|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 «Международных образовательных програм» (ФМОП)

«Информатика и Системы Управления» (ИУ)

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии» (ИУ7)

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОЙ РАБОТЕ***

***НА ТЕМУ:***

***Реализация мобильного приложения для интернет видео разговоров***

Студент \_\_ИУ7И-71Б\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_** К. С. Алмишев**\_\_\_**

Студент \_\_ИУ7И-76Б\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_** А. С. Миневска**\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Н.О. Рогозин**\_\_**

(Подпись, дата) (И.О.Фамилия)

**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: Компьютерные сети

Студент группы: ИУ7И-71Б\_Алмишев Костадин Спасов

Студент группы: ИУ7И-76Б\_Миневска Ани Стоянова\_\_\_

(Фамилия, имя, отчество)

Тема курсовой работы: Реализация мобильного приложения для интернет видео разговоров

Направленность курсовой работы (учебный, исследовательский, практический, производственный, др.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Учебный\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

График выполнения проекта: 25% к \_4\_ нед., 50% к \_7\_ нед., 75% к \_11\_ нед., 100% к 14 нед.

***Задание****\_* *Разработать систему для интернет видео-разговоров, при помощи который будет проведено исследования протоколов для передачи данных.*

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на \_15-30\_ листах формата А4.

На защиту работы должна быть представлена презентация из 10-20 слайдов.

Дата выдачи задания «12» октября 2020 г.

**Руководитель НИР**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_**\_\_**\_\_Н.О. Рогозин\_

(Подпись, дата) (И.О.Фамилия)

**Студенты \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**К. С. Алмишев**\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**А.С. Миневска

(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Оглавление

[Введение 4](#_Toc60156241)

[Цель работы 5](#_Toc60156242)

[1. Аналитический раздел 6](#_Toc60156243)

[1.1 Обзор протоколов 6](#_Toc60156244)

[1.1.1 UDP 6](#_Toc60156245)

[1.1.2 TCP 9](#_Toc60156246)

[1.1.3 UDT 10](#_Toc60156247)

[1.1.4 SRT 11](#_Toc60156248)

[1.1.5 WebRTC 14](#_Toc60156249)

[Вывод 15](#_Toc60156250)

[2. Конструкторский раздел 16](#_Toc60156251)

[2.1 Развертка TURN сервера 16](#_Toc60156253)

[2.2 Добавление firewall 16](#_Toc60156254)

[2.3 Конфигурация домена 17](#_Toc60156255)

[2.4 Настройка SSL шифрование 18](#_Toc60156256)

[2.5 Конфигурация TURN сервера 19](#_Toc60156257)

[2.6 Архитектура приложения 21](#_Toc60156258)

[3. Технологический раздел 23](#_Toc60156259)

[4.1 Язык для Signaling сервера и разработки мобильного приложения 23](#_Toc60156262)

[4.2 Технология для Signaling сервера 23](#_Toc60156263)

[4.3 Turn сервер для обхода NAT 25](#_Toc60156264)

[Вывод 26](#_Toc60156265)

[Список использованной литературы 27](#_Toc60156266)

Введение

Неоспоримым фактом является то, что Интернет сегодня — это самый колоссальный источник информации, который знало человечество. Но его возможности, такие, как оперативность, быстрота и доступность связи между пользователями на дальних и близких расстояниях, позволяют использовать Интернет не только как инструмент для познания, но и как инструмент для общения. Общение – это взаимодействие людей, при котором они познают друг друга, вступают в те или иные взаимоотношения и при котором между ними устанавливается определенное взаимоотношение.

В современном мире удобная, дешевая и качественная связь между людьми невозможно без помощи интернета и электронных услуг. Без них любому человека приходится тратить дополнительные ресурсы и время, чтобы решать жизненно-важные задачи.

Цель работы

Разработать мобильное приложение для интернет видео-разговоров. Для достижения этой цели, необходимо решить следующие задачи:

1. Рассмотреть существующие протоколы для передачи данных
2. Выбрать протокол подходящий для поставленной задачи
3. Разработать мобильное приложение на основе выбранного протокола
4. Аналитический раздел
   1. Обзор протоколов
      1. UDP

UDP – это передача данных без установления соединения, которое не имеет подтверждения связи и, таким образом, не дает никаких гарантий относительно доставки или порядка получения пакетов.

Карл Сегин (Karl Seguin) провел эксперимент, чтобы выяснить, какова же надежность такого способа передачи данных.

Он создал 5 VPS для отправки друг другу нескольких UDP-пакетов в течение 7 часов. Каждый сервер с периодом в 9-11 секунд случайно выбирал получателя и посылал 5-10 пакетов размером от 16 до 1016 байт.

Два сервера располагались в одном центре обработки данных в Нью-Джерси (НД 1 и НД 2) и по одному в Лос-Анджелесе (ЛА), Амстердаме (Нидерланды/НЛД) и Токио (Япония/ЯПН).

**Надежность**

В таблицах 1 и 2 представлены результаты эксперимента связаны с надежностью. К сожалению, нельзя узнать произошла ли потеря пакетов примерно в одно и то же время. Если проанализировать полученные данные, то можно заметить, что из 43 пакетов, которые не дошли из Лос-Анджелеса до второго сервера в Нью-Джерси, 29 были потеряны во время 1 ~ 2-минутных промежутков времени. Потеря пакетов, передаваемых первому серверу, также в значительной степени произошла в 2 коротких периода времени.

Таблица 1 Таблица переданных/полученных пакетов протокола UDP после эксперимента Карла Сегина (процентное соотношение)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Получатель | | | | |
|  | НД1 | НД2 | ЛА | НЛД | ЯПН |
| НД1 | - | 100 | 99.97 | 100 | 99.97 |
| НД2 | 100 | - | 99.97 | 99.97 | 100 |
| ЛА | 98.64 | 98.55 | - | 99.89 | 100 |
| НЛД | 100 | 99.97 | 100 | - | 100 |
| ЯПН | 99.96 | 100 | 100 | 100 | - |

Таблица 2Потери пакетов протокола UDP в зависимости от размера после эксперимента Карла Сегина (в байтах)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0-115 | 116-215 | 216-315 | 316-515 | 516-715 | 716-915 |
| 13 | 11 | 12 | 13 | 23 | 23 |

**Порядок**

Второй аспект, интересовавший Карла, — это порядок получения пакетов. Изначально, автор эксперимента решил измерить инверсию массива. По сути, это число пар, в которых элементы расположены не по порядку. Если есть массив со значениями 10, 8, 3, 7, 4, то в конечном итоге придется сделать 8 перестановок ((10,8), (10,3), (10,7), (10,4), (8,3), (8,7), (8,4), (7,4)).

Таблица 3 Инверсия в протоколе UDP

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Получатель | | | | |
|  | НД1 | НД2 | ЛА | НЛД | ЯПН |
| НД1 | - | 2981 | 1514 | 1658 | 1123 |
| НД2 | 3016 | - | 1627 | 1483 | 1161 |
| ЛА | 1227 | 1259 | - | 1485 | 1067 |
| НЛД | 1407 | 1645 | 1220 | - | 1096 |
| ЯПН | 980 | 1083 | 1141 | 1087 | - |

Таблица 4 Количество упорядоченных пакетов (процентное соотношение)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Получатель | | | | |
|  | НД1 | НД2 | ЛА | НЛД | ЯПН |
| НД1 | - | 100 | 52.40 | 55.94 | 36.77 |
| НД2 | 100 | - | 52.47 | 54.22 | 38.02 |
| ЛА | 47.72 | 42.32 | - | 50.48 | 39.34 |
| НЛД | 46.32 | 59.34 | 44.79 | - | 39.27 |
| ЯПН | 38.40 | 37.53 | 40.20 | 37.65 | - |

**Вывод**

По разельтатам надежность UDP довольно высока. Расстояние обычно включает в себя больше «прыжков» (Каждое встречающееся на пути пакета промежуточное сетевое устройство заставляет пакет «перепрыгивать» на очередное соединение (следующий сетевой сегмент), и каждый «прыжок» увеличивает риск того, что что-то пойдет не так, но если все будет налажено, то расстояние не будет проблемой.

Что касается порядка: в данном случае расстояние играет большую роль. Для систем видео-разговор порядок пакетов очень важен, но, к сожалению, UDP не удовлетворяют эти требования. Если используете потоковую передачу данных, то вполне достаточно сохранять временную метку и выполнять повторное упорядочение на принимающей стороне. Таким образом можно решить эту проблему и получить скорость лучше, чем у TCP

* + 1. TCP

Transmission Control Protocol (TCP, протокол управления передачей) — один из основных протоколов передачи данных интернета, предназначенный для управления передачей данных. Пакеты в TCP называются сегментами.

В стеке протоколов TCP/IP выполняет функции транспортного уровня модели OSI.

Механизм TCP предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым, в отличие от UDP, целостность передаваемых данных и уведомление отправителя о результатах передачи.

Процесс начала сеанса TCP (также называемый «рукопожатие» (англ. handshake)), состоит из трёх шагов.

1. Клиент, который намеревается установить соединение, посылает серверу сегмент с номером последовательности и флагом SYN.

* Сервер получает сегмент, запоминает номер последовательности и пытается создать сокет (буферы и управляющие структуры памяти) для обслуживания нового клиента.
* В случае успеха сервер посылает клиенту сегмент с номером последовательности и флагами SYN и ACK, и переходит в состояние SYN-RECEIVED.
* В случае неудачи сервер посылает клиенту сегмент с флагом RST.

1. Если клиент получает сегмент с флагом SYN, то он запоминает номер последовательности и посылает сегмент с флагом ACK.

* Если клиент одновременно получает и флаг ACK (что обычно и происходит), то он переходит в состояние ESTABLISHED.
* Если клиент получает сегмент с флагом RST, то он прекращает попытки соединиться.
* Если клиент не получает ответа в течение 10 секунд, то он повторяет процесс соединения заново.

1. Если сервер в состоянии SYN-RECEIVED получает сегмент с флагом ACK, то он переходит в состояние ESTABLISHED.

В противном случае после тайм-аута он закрывает сокет и переходит в состояние CLOSED.

Процесс называется «трёхэтапным рукопожатием» (англ. three way handshake), так как несмотря на то, что возможен процесс установления соединения с использованием четырёх сегментов (SYN в сторону сервера, ACK в сторону клиента, SYN в сторону клиента, ACK в сторону сервера), на практике для экономии времени используется три сегмента.

**Вывод**

Протокол TCP является протоколом надежной передачи данных. Пакеты получаются упорядочено, которое идеально подходит для систем видео разговор. Недостаток этого протокола является медленная скорость работы из-за того, что она использует SACK.

* + 1. UDT

UDT - это надежный протокол передачи данных на уровне приложений, основанный на UDP, для распределенных приложений с интенсивным использованием данных в широком диапазоне высокоскоростных сетей. UDT использует UDP для передачи больших объемов данных со своими собственными механизмами контроля надежности и контроля перегрузки. Новый протокол может передавать данные с гораздо большей скоростью, чем TCP. UDT также представляет собой легко настраиваемую структуру, которая может использовать различные алгоритмы управления перегрузкой.

**Преимущества**

* Быстрый. UDT разработан для высокоскоростных сетей и используется для поддержки глобальной передачи наборов данных размером в терабайт. UDT - это основная технология во многих коммерческих продуктах для ускорения WAN.
* Честно и дружелюбно. Параллельные потоки UDT могут справедливо разделять доступную полосу пропускания, в то время как UDT также оставляет достаточно полосы пропускания для TCP.
* Легко использовать. UDT полностью находится на уровне приложения. Пользователи могут просто загрузить программное обеспечение и начать его использовать. Реконфигурация ядра не требуется. Кроме того, API UDT очень похож на традиционный API сокетов, поэтому существующие приложения можно легко изменять.
* Сильно настраиваемый. UDT поддерживает определенные пользователем алгоритмы управления перегрузкой с простой конфигурацией. Пользователи также могут изменять UDT для соответствия различным ситуациям. Эта функция также может использоваться студентами и исследователями для исследования новых алгоритмов управления.
* Совместимость с межсетевым экраном. UDT полностью основан на UDP, что упрощает прохождение межсетевого экрана. Кроме того, несколько потоков UDT могут совместно использовать один порт UDP, поэтому брандмауэр может открыть только один порт UDP для всех соединений UDT. UDT также поддерживает установку рандеву-соединения.

**Недостатки**

Протокол не предоставляет возможность синхронизации времени получения пакетов, которая существенно важно для качества видео разговора

* + 1. SRT

SRT (Secure Reliable Transport) — относительно молодой протокол передачи данных, разработанный на основе UDP. Данный протокол обеспечивает два вида гарантий в разных режимах работы — гарантию доставки данных или гарантию задержки доставки.

**Кроме открытого кода у протокола есть еще несколько преимуществ.**

* Высокая скорость передачи видео

Она достигается благодаря использованию протокола UDP. На скорость влияет и использование временных меток для каждого отдельного пакета данных, которые размещаются в начале. Благодаря этому объемы буфера данных при использовании SRT в разы меньше, чем при применении RTMP.

Но UDP не гарантирует доставку пакетов адресату: его даже прозвали «протокол ненадёжных датаграмм». При разработке SRT эта особенность UDP была учтена и исправлена.

* Интеллектуальный механизм повторной передачи данных быстрый и эффективный

SRT идентифицирует каждый пакет по его порядковому номеру. Если на стороне получателя образуется разница больше чем в 1 порядковый номер, отправителю незамедлительно посылается отрицательное подтверждение получения пакета NACK, и он передается заново — только один пакет, а не группа, как при использовании TCP

* Поддержка стандарта сжатия видео H.265 (HEVC)

Благодаря этому снижается битрейт при сохранении качества изображения. Если сравнивать H.265 с другими стандартами видеокодирования, при его применении битрейт ниже на 35.4%, чем при использовании H.264/MPEG-4 AVC HP, на 63,7 %, чем у MPEG-4 ASP, на 65,1 % и 70,8 %, чем при применении H.263 HLP и H.262/MPEG-2 MP

Уменьшается нагрузка на сеть. В сравнении с тем же H.264 разница — примерно 40%. Также есть возможность включения нескольких полноценных звуковых дорожек

* Использование алгоритма шифрования AES

Протокол поддерживает шифрование AES 128/256 для защиты потокового видео. Алгоритм одобрен АНБ США и структурами, отвечающими за безопасность в других странах. К примеру, чтобы расшифровать ключ такого алгоритма шифрования, суперкомпьютеру, который расшифрует ключ DES за 1 секунду, потребуется более 149 000 000 000 000 лет.

У протокола UDT (на основе которой, разработан SRT) есть возможность восстановления после потери пакетов. Хотя он вообще не предназначен для потоковой передачи в реальном времени, он работает при использовании действительно больших буферов. Цель SRT сделать решение для прямой трансляции с низкой задержкой. Для этого необходимо было переписать всю функцию повторной передачи пакетов, чтобы иметь возможность немедленно реагировать на потерю пакета, когда это произойдет, и что он добавил протокол шифрования, который он определил и реализовал для других случаев использования ранее.

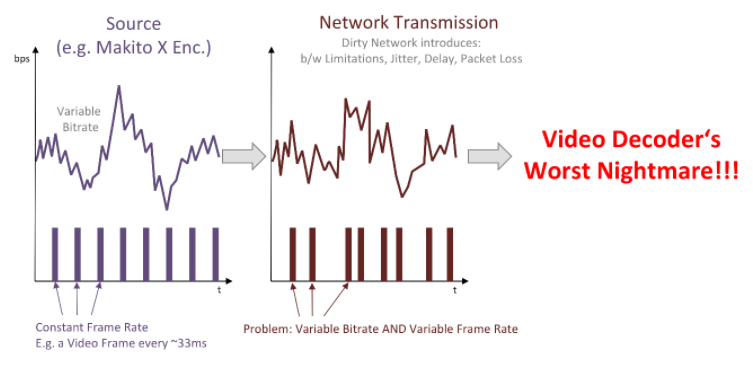
Было сделано тестирование отправки прямых трансляций с низкой задержкой между Германией и Монреалем, и результат был положительным!

Рисунок 1Проблема с частотой получения кадров протокола UDT

Характеристики исходного потока протокола UDT в исходной сети полностью изменились при передаче через общедоступный Интернет. Причины - задержка, джиттер, потеря пакетов и их восстановление в грязной сети. Сигнал на стороне приемника имел совершенно другие характеристики, что приводило к проблемам с декодированием, так как декодеры аудио и видео не получали пакеты в ожидаемое время. Это можно сделать с помощью буферизации, но это не то, что вам нужно в настройках с низкой задержкой.

Решением было придумать механизм, который воссоздает характеристики сигнала на стороне приемника. Таким образом мы смогли значительно уменьшить буферизацию. Эта функция является частью самого протокола SRT, поэтому, как только данные поступают из протокола SRT на стороне получателя, характеристики потока были должным образом восстановлены.

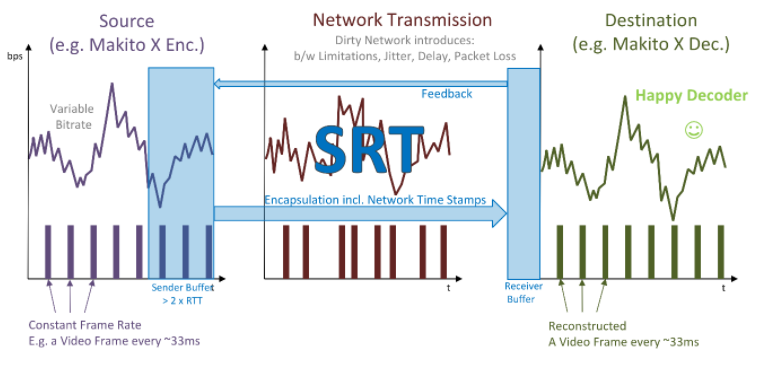


Рисунок 2"Сглаживание" частотой получения кадров, про помощи протокола SRT

* + 1. WebRTC

WebRTC (англ. real-time communications — коммуникации в реальном времени) — проект с открытым исходным кодом, предназначенный для организации передачи потоковых данных между браузерами или другими поддерживающими его приложениями по технологии точка-точка.

**Преимущества**

* Проведение конференции в браузере значительно упрощает процесс проведения конференции — пользователю не нужно устанавливать для этого отдельные приложения;
* Используемые кодеки обеспечивают хорошее качество связи;
* открытый исходный код даёт больше возможностей для использования.
* Работа при помощи UDP и TCP

**Недостатки**

* Технология определяет только общий стандарт передачи данных (видео и звука), но отдельные решения разных браузеров относительно адресации абонентов и прочих управляющих процессов не совместимы между собой. Поэтому даже звонки между парой различных браузеров представляют отдельную сложность.
* Обеспечение групповых конференций требует дополнительных разработок поверх WebRTC.

Вывод

В данном разделе были рассмотрены протоколы передачи данных. TCP является надежным способом для передачи данных, но работает медленно. Чтобы решить эту проблему, нужно использовать протоколы на основе UDP.

UDP не предоставляет механизм надежной передачи данных. Также большинство из пакетов не приходят в упорядоченном виде. По этой причине качество видео и голоса плохие.

Для передачи большой информации данных был реализован протокол UDT, которой работает поверх UDP. Он предоставляет механизм надежной передачи и упорядочивание пакетов. Но у этого протокола для видео передачи в реальном проблемы есть недостаток. Он заключается в том, что кадры не получаются с одинаковой частотой. Из-за этого видео кодеки не всегда успевают решить проблемы со сглаживанием видео.

По этой причине был разработан протокол SRT, предоставляющий механизм синхронизации кадров.

Протокол WebRTC, создан специально для передачи видео и звука в реальном времени и работает поверх TCP и UDP, выбираю оптимальным образом тип связи.

На основе полученной информации, было принято решение использовать протокол WebRTC для реализации поставленной задачи.

1. Конструкторский раздел
2. 1. Развертка TURN сервера

Приложение было развёрнуто на сервисе https://www.digitalocean.com/. Он позволяет создать так называемы Droplet. Это виртуальная машина (ВМ) на базе Linux, которая работает поверх виртуализированного оборудования. Каждый Droplet — это новый сервер, который можно использовать как автономно, так и как часть более крупной облачной инфраструктуры.

Во время настройки Droplet была выбранная следующая конфигурация:

Операционная система – Ubuntu Server 18.04.3 (LTS) x64

* План – Standard 5$ в месяц (1 GB RAM, 1 VCPU, 25 GB SSD disk, 1000 GB transfer)
* Местоположение – Франкфурд
* Добавлен SSH key, для подключения при помощи ssh
* Добавлен домен amempire.ru, который был куплен в https://www.superhosting.bg/

Через несколько минут система подняла новый сервер.

* 1. Добавление firewall

Первый шаг - получить доступ к серверу, используя ssh. Для этого понадобилось IP-адрес сервера, который был создан. Чтобы войти на созданный сервер, в терминале на локальном компьютере была использована следующая команда для SSH в качестве пользователя root (вместо server\_ip\_address, был поставлен IP адрес созданного сервера):

$ ssh root@server\_ip\_address

Серверы Ubuntu могут использовать UFW в качестве farewall, чтобы гарантировать, что разрешены только подключения к определенным службам.

Изначально нужно убедиться, что farewall разрешает SSH-соединения, чтобы после включения firewall, можно снова войти в систему. Чтобы разрешить эти типы подключений, была использована команда:

$ sudo ufw allow OpenSSH

Рекомендуется включить наиболее строгий профиль, который по-прежнему будет разрешать трафик. Поскольку будет настроен SSL для сервера, нужно только разрешить трафик на порт 443. Также для turn сервера нужны порты 3478 и 5343.

Порт можно открыть при помощи команды:

$ sudo ufw allow номер\_порта

После настройки firewall, нужно ее включить командой:

$ sudo ufw enable

* 1. Конфигурация домена

DigitalOcean не является domain name registrar. Это означает, что нельзя приобрести у него доменное имя. Но они предоставляют услугу хостинга DNS, которая упрощает настройку доменного имени на их серверах.

Для покупки домена, была использована служба <https://www.reg.ru>

В веб-сайте DigitalOcean, нужно открыть раскрывающееся меню «Создать» и нажать на ссылку «Домены / DNS».

В разделе «Добавить домен» необходимо ввести куплены домен (обычно это только база: example.com, а не www.example.com). Чтобы сохранить нужно нажать на «Добавить домен». После этого, открывается страница «Создать новую запись».

Теперь нужно добавить записи NS для домена в системах DigitalOcean. Было добавлено запись A, который сопоставляют IPv4-адрес с доменным именем. Это определяет, куда DigitalOcean должен направлять любые запросы на данное доменное имя.

Чтобы использовать DNS DigitalOcean, еще необходимо обновить серверы имен, используемые регистратором домена, чтобы вместо этого они указывали на серверы имен DigitalOcean. Для этого необходимо зайти на <https://www.reg.ru/> и открыть панель со списком всех доменов. В разделе «Серверы имен» появившегося экрана нужно ввести следующие серверы имен:

* ns1.digitalocean.com
* ns2.digitalocean.com
* ns3.digitalocean.com
  1. Настройка SSL шифрование

Для настройки был использован сервис Let's Encrypt и программное обеспечение Certbot для получения и установки бесплатных сертификатов SSL.

Let's Encrypt — это Certificate Authority (CA), который обеспечивает простой способ получения и установки бесплатных сертификатов SSL, тем самым обеспечивая возможность использования зашифрованного HTTPS на веб-серверах. Это упрощает процесс, предоставляя программного клиента Certbot, который пытается автоматизировать большинство (если не все) необходимых шагов.

Certbot предоставляет множество способов получения сертификатов SSL с помощью различных плагинов.

Чтобы получить сертификаты SSL для URL-адресов example.com и www.example.com, нужно использовать следующую команду:

$ sudo certbot certonly --standalone --preferred-challenges http \

--deploy-hook "systemctl restart coturn" \

-d turn.example.com

Эта команда запускает certbot и использует директиву -d для указания имен, для которых нужно, чтобы сертификаты были действительными. После этого Certbot свяжется с сервером Let's Encrypt, а затем запустит запрос, чтобы подтвердить, что вы контролируете домен, для которого запрашиваете сертификат.

Сертификаты Let's Encrypt действительны только в течение девяноста дней. Это должно побудить пользователей автоматизировать процесс обновления сертификатов.

Пакет certbot, который был использован, позаботится об этом, запустив certbot refresh дважды в день с помощью таймера systemd. В дистрибутивах, отличных от systemd, эта функциональность обеспечивается сценарием, размещенным в /etc/cron.d. Эта задача выполняется два раза в день и обновляет любой сертификат, срок действия которого истекает в течение тридцати дней.

* 1. Конфигурация TURN сервера

Для установки coturn на ubuntu необходимо выполнить команду:

$ sudo apt-get install coturn

На сервере coturn необходимо иметь следующие порты (в дополнение к порту 22), (порт 3478 и 443), а также для подключения coturn к клиенту (49152 - 65535).

|  |  |  |
| --- | --- | --- |
| Порты | Протокол | Описание |
| 3478 | TCP/UDP | coturn listening port |
| 443 | TCP/UDP | TLS listening port |
| 49152-65535 | UDP | relay ports range |

Конфигурация coturn хранится в файле /etc/turnserver.conf. Доступно множество опций, все они задокументированы в комментариях к этому файлу. Ниже приведены самые важные части из конфигурации с комментариями, указывающими параметрами:

# TURN listener port for UDP and TCP (Default: 3478).

# Note: actually, TLS & DTLS sessions can connect to the

# "plain" TCP & UDP port(s), too - if allowed by configuration.

#

listening-port=3478

# In more complex case when more than one IP address is involved,

# that option must be used several times, each entry must

# have form "-X <public-ip/private-ip>", to map all involved addresses.

# RFC5780 NAT discovery STUN functionality will work correctly,

# if the addresses are mapped properly, even when the TURN server itself

# is behind A NAT.

#

# By default, this value is empty, and no address mapping is used.

#

external-ip=138.68.69.56

# Uncomment to use fingerprints in the TURN messages.

# By default the fingerprints are off.

#

fingerprint

# Uncomment to use long-term credential mechanism.

# By default no credentials mechanism is used (any user allowed).

#

lt-cred-mech

# 'Static' user accounts for long term credentials mechanism, only.

# This option cannot be used with TURN REST API.

# 'Static' user accounts are NOT dynamically checked by the turnserver process,

# so that they can NOT be changed while the turnserver is running.

#

user=username:password

# Certificate file.

# Use an absolute path or path relative to the

# configuration file.

#

cert=/etc/letsencrypt/live/turn.amempire.ru/fullchain.pem

# Private key file.

# Use an absolute path or path relative to the

# configuration file.

# Use PEM file format.

#

pkey=/etc/letsencrypt/live/turn.amempire.ru/privkey.pem

# Server relay. NON-STANDARD AND DANGEROUS OPTION.

# Only for those applications when we want to run

# server applications on the relay endpoints.

# This option eliminates the IP permissions check on

# the packets incoming to the relay endpoints.

#

server-relay

Пакет ubuntu для coturn требует, чтобы отредактировать файл, чтобы разрешить запуск. Файл / etc / default / coturn был редактирован и была написана следующая строка:

TURNSERVER\_ENABLED=1

После этого можно запустить сервис при помощи командой:

$ systemctl start coturn

* 1. Архитектура приложения

Есть сервер сигнализации, который координирует начало связи. Как только peer-to-peer соединение установлено, сервер сигнализации не нужен для связи. На рисунке 3 показана как инициализируется связь:

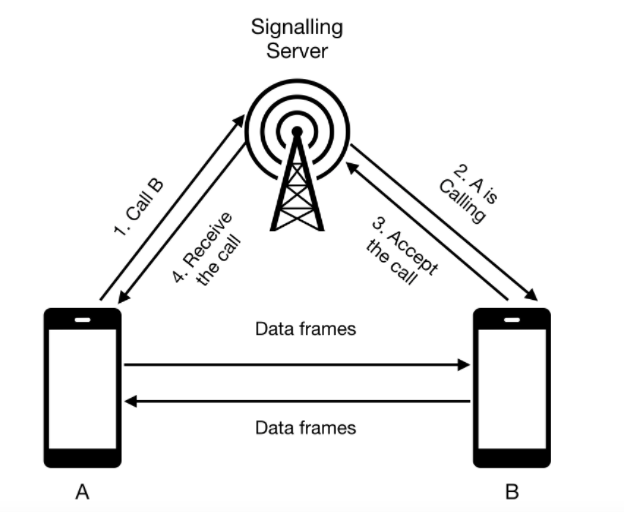


Рисунок 3 Инициализация подключения при помощи Signalling Server

(A) подготавливает так называемый SDP - протокол описания сеанса, который называется «offer».

(A) отправляет этот offer на сервер сигнализации, чтобы запросить подключение к (B).

Затем сигнальный сервер отправляет этот «offer» (B).

(B) получает offer, создает собственный SDP и отправляет его обратно на сервер сигнализации. Он называется «answer».

Затем сервер сигнализации отправляет этот «answer» на (A).

После этого создается peer-to-peer соединение для обмена данными.

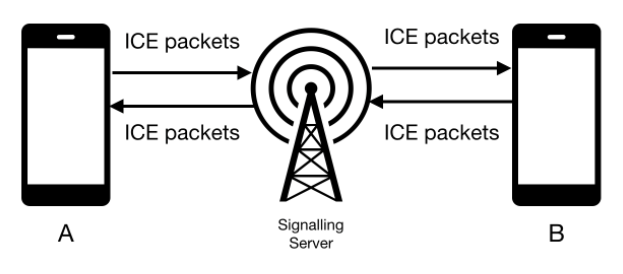
Что-то действительно важное происходит в фоновом режиме - (A) и (B) также обмениваются public IP-адресами (Рисунок 4). Это делается с помощью ICE - Interactive Connectivity Establishment - который использует сторонний сервер для получения общедоступного IP-адреса. Эти серверы известны как STUN или TURN.

Рисунок 4Передача Ip адресов при помощи Signalling Server (Ice Candidates)

1. начинает отправлять «ICE-пакеты» с момента создания «предложения».
2. Точно так же (B) посылать «ответ».
3. Технологический раздел
5. 1. Язык для Signaling сервера и разработки мобильного приложения

Для разработки серверной части было принято решение использовать Kotlin. Он является статически типизированный, объектно-ориентированный язык программирования, работающий поверх Java Virtual Machine. Также компилируется в JavaScript и в исполняемый код ряда платформ через инфраструктуру LLVM.

Kotlin полностью совместим с Java. Все Java-фреймворки и библиотеки доступны в Kotlin и наоборот. Kotlin разработчик обычно пишет до 40% меньше кода, чем java разработчик без потери скорости выполнения. Можно перечислить множество плюсов этого языка, такие как безопасность Null, улучшение лямбда выражения, деструктурирующие объявления, перегрузка операторов, data classes, именованные аргументы и многое другое.

Для разработки андроид приложений можно сказать, что Kotlin является „стандартом“ на сегодняшний день.

* 1. Технология для Signaling сервера

Для разработки серверной части был использован **Spring Framework**. Главное, что позволяет сделать Spring – упростить разработку J2EE приложений для разработчика.

Основные преимущества:

* Spring предоставляет каркас приложения или, другими словами, «заготовку» для приложения. При этом фреймворк диктует правила построения приложения – есть определенная архитектура приложения, в которую нужно встроить свою функциональность. Эта функциональность, собственно, и будет бизнес-логикой приложения. В состав Spring входит много подпроектов, «заточенных» под определенную функциональность (SpringMVC, Spring Security, SpringData и др., из которых разработчик может выбрать наиболее подходящий ему, а остальные не использовать – это модульный принцип построения приложения;
* В приложении на основе Spring объекты слабосвязаны за счет использования внедрения зависимостей (DI – Dependency injection). Одной из целью создания Spring было разорвать зависимость одних объектов от других. Это когда объект Object1 использует методы другого объекта Object2, т.е объект Object1 зависит от объекта Object2, чьи методы он использует. Пока объект Object2 не создан, Object1 не сможет реализовать свою функциональность. Для того, чтобы разорвать зависимость, в объект Object1 можно «инжектировать» ссылку на объект Object2 через конструктор или сеттер. Этот процесс, собственно, и есть внедрение зависимости. При этом важно помнить, что в Spring объекты необходимо строить на основе интерфейсов, чтобы зависимости внедряются в виде интерфейса для возможной последующей замены реализации.
* Не нужно создавать объекты вручную с помощью оператора new. Эту функцию делегировали контейнеру Spring. Это и есть инверсия контроля (IoC) – передача функции инстанциирования необходимых зависимостей (объектов) контейнеру. Какова роль разработчика во всем этом спросите вы? Объявить компонент, что бы он попал в контекст Spring. Контекст Spring-a по простому – мапа, где находятся все бины. Когда говорят, что бин находиться в контексте спринга – можно считать, что бин лежит в мапе, а спринг знает key, чтоб его достать из мапы. Все, что отмечено как бин в xml конфигурации или в классах анотациями @Component - инстанциируется и кладется в мапу вида Map<key,bean> map, т.е. у контейнера есть мапа, где он «складирует» все бины (ключевое понятие в Spring – бин, это управляемая контейнером сущность. Для того, чтобы бин (обычный класс) стал управляемым, он должен попасть в контекст спринга.) и при необходимости внедрения, контейнер делает примерно следующее: map.get(key), в качестве key выступает тип поля;
  1. Turn сервер для обхода NAT

Для TURN сервера была выбрана реализация coturn. Оно является бесплатной реализации TURN и STUN серверов с открытым исходным кодом. Сервер TURN является сервером и шлюзом для прохождения через NAT медиа-трафика VoIP. Его также можно использовать в качестве TURN-сервера общего назначения и шлюза сетевого трафика.

Доступен онлайн-интерфейс управления (через Telnet или HTTPS) для сервера TURN.

Реализация также включает некоторые дополнительные экспериментальные функции.

Вывод

В результате проделанной работы:

* Было написано мобильное приложение для видео интренет-разговоров.
* Был написан и развернут Listening сервер, при помощи которого осуществляется связь между мобильными приложениями
* Был настроен и развернут TURN сервер с персональным доменом и настроено шифрование SSL на выделенном сервере DigitalOcean.

Также были выполнены все задачи для достижения цели этой курсовой работы.

Для дальнейшей развитии можно предложить следующие пункты:

* Добавить дополнительный уровень шифрование
* Добавить удобный способ добавления новых пользователей
* Добавить возможность работы с другими протоколами как SRT

Список использованной литературы

1. tproger. Сообщество программистов [Электронный ресурс]. – Режим доступа: <https://tproger.ru/translations/udp_reliability/amp/> - (дата обращения: 24.11.2020г.)
2. Дуглас Камер. Сети TCP/IP, том 1. Принципы, протоколы и структура. Internetworking with TCP/IP, Vol. 1: Principles, Protocols and Architecture. — М.: «Вильямс», 2003. — С. 880
3. Udt. Документация [Электронный ресурс]. – Режим доступа: https://udt.sourceforge.io/- (дата обращения: 28.11.2020г.)
4. Haivision. Разница между SRT и UDT [Электронный ресурс]. – Режим доступа: <https://www.haivision.com/blog/broadcast-video/created-srt-difference-srt-udt/> (дата обращения: 5.12.2020г.)
5. webrtc. Официальная страница протокола [Электронный ресурс]. – Режим доступа: <https://webrtc.org/> (дата обращения: 7.12.2020г.)