Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Мультисерверный видеостриминг с множественными источниками»

Студент __ИУ7-31М__ _____ ____Е. В. Брянская__
(Группа) (Подпись, дата) (И.О.Фамилия)

Студент __ИУ7-31М__ _____ ____В. А. Иванов_ __
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ ____А.М. Никульшин_
(Подпись, дата) (И.О.Фамилия)

2023 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ7
(Индекс)
_____ И. В. Рудаков
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине Протоколы вычислительных сетей

Студенты группы ИУ7-31М

Брянская Екатерина Вадимовна
(Фамилия, имя, отчество)
Иванов Всеволод Алексеевич
(Фамилия, имя, отчество)

Тема курсового проекта Мультисерверный видеостриминг с множественными источниками

Направленность КП (учебный, исследовательский, практический, производственный, др.)
учебный
Источник тематики (кафедра, предприятие, НИР) Кафедра

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание Разработать протокол для мультлисерверного стриминга. Проанализировать предметную область, выделить целевую аудиторию и сценарии его применения. Провести обзор существующих форматов видеорядов, обосновать выбор используемого формата в разрабатываемом протоколе. Определить участников видеостриминга и их функции. Определить способы их взаимодействия и составить набор соответствующих сообщений протокола и их содержание. Протокол должен поддерживать проверку целостности данных, повторный запрос потерянных и повреждённых кадров. Должен быть реализован механизм перераспределения трафика между источниками и управления скоростью передачи данных. Реализовать и протестировать клиент-серверное приложение, использующее данный протокол.

Оформление курсового проекта:

Расчетно-пояснительная записка на 20-30 листах формата А4.
Расчетно-пояснительная записка должна содержать постановку задачи, введение, аналитическую, конструкторскую, технологическую части, заключение и список литературы.

Дата выдачи задания «11» октября 2023 г.

Руководитель курсового проекта	_____ (Подпись, дата)	А.М.Никульшин (И.О.Фамилия)
Студент	_____ (Подпись, дата)	В.А.Иванов (И.О.Фамилия)
Студент	_____ (Подпись, дата)	Е.В.Брянская (И.О.Фамилия)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

По данным журнала Market Research Report [1] рынок видеостриминга сейчас оценивается более, чем 500 миллиардов долларов, и, предполагается, что к 2030 году эта сумма достигнет 1.9 триллионов. Согласно статистике по всему миру насчитывается около 1.8 миллиарда подписок на сервисы потоковой передачи видео, примерно 26% пользователей которых используют подписку на постоянной основе не реже одного раза в неделю.

Потоковая передача видео в настоящее время более популярна и составляет более, чем 38.1% от общего объёма использования, чем кабельное или широковещательное телевидение, на долю которых приходится 30.9% и 24.7%.

Соответственно, ввиду непрекращающегося спроса на видеоплатформы, необходимо обеспечить эффективный способ взаимодействия многочисленных пользователей с этими сервисами для обеспечения наиболее качественной передачи информации.

Цель работы – разработать протокол для мультисерверного стриминга, обеспечивающий высокую производительность и масштабируемость.

Для достижения цели необходимо решить следующие задачи:

- проанализировать предметную область и выделить целевую аудиторию;
- определить сценарии взаимодействия участников передачи данных, составить набор соответствующих сообщений и их содержание;
- провести обзор существующих форматов видеорядов и обосновать выбор используемого формата в разрабатываемом протоколе;
- идентифицировать и сформулировать основные требования для мультисерверного стриминга;
- разработать протокол в соответствии с выделенным и требованиями;
- реализовать и протестировать прототип, позволяющий проверить правильность работы и эффективность протокола.

1 Аналитическая часть

1.1 Основные понятия

Потоковый трафик – тип трафика, для которого характерен просмотр и/или прослушивание информации по мере её поступления на конечное оборудование (мобильный телефон, компьютер, телевизор с доступом в Интернет и т.д.). [2] Основную часть потокового трафика составляет потоковое видео (или видеопоток).

Видеостриминг (или **стриминг видео**) – технология передачи видеоконтента через Интернет в режиме реального времени, при этом пользователь не должен ждать полной загрузки файла для просмотра. [3] Видео транслируется непрерывным потоком в виде последовательных кадров в специальном формате. Просмотр начинается в момент достаточной буферизации, обеспечивая при этом равномерное отображение данных.

Основой для передачи мультимедийной информации в настоящее время становятся мультисерверные платформы. Они способны отправлять данные разных типов и поддерживать трафик с различными характеристиками. Наибольшее распространение получили два варианта передачи потокового видео по сети.

- *Видео реального времени (real-time streaming)*, запись которого осуществляется одновременно с его просмотром, например, видеоконференции, прямые эфиры.
- *Видео по запросу* (progressive streaming). Предварительно записанные видеоряды хранятся на сервере, запрашиваются приложениями конечного пользователя и воспроизводятся при получении.

Привлечение мультипоточной загрузки, при которой данные загружаются сразу с нескольких серверов или источников, имеет следующие преимущества по сравнению с загрузкой только с одного сервера.

- *Увеличение скорости загрузки*. Мультипоточная загрузка позволяет ис-

пользовать полную пропускную способность нескольких серверов одновременно, ускоряя процесс передачи видеоконтента. Это особенно полезно при работе с большими файлами, например, с видео высокого разрешения.

- *Улучшение стабильности и отказоустойчивости*. Если один из источников недоступен или работает медленно по какой-то причине, то другие могут продолжать предоставлять необходимые данные. Это минимизирует возможные проблемы с недоступностью серверов.
- *Оптимизация использования сетевых ресурсов*. Разделение загрузки данных между несколькими серверами помогает избежать перегрузки одного сервера и эффективно использовать сетевые ресурсы.
- *Адаптация к сетевым ограничениям*. При привлечении нескольких серверов можно адаптировать процесс загрузки к изменениям сетевых условий (например, изменение скорости интернет-соединения), что помогает поддерживать стабильное и качественное воспроизведение.

Все эти преимущества делают мультисерверную загрузку более предпочтительной, особенно в случаях, если важна скорость передачи данных, стабильности и отказоустойчивость системы.

Как правило, система трансляции потоковых видео состоит из четырёх подсистем:

1) **Устройство кодирования** для сжатия видеопотока и загрузки его на медиасервер.
2) **Медиасервер**, отвечающий за хранение видеорядов и передачу пользователям. Он является ключевой единицей во всём процессе передачи. Основная его задача – взаимодействие с транспортной сетью при отправке пакетов в нужное время. Как правило, состоит из трёх компонентов механизма трансляции: *транспортный протокол, операционная система и система хранения*.
3) **Транспортная сеть**, которая транслирует пакеты от медиасервера до кли-

ентского устройства с помощью специально разработанных и стандартизированных протоколов. Последние обеспечивают такие услуги связи, как сетевая адресация, транспортировка и контроль за сеансом связи.

4) **Клиентское приложение**, декодирующее и воспроизводящее мультимедиапоток. Также опционально может присутствовать механизм синхронизации аудио и видео.

Действующий протокол между клиентом и сервером определяет:

1) синтаксис и формат данных

   Фиксируется чёткая структура сообщений или пакетов данных, которые передаются между устройствами. Как правило, это описание формата заголовков и тела сообщения.

2) способ установления соединения

   Протокол может определять процедуру процессов установления, поддержания и завершения соединения между устройствами.

3) основные операции и команды

   Описывается множество доступных операций, команд или запросов, которые могут быть выполнены в рамках существующего протокола.

4) обработку ошибок и контроль целостности данных

   Определяются методы и сценарии обработки некорректных ситуаций, способы контроля целостности данных.

5) управление потоком данных

   Может также описываться алгоритм регулирования скорости передачи потока информации.

В текущей работе будет рассматриваться протокол мультисерверного видеостриминга для передачи данных по запросу.

1.2 Возможные сферы применения и целевая аудитория

Подобный протокол может быть применён в рамках высоконагруженных систем, позволяя увеличивать скорость загрузки данных путём одновременно-

го запроса и получения разных частей контента с нескольких серверов. Такой подход позволяет не только ускорить загрузку видео, но и обеспечивает более плавное воспроизведение, поскольку некоторые части запрашиваются заранее и в случае потери, могут быть получены повторно.

Его привлечение в сервисах видеохостинга особенно полезно в случае нестабильного Интернет-соединения и большого объёма файла. Например, разрабатываемый протокол может быть привлечён в рамках обязательного внеурочного занятия в России, введённого в программы образовательных организаций начального, основного, среднего и профессионального образования. Каждый понедельник на первом уроке учащимся показывают видеофайлы, повествующие о наиболее актуальных событиях в стране, на основе которых строится дальнейшее обсуждение поднятых в них тем.

Таким образом, каждое утро понедельника классные руководители по всей стране скачивают подготовленные ролики с сайта «Разговоры о важном», создавая колоссальную нагрузку на серверы, из-за чего постоянно возникают сбои и сложности с загрузкой. Рассматриваемый в рамках текущей курсовой работы протокол может помочь решить некоторые возникающие проблемы, путём распределения процесса загрузки видеофрагментов по нескольким серверам.

Также его можно привлечь в процессе обмена видеофайлами между платформами для ускорения процесса переноса данных. Это может быть актуально для сервисов, размещающих видео сразу на нескольких платформах, например, YouTube, Rubube, VK Видео.

1.3 Форматы видеорядов

Наиболее часто встречающиеся форматы видео:

- **MP4 (по-другому MPEG-4 Part 14)** – формат, совместимый с большинством браузеров и поддерживаемый сайтами потокового видео, в частности, YouTube.
- **AVI (Audio Video Interleave)** – старый формат, разработанный Microsoft.

Поддерживается большинством популярных браузеров, работающих в системах Windows, Linux.

- **MPG, MPEG, MP2, MPE, MPV** – форматы, в которых обычно записывают видео, которые впоследствии не нужно будет редактировать.
- **MOV** – формат, разработанный Apple. Видео сохраняется в хорошем качестве, но файл занимает много места.
- **WebM** – формат, позволяющий получать видео небольшого размера среднего качества. Видео в таком формате подходят для YouTube и других сайтов потокового видео на платформе HTML5.

Формат видео существенно не влияет на протокол видео-трансляции, поэтому в демонстративных целях будет использоваться только MP4, другие видеоформаты также могут быть привлечены при доработках реализации парсера и проигрывателя медиафайла.

1.4 Основные этапы протокола мультисерверного стриминга с множественными источниками

На рисунке 1.1 схематично представлен общий алгоритм работы протокола. Ключевые моменты отмечены цифрами.

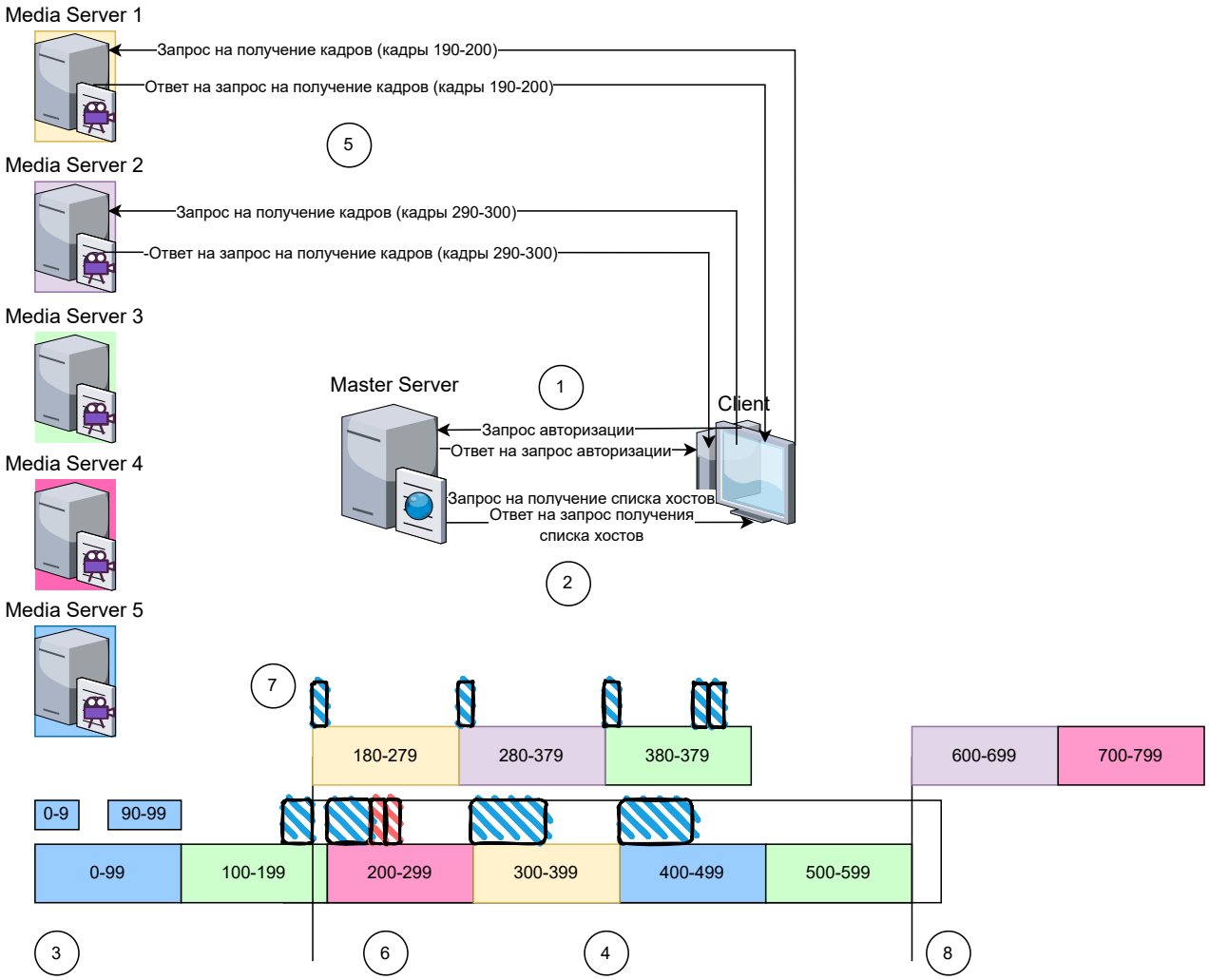

Рисунок 1.1 – Основные этапы протокола.

Весь видеофайл разбивается на несколько фрагментов (на рисунке метка 3), каждый из которых может быть получен от любого медиа-сервера.

Сначала клиент должен пройти процесс авторизации, обозначен цифрой 1, после успешного завершения он может запросить у мастер-сервера распределение интервалов видео по хостам (отмечен 2).

Получив его, клиент делает несколько параллельных запросов к различным медиа-серверам (отмечено как 5) на загрузку n кадров.

В случае m неуспешных запросов к какому-либо медиа-серверу (отмечено 6) клиент должен сообщить мастер-серверу о проблеме, инициировав перераспределение фрагментов между хостами.

Таким образом, клиент получает новый список медиа-серверов с перерас-

пределёнными видео-интервалами загружаемого файла (отмечено на рисунке цифрой 7) и начинает выгрузку ещё не полученных кадров.

Клиент загружает такой объём информации, который помещается в буфер (отмечен на рисунке цифрой 4).

Как только окно загрузки будет передвинуто на некоторую величину кадров, клиент должен отправить запрос на получение списка медиа-серверов для новых видео-фрагментов (цифра 8).

Выводы

Таким образом, в данной работе будет разрабатываться протокол для мультисерверного видеостриминга с множественными источниками. В качестве формата видео был выбран MP4. В разделе был описан принцип работы протокола, выделены необходимые виды сообщений.

2 Конструкторская часть

2.1 Взаимодействие клиента и мастер-сервера

2.1.1 Общая схема

Основными компонентами в разрабатываемом протоколе являются клиент, мастер-сервер и медиасерверы. Сначала клиент устанавливает соединение с мастер-сервером, запрашивает у него список доступных для дальнейшего взаимодействия хостов.

В рассматриваемом протоколе взаимодействие между клиентом и мастер-сервером осуществляется по gRPC, между клиентом и медиа-серверами по UDP.

Общая схема обмена сообщениями между клиентом и мастер-серверов представлена на рисунке 2.1.

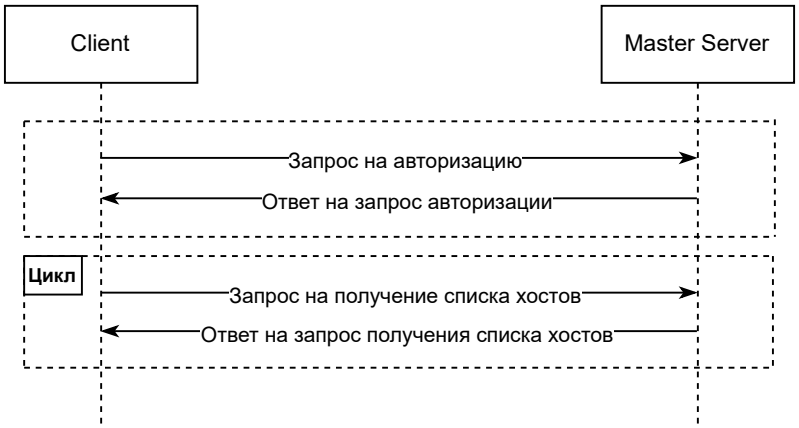

Рисунок 2.1 – Общая схема.

2.1.2 Авторизация

Первый шаг в установке соединения между клиентом и мастер-сервером – авторизация (рисунок 2.2).

Клиент отправляет авторизационный запрос в формате, представленном в таблице 1.

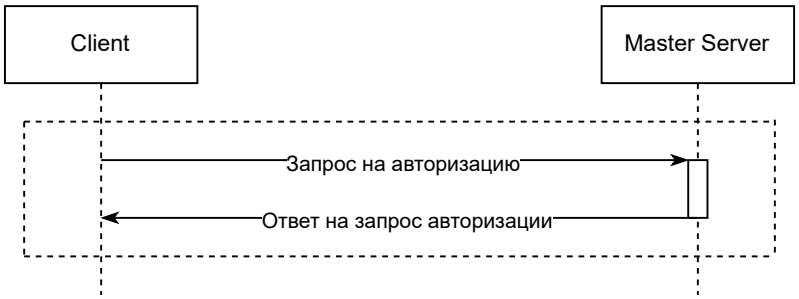

Рисунок 2.2 – Этап авторизации.

Таблица 1 – Атрибутивный состав запроса на авторизацию

Параметр	**Тип**	**Описание**
username	string	Имя пользователя доменной учетной записи
password	string	Пароль пользователя доменной учетной записи
grant_type	string	Тип запроса на получение токена
client_id	string	Имя приложения

В зависимости от полученных данных сервер либо передаёт токен доступа в ответном сообщении (таблица 2), либо возвращает сообщение об ошибке авторизации в формате из таблицы 3.

Таблица 2 – Атрибутивный состав ответа на успешный запрос авторизации

Параметр	**Тип**	**Описание**
access_token	string	Токен доступа
expires_in_sec	integer	Число секунд, после которых токен перестанет быть валидным
token_type	string	Тип токена

Таблица 3 – Атрибутивный состав ответа о непройденной авторизации на соответствующий запрос

Параметр	**Тип**	**Описание**
error	string	Общее описание ошибки
error_description	string	Детальное описание ошибки

2.1.3 Получение списка хостов

Далее в случае успешной авторизации клиент отправляет запрос на получение списка хостов (рисунок 2.3), к которым в последствии будут отправляться запросы на получение фрагментов видео.

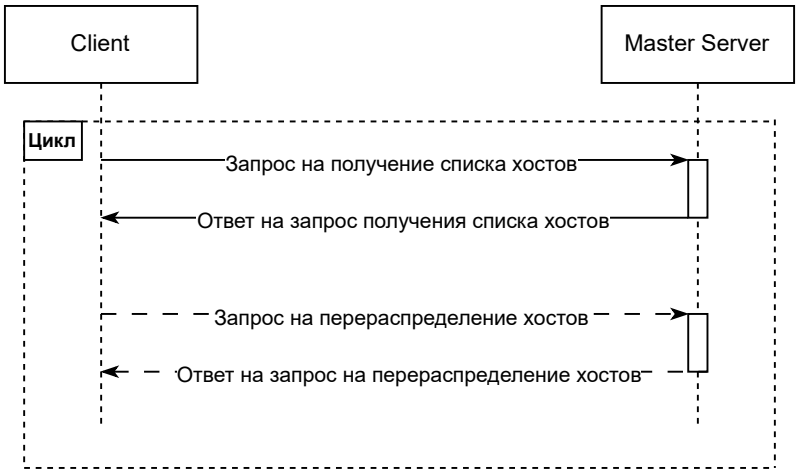

Рисунок 2.3 – Этап получения списка хостов.

Формат такого запроса представлен в таблице 4.

Таблица 4 – Атрибутивный состав запроса на получение списка хостов

Параметр	**Тип**	**Описание**
access_token	string	Токен авторизации, указывается в заголовке запроса
video_filename	string	Название видео-файла

Продолжение на следующей странице

Параметр	**Тип**	**Описание**
start_time	timestamp	Временная метка начала временного окна
end_time	timestamp	Временная метка окончания временного окна

В ответ мастер-сервер отправляет список хостов с указанием диапазона кадров, которые можно с них загрузить, также дополнительно прилагается метаинформация о видеофайле (разрешение, длительность и т.д.). Атрибутивный состав этого сообщения приведён в таблице 5.

Таблица 5 – Атрибутивный состав ответа на запрос получения списка хостов

Параметр	**Тип**	**Описание**
hosts	array[object]	Список хостов с указанием адреса и диапазона кадров
address	string	Адрес хоста
frame_start	integer	Номер первого фрагмента из диапазона
frame_end	integer	Номер последнего фрагмента из диапазона
metadata	object	Блок метаданных
resolution	string	Разрешение
codec	string	Кодек
duration	timestamp	Длительность

Поскольку клиент запрашивает кадры видео только в пределах окна, то в случае его заполнения оно должно быть сдвинуто, и к мастер-серверу должен осуществляться повторный запрос на получение списка хостов для загрузки очередных фрагментов. Эта операция должна повторяться до тех пор, пока не будет получен весь видеофайл.

В случае, если клиенту не удаётся загрузить n частей видеофрагмента от

одного и тоже же медиа-сервера в пределах текущего окна, то он должен отправить на мастер-сервер запрос на перераспределение хостов, в котором указывается проблемный медиа-сервер и временная метка последней успешно загруженной с него части видеофрагмента. Атрибутивный состав сообщения приведён в таблице 6.

Таблица 6 – Атрибутивный состав запроса на перераспределение хостов

Параметр	**Тип**	**Описание**
access_token	string	Токен авторизации, указывается в заголовке запроса
video_filename	string	Название видео-файла
host	object	Проблемный хост
address	string	Адрес хоста
frame_start	integer	Номер первого фрагмента из диапазона
frame_end	integer	Номер последнего фрагмента из диапазона
start_time	timestamp	Временная метка последней успешно загруженной части видеофрагмента
end_time	timestamp	Временная метка окончания временного окна

Отправляя этот запрос, клиент инициирует повторную операцию определения подходящих для взаимодействия хостов. В ответ мастер-сервер отправляет новое перераспределение фрагментов видео между медиа-серверами, состав ответа приведён в таблице 7.

Таблица 7 – Атрибутивный состав ответа на запрос перераспределения хостов

Параметр	**Тип**	**Описание**
hosts	array[object]	Список хостов с указанием адреса и диапазона кадров
address	string	Адрес хоста
frame_start	integer	Номер первого фрагмента из диапазона
frame_end	integer	Номер последнего фрагмента из диапазона
metadata	object	Блок метаданных
resolution	string	Разрешение
codec	string	Кодек
duration	timestamp	Длительность

2.1.4 Получение видеофрагментов

Общая схема загрузки видеофрагмента показана на рисунке 2.4. После получения списка доступных хостов от мастер-сервера клиент отправляет сразу несколько запросов на загрузку частей фрагмента параллельно. Таким образом, осуществляется одновременное скачивание сразу нескольких частей видео-файла в рамках текущего окна загрузки.

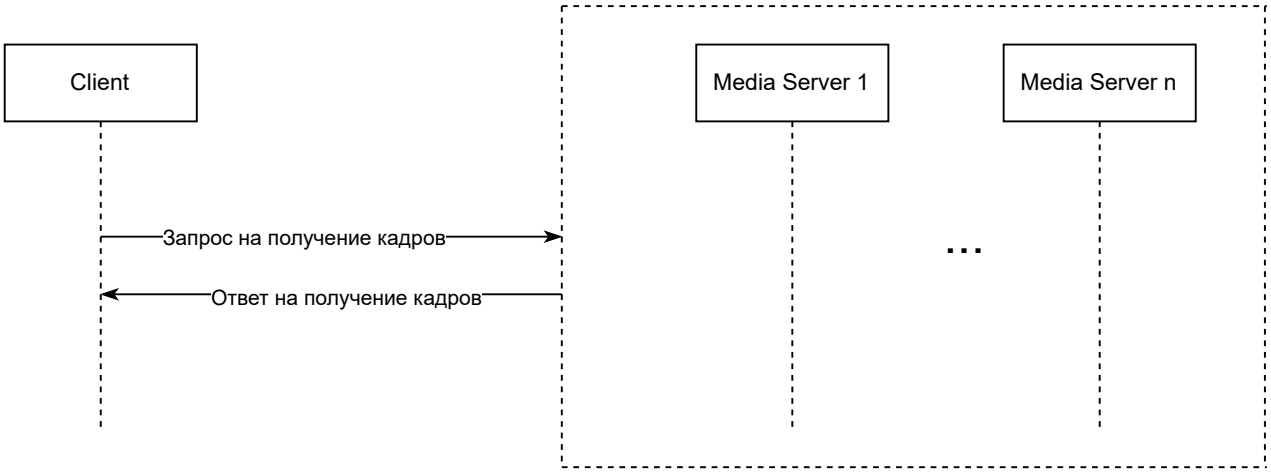

Рисунок 2.4 – Общая схема получения видеофрагментов.

На рисунке 2.5 представлена более детальная схема этого этапа. Параллельные запросы (таблица 8) к медиа-серверам будут осуществляться до тех пор, пока не будет получена большая часть окна загрузки, после чего оно динамически сдвигается, и клиент запрашивает у мастер-сервера список новых хостов.

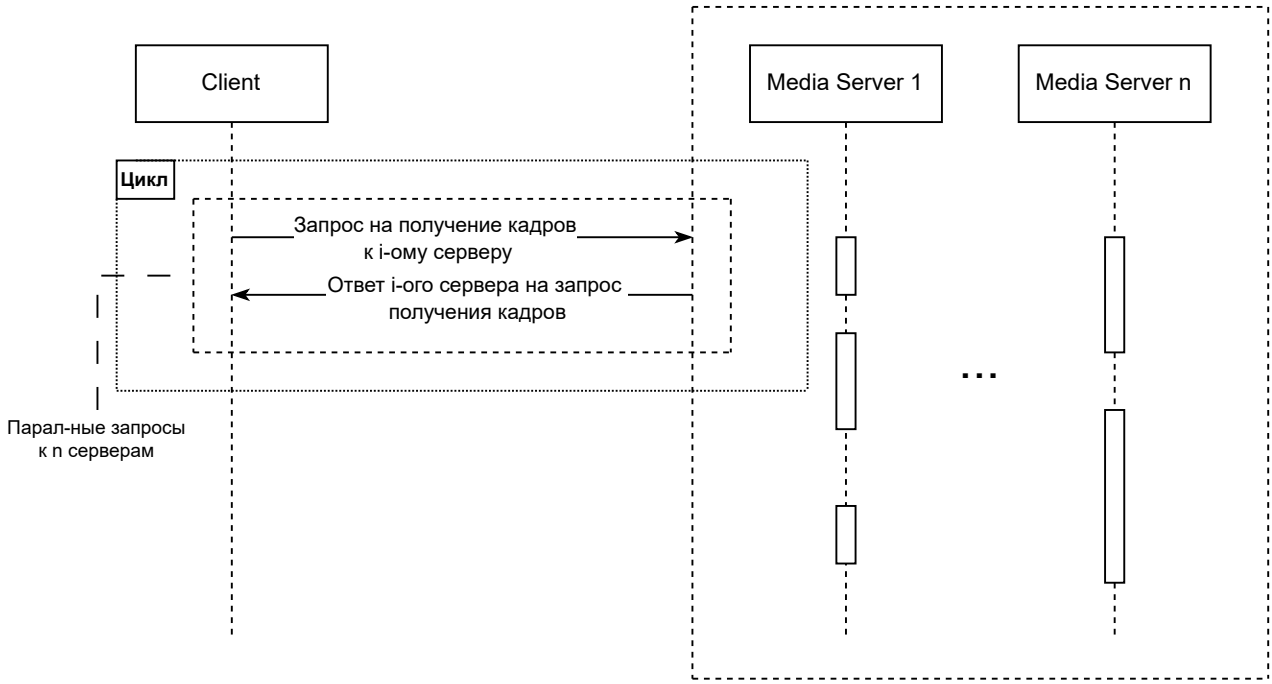

Рисунок 2.5 – Детализированная схема получения видеофрагментов.

Для контроля целостности UDP пакета также привлекается контрольная

сумма содержимого кадров. Для её вычисления используется XOR всех байтов.

Таблица 8 – Атрибутивный состав запроса на получение кадров

Параметр	**Тип**	**Описание**
access_token	string	Токен авторизации, указывается в заголовке запроса
check_sum	string	Контрольная сумма
video_filename	string	Название видео-файла
frame_start	integer	Номер первого фрагмента из диапазона
frame_end	integer	Номер последнего фрагмента из диапазона

В ответ на поступивший запрос медиа-сервер формирует ответ, атрибутивный состав которого представлен в таблице 9.

Таблица 9 – Атрибутивный состав ответа на запрос получения кадров

Параметр	**Тип**	**Описание**
num_frames	integer	Число передаваемых кадров
frames	array[object]	Список кадров
size	integer	Размер в байтах
frame	array[byte]	Кадр

Выводы

В этом разделе были формализованы сообщения с помощью схем и атрибутивного состава.

3 Технологическая часть

3.1 Выбор средств программной реализации

3.1.1 Основные средства

В качестве языка программирования был выбран Python 3 [4], ввиду нескольких причин.

- Язык поддерживает объектно-ориентированный подход, что важно, поскольку в процессе реализации подразумевается использование этой методологии, позволяющей разрабатывать хорошо организованную и просто модифицируемую структуру приложения.
- Кроме того, предоставляются библиотеки для создания графического интерфейса, которые планируется использовать для отладки и наглядной демонстрации работы приложения.
- В дополнение, в процессе обучения был накоплен существенный опыт в использовании этого языка программирования.

В качестве среды разработки был выбран VS Code [5] в силу следующих факторов.

- Бесплатна.
- Предоставляются удобные инструменты для написания, редактирования кода, а также графический отладчик.
- Помимо этого, является хорошо знакомой средой разработки, и какие-либо проблемы с взаимодействием сведены к минимуму, что позволяет сэкономить время.

3.2 Используемые библиотеки

OpenCV – open source библиотека компьютерного зрения, которая применяется для анализа, классификации и обработки изображений и видео. [6]

gRPC – библиотека, которая используется для обеспечения взаимодей-

ствия между клиентом и мастер-сервером по проколу gRPC. [7]

Выводы

В данном разделе для реализации протокола в качестве основного языка программирования был выбран Python, среды разработки – VS Code. Определены основные библиотеки, в том числе и для обработки видеоматериалов.

ЗАКЛЮЧЕНИЕ

Таким образом, в рамках текущего курсового проекта был разработан и реализован протокол мультисерверного видеостриминга с множественными источниками.

В результате проделанной работы были выполнены все поставленные задачи.

- Проанализирована предметная область, выделена целевая аудитория.
- Также определены участники видеотрансляции, возможные сценарии их взаимодействия, составлен набор соответствующих запросов и ответов и их атрибутивный состав.
- Разработан и протестирован соответствующий прототип приложения, позволяющих проверить работоспособность протокола.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Video Streaming Market Size, Market Research Report [Электронный ресурс]. – Режим доступа: https://www.fortunebusinessinsights.com/video-streaming-market-103057 (Дата обращения: 02.12.2023)

2. Пакулова Екатерина Анатольевна Алгоритм распределения потокового трафика и трафика реального времени в гетерогенной беспроводной сети // Известия ЮФУ. Технические науки. 2014. №2 (151). URL: https://cyberleninka.ru/article/n/algoritm-raspredeleniya-potokovogo-trafika-i-trafika-realnogo-vremeni-v-geterogennoy-besprovodnoy-seti (дата обращения: 04.12.2023).

3. Apostolopoulos J. G., Tan W., Wee S. J. Video streaming: Concepts, algorithms, and systems //HP Laboratories, report HPL-2002-260. – 2002. – С. 2641-8770.

4. Документация по Python 3 [Электронный ресурс]. Режим доступа: https://docs.python.org/3/ (Дата обращения 01.12.2023)

5. Документация по Visual Studio Code [Электронный ресурс]. Режим доступа: https://code.visualstudio.com/docs (Дата обращения 01.12.2023)

6. Документация OpenCV [Электронный ресурс]. Режим доступа: https://docs.opencv.org/3.4/d6/d00/tutorial_py_root.html (Дата обращения 02.12.2023)

7. Документация gRPC [Электронный ресурс]. Режим доступа: https://grpc.github.io/grpc/python/ (Дата обращения 02.12.2023)

ПРИЛОЖЕНИЕ А

Ключевые методы реализации

Листинг 1: Protobuf-файл мастер-сервера

```
syntax = "proto3";

package master_server;

service MasterServer {
  rpc GetDistribution (GetDistributionRequest) returns (
    GetDistributionResponse) {}
}

message GetDistributionRequest {
  string filename = 1;
  int64 beginFrame = 2;
  optional int64 endFrame = 3;
}

message FrameDistribution {
  string endpoint = 1;
  int64 beginFrame = 2;
  int64 endFrame = 3;
}

message GetDistributionResponse {
  bool endOfFile = 1;
  repeated FrameDistribution distribution = 2;
}
```

Листинг 2: Основные функции клиента

```
@dataclass
class FrameDistribution:
    filename: str
    endpoint: str
    begin_frame: int
    end_frame: int
```

```
7
8  # Client <-> Media-server communication class
9  class ServerFetcher:
10     def __init__(self, distribution: FrameDistribution) -> None:
11         self.distribution = distribution
12         self.batch_size = 20
13         self.frame_buffer = asyncio.Queue()
14         self.data = bytes()
15         self.all_fetched = False
16
17     async def recv(self, n: int):
18         return await self.reader.read(n)
19
20     async def recv_data(self, size: int):
21         while len(self.data) < size:
22             self.data += await self.recv(CHUNK_SIZE)
23         msg = self.data[:size]
24         self.data = self.data[size:]
25         return msg
26
27     async def fetch_frames(self):
28         packed_frame_count = await self.recv_data(struct.calcsize("L"))
29         frame_count = struct.unpack("L", packed_frame_count)[0]
30
31         frames = []
32         for frame_i in range(frame_count):
33             packed_frame_meta = await self.recv_data(struct.calcsize("LL"))
34             frame_number, frame_size = struct.unpack("LL", packed_frame_meta)
35             frame_data = await self.recv_data(frame_size)
36             # Extract frame
37             frame = pickle.loads(frame_data)
38             frames.append(frame)
39         return frames
40
41     def sync_run(self):
42         asyncio.run(self.run())
43
44     async def run(self):
45         host, port = self.distribution.endpoint.split(":")
46
```

```
47          for frame_index in range(
48              self.distribution.begin_frame,
49              self.distribution.end_frame + 1,
50              self.batch_size,
51          ):
52              self.reader, self.writer = await asyncio.open_connection(host,
    port)
53
54              raw_filename = self.distribution.filename.encode("utf-8")
55              self.writer.write(struct.pack("LL64s", frame_index, self.
    batch_size, raw_filename))
56              await self.writer.drain()
57
58              frames = await self.fetch_frames()
59              if len(frames) == 0:
60                  print(f"recived end of communication message {frame_index}")
61                  break
62
63              print(f"recived frames: {frame_index}-{frame_index+len(frames)-1}
    ")
64              for frame in frames:
65                  await self.frame_buffer.put(frame)
66              self.writer.close()
67              await self.writer.wait_closed()
68          self.all_fetched = True
69
70      def is_done(self):
71          return self.all_fetched and self.frame_buffer.empty()
72
73
74  class DownloadMaster:
75      def __init__(self, filename: str, master_endpoint: str) -> None:
76          self.filename = filename
77          self.master_endpoint = master_endpoint
78
79          self.unfetched_distributions: list[FrameDistribution] = []
80          self.fetchers: list[ServerFetcher] = []
81          self.all_distibution_fetched = False
82          self.first_undistributed_frame = 0
83
```

```
84          self.distibution_size = 2000
85          self.fetchers_n = 6
86
87      def sync_start(self):
88          asyncio.run(self.start())
89
90      async def start(self):
91          await self.fetch_distibution()
92          for _ in range(self.fetchers_n):
93              await self.run_next_fetcher()
94
95      async def fetch_distibution(self):
96          async with grpc.aio.insecure_channel(self.master_endpoint) as channel
    :
97              stub = master_server_pb2_grpc.MasterServerStub(channel)
98              request = master_server_pb2.GetDistributionRequest(
99                  filename=self.filename,
100                 beginFrame=self.first_undistributed_frame,
101                 endFrame=self.first_undistributed_frame+self.distibution_size
    ,
102             )
103             response: master_server_pb2.GetDistributionResponse = await stub.
    GetDistribution(request)
104
105         for distr in response.distribution:
106             print(f"{distr.endpoint}: {distr.beginFrame}-{distr.endFrame}")
107
108         self.unfetched_distributions += [
109             FrameDistribution(self.filename, distr.endpoint, distr.beginFrame
    , distr.endFrame)
110             for distr in response.distribution
111         ]
112         self.first_undistributed_frame += self.distibution_size
113         self.all_distibution_fetched = response.endOfFile
114
115     async def run_next_fetcher(self) -> bool:
116         if len(self.unfetched_distributions) == 0:
117             if self.all_distibution_fetched:
118                 # no more distributions left
119                 return False
```

```
120             await self.fetch_distibution()
121
122         if len(self.unfetched_distributions) == 0:
123             return False
124
125         distribution = self.unfetched_distributions.pop(0)
126         fetcher = ServerFetcher(distribution)
127         self.fetchers.append(fetcher)
128         asyncio.create_task(asyncio.to_thread(fetcher.sync_run))
129         return True
130
131     async def get_current_fetcher(self):
132         while len(self.fetchers) and self.fetchers[0].is_done():
133             done_fetcher = self.fetchers.pop(0)
134             print(f"{done_fetcher.distribution=}")
135
136         if len(self.fetchers):
137             current_fetcher = self.fetchers[0]
138             if len(self.fetchers) < self.fetchers_n:
139                 asyncio.create_task(self.run_next_fetcher())
140         else:
141             launched = await self.run_next_fetcher()
142             if not launched:
143                 return None
144             current_fetcher = self.fetchers[0]
145
146         return current_fetcher
147
148     async def get_next_frame(self, timeout: float | None):
149         current_fetcher = await self.get_current_fetcher()
150         if current_fetcher is None:
151             return None
152
153         return await asyncio.wait_for(
154             current_fetcher.frame_buffer.get(),
155             timeout=timeout,
156         )
```

Листинг 3: Основные функции мастер-сервера

```
ENDPOINTS = [
    f"localhost:{port}"
    for port in range(8090, 8095)
]
# Number of endpoints used in one distribution
PARALLEL_ENDPOINTS_N = 3
DISTRIBUTION_SIZE = 100
MEDIA_DIRECTORY = "/Users/ivavse/temp/nets/"


class MasterServer(master_server_pb2_grpc.MasterServer):
    def GetDistribution(
        self,
        request: master_server_pb2.GetDistributionRequest,
        context,
    ):
        filepath = os.path.join(MEDIA_DIRECTORY, request.filename)
        video_stream = cv2.VideoCapture(filepath)
        total_frames = int(video_stream.get(cv2.CAP_PROP_FRAME_COUNT))

        used_endpoints = random.choices(ENDPOINTS, k=PARALLEL_ENDPOINTS_N)
        endpoints_iterator = itertools.cycle(used_endpoints)

        distribution = []
        end_frame = min(request.endFrame, total_frames)
        for begin_frame in range(request.beginFrame, end_frame,
    DISTRIBUTION_SIZE):
            distribution.append(master_server_pb2.FrameDistribution(
                endpoint=next(endpoints_iterator),
                beginFrame=begin_frame,
                endFrame=begin_frame + DISTRIBUTION_SIZE - 1
            ))

        return master_server_pb2.GetDistributionResponse(
            endOfFile=(end_frame != request.endFrame),
            distribution=distribution,
        )
```

Листинг 4: Основные функции медиа-сервера

```
class Server:
    def get_raw_frame(self, stream: cv2.VideoCapture) -> bytes:
        ret, frame = stream.read()
        data = pickle.dumps(frame)
        return data

    async def start_server(self, host, port):
        self.server = await asyncio.start_server(self.handle_request, host, port)
        print(f"server started on the endpoint {host}:{port}")
        await self.server.serve_forever()

    async def read_request(self, reader: asyncio.StreamReader):
        data = bytes()
        msg_format = "LL64s"
        msg_size = struct.calcsize(msg_format)
        while len(data) < msg_size:
            data += await reader.read(msg_size - len(data))
        frame_offset, frame_count, raw_filename = struct.unpack(msg_format, data)
        filename = bytes(raw_filename).rstrip(b'\x00').decode("utf-8")
        return frame_offset, frame_count, filename

    async def handle_request(
        self,
        reader: asyncio.StreamReader,
        writer: asyncio.StreamWriter,
    ):
        frame_offset, frame_count, filename = await self.read_request(reader)
        print(f"request: {filename=}, frames:{frame_offset}-{frame_offset+frame_count-1}")

        filepath = os.path.join(MEDIA_DIRECTORY, filename)
        stream = cv2.VideoCapture(filepath)
        stream.set(cv2.CAP_PROP_POS_FRAMES, frame_offset)

        raw_frames = []
        for frame_number in range(frame_offset, frame_offset + frame_count):
            data = self.get_raw_frame(stream)
```

```
            if len(data) == 4:
                # end of frames
                break
            frame_number_raw = struct.pack("L", frame_number)
            frame_size_raw = struct.pack("L", len(data))
            raw_frame = frame_number_raw + frame_size_raw + data
            raw_frames.append(raw_frame)

        frame_count_raw = struct.pack("L", len(raw_frames))
        packed_frames = struct.pack(
            "".join([str(len(x)) + "s" for x in raw_frames]), *raw_frames
        )
        raw_message = frame_count_raw + packed_frames

        writer.write(raw_message)
        await writer.drain()
        writer.close()
```