МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет Механико-математический
Кафедра Программирования

Направление подготовки Математика и компьютерные науки

РЕФЕРАТ

Дедов Георгий Юрьевич
(Фамилия, Имя, Отчество автора)

Тема: **SRT (Secure, Reliable, Transport – Протокол)**

«Реферат принят»

Научный руководитель

Доцент кафедры АФТИ ФФ

Таранцев.И.Г. /_____
(фамилия , И., О.) (подпись, МП)

«…»……20…г.

Новосибирск, 2024

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1 Определение SRT-протокола

SRT (Secure Reliable Transport) — это протокол передачи данных, разработанный для обеспечения надежной и безопасной доставки аудио- и видеоконтента через нестабильные сети, такие как интернет. Протокол был изначально разработан компанией Haivision и предназначен для минимизации задержек и потерь данных при передаче в реальном времени. SRT использует передовые алгоритмы коррекции ошибок и адаптации к изменяющимся условиям сети, что позволяет значительно улучшить качество потокового видео и аудио. На данный момент протокол был передан в SRT Alliance – Open Source проект, который продолжает его разработку и улучшение.

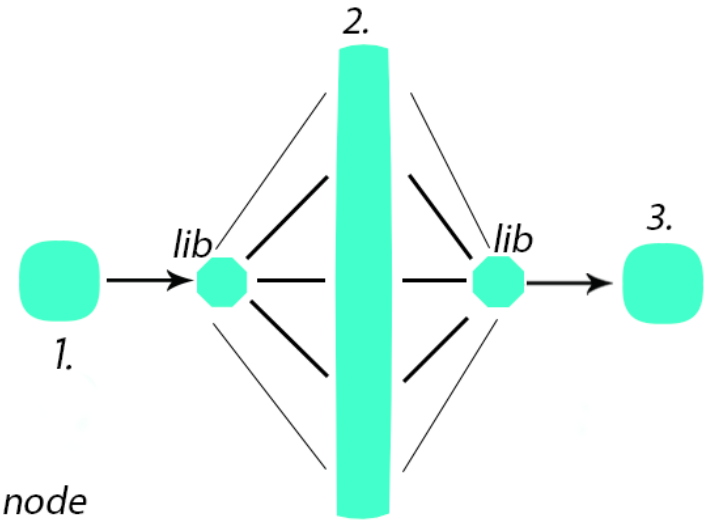
1.2 Значение и актуальность темы

В современном мире медиатехнологий, где потоковая передача данных играет ключевую роль, надежность и безопасность являются основными требованиями к протоколам передачи данных. SRT-протоколы стали неотъемлемой частью экосистемы потокового вещания, предоставляя пользователям возможность передавать контент с минимальными задержками и высоким уровнем безопасности.

Актуальность темы обусловлена несколькими факторами. Во-первых, с ростом популярности онлайн-трансляций, видеоконференций и удаленного обучения возросла потребность в стабильных и безопасных каналах передачи данных. Во-вторых, использование SRT-протоколов позволяет решить множество проблем, связанных с нестабильностью интернет-соединений и высокой задержкой, что делает их незаменимыми в различных сферах, от медиа и развлечений до медицины и образования.

Таким образом, изучение SRT-протоколов не только помогает понять современные технологии потоковой передачи данных, но и открывает новые возможности для их эффективного применения в самых различных областях.

1.3 Схема системы



1. Sender node
2. Генератор Помех в сети
lib. Библиотека
3. Receiver node

В данном проекте разработки библиотеки для динамического перераспределения потоков мультимедиа данных на базе SRT-протокола имеются следующие компоненты: sender (отправитель), receiver (получатель), генератор помех в сети и библиотека, управляющая процессом передачи данных. Подробнее о компонентах:

- Sender (отправитель) является начальной точкой передачи данных в системе. Генерирует и отправляет данные через SRT-протокол. Процесс передачи данных и их распределение по потокам контролируются библиотекой.
- Данные, выходящие из библиотеки, направляются в генератор помех в сети. Генератор помех симулирует реальные условия сети, такие как задержки, потери пакетов. Это позволяет тестировать и оптимизировать работу библиотеки и SRT-протокола в различных сетевых условиях.
- Receiver (получатель) является конечной точкой передачи данных в системе, он принимает и анализирует данные.

На 6-ой семестр обучения были поставлены следующие задачи: изучение SRT – протокола и его документации, разобраться в том

как работают сетевые протоколы, изучить сокеты в C++, разработать инструмент для отладки будущей библиотеки.

1.4 Конечная цель разработки

Конечная цель разработки заключается в реализации динамического перераспределения данных между различными сетевыми путями для передачи мультимедиа контента с использованием протокола SRT (Secure Reliable Transport). Различные сетевые пути – это альтернативные маршруты, по которым данные могут передаваться через интернет. В настоящее время SRT не поддерживает динамическое перераспределение данных между различными сетевыми путями.

2. Обзор SRT протокола

2.1 Основные Характеристики SRT

2.1.1 Надежность передачи данных

SRT (Secure Reliable Transport) разработан с целью обеспечения высокой надежности передачи данных, особенно в условиях нестабильных сетевых соединений. Протокол использует:

2.1.2 Безопасность и шифрование

Одним из ключевых аспектов SRT является его способность обеспечивать безопасную передачу данных. Протокол поддерживает современные методы шифрования, такие как AES (Advanced Encryption Standard), что защищает передаваемую информацию от несанкционированного доступа и взлома. Это делает SRT идеальным выбором для приложений, где безопасность данных является критически важной, включая трансляции конфиденциальных мероприятий, видеоконференции и удаленную медицину.

2.1.3 Гибкость настройки параметров передачи

SRT предоставляет пользователям широкие возможности для настройки параметров передачи в зависимости от конкретных требований и условий сети. Протокол позволяет регулировать такие параметры, как битрейт, задержка, уровень коррекции ошибок и другие, что обеспечивает оптимальное соотношение между качеством и скоростью передачи данных. Это делает SRT универсальным решением, подходящим для различных сценариев использования — от трансляций в реальном времени до записи и воспроизведения видео.

2.1.4 Открытость и доступность протокола

Одним из наиболее значимых преимуществ SRT является его открытость. Исходный код протокола доступен на GitHub, что позволяет разработчикам свободно использовать, изменять и

улучшать его. Это способствует быстрому распространению SRT и его интеграции в различные продукты и платформы. Открытость протокола также означает, что он поддерживается активным сообществом, что ускоряет развитие и улучшение технологии.

2.1.5 Поддержка различных платформ

SRT поддерживается на широком спектре операционных систем и аппаратных платформ, включая Windows, Linux, macOS, iOS и Android. Это обеспечивает его универсальность и позволяет использовать SRT в самых различных приложениях и устройствах. Благодаря этому, разработчики могут легко интегрировать SRT в свои решения, независимо от используемой платформы, что делает протокол чрезвычайно гибким и удобным в применении.

2.2. Как SRT протокол работает

2.2.1 ARQ в SRT

ARQ (Automatic Repeat reQuest) — это метод, используемый в протоколах передачи данных для обеспечения надежности. В SRT (Secure Reliable Transport) ARQ играет важную роль в обеспечении доставки всех пакетов без потерь. Принцип работы ARQ заключается в следующем:

Проверка целостности данных: Приемник проверяет полученные пакеты на наличие ошибок. Если пакет принят корректно, приемник отправляет подтверждение (ACK) обратно отправителю. В случае обнаружения ошибки, пакет не подтверждается.

Повторная передача: Если отправитель не получает подтверждения о получении пакета в течение определенного времени, он повторно отправляет этот пакет. Этот процесс повторяется до тех пор, пока пакет не будет успешно доставлен и подтвержден.

Управление потоком: ARQ в SRT использует скользящее окно (sliding window) для управления потоком данных. Это позволяет

отправителю отправлять несколько пакетов до получения подтверждений, что повышает эффективность передачи данных.

ARQ обеспечивает надежную доставку данных, но может увеличить задержку в условиях высокой потери пакетов, поскольку повторная передача требует дополнительного времени.

2.2.2 FEC в SRT

FEC (Forward Error Correction) — это метод, используемый для коррекции ошибок на стороне приемника без необходимости повторной передачи пакетов. В SRT FEC используется в дополнение к ARQ для повышения надежности и эффективности передачи данных. Принцип работы FEC заключается в следующем:

Добавление избыточности: При отправке данных отправитель добавляет к каждому блоку данных дополнительные избыточные биты. Эти биты используются для обнаружения и коррекции ошибок на стороне приемника.

Обнаружение и коррекция ошибок: Приемник использует избыточные биты для проверки целостности полученных данных. Если обнаруживаются ошибки, приемник может исправить их, используя информацию из избыточных битов, без необходимости запрашивать повторную передачу пакета.

FEC особенно эффективен в условиях низкой и средней потери пакетов, поскольку позволяет избежать задержек, связанных с повторной передачей пакетов. Однако использование FEC увеличивает объем передаваемых данных за счет добавления избыточной информации.

2.2.3 Буфер в SRT

Буфер в SRT является важным компонентом, обеспечивающим надежность и стабильность передачи данных. Основные аспекты работы буффера в SRT включают:

Фиксированный размер буфера: В SRT используется фиксированный, часто достаточно большой буфер для временного хранения данных. Это позволяет компенсировать вариации задержки (джиттера) и потери пакетов в сети.

Сглаживание задержек: Буфер помогает сглаживать колебания задержки, что особенно важно для передачи аудио- и видеопотоков. Накопление данных в буфере позволяет поддерживать стабильный поток данных к конечному приложению.

Уменьшение потерь пакетов: Буфер в SRT также играет роль в уменьшении потерь пакетов, позволяя хранить данные до их успешной доставки. В случае потери пакета SRT использует данные из буфера для повторной передачи, что минимизирует потери и улучшает качество передачи.

Адаптация к сетевым условиям: Хотя размер буфера в SRT фиксирован, его наличие позволяет протоколу более эффективно адаптироваться к текущим условиям сети. Это особенно важно в условиях нестабильных соединений, где колебания задержки и потери пакетов могут быть значительными.

2.3 Роль SRT в современных медиа-системах

SRT занимает важное место в современных медиа-системах, обеспечивая высокое качество и надежность передачи данных. Протокол активно используется в различных сценариях, от прямых трансляций и видеоконференций до OTT (Over-the-Top) сервисов и CDN (Content Delivery Network). Благодаря своей гибкости и эффективности, SRT позволяет медиакомпаниям предоставлять контент высокого качества даже в условиях ограниченной пропускной способности и задержки сетей. Кроме того, открытость и доступность SRT способствуют его распространению и интеграции в разнообразные медиа-платформы, обеспечивая единый стандарт для передачи видео и аудио данных.

3. Архитектура приложения для отладки

3.1 Логика работы приложения

Общая цель и функциональность

Приложение предназначено для отладки и тестирования передачи мультимедиа данных через протокол SRT, далее приложение будет использовано при разработке библиотеки, чтобы производить тестовые выборки.

Приложение должно уметь отправлять тестовый набор данных по заранее заданному алгоритму, а также принимать и анализировать принятые данные.

Приложение должно вызываться из командной строки, в следующем формате:

приложение.exe формат_работы адрес –некоторые параметры работы

3.2 Итоговый инструмент

Приложение для отладки было названо datacontrol. Оно способно при вызове из командной строки работать в 1-ом из указанных 3-ёх различных режимах:

Generate – создание потоковой передачи SRT
Receive - получение потоковой передачи SRT в ноль
Forward - Получать пакеты между двумя сокетами в двунаправленном режиме
Один из режимов работы приложения при вызове указывается обязательно самым первым аргументом.

После указания режима работы обязательно указывается адрес для SRT протокола, базовый локальный адрес выглядит следующим образом: "srt://127.0.0.1:4200?transtype=messageapi=1", где transtype это «тип передачи», то есть заранее указывается то, что

мы будем передавать по этому адресу, в ходе работы основной тип который будет интересовать это messageapi, то есть байтовая передача, также можно указывать тип file.
Следует заметить то, что при двунаправленной передаче в режиме forward обязательно указываются два адреса

После указания двух обязательных аргументов в любом порядке можно указывать не обязательные параметры работы приложения, в тестовом приложении есть следующие дополнительные параметры исполнения:

--msgsize Размер сообщения для отправки, по умолчанию предел не установлен.
--sendrate Битрейт для генерации, по умолчанию 0 - нет предела
--num Сколько раз отправить, по умолчанию -1 - бесконечно много за промежуток времени
--duration Длина промежутка времени отправки пакетов, по умолчанию 0 - нет предела
--lossrate Сколько необходимо терять данных в потоке по умолчанию 0 - нет предела потери
--reconnect, !--no-reconnect - Автоматическое переподключение
--close-listener - Закрыть принимающую сторону, как только будет получен запрос

Пример команд вызова приложения из под командной строки:

datacontrol generate "srt://127.0.0.1:4200?transtype=file&messageapi=1" --msgsize 1456 --num 1000

datacontrol receive "srt://127.0.0.1:4200?transtype=messageapi=1"

datacontrol forward "srt://127.0.0.1:4200?transtype= messageapi=1" "srt://127.0.0.1:4200?transtype= messageapi=1"

3.3 Программные компоненты приложения

Приложение состоит из компонентов, которые и определяют его функционал

Базовые обязательные компоненты программы:

Socket – Класс для работы с сокетами, включая создание, привязку и соединение
SRT Receiver – Класс для получения данных
SRT Sender – Класс для отправки данных
StreamManager – Класс для управления потоками данных и их распределением

Дополнительные компоненты:

FileReceiver и FileSender – Классы для перевода файлов в байты и их передачи/получения массива байтов, составлявших файл
PingReceiver и PingSender – Классы для отправки ряда байтов/их получения
TwoWayCommunication – Класс для двунаправленной передачи данных

3.4 Сборка приложения

Требования для сборки:

C++ 17.0 и выше
OpenSSL
CMake
Pthreads
После чего приложение собирается согласно инструкции изложенной на GitHub проекта:
https://clonexy700.github.io/SRTdatacontroller/installguide/

Установка
На Windows:
```
git clone https://github.com/microsoft/vcpkg.git
cd vcpkg
.\bootstrap-vcpkg.bat
\.vcpkg install pthreads --triplet x64-windows
```

```
\.vcpkg install openssl --triplet x64-windows
\.vcpkg integrate install
cd SRTdatacontroller
md _build
cd _build
cmake ../ -
DCMAKE_TOOLCHAIN_FILE=SRTdatacontroller\vcpkg\scripts\buildsystems\
vcpkg.cmake
cmake --build ./ --config Release
```

После чего исходный код будет скомпилирован, будет создана директория «Release», которая будет содержать .exe исполнительный файл программы отладки.

4. Приложение

Здесь описывается структура и код программы отладки

File-forward.cpp

createNode: Эта функция создает узел SRT с заданной конфигурацией и URI. Она настраивает узел в зависимости от того, является ли он вызывающим или слушающим.
forwardMessage: Эта функция пересылает сообщения от исходного узла к целевому. Она непрерывно принимает сообщения от исходного узла, проверяет наличие ошибок и затем пересылает их целевому узлу. Также обрабатывает логирование ошибок.
startForwarding: Эта функция настраивает процесс пересылки. Она создает узлы исходного и целевого узлов, устанавливает соединения и запускает два потока для двунаправленной пересылки (**thSrcToDst** и **thDstToSrc**).
run: Эта функция непрерывно запускает процесс пересылки до тех пор, пока флаг **forceBreak** не установлен в значение true.
addSubcommand: Эта функция добавляет подкоманду в интерфейс командной строки (CLI) для конфигурации параметров пересылки.

File-receive.cpp

createNode:
Создает узел SRT с заданными параметрами конфигурации и URI.
В зависимости от значения **isCaller** устанавливает режим узла: вызывающий или слушающий.
Эта функция использует объект **UriParser** для разбора URI и настройки соответствующих параметров.
forwardMessage:
Пересылает сообщения от одного узла к другому.
Непрерывно принимает сообщения от исходного узла, проверяет их и пересылает целевому узлу.
Обрабатывает возможные ошибки в процессе передачи.
startForwarding:
Устанавливает начальные параметры для передачи файлов:
Создает и настраивает узлы для исходного и целевого узлов
Устанавливает соединение между ними.
Запускает два потока для двунаправленной передачи сообщений между узлами.
run:
Непрерывно запускает процесс передачи файлов, пока не будет установлен флаг **forceBreak**.
addSubcommand:
Добавляет подкоманду в интерфейс командной строки для настройки параметров передачи файлов.

File-send.cpp
sendFile:
Открывает указанный файл для чтения в бинарном режиме.
Последовательно читает содержимое файла блоками и отправляет их через соединение SRT.
Выводит информацию о ходе передачи и результате.
readDirectory:
Рекурсивно сканирует указанную директорию и составляет список файлов.
Возвращает вектор строк с путями к файлам.

relativePath:
Возвращает относительный путь файла относительно указанной директории.
startFileSender:
Запускает передачу файлов в отдельном потоке.
Получает устанавливает соединение и передает его в функцию отправки файлов.
Запускает поток для логирования статистики передачи.
run:
Выполняет отправку файлов в указанную директорию по указанному URL.
Выводит ошибку, если отправка не удалась.
addSubcommand:
Добавляет подкоманду в интерфейс командной строки для отправки файлов.
Принимает опции для настройки параметров отправки, таких как путь к файлу, размер сегмента, формат статистики и другие.

Generate.cpp

Определения:
Определение LOG_SC_GENERATE для использования в логировании.
run_pipe:
Отправляет данные на заданный сокет.
Генерирует данные для отправки и отправляет их через указанный сокет.
Логирует скорость отправки данных.
Обрабатывает возможные исключения при отправке данных.
run:
Запускает процесс генерации и отправки данных на указанные адреса назначения.
Использует функцию common_run для обработки отправки данных на несколько адресов назначения.
add_subcommand:
Добавляет подкоманду в интерфейс командной строки для настройки параметров генерации и отправки данных.

Принимает опции, такие как размер сообщения, скорость отправки, количество сообщений, продолжительность отправки, управление переподключением и т. д.

misc.cpp

create_connection:
Создает и возвращает сокет для соединения на основе переданных URL-адресов.
Поддерживает несколько URL-адресов и возможность группового соединения.
Поддерживает протокол SRT
Обрабатывает возможные исключения при создании соединения.
common_run:
Запускает процесс установления и управления соединениями.
Позволяет многократно устанавливать и переподключаться к соединениям в зависимости от настроек.
Включает в себя возможность логирования статистики о сокетах при необходимости.
create_addr:
Создает структуру netaddr_any на основе переданного имени хоста и порта.
Поддерживает разрешение имени хоста в IP-адреса.

receive.cpp

traceMessage:
Выводит информацию о принятом сообщении, его длине и идентификаторе соединения.
Это вспомогательная функция для отладки.
metricsWritingLoop:
Записывает метрики о приеме данных в файл или выводит их в лог.
Вызывается в отдельном потоке с определенной частотой.
runPipe:
Основная функция для приема данных.
Читает данные из сокета, обрабатывает их и, при необходимости, отправляет ответное сообщение.

Запускает цикл приема данных, обрабатывая возможные исключения.

srtdatacontroller::receive::run:

Запускает процесс приема данных на основе переданных адресов и настроек.

Включает в себя механизм переподключения и возможность записи метрик.

srtdatacontroller::receive::add_subcommand:

Добавляет подкоманду для приема данных в CLI-интерфейс.

Определяет параметры командной строки для настройки приема данных.

route.cpp

srtdatacontroller::route::route:

Маршрутизирует данные от источника к назначению и обратно.

Читает данные из источника, записывает их в назначение.

Работает в цикле до получения сигнала прерывания.

srtdatacontroller::route::run:

Запускает маршрутизацию данных между источником и назначением.

Создает сокеты для источника и назначения.

Запускает фоновый поток для обратной маршрутизации, если активирована функция двунаправленного соединения

Обрабатывает исключения при работе с сокетами.

srt_socket.cpp

Подключение заголовочных файлов: Код включает необходимые заголовочные файлы для работы с различными функциями и структурами, такими как assert.h, iostream, chrono, future и другие. Также он включает заголовочные файлы из собственных модулей проекта, таких как misc.hpp, uriparser.hpp, srt_node.hpp и другие.

Пространства имен: Используются пространства имен std и srtdatacontroller.

print_ts: Это вспомогательная функция, которая форматирует текущее время с точностью до микросекунды.

Конструктор SrtSocket: Принимает объект UriParser, который содержит информацию об адресе и порте источника, и инициализирует узел SRT. В конструкторе происходит вызов **srt_startup**() для инициализации библиотеки SRT и создание двух экземпляров epoll для обработки подключений и приема данных.

Деструктор ~SrtSocket: Освобождает ресурсы, связанные с узлом SRT, закрывая сокет и ожидая завершения потока, если он был запущен.

Метод AcceptConnection: Асинхронно запускает поток, который ожидает новые соединения.

Метод AcceptingThread: Этот метод представляет поток, который асинхронно принимает новые соединения. Он использует srt_epoll_wait для ожидания событий на сокете, который слушает новые соединения. После принятия нового соединения вызывается метод AcceptNewClient, который конфигурирует сокет и возвращает его.

Метод AcceptNewClient: Принимает новое соединение на сокете, который прослушивает новые соединения, и конфигурирует его с помощью метода ConfigureAcceptedSocket.

Метод ConfigureAcceptedSocket: Конфигурирует параметры сокета после его принятия. Например, устанавливает параметры для блокировки и разблокировки синхронизации передачи данных.

Метод ConfigurePre: Предварительная конфигурация сокета перед установкой соединения. Устанавливает параметры для неблокирующего режима приема и блокирующего режима отправки данных.

Метод EstablishConnection: Метод для установки соединения. Создает сокет, конфигурирует его, и либо подключается к удаленному хосту, либо ожидает подключений от удаленных хостов.

5. ЗАКЛЮЧЕНИЕ

В ходе работы был разработан инструмент для отладки на основе протокола SRT, а также изучены материалы по данному протоколу и сокетам в c++. Итоговые результаты выложены на GitHub. В дальнейшем планируется продолжение разработки, в будущем будет разработана полноценная библиотека динамического перераспределения потоков мультимедиа данных.

СПИСОК ЛИТЕРАТУРЫ

Haivision. (2017). Secure Reliable Transport (SRT): Technical Overview.

SRT Alliance. (n.d.). SRT Protocol Documentation

Lu, Y., Peha, J. M., & Matsuura, K. (2018). Secure and Reliable Transport Protocols for Multimedia Streaming over the Internet. IEEE Communications Magazine, 56(11), 82-89. doi:10.1109/MCOM.2018.1700879

Wowza. (2020). SRT vs. RTMP: Which Protocol Should You Use?

Cisco. (2019). Cisco Visual Networking Index: Forecast and Trends, 2018–2023.

OBS Studio. (2020). Setting Up SRT for High-Quality Streaming in OBS.