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, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Протокол бессерверных конференций»

Студент ИУ7-31М (Группа) _____ (Подпись, дата) Клименко А. К. (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) Никульшин А. М. (И. О. Фамилия)

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является проектирование протокола проведения бессерверных аудио видео конференций. Для достижения поставленной цели необходимо решить следующие задачи:

— изучить схожие протоколы;

— спроектировать протокол бессерверных конференций;

— реализовать ПО осуществляющее взаимодействие по разработанному протоколу;

— протестировать ПО.

1 Аналитический раздел

1.1 Актуальность

Одной из наиболее актуальных задач в современном мире является обеспечение эффективного взаимодействия между удаленными пользователями. Бессерверные аудио-видео конференции представляют собой перспективное решение, позволяющее организовывать виртуальные встречи без необходимости использования централизованных серверов.

Традиционные подходы к организации конференц-связи, основанные на клиент-серверной архитектуре, сталкиваются с рядом ограничений. Во-первых, они требуют наличия мощной серверной инфраструктуры, которая должна обрабатывать весь трафик участников, что приводит к высоким эксплуатационным расходам. Во-вторых, централизованные системы являются уязвимыми к сбоям и атакам, так как отказ сервера может привести к полной потере связи между пользователями. В-третьих, такие решения зачастую ограничены в масштабируемости, что затрудняет их применение в крупномасштабных распределенных средах.

Бессерверные аудио-видео конференции, в свою очередь, обладают рядом преимуществ. Они позволяют организовывать виртуальные встречи напрямую между участниками, без необходимости использования централизованных серверов. Это повышает отказоустойчивость системы, поскольку отказ одного из клиентов не приводит к полной потере связи. Кроме того, данный подход способствует снижению эксплуатационных расходов и упрощает процесс масштабирования.

Разработка протокола для организации бессерверных аудио-видео конференций является важной научно-технической задачей, имеющей высокую практическую значимость. Результаты исследования могут найти применение в различных областях, таких как дистанционное обучение, удаленная работа, проведение виртуальных совещаний и конференций, а также во многих других сценариях, требующих эффективного взаимодействия между удаленными участниками.

1.2 Потоковая передача данных в реальном времени

Говоря о аудио-видео конференциях, следует выделить две основные составляющие потоков данных – аудио поток и видео поток. Аудио поток, как правило, имеет более высокий приоритет по сравнению с видео потоком, поскольку бесперебойная передача звука является критически важной для обеспечения качественного взаимодействия участников.

Основными протоколами транспортного уровня, используемыми при передаче данных в режиме реального времени, являются TCP, UDP. На прикладном уровне часто используются такие протоколы как RTP (Real-Time Transport Protocol [1]), RTCP (Real-Time Transport Control Protocol [1]), SIP (Session Initiation Protocol [2]), H.323 [3].

RTP и RTCP обеспечивают передачу аудио и видео с минимальной задержкой, но не предоставляют встроенных механизмов шифрования. SIP и H.323, напротив, поддерживают управление сессиями и шифрование, но могут быть более сложными в реализации. На основе этого анализа было принято решение рассмотреть два подхода: TCP+TLS и SRTP,

как наиболее подходящие для бессерверных конференций. Рассмотрим преимущества и недостатки каждого из подходов:

Преимущества использования TCP+TLS:

— Надежность и безопасность — TCP гарантирует доставку данных без потерь, а TLS обеспечивает шифрование и аутентификацию, что защищает данные от перехвата и подделки.

— Развитая инфраструктура — TCP и TLS широко поддерживаются, существует множество библиотек и инструментов для их реализации, что упрощает разработку и интеграцию.

Недостатки:

— Более высокая задержка — TCP имеет больше накладных расходов по сравнению с UDP-протоколами, что может повлиять на качество аудио и видео.

— Более высокие требования к ресурсам — TCP-стек требует больше памяти и вычислительных ресурсов, что может быть критично для устройств с ограниченными возможностями, такими как мобильные устройства или встраиваемые системы.

Преимущества использования SRTP:

— Низкая задержка — SRTP, построенный поверх UDP, имеет меньше накладных расходов, что позволяет обеспечить более высокое качество аудио и видео.

— Меньшее потребление ресурсов — SRTP более эффективен в плане использования памяти и вычислительной мощности, что важно для мобильных устройств.

— Специализация на мультимедиа — SRTP разработан специально для защищенной передачи аудио и видео данных, в отличие от более универсального TCP+TLS.

Недостатки:

— Меньшая надежность — UDP, лежащий в основе SRTP, не гарантирует доставку данных, что может привести к потере пакетов в неблагоприятных сетевых условиях.

— Сложность реализации — SRTP требует реализации дополнительных механизмов для управления ключами шифрования, что усложняет разработку.

1.3 Распределение вычислений

При централизованном проведении конференций основная нагрузка ложится на сервер, в то время как при проведении бессерверных конференций она распределяется по клиентам. Для обеспечения наилучшего QoS необходимо учитывать вычислительные возможности каждого клиента, а также пропускную способность каналов, объединяющих их.

В бессерверной архитектуре каждый участник конференции выступает в качестве автономного узла, ответственного за обработку и передачу мультимедийного контента. Это требует от клиентских устройств достаточной вычислительной мощности для кодирования, декодирования и синхронизации аудио, видео и данных в режиме реального времени.

$$Perf_{client} = f(CPU, GPU, RAM) \tag{1.1}$$

Пропускная способность каналов связи также играет критическую роль, определяя максимально возможное качество и разрешение передаваемых потоков. Недостаточная пропускная способность может привести к снижению качества и стабильности соединения, а также к возникновению задержек и потерь пакетов.

$$Bandwidth_{channel} = f(network\ speed, distance, noises) \tag{1.2}$$

Для достижения оптимального QoS в бессерверных конференциях необходимо динамически адаптировать параметры кодирования и передачи мультимедийных потоков к возможностям каждого участника. Это может быть реализовано с помощью механизмов согласования характеристик сессии и адаптивного управления скоростью передачи данных.

1.4 Микширование

При большом числе участников в бессерверных видеоконференциях имеет смысл применять микширование передаваемых видео потоков. Это позволяет уменьшить общий объем передаваемых данных, снизив нагрузку на каналы связи, но при этом требует значительных вычислительных ресурсов от клиентских устройств.

Процесс микширования видео можно представить в виде следующих основных шагов:

1. Получение видео кадров от соседних узлов.
2. Декодирование полученных кадров.
3. Композиция единого кадра путем пространственного объединения (склеивания) исходных кадров.
4. Кодирование скомпонованного кадра.
5. Передача закодированного кадра всем соседним узлам.

При этом, на этапе формирования общего кадра необходимо учитывать, нужен ли тот или иной видео поток каждому из соседних участников. Это позволяет оптимизировать состав композитного кадра и снизить вычислительную нагрузку.

Микширование аудио потоков может быть реализовано схожим образом, с формированием раздельных аудио каналов для каждого участника конференции. Это требует дополнительных вычислений, связанных с микшированием, кодированием и передачей аудио данных.

Эффективность микширования определяется соотношением выигрыша в объеме передаваемых данных и затратами вычислительных ресурсов на клиентских устройствах. Для поддержания высокого качества и низкой задержки передачи мультимедийного контента необходим тщательный подбор алгоритмов микширования и кодирования с учетом возможностей участников конференции.

1.5 Устойчивость конференции

Отключение ключевого участника конференции, через которого проходит большое число потоков данных от других участников, является одной из проблем распределенных вычислений. Решением этой проблемы может послужить механизм, обеспечивающий поддержание «запасных» каналов передачи между клиентами.

1.6 Защита информации

Для защиты конференции необходимо использовать защищенные каналы передачи. На каждое соединение инициализируется собственная сессия, вместо того, чтобы использовать одну сессию на всех участников конференции.

Плюсы такого подхода:

— Повышение безопасности за счет использования индивидуальных криптографических ключей для каждого соединения.

— Улучшение масштабируемости системы, так как каждое соединение обрабатывается независимо.

— Более гибкое управление доступом и привилегиями участников конференции.

Минусы такого подхода:

— Увеличение вычислительной нагрузки на клиентские устройства, связанное с необходимостью поддержания множества криптографических сессий.

— Более высокие требования к пропускной способности канала связи, обусловленные необходимостью передачи дополнительных данных, связанных с криптографической защитой.

В программной реализации протокола применяется библиотека OpenSSL [4] для осуществления защищенных соединений. Использование OpenSSL позволяет обеспечить совместимость с широким спектром криптографических алгоритмов, а также реализовать такие механизмы безопасности, как аутентификация, шифрование и целостность передаваемых данных. Кроме того, OpenSSL предоставляет средства для управления криптографическими ключами, что упрощает интеграцию защищенных соединений в общую архитектуру системы.

1.7 Особенности NAT

Одной из ключевых проблем при реализации бессерверных видеоконференций является преодоление ограничений, связанных с трансляцией адресов. NAT используется для преобразования частных IP-адресов в публичные и наоборот, что позволяет нескольким устройствам в локальной сети использовать один публичный IP-адрес для выхода в интернет. Однако NAT создает сложности для прямого соединения между участниками, находящимися за разными NAT, так как они не могут напрямую обмениваться данными без дополнительных механизмов.

Выделяют следующие типы реализаций NAT по способу обработки UDP пакетов [5]:

— *Конус (Full Cone)*. Все запросы с одного и того же внутреннего IP адреса и порта отображаются на один и тот же внешний IP адрес и порт. Любой внешний хост может отправить пакет внутреннему хосту, отправляя его на отображенный врешний IP адрес.

— *Ограниченный конус (Restricted Cone)*. То же, что и конус, за исключением того, что внешний хост (с IP адресом X) может отправить пакет внутреннему хосту только если ранее внутренний хост отправлял пакет внешнему хосту по IP адресу X.

— *Конус, ограниченный портами (Port Restricted Cone)*. То же, что и ограниченный конус, но включая ограничение на порты — внешний хост (с IP адресом X и портом P) может отправить пакет внутреннему хосту только если ранее внутренний хост отравлял пакет внешнему хосту по IP адресу X и порту P.

— *Симметричный (Symmetric)*. Все запросы с одного и того же внутреннего IP адреса и порта к одному и тому же IP адресу и порту хоста назначения отображаются на один и тот же внешний IP адрес и порт. Если один хост отправляет пакет с тем же IP адресом и портом источника, но с другим адресом назначения, то используется другое отображение. Более того, только внешний хост, получивший пакет, может отправить UDP пакет обратно источнику, находящемуся за NAT.

Для преодоления ограничений NAT в бессерверных конференциях используются следующие подходы:

1. STUN (Session Traversal Utilities for NAT). STUN — это протокол, который позволяет участникам, находящимся за NAT, определить свой публичный IP-адрес и порт, назначенный NAT для исходящего соединения. Это позволяет участникам обмениваться этой информацией и устанавливать прямое соединение.

   Принцип работы STUN заключается в том, что каждый участник отправляет запрос на STUN-сервер, который возвращает ему его публичный IP-адрес и порт. Затем они обмениваются этой информацией через сигнальный канал. После обмена информацией участники пытаются установить прямое соединение через публичные IP-адреса и порты.

STUN Позволяет установить прямое соединение между участниками, находящимися за NAT. Не требует дополнительных серверов для ретрансляции данных. Однако, он не работает с симметричным NAT.

2. TURN (Traversal Using Relays around NAT). TURN — это протокол, который позволяет участникам, находящимся за NAT, использовать ретрансляционный сервер для передачи данных, если прямое соединение невозможно. TURN-сервер выступает в качестве посредника, пересылая данные между участниками.

   Как работает TURN: участники устанавливают соединение с TURN-сервером и получают от него публичный IP-адрес и порт для ретрансляции данных, после чего обмениваются этой информацией через сигнальный канал. Если прямое соединение невозможно, данные передаются через TURN-сервер.

   TURN работает с любыми типами NAT, включая симметричный и обеспечивает надежную передачу данных даже в сложных сетевых условиях. Но данный подход требует наличия TURN-сервера, что увеличивает затраты на инфраструктуру, а также вводит дополнительную задержку из-за ретрансляции данных через сервер.

3. ICE (Interactive Connectivity Establishment). ICE — это технология, которая объединяет STUN и TURN для установления соединения между участниками. ICE автоматически выбирает оптимальный способ соединения: прямое (через STUN) или через ретрансляционный сервер (TURN).

   При использовании ICE участники собирают все возможные кандидаты для соединения (локальные IP-адреса, публичные IP-адреса через STUN, ретрансляционные адреса через TURN) и обмениваются списками кандидатов через сигнальный канал. ICE проверяет каждый кандидат и выбирает оптимальный путь для соединения.

   Преимущества: автоматический выбор лучшего способа соединения, поддержка всех типов NAT, включая симметричный. Ограничения: требуются интеграции STUN и TURN серверов, увеличенная сложность реализации.

4. UPnP (Universal Plug and Play). UPnP — это протокол, позволяющий устройствам автоматически настраивать проброс портов на NAT-устройствах (например, роутерах). Если участник поддерживает UPnP, он может автоматически открыть порт для входящих соединений.

   Как работает UPnP: участник отправляет запрос на роутер через UPnP для открытия порта; роутер открывает порт и назначает его для входящих соединений; участник использует этот порт для приема данных от других участников.

   UPnP позволяет участникам принимать входящие соединения без ручной настройки NAT и упрощает установление прямых соединений. Однако не все роутеры его поддерживают, а также он может быть отключен по соображениям безопасности.

5. Использование P2P-технологий с NAT-пробросом. В некоторых случаях можно использовать P2P-технологии, такие как UDP hole punching, которые позволяют

участникам устанавливать прямое соединение через NAT без использования промежуточных серверов.

Участники обмениваются информацией о своих публичных IP-адресах и портах через сигнальный канал, затем одновременно отправляют UDP-пакеты друг другу, что позволяет установить прямое соединение через NAT.

Данный способ не требует дополнительных серверов для ретрансляции данных и подходит для большинства типов NAT, кроме симметричного, однако для его работы требуется точная синхронизация между участниками.

2 Конструкторский раздел

2.1 Функциональные требования к протоколу

Ниже перечислены функциональные требования к разрабатываемому протоколу.

1. Протокол должен поддерживать проведение одной конференции для произвольного количества участников.
2. Участники должны иметь возможность подключаться и отключаться от конференции во время ее проведения.

2.2 Роли участников конференции

Протокол предусматривает наличие следующих ролей для участников конференции:

— **Организатор** (0) — роль участника, который первым отправил приглашение для данной конференции. Организатор имеет право отправлять приглашения и отключать участников от конференции.

— **Слушатель** (1) — роль, назначаемая по-умолчанию всем остальным участникам конференции кроме организатора.

Обеспечение запрета приглашения слушателями осуществляется с помощью маркировки приглашения. При создании конференции организатор формирует идентификатор конференции следующим образом:

$$id_{\text{конф.}} = \text{MD5}(id_{\text{орг.}} + T_{\text{старт.}})[0:8], \tag{2.1}$$

где id_x — идентификаторы, $T_{\text{старт.}}$ — время старта конференции в миллисекундах, $MD5$ — хеш-функция, $X[0:n]$ — операция взятия первых (младших) n байт от X.

Таким образом, идентификатор конференции будет отражать фактического организатора и приглашения можно проверять на предмет достоверности путем пересчета хеш-функции.

2.3 Типы сообщений

Разрабатываемый протокол предусматривает следующие типы сообщений:

— `INVITE` (0) — приглашение в конференцию. Получатель сообщения подключается к конференции, в которой участвует отправитель.

— `INVITE_ACCEPT` (1) — участник принимает приглашение.

— `INVITE_REJECT` (2) — участник отклоняет приглашение.

— `PART_PRESENCE` (3) — подключение/отключение участников конференции.

— `PART_INFO` (4) — изменение состояния отдельного участника конференции.

— `REENTER` (5) — отправитель намеревается подключиться к существующей конференции (после потери соединения).

— `LEAVE` (6) — отправитель намеривается покинуть конференцию. Получатели должны финализировать все ресурсы, связанные с этим участником и разорвать соединения. Без использования данного типа сообщения будет происходить ожидание переподключения участника.

— `TEXT` (7) — текстовое сообщение.

— `AUDIO` (8) — сообщение с аудио пакетом.

— `VIDEO` (9) — сообщение с видео пакетом.

Для передачи данных между участниками используется бинарный формат сообщений.

Схема типов сообщений представлена в таблицах 2.1—2.10. Первым полем любого сообщения передается его размер, затем поле с типом сообщения, в зависимости от которого происходит анализ содержимого сообщения.

Идентификаторы участников представляются 8 байтными беззнаковыми целыми и генерируются случайным образом. Идентификатор участника генерируется при инициализации участника и закрепляется за ним при переходах между конференциями. Идентификатор конференции генерируется каждый раз при старте конференции на основе формулы 2.1.

Таблица 2.1 – Структура сообщения типа `INVITE`

Поле	**Размер**	**Описание**
`size`	8 байт	Размер сообщения, равен 36 + N
`type`	4 байт	Тип сообщения, равен 0
`conf_id`	8 байт	Идентификатор конференции
`part_id`	8 байт	Идентификатор приглашающего участника
`conf_start_ts`	8 байт	Временная метка UTC начала конференции
`part_name`	N байт	Строковое имя приглашающего участника

Таблица 2.2 – Структура сообщения типа `INVITE_ACCEPT`

Поле	**Размер**	**Описание**
`size`	8 байт	Размер сообщения, равен 20 + N
`type`	4 байт	Тип сообщения, равен 1
`part_id`	8 байт	Идентификатор присоединившегося участника
`part_name`	N байт	Строковое имя приглашенного участника

Таблица 2.3 – Структура сообщения типа `INVITE_REJECT`

Поле	**Размер**	**Описание**
`size`	8 байт	Размер сообщения, равен 12
`type`	4 байт	Тип сообщения, равен 2

Таблица 2.4 – Структура сообщения типа `PART_PRESENCE`

Поле	**Размер**	**Описание**
`size`	8 байт	Размер сообщения, равен 28
`type`	4 байт	Тип сообщения, равен 3
`part_id`	8 байт	Идентификатор участника
`part_role`	4 байт	Роль участника
`state`	4 байт	0 — участник присоединился; 1 — иначе

Таблица 2.5 – Структура сообщения типа `PART_INFO`

Поле	**Размер**	**Описание**
`size`	8 байт	Размер сообщения, равен 24 + N
`type`	4 байт	Тип сообщения, равен 4
`part_id`	8 байт	Идентификатор участника
`part_role`	4 байт	Новая роль участника
`part_name`	N байт	Новое строковое имя участника

Таблица 2.6 – Структура сообщения типа `REENTER`

Поле	**Размер**	**Описание**
`size`	8 байт	Размер сообщения, равен 28
`type`	4 байт	Тип сообщения, равен 5
`conf_id`	8 байт	Идентификатор конференции
`part_id`	8 байт	Идентификатор перезаходящего участника

Таблица 2.7 – Структура сообщения типа `LEAVE`

Поле	**Размер**	**Описание**
`size`	8 байт	Размер сообщения, равен 20
`type`	4 байт	Тип сообщения, равен 6
`part_id`	8 байт	Идентификатор уходящего участника

Таблица 2.8 – Структура сообщения типа `TEXT`

Поле	**Размер**	**Описание**
`size`	8 байт	Размер сообщения, равен 20 + N
`type`	4 байт	Тип сообщения, равен 7
`part_id`	8 байт	Идентификатор участника–источника
`text`	N байт	Текстовое сообщение

Таблица 2.9 – Структура сообщения типа `AUDIO`

Поле	**Размер**	**Описание**
`size`	8 байт	Размер сообщения, равен 20 + N
`type`	4 байт	Тип сообщения, равен 8
`part_id`	8 байт	Идентификатор участника–источника
`packet`	N байт	Аудио пакет

Таблица 2.10 – Структура сообщения типа `VIDEO`

Поле	**Размер**	**Описание**
`size`	8 байт	Размер сообщения, равен 20 + N
`type`	4 байт	Тип сообщения, равен 9
`part_id`	8 байт	Идентификатор участника–источника
`packet`	N байт	Видео пакет

2.4 Медиа форматы

Снижение объемов передачи аудио/видео данных по сети требует использования кодеков, сжимающих медиа данные. Для разрабатываемого протокола применяются кодеки AAC [6] и H.264 [7]. Аудио кадр кодируется в стерео формате с частотой 48кГц. Видео кадр кодируется в разрешении 320 на 180 пикселей и частотой в 30 кадров в секунду.

Закодированный аудио/видео кадр записывается в поле `packet` сообщений `AUDIO`/`VIDEO`.

2.5 Взаимодействия сторон

Рассмотрим сценарий с двумя участниками — «А» и «Б». При создании конференции ей назначается случайный идентификатор. Для организации конференции, участник «А» отправляет участнику «Б» сообщение типа `INVITE`, в котором записывается идентификатор конференции, а также идентификатор и имя приглашающего участника.

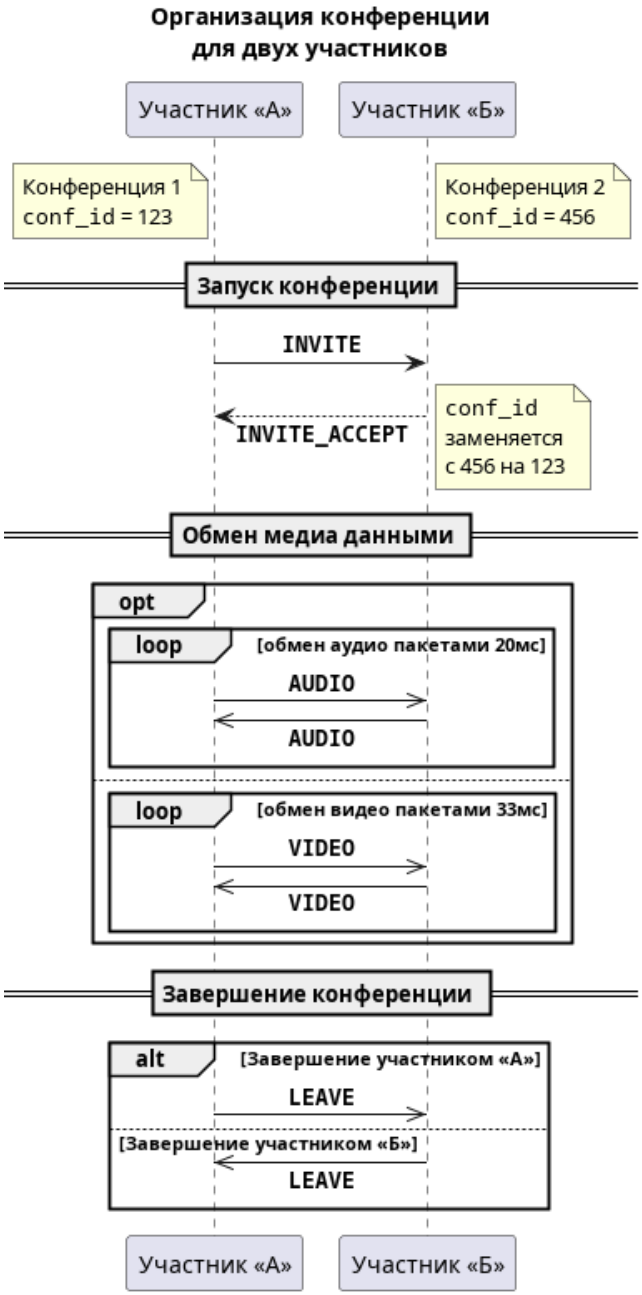


Рисунок 2.1 – Диаграмма последовательности для организации конференции для двух участников

После чего участник «Б» должен ответить на него сообщением `INVITE_ACCEPT` для

подключения к конференции участника «А», при этом он покидает свою текущую конференцию, если ее идентификатор не совпадает с идентификатором конференции в приглашении. После успешного подключения, происходит инициализация каналов передачи медиа данных.

Аудио и видео пакеты передаются с использованием сообщений типов `AUDIO` / `VIDEO`. Диаграмма последовательности описанного процесса представлена на рисунке 2.1.

Подключение последующих участников происходит аналогичным образом — любой из активных участников конференции отправляет сообщение типа `INVITE` с приглашением в конференцию новому участнику, и, после получения ответного сообщения `INVITE_ACCEPT`, отправляет сообщение `PART_PRESENCE` остальным участникам конференции для уведомления о подключении нового участника.

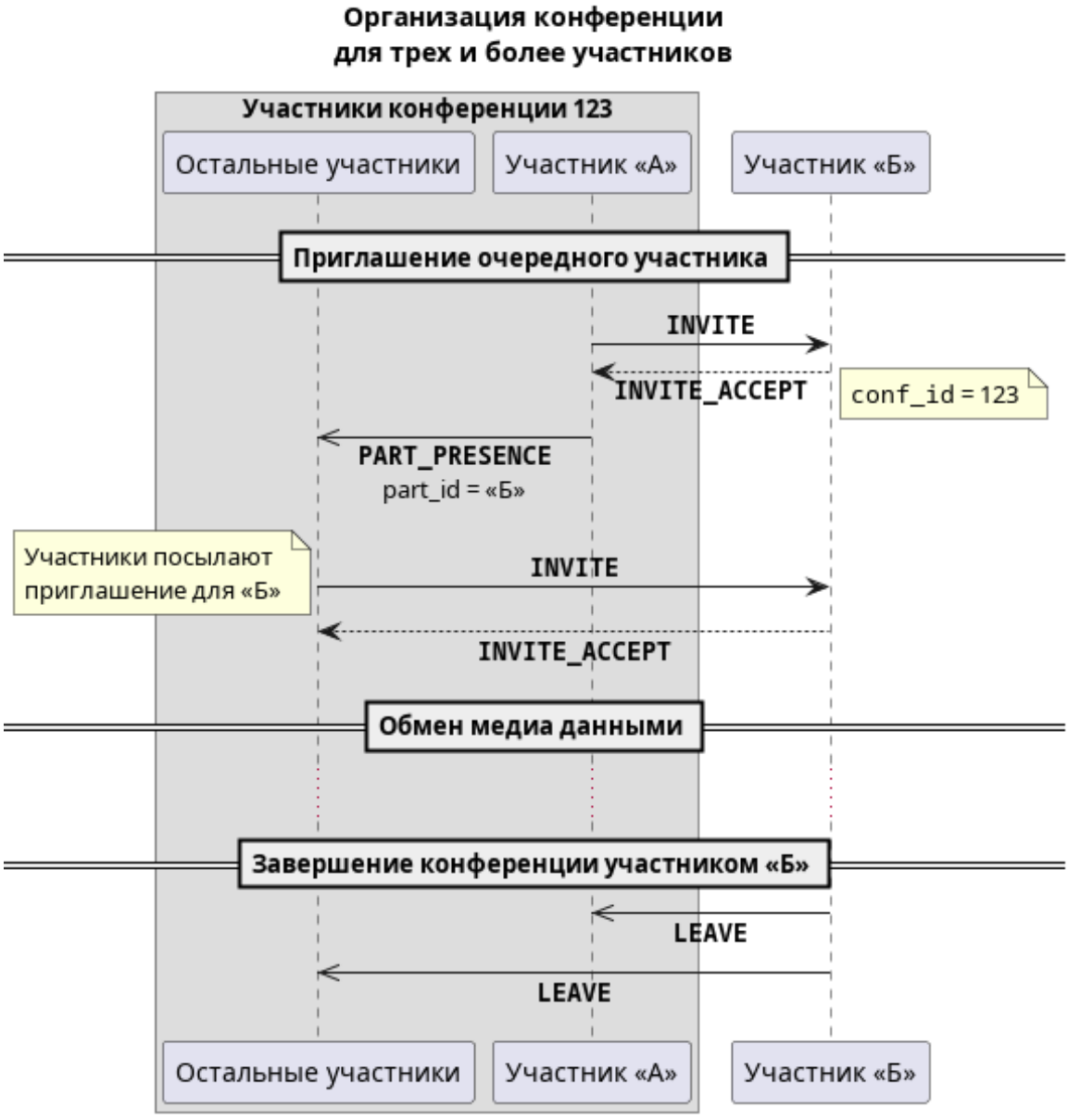


Рисунок 2.2 – Диаграмма последовательности для организации конференции для трех и более участников

Каждый из остальных участников, получив сообщение `PART_PRESENCE` отправляет приглашение `INVITE` новому участнику для инициализации каналов связи между всеми парами участников конференции. Диаграмма последовательности описанного процесса представлена на рисунке 2.2.

Для обеспечения уникальной идентификации участников в условиях, когда несколько клиентов могут находиться за NAT с одним IP-адресом, в протоколе используются уникальные идентификаторы участников. Эти идентификаторы генерируются случайным образом при подключении участника к конференции и используются для однозначной идентификации в рамках сессии.

2.6 Шифрование

Шифрование используется для обеспечения целостности и конфиденциальности передаваемых данных. (Защита от подслушивания, порчи и/или подделки сообщений). Проверка сертификатов разрабатываемым протоколом не предусмотрена, но может быть проведена при повторном соединении с участником. В качестве протокола шифрования используется TLS v1.3 [8].

2.7 Переподключение

При потере соединения без получения сообщения `LEAVE` небходимо ожидать переподключение участника в течении 30 секунд. Если участник так и не подключился к конференции, то он считается вышедшим из конференции и в последующем сможет подключиться к ней только через приглашение от другого участника конференции.

Переподключение производится отправкой сообщения `REENTER` с идентификатором текущей конференции и идентификатором участника, потерявшего соединение. Если сообщение было получено по истечению 30 секундного интервала — оно отбрасывается, соединение закрывается и участник считывается неподключившимся.

2.8 Определение качества каналов связи

Для реформирования топологии связей в бессерверных видео конференциях необходимо определить качество существующих каналов связи между участниками. Это может быть выполнено с использованием метрик, которые зависят от следующих параметров: временные метки отправки/приема аудио/видео пакета; размер пакета в байтах; количество переданных и повторно переданных пакетов.

На основе этих параметров могут быть вычислены такие показатели качества как:

— задержка передачи — временная разница между отправкой и получением пакета;

— джиттер — вариация задержки между последовательными пакетами;

— пропускная способность — объем данных, переданных за определенный период времени.

3 Технологический раздел

3.1 Средства разработки

Для реализации поддерживающего спроектированный протокол ПО был выбран язык C. В качестве средства сборки используется средство сборки CMake [9].

Для работы с медиа данными используется библиотека с открытым исходным кодом `ffmpeg` [10].

Шифрование сообщений осуществляется с использованием библиотеки `OpenSSL`.

Для автотестов используется фреймворк `GoogleTest` [11].

3.2 Листинги процедуры приглашения

Листинг 3.1 — Процедура приглашения со стороны приглашающего

```
static enum SError do_handshake_client(struct SConnection *con,
    struct SMessage *msg, struct SMsgInviteAccept **acceptMsg) {
  enum SError err = sconn_send(con, msg);
  if (err != SELECON_OK)
    return err;
  err = sconn_recv(con, &msg);
  if (err != SELECON_OK)
    message_free(&msg);
  else if (msg->type == SMSG_INVITE_ACCEPT)
    *acceptMsg = (struct SMsgInviteAccept *)msg;
  else {
    message_free(&msg);
    err = SELECON_INVITE_REJECTED;
  }
  return err;
}
```

Листинг 3.2 — Процедура приглашения со стороны приглашаемого

```
static enum SError do_handshake_srv(struct SParticipant *self,
    struct SEndpoint *listen_ep, struct SConnection *con,
    struct SMsgInvite **invite, invite_handler_fn_t handler) {
  struct SMessage *msg = NULL;
  enum SError err      = sconn_recv(con, &msg);
  if (err != SELECON_OK || msg->type != SMSG_INVITE)
    goto cleanup;
  bool accepted = handler((struct SMsgInvite *)msg);
  if (!accepted) {
    msg->size = sizeof(struct SMessage);
    msg->type = SMSG_INVITE_REJECT;
    err       = sconn_send(con, msg);
    goto cleanup;
  }
  *invite = (struct SMsgInvite *)msg;
  msg     = message_invite_accept_alloc(self->id, self->name, listen_ep);
  err     = sconn_send(con, msg);
cleanup:
  message_free(&msg);
  return err;
}
```

Листинг 3.3 — Воркер, обрабатывающий входящие приглашения в конференции

```
static void *invite_worker(void *arg) {
  struct SContext *ctx           = arg;
  struct SConnection *listen_con = NULL;
  enum SError err                = sconn_listen(&listen_con, &ctx->listen_ep);
  while (ctx->initialized && (err == SELECON_OK || err == SELECON_CON_TIMEOUT)) {
    struct SConnection *con = NULL;
    err = sconn_accept_secure(listen_con, &con, SELECON_DEFAULT_SECURE_TIMEOUT);
    if (err == SELECON_OK) {
      struct SMsgInvite *invite = NULL;
      err = do_handshake_srv(&ctx->self, &ctx->listen_ep, con, &invite,
              ctx->invite_handler);
      if (err == SELECON_OK && invite != NULL)
        err = handle_invite(ctx, con, invite);
      if (err != SELECON_OK || invite == NULL)
        sconn_disconnect(&con);
      message_free((struct SMessage **)&invite);
    }
  }
  sconn_disconnect(&listen_con);
  return NULL;
}
```

Листинг 3.4 — Процедура приглашения участника в конференцию

```
static enum SError invite_connected(struct SContext *context,
    struct SConnection *con, part_id_t *out_part_id) {
  struct SMessage *inviteMsg = message_invite_alloc(
      context->conf_id, context->conf_start_ts, context->self.id, context->self.name);
  struct SMsgInviteAccept *acceptMsg = NULL;
  enum SError err                    = do_handshake_client(con, inviteMsg, &acceptMsg);
  if (err != SELECON_OK || acceptMsg == NULL)
    return err;
  // send other participants info about invitee
  struct SMsgPartPresence *msg = message_part_presence_alloc();
  msg->part_id                 = acceptMsg->part_id;
  msg->ep                      = acceptMsg->ep;
  msg->part_role               = SROLE_CLIENT;
  msg->state                   = PART_JOIN;
  pthread_rwlock_wrlock(&context->part_rwlock);
  for (size_t i = 0; i < context->nb_participants - 1; ++i)
    sconn_send(context->participants[i].connection, (struct SMessage *)msg);
  add_participant(context, acceptMsg->id, acceptMsg->name, con);
  if (context->nb_participants == 2 && !context->conf_thread_working) {
    pthread_create(&context->conf_thread, NULL, conf_worker, context);
    pthread_setname_np(context->conf_thread, "conf");
  }
  pthread_rwlock_unlock(&context->part_rwlock);
  *out_part_id = acceptMsg->part_id;
  message_free((struct SMessage **)&msg);
  message_free((struct SMessage **)&acceptMsg);
  return SELECON_OK;
}
```

Листинг 3.5 — Процедура повторного подключения к конференции

```
enum SError selecon_reenter(struct SContext *context) {
  struct SMsgReenter *msg = message_reenter_alloc(context->conf_id, context->self.id);
  enum SError err         = SELECON_OK;
  pthread_rwlock_rdlock(&context->part_rwlock);
  for (size_t i = 0; i < context->nb_participants - 1; ++i) {
    struct SConnection *con = NULL;
    err = sconn_connect_secure(&con, &context->participants[i].listen_ep);
    if (err != SELECON_OK) {
```

```
      sconn_disconnect(&con); break;
    }
    err = sconn_send(con, (struct SMessage *)msg);
    if (err != SELECON_OK) {
      sconn_disconnect(&con); break;
    }
    err = sconn_recv(con, (struct SMessage **)&msg);
    if (err != SELECON_OK) {
      sconn_disconnect(&con); break;
    }
    // restore participant state and streams
    if (!spart_hangup_validate(&context->participants[i], con))
      err = SELECON_CON_TIMEOUT;
    else {
      sstream_id_t audio_stream = NULL;
      sstream_id_t video_stream = NULL;
      err = scont_alloc_stream(&context->streams, context->participants[i].id,
        context->conf_start_ts, SSTREAM_AUDIO, SSTREAM_INPUT, &audio_stream);
      assert(err == SELECON_OK);
      err = scont_alloc_stream(&context->streams, context->participants[i].id,
        context->conf_start_ts, SSTREAM_VIDEO, SSTREAM_INPUT, &video_stream);
      assert(err == SELECON_OK);
    }
  }
  pthread_rwlock_unlock(&context->part_rwlock);
  message_free((struct SMessage **)&msg);
  if (err != SELECON_OK) {
    // give up reentering - reset context state to new conference
    selecon_leave_conference(context);
  }
  return err;
}
```

Листинг 3.6 — Функции подключения с использованием шифрования

```
static atomic_int ssl_usage_counter = 0;
static void ssl_init(void) {
 if (atomic_fetch_add(&ssl_usage_counter, 1) == 0) {
  SSL_load_error_strings();
  OpenSSL_add_all_algorithms();
 }
}
static void ssl_destroy(void) {
 if (atomic_fetch_add(&ssl_usage_counter, -1) == 1) {
  ERR_free_strings();
  EVP_cleanup();
 }
}
static SSL_CTX *ssl_new_server_ctx(void) {
 const char *cert = cert_get_cert_path();
 const char *key  = cert_get_key_path();
 if (cert == NULL || key == NULL)
  return NULL;
 SSL_CTX *ctx = SSL_CTX_new(SSLv23_server_method());
 if (SSL_CTX_use_certificate_file(ctx, cert, SSL_FILETYPE_PEM) > 0) {
  if (SSL_CTX_use_PrivateKey_file(ctx, key, SSL_FILETYPE_PEM) > 0) {
   return ctx;
  }
 }
 SSL_CTX_free(ctx);
 return NULL;
}
static SSL_CTX *ssl_new_client_ctx(void) {
 SSL_CTX *ctx = SSL_CTX_new(SSLv23_client_method());
```

```
 SSL_CTX_set_mode(ctx, SSL_MODE_AUTO_RETRY);
 SSL_CTX_set_verify(ctx, SSL_VERIFY_NONE, NULL);
 return ctx;
}

enum SError sconn_accept_secure(struct SConnection *con,
    struct SConnection **out_con, int timeout_ms) {
  enum SError err = sconn_accept(con, out_con, timeout_ms);
  if (err != SELECON_OK)
    return err;
  ssl_init();
  if (((*out_con)->ssl_ctx = ssl_new_server_ctx()) != NULL) {
    (*out_con)->ssl = SSL_new((*out_con)->ssl_ctx);
    SSL_set_fd((*out_con)->ssl, (*out_con)->fd);
    if (SSL_accept((*out_con)->ssl) > 0)
      return SELECON_OK;
  }
  ERR_print_errors_fp(stderr);
  sconn_disconnect(out_con);
  return SELECON_SSL_ERROR;
}

enum SError sconn_connect_secure(struct SConnection **con, struct SEndpoint *ep) {
 enum SError err = sconn_connect(con, ep);
 if (err != SELECON_OK)
  return err;
 ssl_init();
 if (((*con)->ssl_ctx = ssl_new_client_ctx()) != NULL) {
  (*con)->ssl = SSL_new((*con)->ssl_ctx);
  SSL_set_fd((*con)->ssl, (*con)->fd);
  if (SSL_connect((*con)->ssl) > 0)
   return SELECON_OK;
 }
 ERR_print_errors_fp(stderr);
 sconn_disconnect(con);
 return SELECON_SSL_ERROR;
}
```

Листинг 3.7 — Функции работы с медиа потоками

```
static void stream_input_worker(struct SStream *stream) {
  int64_t pts           = 0;
  struct AVFrame *frame = av_frame_alloc();
  enum AVMediaType mtype = stream->type == SSTREAM_AUDIO
    ? AVMEDIA_TYPE_AUDIO : AVMEDIA_TYPE_VIDEO;
  struct AVPacket *packet = NULL;
  do {
    av_packet_free(&packet);
    packet  = pop_packet(stream);
    int ret = avcodec_send_packet(stream->codec_ctx, packet);
    if (ret < 0) {
      av_frame_free(&frame); break;
    }
    while (ret == 0) {
      ret = avcodec_receive_frame(stream->codec_ctx, frame);
      if (ret < 0) break;
      frame->time_base = stream->codec_ctx->time_base;
      if (mtype == AVMEDIA_TYPE_AUDIO) {
        frame->pts = frame->pkt_dts = pts;
        pts += frame->nb_samples;
      }
      stream->media_handler(stream->media_user_data, stream->part_id, mtype, frame);
      av_frame_unref(frame);
    }
```

```
    if (ret != AVERROR(EAGAIN) && ret != AVERROR_EOF)
      perror("avcodec_receive_packet");
  } while (packet != NULL);
  av_frame_free(&frame);
}

static void stream_output_worker(struct SStream *stream) {
 struct AVPacket *packet = av_packet_alloc();
 timestamp_t ts          = 0;
 while (true) {
  struct AVFrame *frame = pop_frame(stream);
  if (frame == NULL) {  // close requested
   av_packet_free(&packet); break;
  }
  int ret = mfgraph_send(&stream->filter_graph, frame);
  if (ret < 0) {
   fprintf(stderr, "mgraph_send: err = %d\n", ret);
   continue;
  }
  while ((ret = mfgraph_receive(&stream->filter_graph, frame)) >= 0) {
   if (frame != NULL) {
    assert(stream->codec_ctx->time_base.num != 0);
    frame->time_base = stream->codec_ctx->time_base;
    frame->pts = av_rescale_q(get_curr_timestamp()
          - stream->start_ts, av_make_q(1, 1000000000), frame->time_base);
    frame->pkt_dts = frame->pts;
   }
   timestamp_t expected_delta = 0; // reduce processing rate
   if (frame->nb_samples > 0)
    expected_delta = av_rescale(frame->nb_samples, 1000000000LL,
          frame->sample_rate);
   else expected_delta = 1000000000LL / SELECON_DEFAULT_VIDEO_FPS;
   reduce_fps(expected_delta, &ts);
   int ret = avcodec_send_frame(stream->codec_ctx, frame);
   if (ret < 0) {
    av_packet_free(&packet);
    av_frame_free(&frame);
    return;
   }
   while (ret == 0) {
    ret = avcodec_receive_packet(stream->codec_ctx, packet);
    if (ret == AVERROR(EAGAIN)) break;
    else if (ret < 0) break;
    stream->packet_handler(stream->packet_user_data, stream, packet);
    av_packet_unref(packet);
   }
   if (ret != AVERROR(EAGAIN) && ret != AVERROR_EOF)
    perror("avcodec_receive_packet");
  }
  if (ret != AVERROR(EAGAIN))
   fprintf(stderr, "mfgraph_receive: err = %d\n", ret);
  av_frame_free(&frame);
 }
}
```

3.3 Сборка и запуск

Сборка основного проекта осуществляется с использованием утилиты cmake:

```
mkdir build
cd build
cmake ..
make цель
```

где *«цель»* может быть `selecon_cli` для сборки консольной утилиты, либо `unittests` для сборки тестов.

Консольная утилита имеет следующие параметры запуска:

```
$ ./selecon_cli --help
./selecon_cli [OPTIONS]

  example demo application for selecon protocol

OPTIONS:
  -h|--help               show this message
  -l|--listen-on address  current participant address (default 0.0.0.0:11235)
  -u|--user username      set user name
  --version               print version and exit
  --stub filename         stream given media file in a loop

DESCRIPTION:
  Address can be IPv4/IPv6 (eg: 192.168.100.1:11235) or socket file path
  in format file://<os-path-to-socket>.
```

Список команд, поддерживаемых утилитой:

```
$ ./selecon_cli
> help
list of available commands:
  dev     manage IO devices
  dump    print info about current selecon context state
  exit    end active conference and close cli tool
  help    show this message
  invite  send invitation for joining active conference to other client
  leave   exit conference without exiting cli tool
  quit    same as exit
  say     send text message to conference chat
  sleep   sleep
  stub    set stub media file for playing in conference
```

В качестве видео заглушки можно использовать видео файлы в формате MP4.

3.4 Тестирование ПО

Ниже приведен список автотестов, разработанных для проверки реализованного протокола.

— Подключение второго участника к конференции, второй участник отклонил приглашение.

— Подключение второго участника к конференции, второй участник принял приглашение.

— Отправка аудио данных от одного участника другому.

— Отправка видео данных от одного участника другому.

— Подключение четырех участников к конференции.

— Отправка аудио данных от одного участника к другим трем.

— Отправка видео данных от одного участника к другим трем.

3.5 Тестирование с использованием контейнеризации

На рисунке 3.1 представлена схема сети, используемой для тестирования работоспособности реализованного протокола для трех участников.

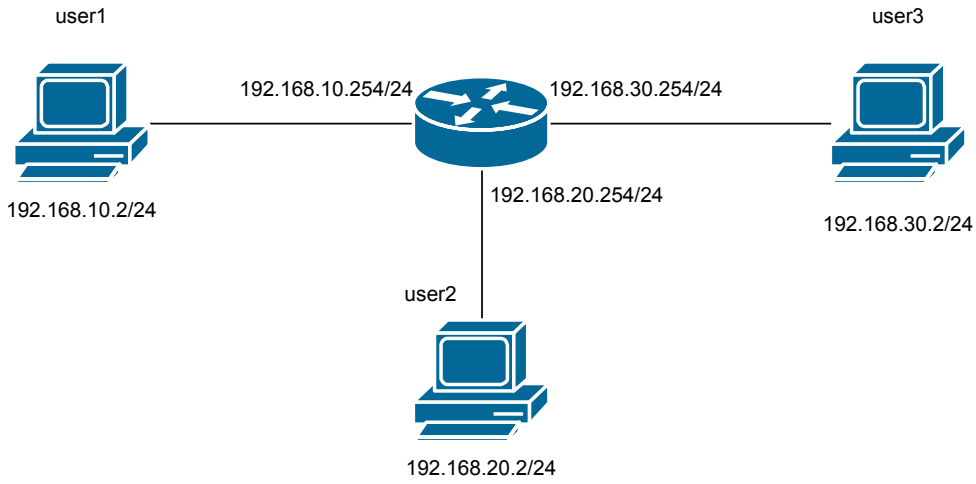


Рисунок 3.1 – Схема сети для тестирования

В качестве роутера выступает отдельный контейнер, использующий ip-переадресацию (ip forwarding). Ниже приведен `compose.yml` файл для запуска контейнеров и создания окружения.

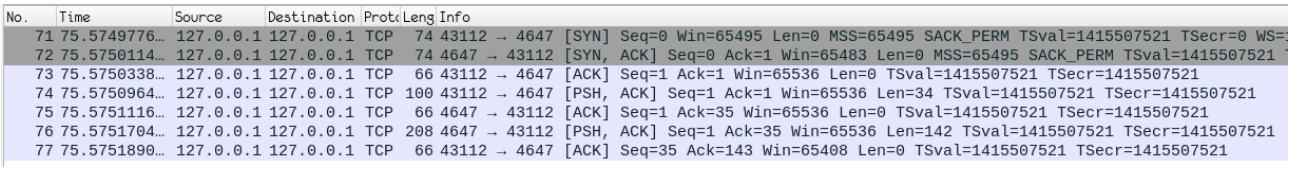
```
services:
  router:
    image: alpine:3.20.1
    cap_add: [ NET_ADMIN ]
    sysctls: [ net.ipv4.ip_forward=1 ]
    networks:
      net1: ipv4_address: 192.168.10.254
      net2: ipv4_address: 192.168.20.254
      net3: ipv4_address: 192.168.30.254
    command: >
      /bin/sh -c "apk update && apk add tcpdump &&
                  tcpdump -i eth0 -i eth1 -i eth2 -v"
  user1:
    build: .
    cap_add: [ NET_ADMIN ]
    environment:
      GW_IP: 192.168.10.254
      USERNAME: user1
    networks: net1: ipv4_address: 192.168.10.2
    command: /bin/bash
  user2:
    build: .
    cap_add: [ NET_ADMIN ]
    environment:
      GW_IP: 192.168.20.254
      USERNAME: user2
    networks: net2: ipv4_address: 192.168.20.2
    command: /bin/bash
  user3:
    build: .
    cap_add: [ NET_ADMIN ]
    environment:
      GW_IP: 192.168.30.254
      USERNAME: user3
    networks: net3: ipv4_address: 192.168.30.2
networks:
  net1:
    driver: bridge
    ipam: config: [ subnet: 192.168.10.0/24 ]
  net2:
    driver: bridge
    ipam: config: [ subnet: 192.168.20.0/24 ]
  net3:
    driver: bridge
    ipam: config: [ subnet: 192.168.30.0/24 ]
```

В данной конфигурации протестировано подключение участников к конференции и корректная отправка медиа данных.

3.6 Трассировка пакетов

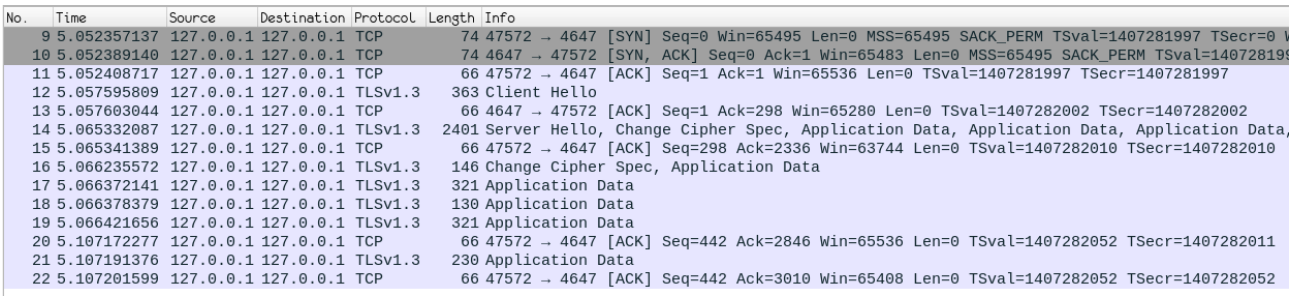
Для более детального рассмотрения пересылаемых пакетов, было организовано по 2 теста с двумя и тремя участниками в локальной сети с применением ПО wireshark. Сначала программа была собрана без использования шифрования сообщений, а затем с использованием шифрования.

На рисунке 3.2 представлен сниф пакетов с использованием программы wireshark во время инициализации конференции с двумя участниками. На рисунке 3.3 представлен сниф того же сценария, но с использованием TLS протокола.



No.	Time	Source	Destination	Proto	Leng	Info
71	75.5749776…	127.0.0.1	127.0.0.1	TCP	74	43112 → 4647 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=1415507521 TSecr=0 WS=
72	75.5750114…	127.0.0.1	127.0.0.1	TCP	74	4647 → 43112 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=1415507521
73	75.5750338…	127.0.0.1	127.0.0.1	TCP	66	43112 → 4647 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1415507521 TSecr=1415507521
74	75.5750964…	127.0.0.1	127.0.0.1	TCP	100	43112 → 4647 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=34 TSval=1415507521 TSecr=1415507521
75	75.5751116…	127.0.0.1	127.0.0.1	TCP	66	4647 → 43112 [ACK] Seq=1 Ack=35 Win=65536 Len=0 TSval=1415507521 TSecr=1415507521
76	75.5751704…	127.0.0.1	127.0.0.1	TCP	208	4647 → 43112 [PSH, ACK] Seq=1 Ack=35 Win=65536 Len=142 TSval=1415507521 TSecr=1415507521
77	75.5751890…	127.0.0.1	127.0.0.1	TCP	66	43112 → 4647 [ACK] Seq=35 Ack=143 Win=65408 Len=0 TSval=1415507521 TSecr=1415507521

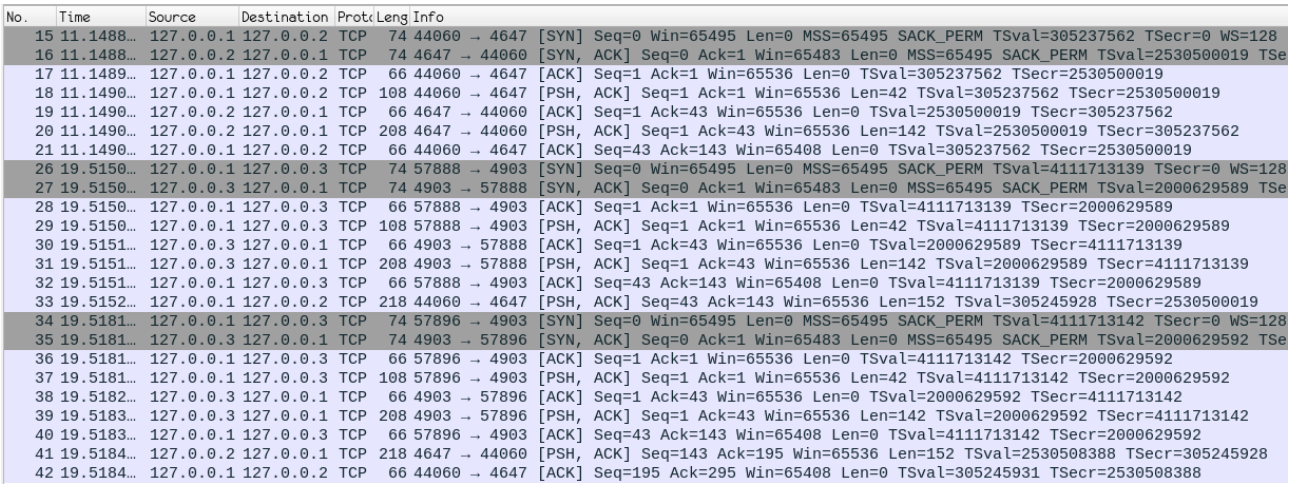
Рисунок 3.2 – Сниф wireshark соединения двух участников без использования TLS



No.	Time	Source	Destination	Protocol	Length	Info
9	5.052357137	127.0.0.1	127.0.0.1	TCP	74	47572 → 4647 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=1407281997 TSecr=0
10	5.052389140	127.0.0.1	127.0.0.1	TCP	74	4647 → 47572 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=14072819
11	5.052408717	127.0.0.1	127.0.0.1	TCP	66	47572 → 4647 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1407281997 TSecr=1407281997
12	5.057595809	127.0.0.1	127.0.0.1	TLSv1.3	363	Client Hello
13	5.057603044	127.0.0.1	127.0.0.1	TCP	66	4647 → 47572 [ACK] Seq=1 Ack=298 Win=65280 Len=0 TSval=1407282002 TSecr=1407282002
14	5.065332087	127.0.0.1	127.0.0.1	TLSv1.3	2401	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data,
15	5.065341389	127.0.0.1	127.0.0.1	TCP	66	47572 → 4647 [ACK] Seq=298 Ack=2336 Win=63744 Len=0 TSval=1407282010 TSecr=1407282010
16	5.066235572	127.0.0.1	127.0.0.1	TLSv1.3	146	Change Cipher Spec, Application Data
17	5.066372141	127.0.0.1	127.0.0.1	TLSv1.3	321	Application Data
18	5.066378379	127.0.0.1	127.0.0.1	TLSv1.3	130	Application Data
19	5.066421656	127.0.0.1	127.0.0.1	TLSv1.3	321	Application Data
20	5.107172277	127.0.0.1	127.0.0.1	TCP	66	47572 → 4647 [ACK] Seq=442 Ack=2846 Win=65536 Len=0 TSval=1407282052 TSecr=1407282011
21	5.107191376	127.0.0.1	127.0.0.1	TLSv1.3	230	Application Data
22	5.107201599	127.0.0.1	127.0.0.1	TCP	66	47572 → 4647 [ACK] Seq=442 Ack=3010 Win=65408 Len=0 TSval=1407282052 TSecr=1407282052

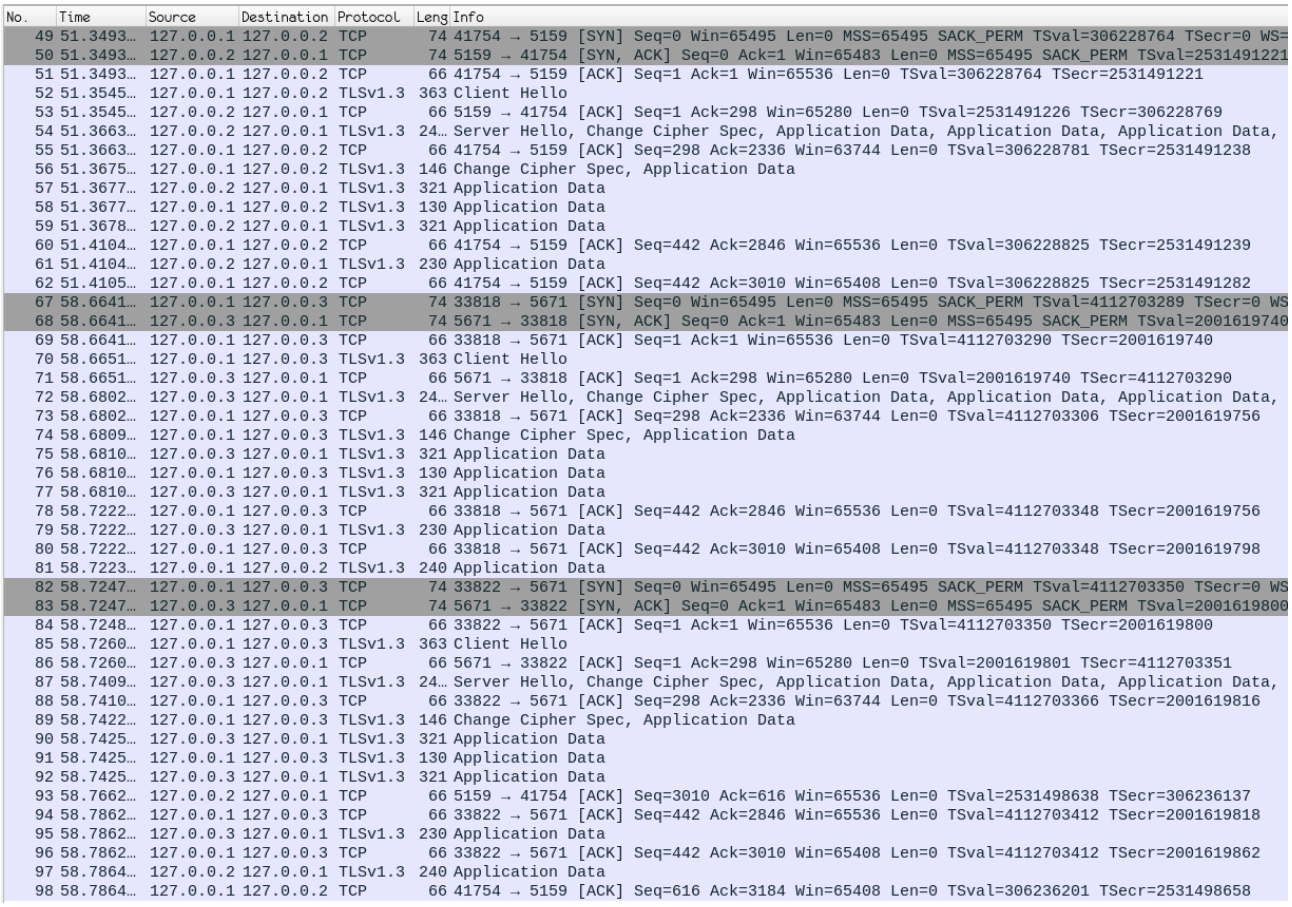
Рисунок 3.3 – Сниф wireshark соединения двух участников с использованием TLS

На рисунках 3.4–3.5 представлены снифы пакетов во время инициализации конференции с тремя участниками. По рисунку видны 3 этапа соединения: инициализация конференции двумя участниками; отправка и получение приглашения для третьего участника и информирование второго участника о новом подключившемся участнике; рукопожатие второго и третьего участников.



No.	Time	Source	Destination	Proto	Leng	Info
15	11.1488…	127.0.0.1	127.0.0.2	TCP	74	44060 → 4647 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=305237562 TSecr=0 WS=128
16	11.1488…	127.0.0.2	127.0.0.1	TCP	74	4647 → 44060 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=2530500019 TSe
17	11.1489…	127.0.0.1	127.0.0.2	TCP	66	44060 → 4647 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=305237562 TSecr=2530500019
18	11.1490…	127.0.0.1	127.0.0.2	TCP	108	44060 → 4647 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=42 TSval=305237562 TSecr=2530500019
19	11.1490…	127.0.0.2	127.0.0.1	TCP	66	4647 → 44060 [ACK] Seq=1 Ack=43 Win=65536 Len=0 TSval=2530500019 TSecr=305237562
20	11.1490…	127.0.0.2	127.0.0.1	TCP	208	4647 → 44060 [PSH, ACK] Seq=1 Ack=43 Win=65536 Len=142 TSval=2530500019 TSecr=305237562
21	11.1490…	127.0.0.1	127.0.0.2	TCP	66	44060 → 4647 [ACK] Seq=43 Ack=143 Win=65408 Len=0 TSval=305237562 TSecr=2530500019
26	19.5150…	127.0.0.1	127.0.0.3	TCP	74	57888 → 4903 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=4111713139 TSecr=0 WS=128
27	19.5150…	127.0.0.3	127.0.0.1	TCP	74	4903 → 57888 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=2000629589 TSe
28	19.5150…	127.0.0.1	127.0.0.3	TCP	66	57888 → 4903 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=4111713139 TSecr=2000629589
29	19.5150…	127.0.0.1	127.0.0.3	TCP	108	57888 → 4903 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=42 TSval=4111713139 TSecr=2000629589
30	19.5151…	127.0.0.3	127.0.0.1	TCP	66	4903 → 57888 [ACK] Seq=1 Ack=43 Win=65536 Len=0 TSval=2000629589 TSecr=4111713139
31	19.5151…	127.0.0.3	127.0.0.1	TCP	208	4903 → 57888 [PSH, ACK] Seq=1 Ack=43 Win=65536 Len=142 TSval=2000629589 TSecr=4111713139
32	19.5151…	127.0.0.1	127.0.0.3	TCP	66	57888 → 4903 [ACK] Seq=43 Ack=143 Win=65408 Len=0 TSval=4111713139 TSecr=2000629589
33	19.5152…	127.0.0.1	127.0.0.2	TCP	218	44060 → 4647 [PSH, ACK] Seq=43 Ack=143 Win=65536 Len=152 TSval=305245928 TSecr=2530500019
34	19.5181…	127.0.0.1	127.0.0.3	TCP	74	57896 → 4903 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=4111713142 TSecr=0 WS=128
35	19.5181…	127.0.0.3	127.0.0.1	TCP	74	4903 → 57896 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=2000629592 TSe
36	19.5181…	127.0.0.1	127.0.0.3	TCP	66	57896 → 4903 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=4111713142 TSecr=2000629592
37	19.5181…	127.0.0.1	127.0.0.3	TCP	108	57896 → 4903 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=42 TSval=4111713142 TSecr=2000629592
38	19.5182…	127.0.0.3	127.0.0.1	TCP	66	4903 → 57896 [ACK] Seq=1 Ack=43 Win=65536 Len=0 TSval=2000629592 TSecr=4111713142
39	19.5183…	127.0.0.3	127.0.0.1	TCP	208	4903 → 57896 [PSH, ACK] Seq=1 Ack=43 Win=65536 Len=142 TSval=2000629592 TSecr=4111713142
40	19.5183…	127.0.0.1	127.0.0.3	TCP	66	57896 → 4903 [ACK] Seq=43 Ack=143 Win=65408 Len=0 TSval=4111713142 TSecr=2000629592
41	19.5184…	127.0.0.2	127.0.0.1	TCP	218	4647 → 44060 [PSH, ACK] Seq=143 Ack=195 Win=65536 Len=152 TSval=2530508388 TSecr=305245928
42	19.5184…	127.0.0.1	127.0.0.2	TCP	66	44060 → 4647 [ACK] Seq=195 Ack=295 Win=65408 Len=0 TSval=305245931 TSecr=2530508388

Рисунок 3.4 – Сниф wireshark соединения трех участников без использования TLS



No.	Time	Source	Destination	Protocol	Leng	Info
49	51.3493…	127.0.0.1	127.0.0.2	TCP	74	41754 → 5159 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=306228764 TSecr=0 WS=
50	51.3493…	127.0.0.2	127.0.0.1	TCP	74	5159 → 41754 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=2531491221
51	51.3493…	127.0.0.1	127.0.0.2	TCP	66	41754 → 5159 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=306228764 TSecr=2531491221
52	51.3545…	127.0.0.1	127.0.0.2	TLSv1.3	363	Client Hello
53	51.3545…	127.0.0.2	127.0.0.1	TCP	66	5159 → 41754 [ACK] Seq=1 Ack=298 Win=65280 Len=0 TSval=2531491226 TSecr=306228769
54	51.3663…	127.0.0.2	127.0.0.1	TLSv1.3	24…	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data,
55	51.3663…	127.0.0.1	127.0.0.2	TCP	66	41754 → 5159 [ACK] Seq=298 Ack=2336 Win=63744 Len=0 TSval=306228781 TSecr=2531491238
56	51.3675…	127.0.0.1	127.0.0.2	TLSv1.3	146	Change Cipher Spec, Application Data
57	51.3677…	127.0.0.2	127.0.0.1	TLSv1.3	321	Application Data
58	51.3677…	127.0.0.1	127.0.0.2	TLSv1.3	130	Application Data
59	51.3678…	127.0.0.2	127.0.0.1	TLSv1.3	321	Application Data
60	51.4104…	127.0.0.1	127.0.0.2	TCP	66	41754 → 5159 [ACK] Seq=442 Ack=2846 Win=65536 Len=0 TSval=306228825 TSecr=2531491239
61	51.4104…	127.0.0.2	127.0.0.1	TLSv1.3	230	Application Data
62	51.4105…	127.0.0.1	127.0.0.2	TCP	66	41754 → 5159 [ACK] Seq=442 Ack=3010 Win=65408 Len=0 TSval=306228825 TSecr=2531491282
67	58.6641…	127.0.0.1	127.0.0.3	TCP	74	33818 → 5671 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=4112703289 TSecr=0 WS
68	58.6641…	127.0.0.3	127.0.0.1	TCP	74	5671 → 33818 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=2001619740
69	58.6641…	127.0.0.1	127.0.0.3	TCP	66	33818 → 5671 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=4112703290 TSecr=2001619740
70	58.6651…	127.0.0.1	127.0.0.3	TLSv1.3	363	Client Hello
71	58.6651…	127.0.0.3	127.0.0.1	TCP	66	5671 → 33818 [ACK] Seq=1 Ack=298 Win=65280 Len=0 TSval=2001619740 TSecr=4112703290
72	58.6802…	127.0.0.3	127.0.0.1	TLSv1.3	24…	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data,
73	58.6802…	127.0.0.1	127.0.0.3	TCP	66	33818 → 5671 [ACK] Seq=298 Ack=2336 Win=63744 Len=0 TSval=4112703306 TSecr=2001619756
74	58.6809…	127.0.0.1	127.0.0.3	TLSv1.3	146	Change Cipher Spec, Application Data
75	58.6810…	127.0.0.3	127.0.0.1	TLSv1.3	321	Application Data
76	58.6810…	127.0.0.1	127.0.0.3	TLSv1.3	130	Application Data
77	58.6810…	127.0.0.3	127.0.0.1	TLSv1.3	321	Application Data
78	58.7222…	127.0.0.1	127.0.0.3	TCP	66	33818 → 5671 [ACK] Seq=442 Ack=2846 Win=65536 Len=0 TSval=4112703348 TSecr=2001619756
79	58.7222…	127.0.0.3	127.0.0.1	TLSv1.3	230	Application Data
80	58.7222…	127.0.0.1	127.0.0.3	TCP	66	33818 → 5671 [ACK] Seq=442 Ack=3010 Win=65408 Len=0 TSval=4112703348 TSecr=2001619798
81	58.7223…	127.0.0.1	127.0.0.2	TLSv1.3	240	Application Data
82	58.7247…	127.0.0.1	127.0.0.3	TCP	74	33822 → 5671 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=4112703350 TSecr=0 WS
83	58.7247…	127.0.0.3	127.0.0.1	TCP	74	5671 → 33822 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=2001619800
84	58.7248…	127.0.0.1	127.0.0.3	TCP	66	33822 → 5671 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=4112703350 TSecr=2001619800
85	58.7260…	127.0.0.1	127.0.0.3	TLSv1.3	363	Client Hello
86	58.7260…	127.0.0.3	127.0.0.1	TCP	66	5671 → 33822 [ACK] Seq=1 Ack=298 Win=65280 Len=0 TSval=2001619801 TSecr=4112703351
87	58.7409…	127.0.0.3	127.0.0.1	TLSv1.3	24…	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data,
88	58.7410…	127.0.0.1	127.0.0.3	TCP	66	33822 → 5671 [ACK] Seq=298 Ack=2336 Win=63744 Len=0 TSval=4112703366 TSecr=2001619816
89	58.7422…	127.0.0.1	127.0.0.3	TLSv1.3	146	Change Cipher Spec, Application Data
90	58.7425…	127.0.0.3	127.0.0.1	TLSv1.3	321	Application Data
91	58.7425…	127.0.0.1	127.0.0.3	TLSv1.3	130	Application Data
92	58.7425…	127.0.0.3	127.0.0.1	TLSv1.3	321	Application Data
93	58.7662…	127.0.0.2	127.0.0.1	TCP	66	5159 → 41754 [ACK] Seq=3010 Ack=616 Win=65536 Len=0 TSval=2531498638 TSecr=306236137
94	58.7862…	127.0.0.1	127.0.0.3	TCP	66	33822 → 5671 [ACK] Seq=442 Ack=2846 Win=65536 Len=0 TSval=4112703412 TSecr=2001619818
95	58.7862…	127.0.0.3	127.0.0.1	TLSv1.3	230	Application Data
96	58.7862…	127.0.0.1	127.0.0.3	TCP	66	33822 → 5671 [ACK] Seq=442 Ack=3010 Win=65408 Len=0 TSval=4112703412 TSecr=2001619862
97	58.7864…	127.0.0.2	127.0.0.1	TLSv1.3	240	Application Data
98	58.7864…	127.0.0.1	127.0.0.2	TCP	66	41754 → 5159 [ACK] Seq=616 Ack=3184 Win=65408 Len=0 TSval=306236201 TSecr=2531498658

Рисунок 3.5 – Сниф wireshark соединения трех участников с использованием TLS

В результате была протестирована реализация спроектированного протокола.

4 Исследовательский раздел

4.1 Постановка исследования

Цель исследования — выявление зависимости задержки аудио и видео, а также общей нагрузки на процессор от числа участников в конференции.

Исследование проведено на ноутбуке Intel i5-8365U 4.1GHz, 16GB оперативной памяти, под операционной системой Debian 12.

Количество участников варьируется от 2 до 7, время конференции — 1 минута. Во время конфереции все участники транслировали одно и то же видео.

4.2 Результаты

Результаты представлены на графике 4.1.

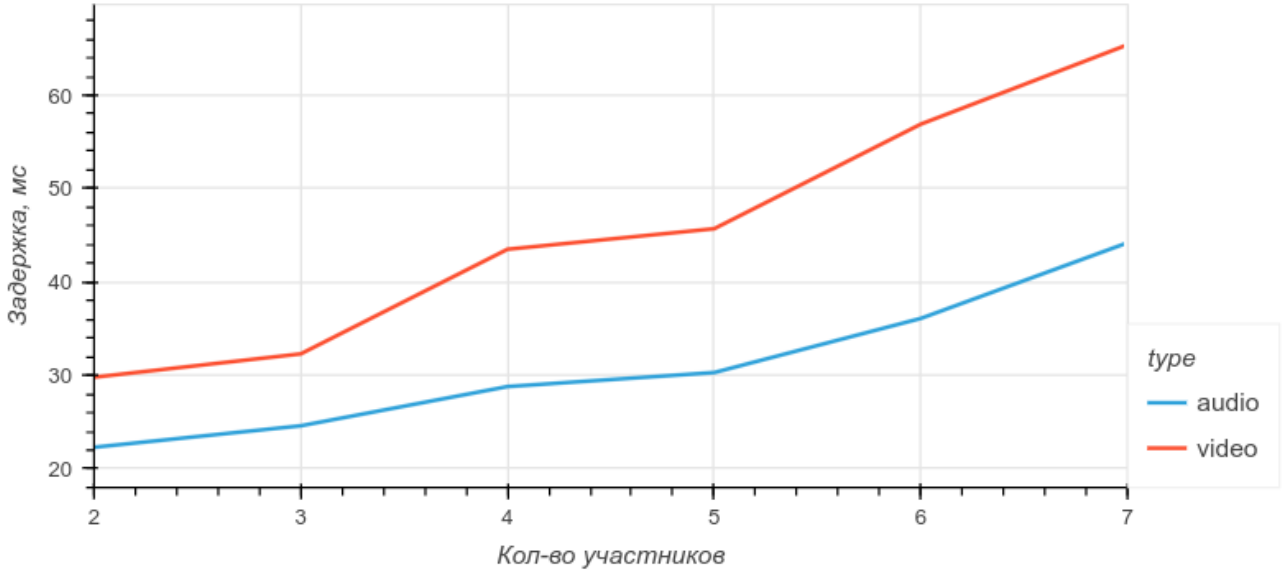


Рисунок 4.1 – Зависимость средней задержки аудио и видео от числа участников конференции

На основе полученных результатов можно предположить, что увеличение задержки связанно с увеличивающейся нагрузкой на процессор при увеличении количества участников.

ЗАКЛЮЧЕНИЕ

В результате работы был спроектирован протокол бессерверных конференций. Разработана программная реализация спроектированного протокола и реализовано ПО, осуществляющее пересылку видео и аудио потоков в рамках бессерверной конференции. Проведено тестирование и исследование ПО. В результате исследования была выявлена зависимость задержки видео и аудио потоков от количества участников конференции.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. RTP: A Transport Protocol for Real-Time Applications / H. Schulzrinne [и др.]. — 07.2003. — DOI: 10.17487/RFC3550. — URL: https://www.rfc-editor.org/info/rfc3550. RFC 3550.
2. SIP: Session Initiation Protocol / E. Schooler [и др.]. — 07.2002. — DOI: 10.17487/RFC3261. — URL: https://www.rfc-editor.org/info/rfc3261. RFC 3261.
3. *Agboh C.*, *Schulzrinne H.* Session Initiation Protocol (SIP)-H.323 Interworking Requirements. — 07.2005. — DOI: 10.17487/RFC4123. — URL: https://www.rfc-editor.org/info/rfc4123. RFC 4123.
4. OpenSSL — https://www.openssl.org/ (дата обращения: 10.12.2024). — [Электронный ресурс].
5. Understand NAT to Enable Peer-To-Peer Communication on IOS and IOS XE Routers - Cisco — https://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/217599-understand-nat-to-enable-peer-to-peer-co.html (дата обращения: 12.12.2024). — [Электронный ресурс].
6. RTP Payload Format for MPEG-4 Audio/Visual Streams / Y. Matsui [и др.]. — 11.2000. — DOI: 10.17487/RFC3016. — URL: https://www.rfc-editor.org/info/rfc3016. RFC 3016.
7. H.264: Advanced video coding for generic audiovisual services — https://www.itu.int/rec/T-REC-H.264 (дата обращения: 10.12.2024). — (Электронный ресурс).
8. *Rescorla E.* The Transport Layer Security (TLS) Protocol Version 1.3. — 08.2018. — DOI: 10.17487/RFC8446. — URL: https://www.rfc-editor.org/info/rfc8446. RFC 8446.
9. CMake — Upgrade Your Software Build System — https://cmake.org/ (дата обращения: 20.12.2024). — (Электронный ресурс).
10. Ffmpeg — https://www.ffmpeg.org/ (дата обращения: 21.12.2024). — (Электронный ресурс).
11. GoogleTest User's Guide — https://google.github.io/googletest/ (дата обращения: 21.12.2024). — (Электронный ресурс).