ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Сложно представить жизнь современного человека без интернета и технологий, которые его образуют. Существует огромное количество возможностей, которые предоставляет нам интернет: доступ к свободной информации в любое удобное время, возможность отсылать и получать электронные письма, возможность участвовать в телеконференциях и удаленных встречах, возможность совершать покупки, не выходя из дома, играть в сетевые компьютерные игры и многое другое.

Среди прочего большую значимость для людей имеет потоковое вещание. Потоковое вещание — это мультимедиа контент (аудио-, видео-, а также текстовые данные), которые пользователи непрерывно получают от поставщика услуг [1]. Это в том числе применимо к аналоговым средствам вещания наподобие радио или телевидения, хотя таковые отношение к интернету имеют посредственное.

В рамках данной работы наибольшее значение имеет потоковое видео вещание в режиме реального времени. Наиболее важные решения, интересующие нас в данной работе таковы:

1) Массовый онлайн стриминг. Под данной категорией подразумеваются платформы способные транслировать видео в реальном времени во всемирную паутину обширному, условно неограниченному количеству пользователей (ограничивают количество пользователей технические возможности платформы). Наиболее популярные сервисы — YouTube, Twitch, Facebook Live и прочие.

2) Видеоконференции. Под данной категорией подразумеваются решения, обеспечивающие двустороннюю передачу, обработку, преобразование и представление видеоданных между несколькими пользователями, количество

которых ограничено, независимо от их месторасположения в режиме реального времени. Skype, Zoom, Google Hangouts Meet — наиболее яркие решения, которые предоставляют данный формат вещания.

Можно сказать, что данные категории узконаправлены в некотором понимании. Массовый онлайн стриминг предоставляет возможность вести трансляцию только одного медиа-потока, но этот поток доступен неограниченному количеству пользователей. Более того, данный видеопоток может обладать высоким качеством, например иметь высокое разрешение. Это в свою очередь говорит о том, что данный поток будет иметь больший объем данных, и, если мы захотим производить трансляцию видео в реальном времени в высоком качестве будет присутствовать ощутимая задержка.

Видеоконференции так же позволяют производить только один медиа-поток данных для одного пользователя, но так как количество пользователей больше одного, таких потоков может быть несколько. Так же стоит отметить, что количество участников ограничено, а конференция зачастую закрыта для широкой публики. Видео качество же тут сравнимо ниже, чем у массового онлайн стриминга, что дает возможность сократить задержку видео.

Отдельного упоминания достойна проблема создания так называемой сцены (отображения того, как будет отображаться видео, если помимо кадров, захваченных устройством захвата, будут присутствовать дополнительные кадры или элементы) – отсутствует возможность многофункционального администрирования таковой по умолчанию. В видеоконференциях возможен вариант локального упорядочивания медиа-потоков отдельных людей. В общем случае сцену необходимо создавать лично, с использованием сторонних приложений.

Возможность участия нескольких людей в одном медиа-потоке также ограничена и зачастую сводится к одному из следующих вариантов:

1) непосредственное нахождение людей в пределах видимости камеры захвата медиа-потока;
2) комбинирование меда-потоков в один прим помощи стороннего программного обеспечения, при этом возможны следующие варианты захвата этих потоков:

    a. производится запись на одном аппаратном устройстве с использованием разных устройств захвата кадров (например, разные камеры на одном персональном компьютере; захват экрана параллельно с захватом изображения камерой так же можно отнести к данной категории, несмотря на то что участник может быть один),

    b. производится запись на различных аппаратных устройствах внутри локальной сети, при этом производится вещание одного потока в аппаратное устройство поставщика основного медиа-потока,

    c. аналогично предыдущему за исключением нахождения устройств в сети интернет.

Очевидны недостатки данных подходов: для первого необходимо физическое присутствие людей в одном месте, для второго необходима установка стороннего программного обеспечения, а также сложности с динамическим количеством участников (выходом может быть использование различных предустановленных сцен для определенного количества участников, что так же неудобно).

Среди существующих платформ нет такой, которая сочетала бы свойства обеих категорий платформ потокового вещания в режиме реального времени:

возможность участия нескольких пользователей, медиа-поток которых бы мог собираться в единый, возможность администрирования потоков пользователей и создания сцены (например, динамическое размещение потоков в одной сцене при изменении количества потоков). Такие платформы здесь и далее будем называть виртуальной аппаратной.

Целью данной работы является обеспечение возможности динамической композиции транслируемого видеопотока под управлением нескольких пользователей.

Для достижения данной цели были поставлены следующие задачи:

1) проанализировать существующие средства организации передачи потокового видео в реальном времени (массовый онлайн стриминг и конференции);
2) изучить способы организации совместных трансляций с помощью существующих средств и выявить недостатки;
3) проанализировать средства, используемые для организации передачи потокового видео (композиция сцены, библиотеки кодеков для транскодирования, протоколы передачи);
4) разработать решение для совместного видео-стриминга.

1 ДОСТАВКА ВИДЕО КОНТЕНТА

1.1 Сценарии доставки

1.1.1 Просмотр видео в интернете

Как известно, в интернете найдется все — нужно лишь хорошенько поискать. Пользователи интернета имеют доступ к миллионам видео файлов, расположенных в глобальной сети. В любое удобное время можно посмотреть любимый фильм или сериал, смешной видеоролик именно благодаря данному сценарию доставки видео контента.

Данный сценарий доставки контента является наиболее распространенным и востребованным. Он представляет из себя предоставление видео файлов по запросу по распространенным протоколам передачи, например по UDP или TCP. Наиболее простой пример – HTML-5 видео, который используется в веб приложениях, где видео интегрировано при помощи тега <video> с указанием расположения видеофайла. Количество решений, предоставляющих данный сценарий доставки видео контента, довольно обширно. Самый популярный поставщик данного сценария – YouTube.

Среди прочих также популярны стриминговые сервисы и онлайн кинотеатры, предоставляющие доступ к видео по подписке. Примерами таких сервисов являются Netflix, Okko, Ivi, Megogo и прочие.

На рисунке 1 приведен пример данного сценария.

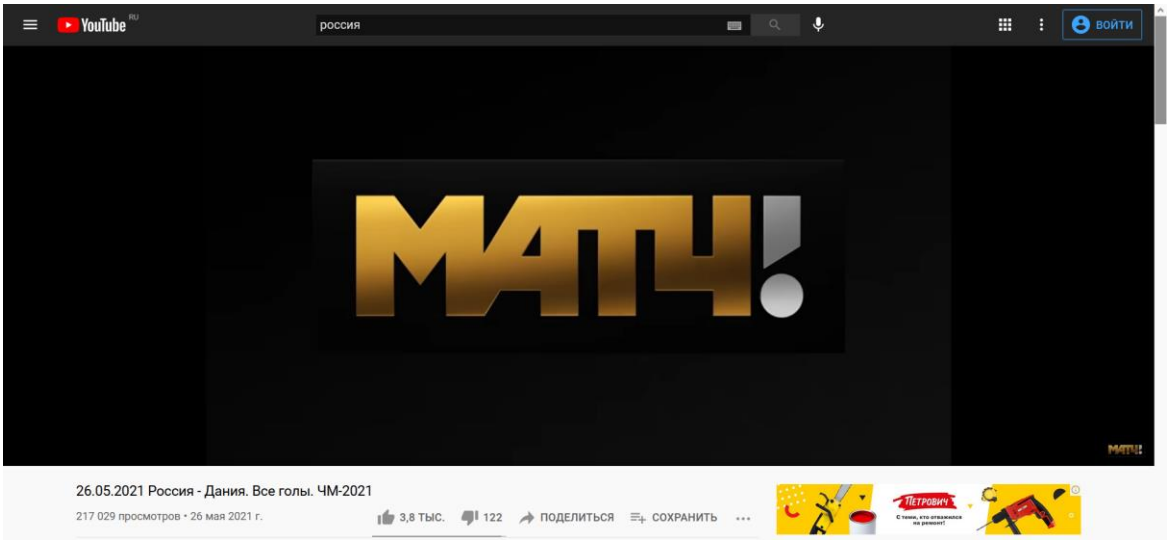

Рисунок 1 – Сценарий доставки видео «просмотр видео» на примере видео платформы YouTube

Имеет смысл рассмотреть возможную организацию данного сценария. Имеется сервер, который предоставляет доступ к видео материалу, например к файлу формата mp4 который хранится где-то на сервере. Клиентом данного сервера является любой имеющий доступ к серверу пользователь.

1.1.2 Видеосвязь, конференции

Видеосвязь — это сессия переговоров между ее участниками, во время которой участники могут слышать и видеть друг друга. Данное понятие очень схоже с видеоконференцсвязью, но значения у них разительные, в основном из-за сферы их применения. На данный момент видеосвязь является более публичным, массовым сценарием, и используется преимущественно в личных целях. Когда мы общаемся по видеозвонку со своими друзьями или родственниками в непринужденной обстановке, нам нужно слышать и видеть их в первую очередь. Качество звука обеспечивается посредством хороших микрофона и динамиков, а для четкой картинки мы выбираем веб-камеру, чего вполне достаточно. Для сеанса видеоконференцсвязи же такого оборудования

будет не хватать. Видеоконференцсвязь является инструментом ведения бизнеса, поэтому требования у нее гораздо выше.

Существует множество сервисов, которые предоставляют видеосвязь. Те же социальные сети, например Facebook, VK или WhatsApp, уже давно завоевали доверие пользователей в данной области. Стоит отметить, что существуют такие решения, которые не требуют наличия другого ПО кроме браузера для обеспечения видеосвязи. Одно из таких решений – это Google Hangout Meet, пример которого приведен на рисунке 2.

Рисунок 2 – Сценарий доставки видео «видеосвязь, конференции» на примере сервиса Google Duo

Рассмотри один из способов организации данного сценария. Клиентом является пользователь, который участвует в видеосвязи. Медиа поток клиента отправляется на сервер, в роли которого может выступать как другой клиент, так

и специализированная под такой тип задач ЭВМ, после чего медиа поток рассылается другим клиентам конференции.

1.1.3 Трансляции с веб-камер

Не покидая собственной квартиры, можно побывать на известных достопримечательностях, в далеких городах или даже посмотреть на нашу планету с борта МКС. Все это доступно благодаря возможности просматривать трансляции с веб-камер, хотя и не ограничивается только развлечением: в последнее время все большую актуальность обретает возможность наблюдения за заседанием государственной думы или, например, наблюдение за избирательными участками во время выборов, что помогает обнаружить нарушения.

Пример такого сценария приведен на рисунке 3.

В роли сервера в данном случае выступает веб-камера, доступ к видео у которой будет просить клиент. Клиентом может быть как конечный интернет-пользователь, так и другой сервер, который далее будет оперировать данным потоком неким образом.

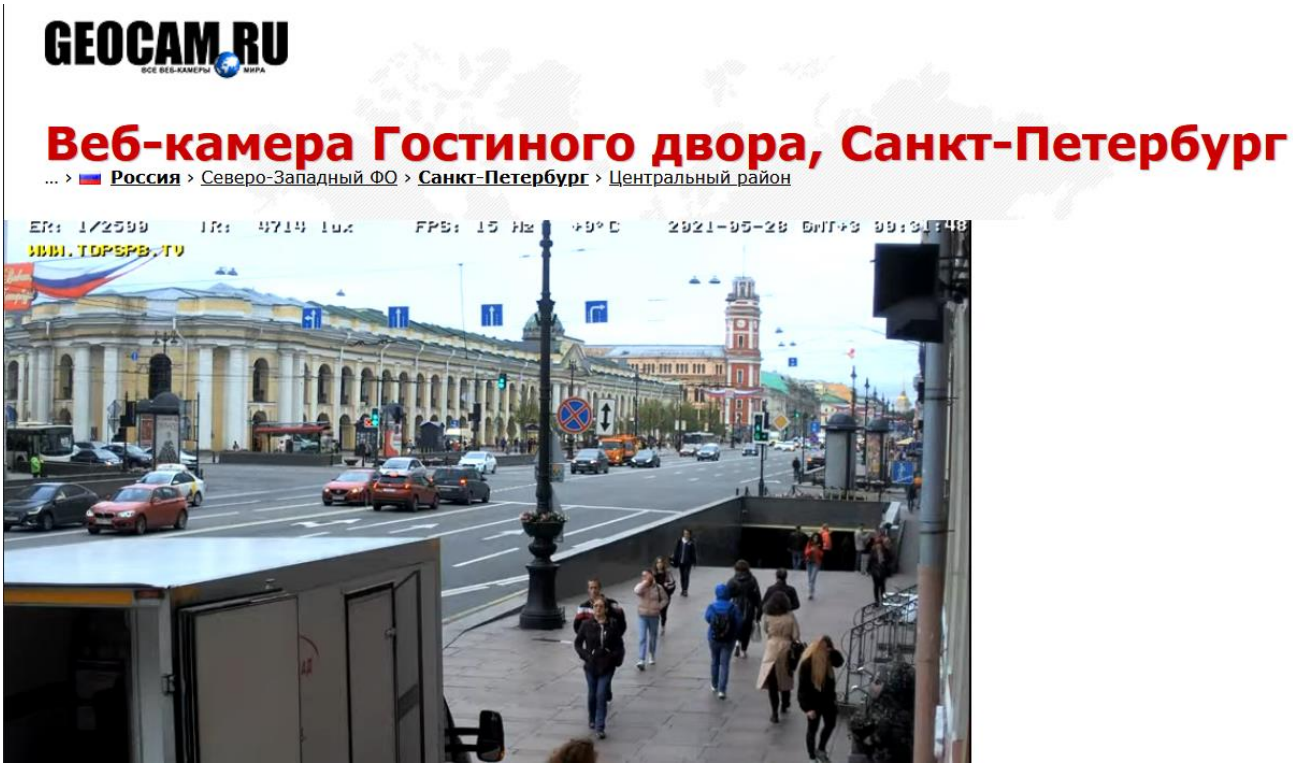

Рисунок 3 –Сценарий доставки видео «трансляции с веб-камер» на примере сервиса geocam.ru

1.1.4 Стриминг

Благодаря современным технологиям любому пользователю глобальной сети доступна возможность создания собственной трансляции при наличии соответствующего оборудования. Более того, ощутимого числа людей эта возможность стала основным источником дохода: зритель может материально поддержать стримера либо прямым пожертвованием, либо подпиской на данного стримера, если стриминговая платформа поддерживает монетизацию данной функции, либо самым обычным просмотром рекламы. Стоит отметить, что основное количество людей собраны вокруг киберспортивного контента, который представляет из себя трансляцию игровой деятельности.

Пример сценария приведен на рисунке 4.

Рисунок 4 –Сценарий доставки видео «стриминг» на примере видео кадра, взятого с платформы Twitch у пользователя yogscast

Несмотря на то, что в данном сценарии так же генерируется медиа-поток в реальном времени, как и в предыдущем сценарии, сервером в данном случае является специализированное ПО/ЭВМ, на который публикуется генерируемый медиа поток. Так же клиентом сервера считается пользователь, который запрашивает медиа поток стримера.

Еще одно отличие состоит в том, что веб-камеры могут поставлять медиа поток одного типа — захватываемое данной камерой видео, а в данном сценарии медиа поток может быть сгенерирован как камерой стримера, так и специализированным ПО, которое может комбинировать различные видео элементы, видео захвата экрана, видео с камеры стримера, посторонние изображения и тому подобное.

1.2 Существующие решения

1.2.1 Определение области рассматриваемых решений

Как уже упоминалось ранее, существует огромное количество решений, которые осуществляют вышеперечисленные сценарии. Это могут быть серверы, оперирующие как статичными файлами, так и динамическим контентом пользователя.

В рамках поставленной цели имеет смысл рассматривать только те решения, которые позволяют генерировать медиа поток. Категорий таких решений 2:

- решения видеосвязи и конференция, которые в основном генерируют медиа поток основываясь на камерах (usb-камерах, web-камерах, а также камерах захвата экрана);
- программные энкодеры медиа потока, которые могут генерировать медиа поток основываясь не только на камерах, но и других внутренних и внешних источниках (например, внешний rtmp сервер или внутреннее статичное видео).

Для ясности и дальнейшего понимания стоит определиться с понятием программные энкодеры. Под данным понятием подразумевается класс программного обеспечения, которое способно генерировать медиа поток из других исходных данных и затем публиковать его на удаленные медиа сервера.

1.2.2 Программные энкодеры

1.2.2.1 Open Broadcast Software

Наиболее популярным решением является Open Broadcaster Software (OBS). OBS – это бесплатная программа с открытым исходным кодом для записи видео и потокового вещания. Доступна на многих операционных системах, среди которых присутствуют Windows, Linux и Mac. Среди конкурентов решение выделяют:

- возможность стримить на любые стриминговые платформы,
- большой выбор плагинов, число которых постоянно растет,
- локализация для 40 языков.

Пример решения приведен на рисунке 5.

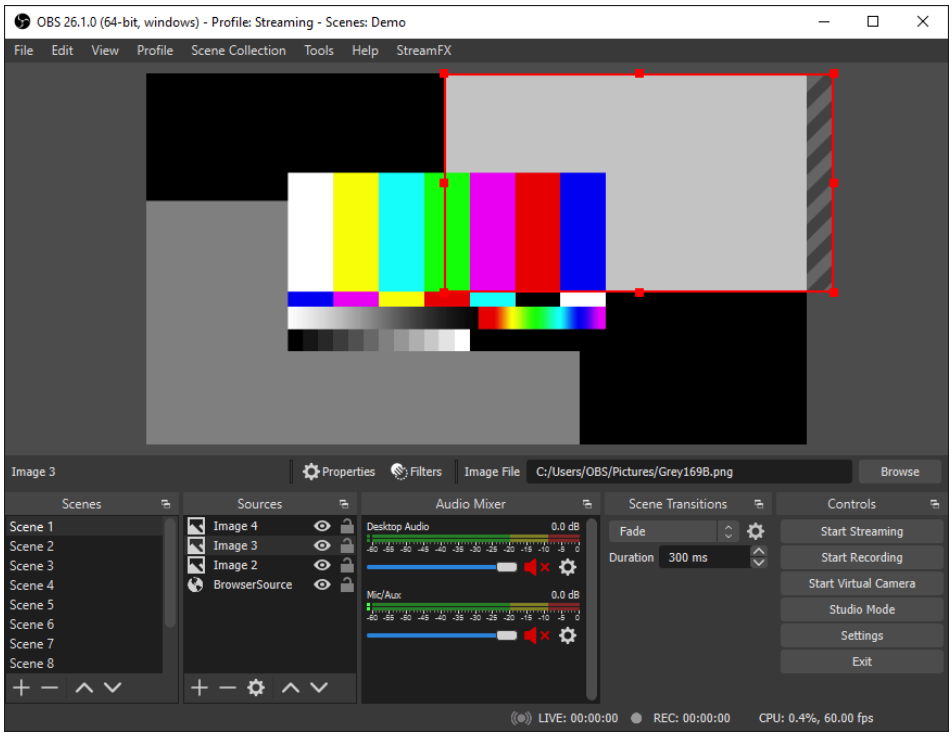

Рисунок 5 – Решение по реализацию сценария «стриминг» на примере ПО OBS

Среди минусов можно выделить отсутствие возможности одновременно стримить на несколько платформ из коробки, а также отсутствие возможности смены битрейта на лету.

В целом, данное решение позволяет собирать локальные элементы и комбинировать их в единый медиа поток и публиковать его на rtmp сервер из коробки, но может быть расширено плагинами. Например, можно вещать по другим транспортным протоколам или подключать внешние источники данных. Стоит упомянуть плагин OBS-studio-webrtc [2], который способен получать данные из видеоконференций, основанных на стандарте WebRTC реализованных

в медиа сервере Janus. С подобными плагинами возможность создания виртуальной аппаратной становится реальной, хотя и требует большого количества накладных ресурсов и сторонних серверов.

Стоит также отметить наличие сцен и возможности переключаться между ними не прерывая поток. Это дает возможность создания пред настроенных сцен для решения разного рода задач. Например, в процессе генерации потока видеосвязи нескольких людей можно использовать разные сцены, отвечающие за композицию разного количества входящих потоков. Возможность динамической смены сцен и вариантов композиции отсутствует, поэтому переключаться между ними придется вручную.

В случае, если мы захотим создать виртуальную аппаратную, в которой будет присутствовать возможность генерации сцен удаленно или посредством нескольких людей, нам придется создавать поток на каждый необходимый сценарий по отдельности и публиковать на промежуточный сервер. Таким образом будет возможность выбора потока, который генерируется отдельным человеком, который размещен на отдельном от остальных потоков адресе, с возможностью смены данного потока другим у любой момент времени.

Данный подход, очевидно, накладывает накладные расходы. Так, например, чтобы иметь возможность выбирать из 3 сцен, нам придется кодировать и декодировать данные как минимум 4 раза, что в 4 раза больше в сравнении с решением, которое предоставило бы возможность не использовать отдельные сервера, а динамически переключаться, используя только один набор входных данных.

1.2.2.2 XSplit

XSplit Broadcaster и XSplit Gamecaster – ПО, предназначенное для трансляции видеопотоков, генерируемых преимущественно посредством захвата

окна компьютерной игры. Данное ПО заточено под геймеров и позволяет начать трансляцию буквально одной кнопкой вне зависимости от вида ЭВМ – будь то персональный компьютер или игровая консоль. Для изменения битрейта или других настроек останавливать данное решение не требуется. Из минусов можно отметить отсутствие использования данного по на операционных системах кроме Windows, а также малое количество плагинов.

Композиция сцены также присутствует, но обладает меньшей гибкостью из-за меньшего количества плагинов.

1.2.3 Программы для видеоконференции

1.2.3.1 Google Meet

Google Meet – решение, которое предоставляет возможность участия в аудио- и видеоконференциях, а также предоставляет возможность мгновенного обмена сообщениями. Решение основано на стандарте WebRTC и не требует стороннего ПО. Примечательно тем, что использует скрытые для широкого пользования возможности WebRTC, являясь своего рода входной точкой для новых возможностей данного стандарта в широкое использование. Так, например, именно в Google Meets впервые были использованы возможности по захвату экрана при помощи функций, описанных стандартом WebRTC.

1.2.3.2 Zoom

Zoom, как и многие другие решения, предоставляет возможность участия в видео конференциях, причем приложение обеспечивает возможности использования HD видеопотока и подключения до 100 участников к одной конференции в бесплатной версии. Среди минусов можно выделить обширное количество уязвимостей безопасностей (например, отсутствие сквозного

шифрования), а также ограничение длительности конференции в 40 минут для бесплатной версии.

Данное решение интересно в рамках данной работы тем, что здесь присутствует минимальные возможности управления сценой, хоть таковые возможности и локальны. В Zoom возможно перемещать видео блоки с участниками по своему усмотрению, а также возможно управлять входным медиа потоком (включать и выключать аудио и видео), а также наличием залов, которые помогают разбивать участников на изолированные группы.

Касательно композиции, в Zoom существуют фильтры видео, которые могут изменять фон позади участника или накладывать объекты поверх участника (например, шляпа в качестве головного убора или усы), хотя практической пользы от этого в рамках данной работы нет.

1.3 Выводы

После анализа сценариев доставки видео контента была составлена таблица 1, приведенная ниже, которая отображает ключевые особенности каждого сценария. По результатам анализа были выделены интересующие в рамках данной работы сценарии:

- видеосвязь, конференции,
- стриминг.

Данные сценарии были выделены так как только в них присутствует генерация потока клиента, во время которой возможно применение композиции сцены.

Таблица 1 – результаты анализа сценариев поставки меди потока

	Клиент	Сервер	Генерация потока клиентом
Просмотр видео в интернете	Пользователь сети интернет	Специализированное ПО/ЭВМ	Нет
Видеосвязь, конференции	Пользователь сети интернет	Пользователь сети интернет Специализированное ПО/ЭВМ	Да
Трансляции с веб-камер	Пользователь сети интернет Другой сервер	Веб-камера	Нет
Стриминг	Пользователь сети интернет	Специализированное ПО/ЭВМ	Да

После анализа решений, использующих интересующие нас сценарии, была составлена таблица 2, приведенная ниже, в которой отображены ключевые особенности каждого решения.

Среди рассмотренных решений наиболее близким к обеспечению поставленной цели является OBS с использованием плагинов. Однако данное решение не является оптимальным, так как требует внешних серверов либо для подключения к ним и сбора данных, либо для публикации одной из возможных композиций для обеспечения возможности удаленного администрирования каждой из них.

Можно отметить, что использовать композицию сцен в рамках видеоконференций возможно, при условии композиции сцены программным энкодером и генерации виртуальной камеры, на основе которой будет генерироваться поток для видеосвязи.

Таблица 2 – анализ решений, использующих интересующие нас сценарии

	Тип сценария	Особенности	Возможности композиции сцен
OBS	стриминг	большое количество плагинов	присутствуют
XSplit	стриминг	ориентированность на геймеров	присутствуют
Google Meets	видеосвязь	ранний доступ к WebRTC функциям	отсутствуют
Zoom	видеосвязь	до 100 (500 в платной версси) участников	несущественные

По результатам анализа существующих сценариев и сцен, было выявлено отсутствие возможности динамической композиции транслируемого видеопотока под управлением нескольких пользователей, что подтверждает целесообразность данной работы.

2 СРЕДСТВА ОРГАНИЗАЦИИ ПЕРЕДАЧИ ПОТОКОВОГО ВИДЕО

2.1 Средства композиции сцены

2.1.1 Вручную

В процессе обработки видео потока, получив сырые, или как их еще называют, декодированные данные, можно получить доступ к данным о пикселях на кадре посредством вызовов соответствующих методов используемого API.

Существует несколько вариантов расположения битов и их значения для одного пикселя. Наиболее распространенные варианты это RGB и I420. На рисунке 6 приведен приме организации порядка байтов в формате RGB

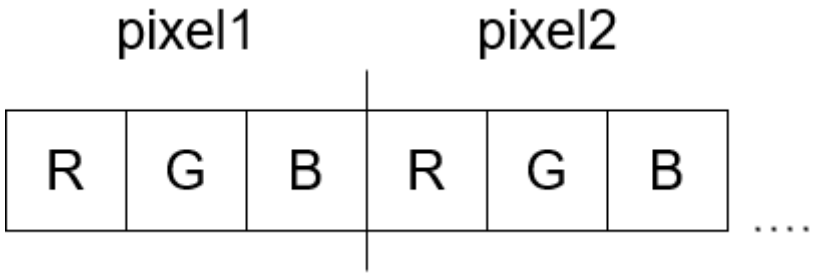

Рисунок 6 –пример порядка байтов в формате RGB

Зная то, как расположены биты для соответствующих пикселей, и что они означают, можно с легкостью манипулировать ими:

- масштабировать,
- изменять размер добавляя или обрезая граничные пиксели,
- заменять пиксели другими с целью создания наложения видео компонентов друг на друга

Для наглядности, рассмотрим один из методом манипуляции над данным кадром. В качестве метода манипулирования изображением был выбран метод масштабирования «метод ближайшего соседа» [3].

На рисунке 6 приведен пример работы метода масштабирования. Из изначального кадра 4 на 4 пикселя, мы получаем новый размером 8 на 8. Черным цветом помечены новые пиксели, которые появятся в кадре.

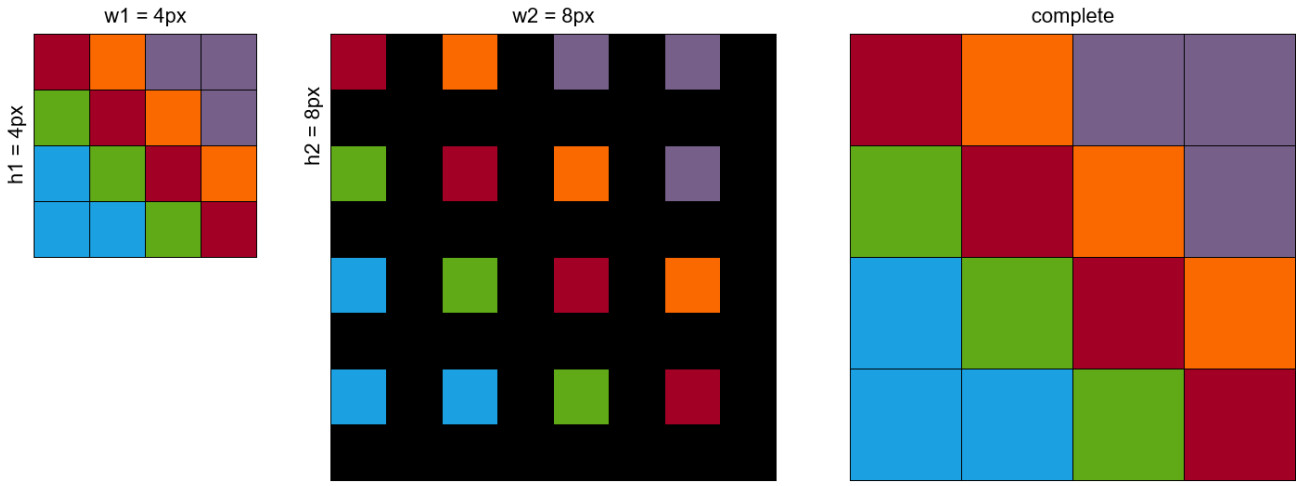

Рисунок 7 – иллюстрация работы метода ближайших соседей

Ниже приведен пример кода 1, в котором описан процесс масштабирования кадра методом ближайшего соседа.

Пример кода 1 — пример метода масштабирования кадра методом ближайшего соседа на языке Java

```
public int[] resizePixels(byte[] pixels,int w1,int h1,int w2,int h2) {
    byte [] temp = new int[w2*h2] ;
    double x_ratio = w1/(double)w2, y_ratio = h1/(double)h2;
    double px, py;
    for (int i=0;i<h2;i++) {
        for (int j=0;j<w2;j++) {
            px = Math.floor(j*x_ratio) ;
            py = Math.floor(i*y_ratio) ;
            temp[(i*w2)+j] = pixels[(int)((py*w1)+px)] ;
        }
    }
    return temp ;
}
```

2.1.2 Использование библиотек

В целом, подход в операциях изменения размера и замены пикселей кадра схожи с подходом из примера указанного выше. Единственное отличие заключается в том, что нет необходимости самим работать с пикселями, ведь ошибиться в способе организации байтов очень легко, а понять в чем ошибка бывает достаточно сложно.

2.2 Методы доставки видео контента

2.2.1 Протоколы управления потоком, передачи и описания данных

В общем случае протоколы управления потоком описывают то, как два участника видеосвязи будут обмениваться данными с целью получения данных. В рамках обмена медиа контентом подразумевается реализация хотя бы команды Play, но иногда встречаются и расширенные варианты: пауза, стоп, продолжение и другие. Примерами таких протоколов являются RTSP, RTMP.

Протоколы передачи или транспортные протоколы описывают то, как сжатое видео упаковывается в так называемый контейнер, отвечающий за способ организации передачи данных, который далее будет отправлены от одного участника другому. Примерами таких протоколов являются RTMP, RTP.

Стоит заметить, что RTMP может выполнять функции двух других протоколов: RTP + RTSP.

Протокол описания данных предназначен для описания сессии и потоковых данных, которые передаются в ее контексте. Примером такого протокола является SDP. Сессия SDP может описывать потоки данных, среди которых могут быть аудио- и видеоданные, а также данные управления и поточные приложения. В общем случае SDP сообщение содержит в себе

- адрес места назначения,
- номера UDP портов отправителя и получателя,
- описание медиа форматов (кодеки),
- описание атрибутов сессии (например, дополнительные атрибуты кодека или режим приема-передачи),
- в случае, если сообщение описывает не прямую трансляцию может содержать время старта и остановки.

2.2.2 WebRTC

WebRTC – это стандарт, который описывает то, как необходимо организовывать передачу потоковых аудио- и видеоданных, а также данных произвольного типа (например, текста) между браузерами или другими поддерживающими его приложениями в режиме реального времени без обязательного использования посредников [4]. Ключевой особенностью является отсутствие необходимости в стороннем ПО: набор стандартов, которые включает в себя WebRTC, позволяет проводить пиринговые конференции и обмен данными без необходимости пользователю устанавливать плагины.

Эта особенность возможна благодаря набору функций Adapter.js, которые обеспечивают бесшовную стыковку различных браузеров и приложений.

WebRTC работает поверх протоколов RTP/RTCP.

Ключевым звеном в стандарте WebRTC является PeerConnection. Именно этот API позволяет устанавливать пиринговое соединение между браузерами. Настроить общение между оконечными пользователями довольно просто. Необходимо лишь передать сообщение о намеренности одного пользователя начать общаться с другим. Реализуется это в большинстве случаев при помощи простых сторонних серверов, часто называемых сигнальными, задача которых

состоит лишь в предоставлении возможности пользователям отправлять и получать сообщения.

На рисунке 8 представлена вариация того, как может выглядеть общение между двумя пользователями. Упрощенная (без использования STUN и TURN серверов) же последовательность действий для инициализации обмена медиа данными такова:

1. PeerA хочет инициировать общение с пользователем PeerB,
2. PeerA посылает SDP сообщение Offer, то есть описание, включающее всю информацию о предлагаемой конфигурации вызывающего абонента для соединения,
3. PeerB в может согласиться или отказаться инициировать сообщение,
4. В случае, если PeerB хочет инициировать сообщение, он посылает SDP сообщение Answer, аналогичное сообщению Offer.
5. После этого пользователи начинают обмениваться так называемыми IceCandidate сообщениями, в которых заключается информация о сетевом соединении (Internet Connectivity Establishment) которая может быть использована для установки соединения. В данном сообщении описываются протокол и способ взаимодействия.

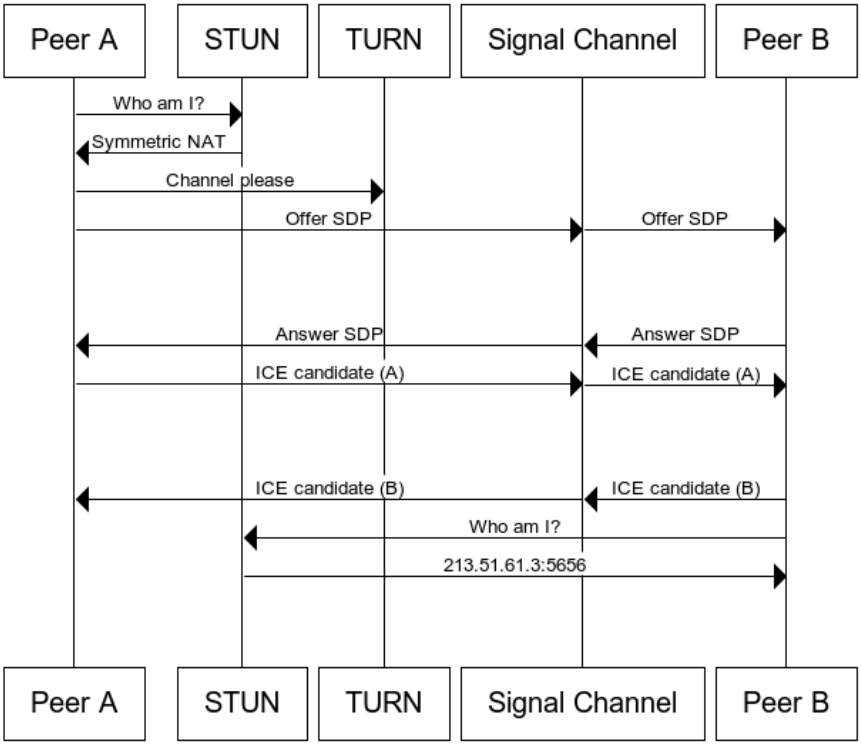

Рисунок 8 – Пример общения между двумя пользователями WebRTC

2.2.3 Технологии адаптивной потоковой передачи данных передачи данных

Основной концепт данных технологий заключается в разбиении контента на последовательность небольших файловых сегментов, каждый из которых содержит небольшой отрывок содержимого. Данные сегменты могут быть описаны разными способами, например при помощи m3u8-плейлистов или MPD-манифеста в виде XML-документа. Основные представители данной категории это HLS и MPEG-DASH.

HLS строится поверх HTTP, по которому передаются MPEG-TS контейнеры, в которых содержатся сжатые закодированные данные, с использованием H264-видео [5] и AAC- или MP3-аудио. Он был разработан в

компании Apple, в связи с чем изначально работал только в браузер Safari под управлением операционной системы семейства Mac. Это создало сложности в использовании стандарта сторонними производителями. Так, например, были реализованы собственные способы передачи разных аудиодорожек, из-за чего теперь приходится отдавать разные форматы плейлистов для разных плееров.

MPEG-DASH обычно строится с использованием контейнеров MP4 и H264/H265-видео и AAC-аудио внутри него или с контейнером WEBM и VP/VP9 видео внутри, хотя строгой привязки к технологиям нет. MPEG-DASH хорош тем, что в большинстве браузеров работает нативно, без использования стороннего ПО.

2.3 Библиотеки кодеков для транскодирования

2.3.1 libav (FFmpeg)

С помощью FFmpeg можно выполнять большое количество задач по обработке видео: кодирование, декодирование, транскодирование, мультиплексирование и демультиплексирование. Использование данной программы очень просто — FFmpeg представляет из себя консольное приложение, а процесс работы над видео задается при помощи флагов и параметров.

Библиотека libav предоставляет кроссплатформенные инструменты и API для конвертирования, манипуляции и передачи потокового медиа в широком спектре форматов по распространенным протоколам для программного кода. Библиотека libav является ответвлением программы FFmpeg и разрабатывается отдельно.

В общем случае процесс работы с видео сводится к общему шаблону, который представлен на рисунке 9.

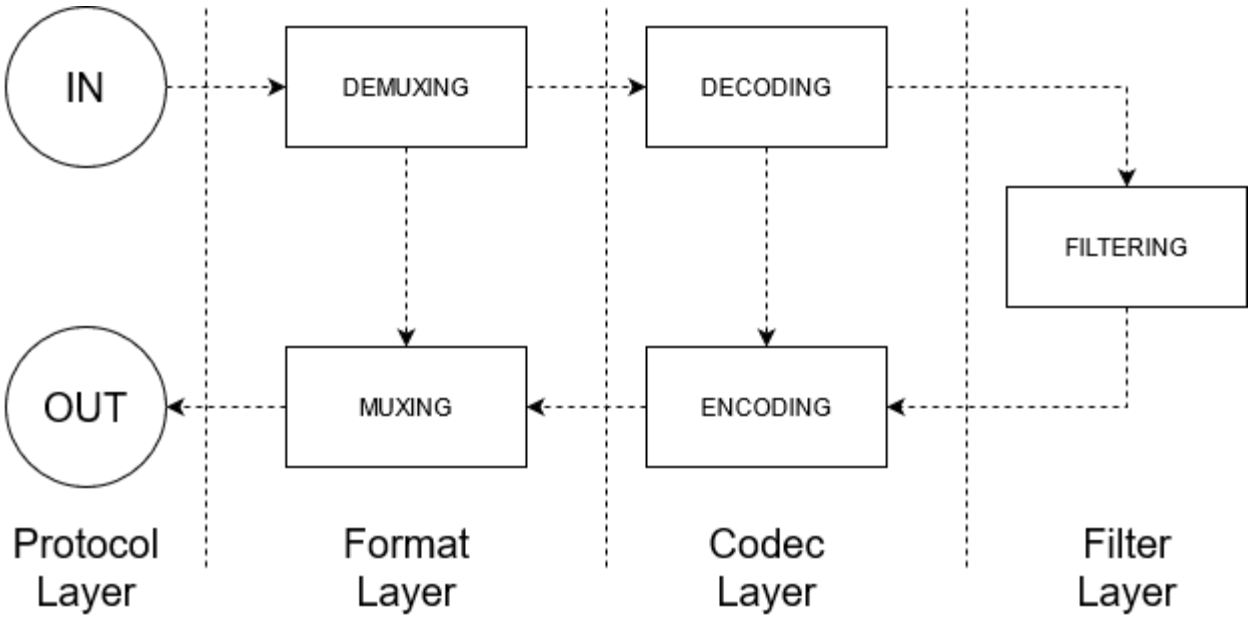

Рисунок 9 – Шаблон рабочего процесса работы с видео в libav

Protocol Layer оперирует, но не ограничивается протоколами HTTP, RTMP. Вместо оперирования протоколами допустимо оперирование файлами или сырыми данными из приложения.

Format Layer оперирует, но не ограничивается форматами видео потока MKV, MP4.

Codec Layer оперирует, но не ограничивается видами кодировки AV1, VP9, H264.

Минусом libav можно отметить, что с развитием и выпуском новых версий API старые ее версии могут быть помечены как устаревшие, что может вести к модифицированию кода, который использовал старый API.

2.3.2 vlclib

vlclib – это главное ядро и интерфейс на котором основан фреймворк VLC Media Player. Под libvlc написано большое количество плагинов, которые могут быть загружены во время выполнения кода.

Данная библиотека поддерживает все распространенные форматы файлов, а также все кодеки и протоколы вещания медиа. Имеет API для всех популярных языков программирования.

Тем не менее, шаблон процесса работы с видео не отличается от шаблона libav, и предоставляет ту же функциональность.

2.3.3 gstreamer

Gstreamer – мультимедийный фреймворк, который использует собственную систему объектов GObject, является основой для многих мультимедийных приложений от видеоредакторов до потоковых серверов и медиа проигрывателей [6].

Данный фреймворк позволяет создавать так называемые pipeline-ы (далее конвейер), которые позволяют гибко работать с видео, изменять его, передавать и принимать. Конвейер выстраивается из элементов GObject, каждый из которых может изменять поступающие в него данные и передавать на обработку следующему элементу. Основной смысл конвейера заключен в инкапсуляции работы над данными. Все, что интересует пользователя в момент работы с конвейером – его состав и порядок элементов внутри него.

Элементы GObject связаны с программным кодом, который обрабатывает данные определенным образом. Существует огромное количество реализаций данного класса под любые задачи: будь то передача сырых кадров из приложения в конвейер, разветвление данных или отправка данных на удаленный сервер. Если есть какая-то стандартная задача работы с видео данными под нее скорее всего есть готовое решение в виде элемента GObject.

На рисунке 10 представлен пример простого конвейера. Как видно на рисунке у каждого элемента имеется два шлюза:

- src – входная точка данных в элемент
- sink – выходная точка данных из элемента

Можно заметить, что у конвейера нет шлюзов вовсе, а у оконечных элементов отсутствует по одному. Это обусловлено назначением этих шлюзов, а именно передачей данных от одного элемента к другому.

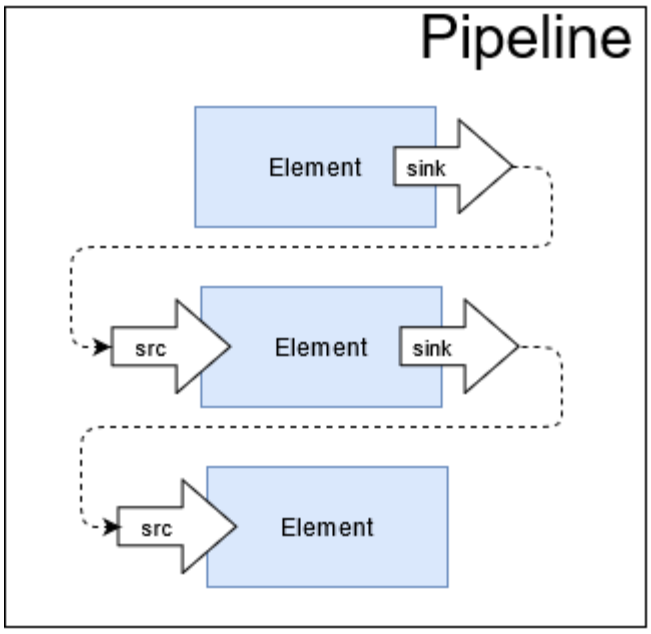

Рисунок 10 – Простой пример конвейера в фреймворке Gstreamer

Помимо вышеперечисленного, можно отметить, что помимо API, Gstreamer поставляет консольное приложение, которое позволяет создавать и управлять контейнерами без необходимости использования API. Данное приложение называется gst-launch и используется гораздо чаще в сравнении с API, исходя из количества найденных обсуждений в открытом доступе сети интернет.

2.4 Выводы

По результатам анализа средств организации передачи потокового видео был определен набор инструментов, которые будут задействованы в реализации решения. В качестве ядра программы решено использовать фреймворк Gstreamer

из-за его универсальности и гибкости. Так как решение, разрабатываемое в рамках работы, должно быть максимально гибким и иметь возможность собирать медиа потоки из всевозможных источников, среди которых могут быть и источники видеоконференцсвязи, конкуренты фреймворка Gstreamer отпадают. Как упоминалось в ходе анализа, во фреймворке Gstreamer существуют реализации элементов на любую стандартную задачу. К сожалению, реализация конференцсвязи не является стандартной проблемой. Среди доступных плагинов не было найдено такого, который позволил бы получать медиа поток от каждого нового участника видеосвязи, в связи с чем возникает необходимость в удобном способе организации видеосвязи, у которого есть простой и надежный API, который желательно был бы широко используемым и унифицируемым. Среди всех проанализированных решений только WebRTC подходит под эти условия. Таким образом были определены используемые решения для данной работы, основанное на комбинации двух технологий:

- Gstreamer – для удобного манипулирования видео,
- WebRTC – для поддержки видеоконференцсвязи.

3 РАЗРАБОТКА РЕШЕНИЯ ДЛЯ СОВМЕСТНОГО ВИДЕО СТРИМИНГА

3.1 Ядро решения — Gstreamer

Как было описано в подразделе выводов предыдущего раздела, ядром разрабатываемого решения служит Gstreamer. Что примечательно, так это то, что для некоторых сценариев манипуляции с видео нам даже не потребуется использовать API. Примером может быть сценарий, в котором данное решение выступает в качестве посредника и передает медиапоток с RTSP сервера на RTMP сервер, при этом возможны манипуляции с видео, в том числе изменение его кадров. В примере кода 2 приведена консольная команда для создания подобного конвейера.

Пример кода 2 — консольная команда для создания простого конвейера

```
gst-launch-1.0 \
      rtspsrc location="rtsp://localhost/input" protocols=tcp latency=0 ! \
      rtph264depay ! \
      h264parse ! \
      x264enc ! \
      flvmux streamable=true ! \
      rtmpsink location="rtmp://localhost/output live=true"
```

Исходя из вышеуказанного примера, имеем следующий медапоток:

1. данные поступают в конвейер из RTSP сервера расположенного по адресу rtsp://localhost/input, при этом медипоток укомлектован в RTP контейнер,
2. далее данные извлекаются из контейнера RTP, при этом медапоток определяется форматом H264,
3. из формата H264 мы получаем сырые данные,
4. медиапоток преобразуется из формата сырых данных в формат H264,
5. медипоток упаковывается в контейнер FLV,

6. медипоток поставляется на сервер RTMP, расположенный по адресу rtmp://localhost/output, при этом указывается флаг того, что данные медапоток является прямой трансляцией, то есть не имеет заранее известного момента окончания.

Именно в момент между 3 и 4 шагом, когда у нас имеются сырые данные мы можем изменять их как нам захочется. Для таких целей есть специальный элемент videofilter, при помощи которого можно, например, добавит эффект сепии к видео.

Можно заметить, что в данном примере все элементы разделены восклицательным знаком — это особенность построения конвейеров при помощи консольной программы. Здесь все элементы расположены в таком порядке, в котором будет протекать медиапоток. Следует понимать, что каждый элемент принимает на вход только определенный вид данных. Так, например, нельзя подать сразу сырые данные на вход элементу rtmpsink, который отправляет данные на RTMP сервер – эти данные нужно упаковать в flv контейнер, который как раз таки принимает элемент rtmpsink.

Несмотря на то, что Gstremer можно использовать без API, данный подход не интересен в рамках данный работы. То, что действительно нам интересно это использования API в качестве src- или sink-элемента в конвейере Gstreamer.

На практике процесс работы над медапотоком в фреймворке Gstreamer очень похож на процесс работы упомянутый в подразделе, в котором описывалась библиотека libav:

- также присутствуют входные и выходные токи,
- присутствуют фазы, во время которых медапоток помещается и извлекается из контейнеров,
- присутствуют фазы кодирования и декодирования,

- а также фаза работы с сырыми данными.

Отличие процесс работы в фреймворке Gstreamer заключается в том, что данный процесс может иметь множество ответвлений и соединений на любой из этих фаз. Например, в конвейере может иметься множество входных точек в виде элементов appsrc, далее эти потоки будут сливаться в единый поток, который затем отправится на единственный выходной элемент rtmpsink. Или же наоборот имеющийся единственный входной элемент rtmpsrc будет продублирован на множество выходных элементов appsink. Примеры реализации таких сценариев будут приведены в следующем разделе

Для понимания того, как может использоваться Gstreamer API, приведем пример создания конвейера, вид которого представлен на рисунке 11. В данном конвейере присутствуют:

- входной элемент appsrc, который является входной точкой в конвейер,
- далее видео данные поступают в элемент videiconvert, который обеспечивает гибкость в обработке данных вне зависимости от вида представления пикселей, например для одинаковых сценариев как RGB, так и I420 пикселей,
- затем видеопоток поступает в элемент autovideosink, который выбирает выходную точку из конвейера основываясь на типе поступивших данных, но так как в данном конвейере отсутствуют определения каких-либо выходных точек, будет использован стандартный вывод видеопотока, который будет отображен во всплывающем окне.

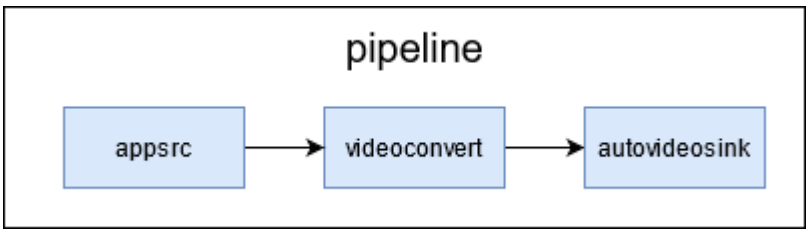

Рисунок 11 – Пример конвейера, в котором входной элемент представлен в виде appsrc

Следует заметить, что стандартный Gstreamer API разрабатывался для языка C/. Методы и классы, рассматриваемые здесь и далее во всей работе предоставляются оберткой GstSharp для языка C# и программной платформы .NET [7] над стандартным API Gstreamer.

Рассмотрим пример создания конвейера, описанного выше на примере кода 3.

Пример кода 3 — пример создания конвейера

```
public static void Main(string[] args) {
    Gst.Application.Init();
    pipeline = new Gst.Pipeline("pipeline");
    var appsrc = Gst.ElementFactory.Make("appsrc", "source");
    var videoconvert = Gst.ElementFactory.Make("videoconvert", "videoconvert");
    var sink = Gst.ElementFactory.Make("autovideosink", "sink");
    var caps = Gst.Global.CapsFromString("video/x-
raw, format=RGBA, width=640, height=480, framerate=30/1");
    appsrc.SetProperty("caps", new GLib.Value(caps));

    pipeline.Add(appsrc, videoconvert, sink);
    Gst.Element.Link(appsrc, videoconvert, x264enc, flvmux, queue1, rtmpsink);

    appsrc.SetProperty("stream-type", new GLib.Value(0));
    appsrc.SetProperty("format", new GLib.Value(Gst.Format.Time));
    appsrc.SetProperty("is-live", new GLib.Value(true));

    var needDataDelegate = new Action<object, EventArgs>(PushData);
    appsrc.AddSignalHandler("need-data", needDataDelegate);

    pipeline.SetState(Gst.State.Playing);
}
```

Для начала необходимо создать конвейер и все элементы, которые будут участвовать в работе на видео. Конвейер создается при помощи инициализации

нового класса Gst.Pipeline, с указанием желаемого названия данного конвейера. Элементы создаются при помощи метода Make у статичного класса Gst.ElementFactory.

Затем необходимо определить свойства производимых нави кадров видео. Делается это при помощи создания элемента Gst.Caps и дальнейшего присвоения его к свойству caps элемента appsrc.

После того как элементы будут созданы, их необходимо поместить внутрь конвейера и свзять между собой. Важно иметь в виду, что добавлять и связывать элементы необходимо именно в том потоке, в каком должен идти ведеопоток.

Последним подготовительным действием является указание метода, который будет вызываться после того, как произойдет need-data, которое говорит о том, что элементу appsrc нужен новый видеокадр.

После данных действий все что нам остается сделать, чтобы данные начали попадать из одного крайнего элемента в другой, необходимо установить состояние конвейера равным Gst.State.Playing. Вожно отметить, что видеопоток не остановится в текущем конвейере, так как не установлен никакой сигнал, который смог бы остановить видеопоток.

Помимо рассмотрения создания конвейера имеет смысл привести пример кода 4, в котором описано то, каким образом данные будут помещаться во входной элемент конвейера appsrc.

Пример кода 4 — описание возможности отправки сырых кадров в элемент appsrc

```
private private void PushData(object o, System.EventArgs args)
    {
        if (!_seg)
        {
            _seg = true;
```

```
            _pad.PushEvent(Gst.Event.NewSegment(new Gst.Segment()
            {
                Rate = 30,
                AppliedRate = 30,
                Time = 0,
                Format = Gst.Format.Default,
                Duration = ulong.MaxValue
            }));
        }

        var data = GetData();
        var buffer = new Gst.Buffer(data);
        buffer.Duration = 33333333; //ns
        buffer.Pts = _time;
        _time += 33333333;
        appsrc.Emit("push-buffer", buffer);
        buffer.Dispose();
    }
```

В представленном примере присутствует сигнал, который говорит о том, что начинается новый сегмент данных. Данный сигнал нужен для корректной обработки поступающих данных.

Пример получения сырого GetData() кадра будет представлен в следующем подразделе.

После того, как мы подготовили новый сырой видеокадр, его необходимо обернуть в класс Gst.Buffer, который можно будет передать на вход элементу appsrc. При этом не следует забывать о необходимости назначения полям класса Duration и Pts значений, который говорят о том сколько следует показывать данный кадр в наносекундах и начиная с какого момента времени после начала воспроизведения соответственно.

Непосредственно сама передача сырого кадра в элемент производится прим помощи метода Emit, с указанием соответствующего названия события.

3.2 Обеспечение видеоконференцсвязи при помощи WebRTC

Как упоминалось в ходе анализа, данное решение позволяет устанавливать видеосвязь внутри браузера без необходимости пользователю в инсталляции сторонних плагинов. Рассмотрим пример создания соединения между двумя клиентами, один из которых является пользователем веб-приложения, интерфейс которого представлен на рисунке 12, а второй – нашим решением, который сможет использовать медиапоток первого внутри конвейера Gstreamer.

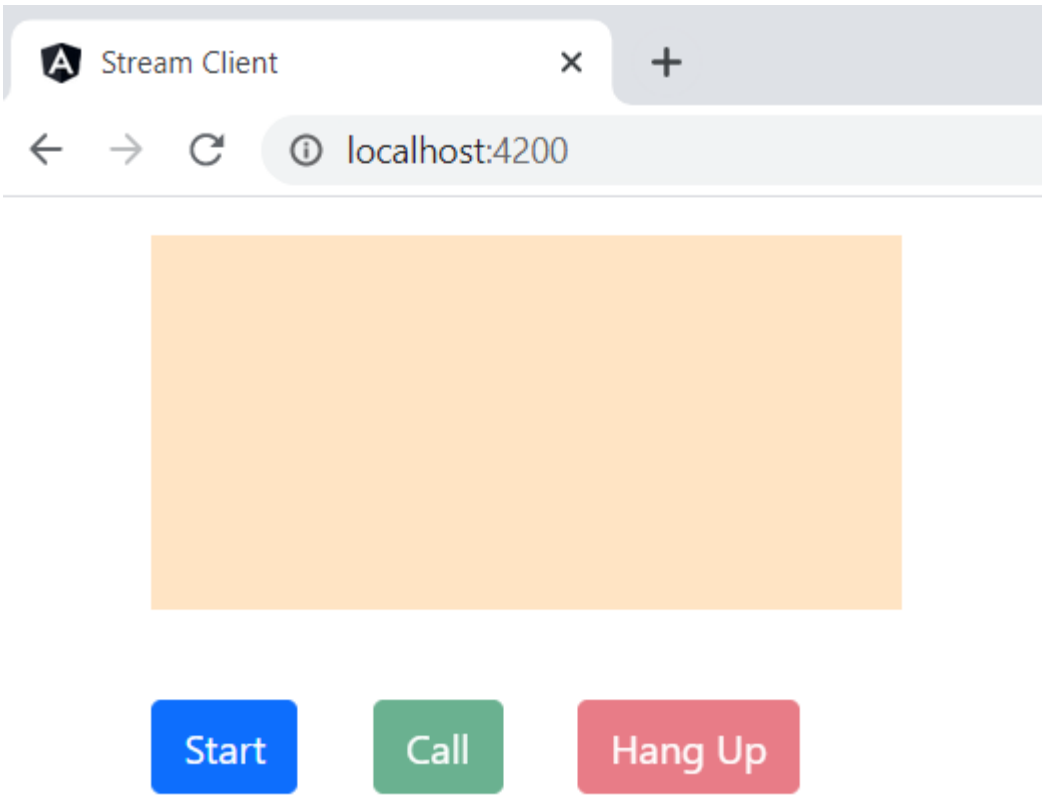

Рисунок 12 – Пример пользовательского интерфейса веб-приложения, которое выступает в пиринговой связи

Для начала приведем пример кода 5 на языке TypeScript, в котором будет рассмотрен способ получения медиаданных пользователя.

Пример кода 5 — Получение медиаданных пользователя

```
  async start() {
    this.setupWebSocket();
    console.log('Requesting local stream');
    this.startButton.disabled = true;
    try {
      const stream = await navigator.mediaDevices.getUserMedia({audio: true, video:
 true});
      console.log('Received local stream');
      this.localVideo.srcObject = stream;
      this.localStream = stream;
      this.callButton.disabled = false;
    } catch (e) {
      console.log(e);
      alert(`getUserMedia() error: ${e.name}`);
    }
  }
```

В данном примере данные доступны благодаря классу навигатор, который предоставляет adapter.js. Данный метод вызывается после нажатия на кнопку Start. После этого пользователю становится доступна для нажатия кнопка Call, при нажатии на которую происходит установления соединения со вторым участником связи. Ниже представлен пример кода 6, который описывает процесс установления соединения.

Пример кода 6 — инициализация пиринговой связи

```
async call() {
    this.callButton.disabled = true;
    this.hangupButton.disabled = false;
    console.log('Starting call');
    this.startTime = window.performance.now();
    const videoTracks = this.localStream.getVideoTracks();
    const audioTracks = this.localStream.getAudioTracks();
    const configuration: RTCConfiguration = {
      iceServers: [
        {
          urls: "stun:stun.l.google.com:19302"
        }
      ]
    };
    this.pc1 = new RTCPeerConnection(configuration);
    this.pc1.addEventListener('icecandidate', e => this.onIceCandidate(this.pc1, e)
);
    this.pc1.addEventListener('iceconnectionstatechange', e => this.onIceStateChang
e(this.pc1, e));
```

```
    this.localStream.getTracks().forEach(track => this.pc1.addTrack(track, this.loc
alStream));

    try {
      console.log('pc1 createOffer start');
      const offer = await this.pc1.createOffer(this.offerOptions);
      await this.onCreateOfferSuccess(offer);
    } catch (e) {
      this.onCreateSessionDescriptionError(e);
    }
  }
```

В данном примере описывается создания объекта PeerConnection, который отвечает за отправку и доставку медиаданных, а также за установку канала связи между двумя клиентами. Инициализация соединения происходит после создания SDP пакета Offer, описание создание которого также приведено в данном примере.

После успешного создания Offer-а необзходимо отправить его на сигнальный сервер. В нашем случае в его роли выступает веб-сокет.

После того, как другой клиент получит пакет Offer, он сформирует аналогичный пакет Answer, после чего начнется пиринговая связь.

В случае, если пользователи не знает значение своего публичного ip, необходимо воспользоваться услугами STUN сервера, который предоставит соответствующую информацию пользователю. Пример использования сервера STUN представлен на рисунке 13.

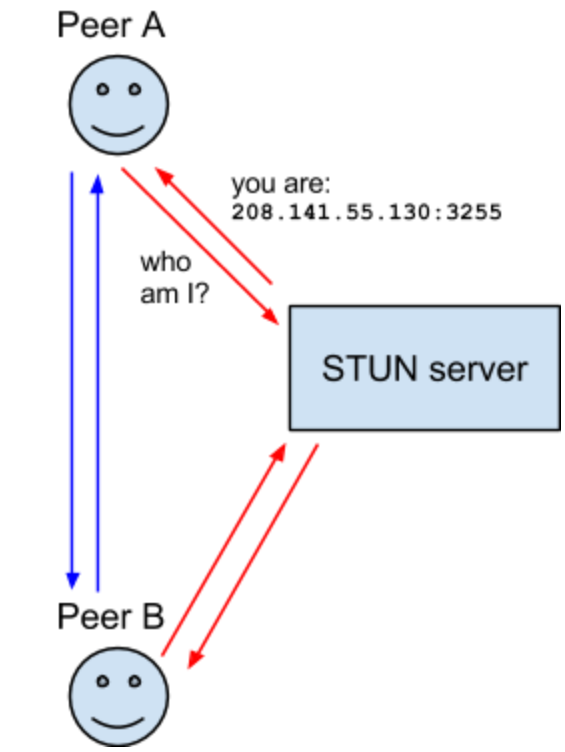

Рисунок 13 – пример использования сервера STUN

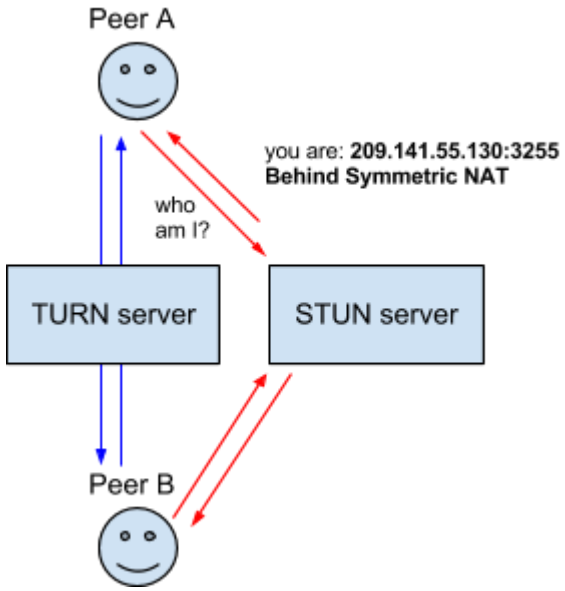

Рисунок 14 – пример использования сервера TURN

В случае если клиенты расположены не в одной локальной сети, необходимо воспользоваться услугами Turn сервера, который позволит преодолеть ограничения накладываемые Symetric NAT. В данном примере сервер

TURN не используется. Пример использования сервера TURN представлен на рисунке 14.

Далее рассмотрим то, как можно получать доступ к сырым из объекта PeerConnection в рамках второго клиента пиринговой связи, и дальнейшую их передачу в конвейер Gstreamer.

Получение кадров доступно из объекта PeerConntection. В данном объекте присутствует событие, которое срабатывает при получении нового кадра. Пример получения сырого кадра представлен в примерах кода 7 и 8.

Пример кода 7 — установка обработчика события получения нового кадра

```
pc.VideoTrackAdded += (RemoteVideoTrack track) =>
{
    Console.WriteLine("Track added");
    track.I420AVideoFrameReady += HandleFrame;
};
```

Пример кода 8 — обрабочик события получения кадра

```
public void HandleFrame(I420AVideoFrame frame)
{
    int pixelSize = (int)frame.width * (int)frame.height;
    int byteSize = (pixelSize / 2 * 3); // I420 = 12 bits per pixel
    byte[] frameBytes = new byte[byteSize];
    frame.CopyTo(frameBytes);
    rgbBytes = new byte[pixelSize * 3];
    Console.WriteLine($"WebRtc frame received of {frame.width}x{frame.height}");

    _queue.Enqueue(frameBytes);
}
```

После данных манипуляций кадры будут поступать в очередь, из которой их можно забирать, когда произойдет событие need-data в элементе appsrc. Данные действия приведены на примере кода 9.

Пример кода 9 — пример забирания кадров из очереди

```
byte[] GetData() { return _queue.Count > 2 ? _queue.Dequeue() : _queue.Peek();}
```

4 РЕЗУЛЬТАТЫ РАЗРАБОТКИ

4.1 Сценарий использования WebRTC – RTMP

Для простоты описания здесь и далее будем использовать термин «конвейерное приложение», который означает то приложение разработанное исходя из нашего решения, то есть объединяющее внутри себя конвейер Gstreamer и клиента WebRTC.

Данный сценарий представляет из себя процесс генерации видеопотоков у 3-х различных пользователей веб-приложения. Веб-приложение при этом отправляет сгенерированные медиа-потоки по видеосвязи другому клиенту, которым во всех случаях является конвейерное приложение. Приложение, получая медиаданные от каждого пользователя, передает их в конвейер, где далее все медиапотоки будут совмещены в один единственный, который далее будет направлен в rtmp сервер.

Конвейерное приложение конфигурирует конвейер и WebRTC соединения исходя из переданного ей на вход XML файла, в котором описываются все участники. Пример такого XML файла приведен в примере кода 10.

Пример кода 10 — XML файл описывающий участников соединения

```
<participators>
    <participator name="participator-1" />
        <video-props>
            <framerate>
                30
            </framerate>
            <width>
                640
            </width>
            <height>
                480
            </height>
        </video-props>
        <canvas-props>
            <x-pos>0</xpos>
            <y-pos>0</ypos>
```

```
        <canvase-props>
    </participator>
    ...
</participators>
```

Под каждого пользователя добавляется web-socket адрес, который будет прослушивать конвейерное приложение, исходя из имени пользователя.

Так же данные из файла XML необходимы для создания элементов внутри конвейера и их уникальной идентификации. Схема расположения элементов внутри конвейера приведена на рисунке.

В примере кода 11 приведен пример инициализации конвейерного приложения учитывая XML файл конфигурации, которое реализует такой сценария.

Пример кода 11 — инициализация конвейерного приложения

```
static async Task Main(string[] args)
{
    var config = GetConfig(args[0]);
    CancellationTokenSource source = new CancellationTokenSource();
    CancellationToken token = source.Token;
    List<Task> tasks = new List<Task>();
    TaskFactory factory = new TaskFactory(token);
    foreach(var participator  in config.GetParticipators()) {
        tasks.Add(factory.StartNew((Participator participator) => {
            var svcAddress = $"http://127.0.0.1/{participator.GetName()}/";
            using (var interactor = new GstInteractor(participator))
            using (var svc = new WebSvc(svcAddress))
            {
                svc.OnWebSocketConnection += ws =>
                {
                    var signaller = new MyWsSignaller(ws);
                    var pipeline = new Conv();
                    var receiver = new MyWebRtcStreamReceiver(signaller);
                    receiver.OnFrameReceived += frame =>
                    {
                        interactor.HandleFrame(frame);
                        if (!working)
                        {
                            working = true;
                            interactor.Interact(participator, pipeline.AddBin, fals
e);
                        }
```

```
                    };
                    receiver.Start();
                };
                while (!token.IsCancellationRequested);
            }
        }, token));
    }
    ...
}
```

На примере кода 12 приведен пример инициализации bin-а под каждого пользователя исходя из параметров, переданных конвейерному приложению.

Пример кода 12 — инициализация bin-а под каждого пользователя

```
public void Interact(Participator participator, Delegate addBin,  bool waitForEnd =
 true)
        {
            appsrc = Gst.ElementFactory.Make("appsrc", $"source-
{participator.GetName()}");
            var videoconvert = Gst.ElementFactory.Make("videoconvert", "videoconver
t-{participator.GetName()}");

            var capsStr = $"video/x-raw, width={participator.GetWidth()}, " +
                                " height={participator.GetHeight()}, format=I420, "
 +
                                "framerate={participator.GetFramerate()}/1";
            var caps = Gst.Global.CapsFromString(capsStr);
            appsrc.SetProperty("caps", new GLib.Value(caps));
            var bin = new Gst.Bin(participator.Name);
            bin.Add(appsrc, videoconvert);
            Gst.Element.Link(appsrc, videoconvert);
            appsrc.SetProperty("stream-type", new GLib.Value(0));
            appsrc.SetProperty("format", new GLib.Value(Gst.Format.Time));
            appsrc.SetProperty("is-live", new GLib.Value(true));
            var needDataDelegate = new Action<object, EventArgs>(PushData);
            appsrc.AddSignalHandler("need-data", needDataDelegate);

            addBin.DynamicInvoke(bin);
        }
```

В данном примере на каждого пользователя создается новый bin, который по функциональности схож с конвейером за исключением наличия входных и выходных точек.

В примере кода 13 приведен пример инициализации конвейера.

Пример кода 13 — инициализация конвейера

```
public Conv()
        {
            _videomixer = Gst.ElementFactory.Make("videomixer", "mix");
            var queue = Gst.ElementFactory.Make("queue", "queue");
            var videoconvert = Gst.ElementFactory.Make("videoconvert", "videoconver
t");
            var x264enc = Gst.ElementFactory.Make("x264enc", "x264enc");
            var flvmux = Gst.ElementFactory.Make("flvmux", "flvmux");
            var queue1 = Gst.ElementFactory.Make("queue", "queue1");
            var rtmpsink = Gst.ElementFactory.Make("rtmpsink", "rtmpsink");
            flvmux.SetProperty("streamable", new GLib.Value(true));
            rtmpsink.SetProperty("location", new GLib.Value("rtmp://localhost/live"
));

            _pipeline.Add(_videomixer,  queue, videoconvert, x264enc, flvmux, queue
1, rtmpsink);
            Gst.Element.Link(_videomixer,  queue, videoconvert, x264enc, flvmux, qu
eue1, rtmpsink)
        }
```

В примере кода выше моджно заметить, что отправной точкой для данных, попадающих в конвейер, является элемент videmixer, который соединяет все потоки в один. Это утверждение не совсем верно. На самом деле, отправной точкой служит bin, из которого данные попадают в видеомиксер.

Стоит отметить, что несмотря на то, что мы создали все bin-ы и конвейер по отдельности, данные внутри них протекать не будут изз-за отсутствия связей между ними. В примере кода 14 приведен пример установления такой связи.

Пример кода 14 — связывание bin-а и конвейера

```
        public void AddBin(Gst.Bin bin, Participator participator) {
            _pipeline.Add(bin);
            var mixerSinkPadTemplate = _videomixer.GetPadTemplate("sink_%u");
            var mixerSinkPad = _videomixer.RequestPad(mixerSinkPadTemplate);
            mixerSinkPad.SetProperty("ypos", new GLib.Value(participator.GetYPos())
);
            mixerSinkPad.SetProperty("xpos", new GLib.Value(participator.GetXPos())
);
            var srcpad1 = videobox1.GetStaticPad("src");
            srcpad1.Link(mixerSinkPad);
        }
```

Результат работы данного сценария представлен на рисунке 15.

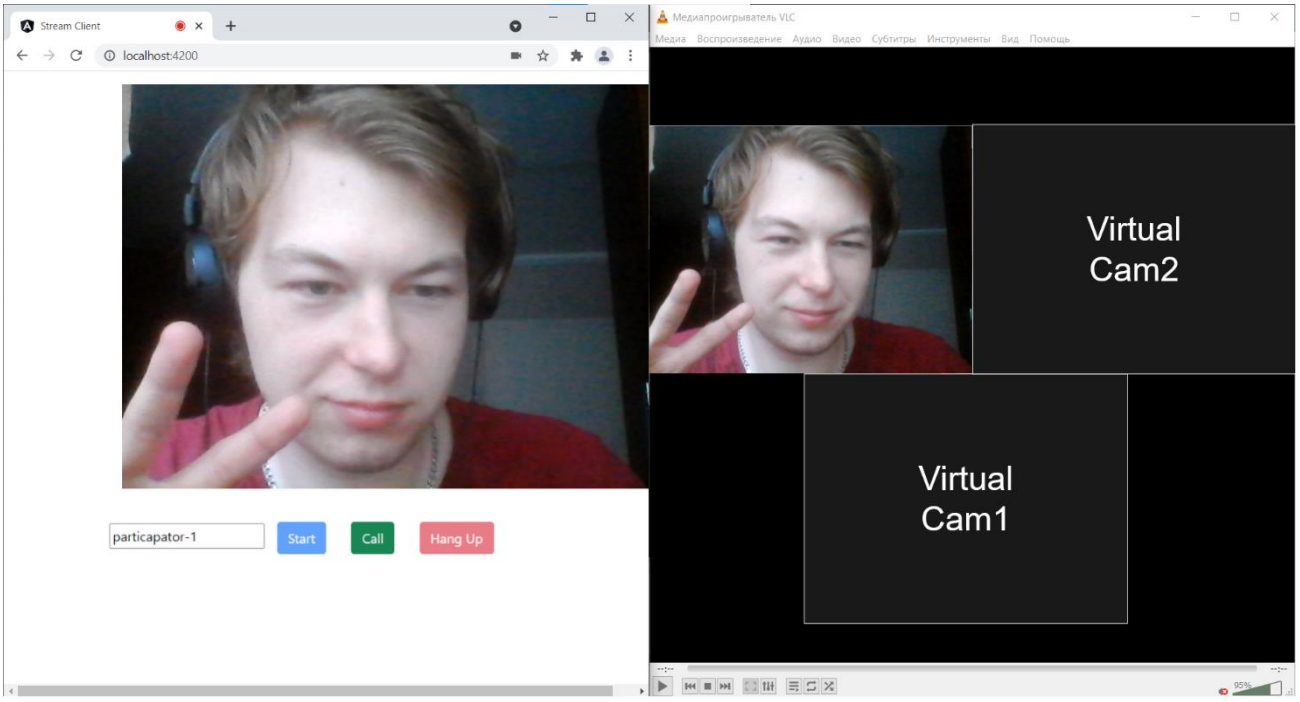

Рисунок 15 – результат работы сценария WebRTC – RTMP

Человек, представленный на изображении, является автором данной работы, то есть мной. К сожалению, количество физических камер в моем распоряжении было ограничено одной штукой, поэтому мне пришлось использовать виртуальные камеры, хотя это никаким образом не повлияло на роботы данного решения. Можно заметить, что кадр со мной, сгенерированный на стороне клиента, отличается от кадра со мной на стороне RTMP сервера. Это обусловлено принципом работы RTMP протокола. Дело в том, что перед тем, как видео можно будет отдать потребителю RTMP трафика, необходимо накопить некий буфер, который и будет отправлен потребителю. Накопление этого буфера сильно отличается, и может даже доходить до нескольких секунд.

4.2 Сценарий использования RTMP – WebRTC

Данный сценарий использования разработанного решения позволяет получать видеопоток из RTMP сервера, и затем направлять данный поток

каждому из 3-х пользователей, подключенных к конвейерному приложению посредство стандарта WebRTC.

В данном случае также используется файл конфигурации XML описанный в предыдущем подразделе, но используется он только с целью добавления нового адреса для прослушивания веб-сокетом с целью обеспечения общения каждой пары клиентов.

Отличается данный сценарий от предыдущего отличается очевидно направлением. Данное направление отражается на конвейере. На примере кода показано создание конвейера, направление которого обратно направлению конвейера из предыдущего примера.

Пример кода 15 — создание конвейера с другим направлением.

```
Gst.Application.Init();
_pipeline = new Gst.Pipeline("pipeline");

var rtmpsrc = Gst.ElementFactory.Make("rtmpsrc", "rtmpsrc");
var flvdemux = Gst.ElementFactory.Make("flvdemux", "flvdemux");
var avdec_h264 = Gst.ElementFactory.Make("avdec_h264", "avdec_h264");
var queue = Gst.ElementFactory.Make("queue", "queue");
var tee = Gst.ElementFactory.Make("tee", "tee");
flvdemux.SetProperty("streamable", new GLib.Value(true));
rtmpsrc.SetProperty("location", new GLib.Value("rtmp://localhost/live"));

_pipeline.Add(rtmpsrc, flvdemux, avdec_h264, queue, tee);
Gst.Element.Link(rtmpsrc, flvdemux, avdec_h264, queue, tee);
_pipeline.SetState(Gst.State.Playing);
```

Можно заметить, что теперь мы используем элемент rtmpsrc вместо rtmpsink, а также элемент tee вместо videomixer. Это обусловленно направлением потока данных и назначением элементов. Так, например, videomixer совмещал потоки в единый и мог при этом производить композицию сцены, элемент tee же только разветвляет поступающий в него поток. В данном сценарии необходимо связывать bin именно с элементом tee.

За исключением направления все другие стороны остаются похожими. Результат работы данного сценария приведен на рисунке 16.

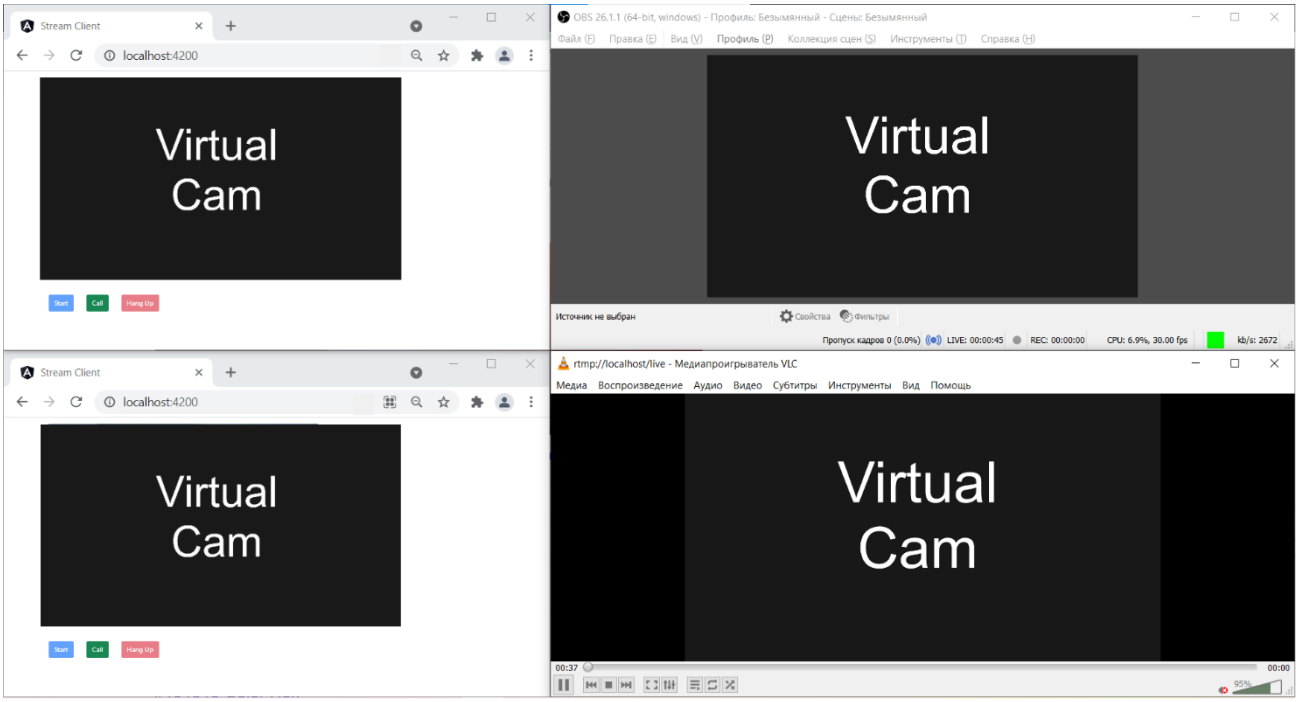

Рисунок 16 – результат работы сценария RTMP – WebRTC

На рисунке 16 видны следующие элементы:

- программа OBS, которая генерирует видеопоток, схожий с тем, что использовался в виртуальных камерах, и далее его публикует,
- проигрыватель vlc, который подключен к серверу, на который идет публикация видеопотока,
- 2 клиента веб-приложения, которое, к слову, отличается от использованного в предыдущем сценарии, которые получают данные от конвейерного приложения, общаясь с ним по стандарту WebRTC.

ЗАКЛЮЧЕНИЕ

В ходе данной работы были проанализированы существующие решения осуществляющие различные сценарии по доставке видео, среди которых не было выявлено тех, которые могут совмещать в себе сценарии стриминга, а также видеоконференцсвязи. Возможности по созданию виртуальной аппаратной были выявлены только у одного решения, но только с существенными ограничениями. Таким образом было подтверждена актуальность проблем и важность поставленной цели.

Среди проанализированных решений организации потокового видео были выявлены 2 решения, которые помогли бы достичь поставленной цели. На стыке выбранных технологий было реализовано решение, которое может осуществлять получение данных из любого необходимого источника, будь то удаленный rtmp-сервер или сырые данные из приложения. Важно отметить то, что среди данных из приложения могут быть данные, полученные из видеоконференцсвязи по стандарту WebRTC.

Работоспособность данного решения также была установлена и продемонстрирована на примере двух сценариев:

- WebRTC – RTMP
- RTMP – WebRTC

СПИСОК ЛИТЕРАТУРЫ

1. E. P. J. Tozer. Broadcast Engineer's Reference Book – 2013 – 1049 с.
2. Исходный код расширения для ПО OBS – studio WebRTC – URL: https://github.com/CoSMoSoftware/OBS-studio-webrtc
3. А. О. Трубаков, М. О. Селейкович. Сравнение интерполяционных методов масштабирования растровых изображений – 2017 – 97с.
4. Altanai. WebRTC Integrator's Guide - 2014 - 382 с.
5. Iain E. G. Richardson. The H.264 Advanced Video Compression Standard - 2011 - 346 с.
6. Документация фреймворка Gstreamer – URL: https://gstreamer.freedesktop.org/documentation
7. Эндрю Троелсен. Язык Программирования C# 5.0 и платформа .NET 4.5 - 6-е издание – 2013–1310 с.