт в виртуальную камеру.

3.1 Приём подключений и диспетчеризация

По нажатию кнопки «Запустить» приложение определяет локальный IP-адрес компьютера, проверяет, что порт 8888 свободен, и создаёт `TcpListener`. Параллельно проверяется наличие в брандмауэре Windows правила, разрешающего входящие подключения, и при отсутствии предлагается создать его автоматически. После старта открывается окно с QR-кодом, генерация QR выполняется библиотекой `QRCoder`.

В цикле приёма каждое подключение оборачивается в `TcpClient` с отключённым алгоритмом Нейгла, приёмным буфером 128 КБ и таймаутами. Затем передаётся в метод-диспетчер, читающий из потока один байт для определения формата. Для JPEG особый случай: байт `0x01` совпадает с первым байтом маркера начала кадра, поэтому диспетчер не отбрасывает прочитанный байт, а оборачивает поток в класс-обёртку `PrefixedStream`, который при первом чтении вернёт уже прочитанный байт, а дальше начнёт читать из настоящего сокета.

3.2 Приём и декодирование кадров

Метод `ProcessClientFast` для JPEG в цикле читает маркер начала, размер полезной нагрузки, саму нагрузку и маркер конца. Чтение выполняется методом `ReadExactAsync`, дочитывающим указанное число байт даже если они приходят несколькими TCP-пакетами. Метод `ProcessClientVideo` для H.264 и H.265 устроен сложнее: внутри создаётся ограниченная очередь блоков NAL вместимостью 16, и стартует `VideoDecodeLoopAsync` — отдельная задача декодирования. Такое разделение приёма и декодирования важно для производительности: на разрешении 4K с HEVC декодирование одного блока NAL может занимать около 50 миллисекунд, и если бы декодирование выполнялось в потоке приёма, окно TCP закрывалось бы, а на смартфоне приходилось бы тормозить отправку. В цикле приёма после чтения каждого блока NAL принимается решение, что с ним делать: если виртуальная камера активна и потребитель выбрал

тот же формат, что и текущий поток, блок отправляется в виртуальную камеру как есть, без декодирования; иначе или дополнительно, если включено превью, блок идёт в декодер.

Декодирование вынесено в класс `H264Decoder`, инкапсулирующий работу с `libavcodec`. Используются три объекта: `CodecParserContext` выделяет из произвольно разрезанного потока полноценные пакеты NAL, `CodecContext` декодирует пакеты в кадры, `VideoFrameConverter` (`libswscale`) преобразует внутренний YUV-формат в NV12. Эмпирически подобраны значения по умолчанию: для H.264 включён режим `LowDelay`, для H.265 — два потока `frame-threading`, иначе процессор не успевает в 2K и 4K разрешениях.

3.3 Передача кадров в виртуальную камеру и интерфейс

Отправка кадров в виртуальную камеру выделена в отдельный поток `VCamSend`, кадры попадают в него через очередь из одного слота с вытеснением старого новым. На 4K-кадрах одна запись в разделяемую память (копирование около 12 МБ) занимала слишком много времени, и в потоке приёма это приводило бы к копящейся задержке.

Обмен с DirectShow-фильтром идёт через разделяемую память с именем `Global\vCamShm`. При создании на неё накладывается разрешение DACL `D:(A;;GRGW;;;WD)`, разрешающее чтение и запись для всех пользователей. Без него сторонние приложения без прав администратора не смогли бы записать поле `consumerFormat`. Раскладка состоит из заголовка и двух буферов под кадры (каждый рассчитан на 4K NV12, около 12 МБ). Заголовок хранит счётчик кадров, индекс активного буфера, метаданные кадра и `consumerFormat`. Два буфера нужны, чтобы избежать разрывов изображения. Серверное приложение пишет в неактивный буфер, обновляет метаданные, атомарно меняет `currentBuffer` и инкрементирует `frameId`. Между шагами стоит `Thread.MemoryBarrier()`, предотвращающий изменение порядка инструкций процессором и компилятором.

Регистрация виртуальной камеры выполняется системной утилитой `regsvr32`, для чего нужны права администратора. Серверное приложение запускается уже с правами администратора, поэтому регистрация выполняется автоматически. Снятие регистрации выполняется при закрытии приложения. Главное окно разделено на три горизонтальные зоны:

— в верхней панели отображаются статусная строка, статус виртуальной ка-

меры с пометкой текущего кодека, счётчик кадров в секунду, разрешение и битрейт;

— центральная зона — область отображения превью;

— нижняя содержит кнопки запуска и остановки приёма подключений, открытия окна с QR-кодом, галочку «Показывать превью» и кнопку перехода в настройки.

В серверном приложении системная шапка отключена (`WindowStyle = None`), а вместо неё рисуется собственная — реализация вынесена в статический класс `WindowDarkChrome`. Помимо тёмной, поддерживается светлая тема по той же схеме, что и в мобильном приложении.

4 Разработка виртуальной камеры

Виртуальная камера завершает цепочку передачи видеопотока. Она регистрируется в Windows как стандартное устройство видеозахвата и становится доступна любому приложению, работающему с веб-камерами. Реализована на C++ как DirectShow-фильтр-источник. В отличие от двух других компонентов программного комплекса, виртуальная камера работает внутри чужого процесса: Windows загружает её DLL в адресное пространство сторонних приложений.

4.1 Структура классов и зависимость от BaseClasses

DirectShow BaseClasses реализует рутинную часть работы любого DirectShow-фильтра: подсчёт ссылок COM, интерфейсы `IUnknown` и `IBaseFilter`, поведение пинов при соединении и отсоединении, аллокацию выходных буферов. На основе BaseClasses реализованы два класса. `CVCamSource` наследуется от `CSource` и реализует `IAMFilterMiscFlags`: в нём только конструктор, создающий единственный выходной пин, и метод `GetMiscFlags`, сообщающий DirectShow, что фильтр является источником. `CPushPinVCam` наследуется от `CSourceStream` (базовый класс выходного пина) и реализует `IAMStreamConfig` (для согласования формата с потребителем) и `IKsPropertySet` (для объявления пина как пина захвата). В нём переопределены методы `GetMediaType`, `GetStreamCaps`, `SetMediaType`, `DecideBufferSize` и главный — `FillBuffer`. Цикл выдачи кадров запускает базовый класс `CSourceStream`: после старта графа он создаёт отдельный рабочий поток, в котором циклически берёт из заранее выделенного пула пустой медиасэмпл (`IMediaSample`), вызывает переопределённый `FillBuffer` и отправляет заполненный медиасэмпл дальше по графу.

4.2 Регистрация, форматы и заполнение кадра

При вызове `regsvr32 VirtualCamFilter.dll` система вызывает функцию `DllRegisterServer`, в которой через BaseClasses регистрируются COM-класс фильтра и его свойства: GUID, имя в системе, `KSCATEGORY_VIDEO_CAPTURE` категория, благодаря которой фильтр попадает в стандартный список веб-камер. Фильтр объявляет пять форматов: NV12, I420, RGB24, H.264 и H.265. Для каждого формата объявлены семь разрешений (от 640×480 до 3840×2160) и две частоты кадров (30 и 60), общее количество комбинаций — 70. Объявление

H.264 и H.265 необходимо, чтобы отдавать их напрямую, без декодирования в серверном приложении, если потребитель такое поддерживает.

Когда потребитель выбирает виртуальную камеру в качестве источника видео, DirectShow вызывает у фильтра `SetMediaType` или `SetFormat` с выбранным типом. Фильтр запоминает выбранный формат, ширину, высоту и длительность кадра, и записывает в поле `consumerFormat` соответствующий код. Серверное приложение опрашивает `consumerFormat`: если выбран код того же кодека, что и текущий поток, отправляется закодированный блок NAL без декодирования; если выбран `FORMAT_NV12`, в виртуальную камеру попадает декодированный NV12.

Главный метод фильтра — `FillBuffer`. DirectShow вызывает его каждый раз, когда потребителю нужен очередной кадр. У метода две ветки. В ветке для H.264 и H.265 фильтр сначала проверяет, что в разделяемой памяти лежат данные именно того кодека, что выбрал потребитель, дальше ждёт обновления `frameId`, опрашивая поле с шагом 2 миллисекунды и потолком в 50 миллисекунд. Без этого ожидания `FillBuffer` вызывался бы DirectShow быстрее, чем серверное приложение успевает класть новые кадры, и один и тот же блок NAL уходил бы потребителю несколько раз с разными метками. После прихода нового кадра выполняется проверка точки синхронизации: до первого IDR все промежуточные P-кадры отбрасываются. Сам блок копируется одним `CopyMemory`, выставляется флаг `sync-point` и метка времени.

В ветке для несжатых форматов фильтр после проверки и ожидания нового кадра выполняет одно из трёх действий. Для NV12 при совпадении исходного и выходного разрешений выполняются два `CopyMemory`, при разных — масштабирование по ближайшему соседу. Для I420 копируется Y как есть, и UV-плоскость разворачивается в две раздельные плоскости. Для RGB24 выполняется матричная конверсия NV12 в BGR24 по формуле BT.601 с переворотом по вертикали, так как DirectShow требует порядка строк снизу вверх. Временные метки у медиасэмпла проставляются на основе показаний `QueryPerformanceCounter`, пересчитанных в единицы DirectShow (100 наносекунд). Метка первого медиасэмпла берётся за ноль, метки последующих — как разница с этим нулём.

5 Взаимодействие компонентов и тестирование

Разработанный программный комплекс состоит из трёх компонентов — мобильного приложения, серверного приложения и DirectShow-фильтра виртуальной камеры, образующих единую цепочку передачи видеопотока от камеры смартфона до любого приложения, работающего с камерами на компьютере. Прежде чем первый кадр пойдёт по цепочке, должны быть подняты три канала связи:

- между мобильным и серверным приложениями — по QR-коду и TCP-соединению;
- между серверным приложением и виртуальной камерой — через разделяемую память `Global\vCamShm`;
- между виртуальной камерой и приложением-потребителем — автоматически операционной системой.

Путь одного кадраначинаетсяс того, что в режиме H.264/H.265 кадр с матрицы попадает напрямую в `MediaCodec` через входную поверхность, а в в JPEG-режиме `CameraX` отдаёт сырой YUV, мобильное приложение сжимает его в JPEG. Готовый кадр инкапсулируется в пакет и уходит в TCP-сокет. Серверное приложение читает пакет и заглядывает в `consumerFormat`, если потребитель выбрал тот же кодек, NAL-блок переписывается в неактивный буфер разделяемой памяти, иначе блок передаётся в очередь декодирования. DirectShow периодически вызывает у виртуальной камеры `FillBuffer`, который копирует данные в медиасэмпл и отдаёт его дальше по графу. Параллельно идёт обратная связь: от виртуальной камеры к серверному приложению — через поле `consumerFormat`; от серверного к мобильному — через состояние очередей: при их заполнении окно TCP закрывается, и адаптивный контроллер на смартфоне снижает битрейт.

Для проверки работоспособности пройден полный сценарий — от запуска приложений до получения изображения в OBS Studio и Яндекс Телемост. После нажатия «Запустить» открывается окно с QR-кодом, на смартфоне необходимо нажать «Подключиться». После распознавания QR, приложение автоматически подключается, а серверное приложение регистрирует виртуальную камеру в системе. После нажатия `Start` в окне серверного приложения появляется превью с камеры. После выбора `Phone Camera` в Яндекс Телемост в превью появляется видеопоток — это подтверждает работу режима с декодированием

в NV12. В OBS Studio, при выборе пользовательских настроек с необходимым разрешением и форматом, видеопоток появляется в превью, что подтверждает режим прямой передачи сжатого H.265 без декодирования. Прохождение обоих сценариев подтверждает работоспособность программного комплекса — поставленные цель и задачи достигнуты.