**Изображение выглядит как логотип

Автоматически созданное описаниеМИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**(национальный исследовательский университет)»

**Институт (Филиал)** № 8 «Компьютерные науки и прикладная математика» **Кафедра** 806

**Группа** М8О-406Б-20 **Направление подготовки** 01.03.02 Прикладная математика и информатика

**Профиль** Информатика

**Квалификация: бакалавр**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

на тему: «Система анализа транзитного трафика с целью выявления актуальных цепочек взаимодействий в коммутационном оборудовании»

Автор ВКРБ: Почечура Артемий Андреевич (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Руководитель: Дзюба Дмитрий Владимирович (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Консультант: Зайцев Валентин Евгеньевич (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Консультант: - (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Рецензент:  (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

**К защите допустить**

Заведующий кафедрой № 806 «Вычислительная математика  
и программирование» Крылов Сергей Сергеевич (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

*\_\_\_\_* \_\_\_\_\_\_\_\_\_\_\_ 2024 года

Москва, 2024

## **РЕФЕРАТ**

Выпускная квалификационная работа бакалавра состоит из 43 страниц, 14 рисунков, 2 таблиц, 28 использованных источников, 1 приложения.

КОММУТАТОР, СЕТЬ, ВОССТАНОВЛЕНИЕ ПУТИ, CDR, ЗВОНОК, СВЯЗЬ, ВЫЗОВ, ТРАНК

Объектом разработки в данной работе является коммутационная сеть, предназначенная для передачи голосовых вызовов.

Цель работы – разработать программное обеспечение, которое будет загружать CDR коммутаторов из заданного источника, проводить их корреляцию и выгружать информацию о построенных цепочках.

Основное содержание работы состояло в разработке алгоритма, позволяющего сопоставить несколько CDR, созданных на разных коммутаторах, одному конкретному звонку.

Основным результатом работы является программное обеспечение, способное найти путь звонка внутри коммутационной сети на языке C++ с использованием СУБД PostgreSQL.

Данные результаты разработки предназначены для осуществления анализа коммутационной сети, благодаря которому можно будет оперативно проводить оптимизацию маршрутов трафика сети оператора связи в соответствии с меняющимися тарифами присоединенных операторов и состоянием загрузки каналов.

Использование программного обеспечения позволит получить данные, на основе которых можно будет предпринять действия по оптимизации использования коммутационной сети.

# **СОДЕРЖАНИЕ**

[РЕФЕРАТ 2](#_Toc167823806)

[ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ 4](#_Toc167823808)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 5](#_Toc167823809)

[ВВЕДЕНИЕ 6](#_Toc167823810)

[1 ТРАНСПОРТИРОВКА ЗВОНКОВ ВНУТРИ СЕТИ 9](#_Toc167823811)

[1.1 Устройство современных сетей связи 9](#_Toc167823812)

[1.2 Проблема восстановления пути звонка 12](#_Toc167823813)

[1.3 Техническое задание 13](#_Toc167823814)

[2 РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 15](#_Toc167823815)

[2.1 Модель организации информации о коммутационной сети и CDR записях. Восстановление пути звонка. 15](#_Toc167823816)

[2.2 Стек используемых технологий 19](#_Toc167823817)

[2.3 Алгоритм восстановления пути звонка внутри сети 21](#_Toc167823818)

[2.4 Исходный код программного обеспечения 24](#_Toc167823819)

[3 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ 30](#_Toc167823820)

[3.1 Область применения программного обеспечения 30](#_Toc167823821)

[3.2 Демонстрация программного обеспечения 31](#_Toc167823822)

[3.3 Тесты производительности 35](#_Toc167823824)

[ЗАКЛЮЧЕНИЕ 39](#_Toc167823825)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 40](#_Toc167823826)

[ПРИЛОЖЕНИЕ А Исходный код 43](#_Toc167823827)

# **ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ**

В настоящей выпускной квалификационной работе бакалавра применяют следующие термины с соответствующими определениями:

|  |  |
| --- | --- |
| Call Detail Record | – сервис в телекоммуникационной сфере, обеспечивающий журналирование работы телекоммуникационного оборудования |
| Транк | – виртуальный канал связи, передающий трафик между сетевыми устройствами |
| Коммутатор | – устройство в компьютерной сети, которое соединяет другие устройства вместе |
| Оператор | *–* человек (компания), осуществляющий коммутацию на телефонной станции, то есть соединяющий телефонных абонентов при звонке по телефонной линии |
| Звонок (вызов) | *–* действия, совершаемые абонентом или пользователем в целях установления соединения своего пользовательского оборудования с пользовательским оборудованием другого абонента или пользователя, и совокупность операций, порождаемых этими действиями в сети электросвязи |
| Вес маршрута звонка | *–* совокупность факторов, влияющих на выбор маршрута при транспортировке звонка. Таких факторами являются: цена (себестоимость) маршрута, качество, лицензионные ограничения в отрасли, технические возможности/ограничения сети компании и её партнеров, коммерческие условия взаимодействия с партнерами, объемы трафика, согласованные исключения (спец. маршруты) |
|  |  |

# **ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

В настоящей выпускной квалификационной работе бакалавра применяют следующие сокращения и обозначения:

|  |  |
| --- | --- |
| СУБД | – система управления базой данных |
| CDR | *–* call detail record |
| АТС | *–* автоматическая телефонная станция |
|  |  |
|  |  |

# **ВВЕДЕНИЕ**

Построение оптимальных маршрутов звонков между коммутаторами оператора связи является важной задачей, поскольку позволяет эффективно использовать ресурсы сети и обеспечивать качественное обслуживание абонентов. Одной из основных проблем, с которой сталкиваются операторы связи, является загруженность каналов связи. При неоптимальном распределении нагрузки возможны перегрузки и потери качества связи.

Для решения проблемы загруженности каналов операторы связи применяют различные методы оптимизации маршрутов звонков. Одним из таких методов является динамическое управление трафиком, при котором маршруты звонков пересматриваются и оптимизируются в реальном времени в зависимости от текущей нагрузки на сеть. Это позволяет операторам связи быстро реагировать на изменения в трафике и предотвращать перегрузки каналов.

Для оптимизации маршрутов звонков с учетом себестоимости операторы связи могут использовать различные алгоритмы, например, алгоритмы минимальной стоимости или алгоритмы оптимального пути. При этом учитывается не только вес маршрута, но и его стоимость, что позволяет экономить ресурсы и повышать эффективность использования сети.

Операторы связи также могут использовать присоединенных операторов для оптимизации маршрутов звонков. Например, при наличии у оператора связи нескольких альтернативных маршрутов для передачи звонков, можно выбрать тот маршрут, который принадлежит присоединенному оператору и обеспечивает более выгодные условия сотрудничества.

Важным аспектом при построении оптимальных маршрутов звонков является также обеспечение надежности и качества обслуживания. Операторы связи должны учитывать не только стоимость маршрута, но и его надежность, задержку и другие параметры, которые влияют на качество связи. Поэтому при выборе оптимального маршрута необходимо учитывать не только экономические, но и технические аспекты.

Отдельную сложность при анализе реальных маршрутов трафика представляет сложность технического ландшафта с точки зрения большого количества производителей коммутационного оборудования, и соответственно различия в логике обработки звонков в зависимости от типа оборудования.

В целом, построение оптимальных маршрутов звонков между коммутаторами оператора связи является сложной задачей, требующей комплексного подхода и использования различных методов оптимизации. Правильно построенные маршруты позволяют операторам связи эффективно использовать ресурсы сети, обеспечивать качественное обслуживание абонентов и повышать конкурентоспособность на рынке телекоммуникаций.

Таким образом, выполненная работа актуальна и с теоретической, и с практической точек зрения.

Цель работы – разработать программное обеспечение, которое будет загружать CDR коммутаторов из заданного источника, проводить их корреляцию и выгружать информацию о построенных цепочках.

Для достижения поставленной цели предстоит решить следующие задачи:

* настроить работу СУБД, в которой будет храниться информация о текущем состоянии коммутаторной сети и CDR записях;
* разработать алгоритм генерации данных, с помощью которого будет осуществляться маршрутизация звонков по оптимальным путям внутри коммутационной сети;
* придумать и реализовать алгоритм восстановления пути звонка внутри коммутационной сети посредством сопоставления CDR записей, относящихся к данному звонку, с сохранением порядка очерёдности их создания;
* провести полное тестирование программного обеспечения, варьируя входные тестовые данные относительно размера и степени диффузии, которая в них вносится;
* выбрать стек технологий, наиболее подходящий для выполнения вышепоставленных задач.

Результаты работы предназначены для нахождения прибыли компании, которая будет получена за транспортировку звонков внутри своей сети, а также её расходы за дальнейшую передачу звонка внешнему оператору.

Использование разработки позволит компании на основе полученных данных обнаружить недостатки в работе по транспортировке звонков внутри сети, вследствие которых компания терпит убытки.

# **1 ТРАНСПОРТИРОВКА ЗВОНКОВ ВНУТРИ СЕТИ**

## **1.1 Устройство современных сетей связи**

Сети телефонной связи [1; 2] представляют собой сложные инфраструктуры, которые позволяют обеспечить связь между абонентами, независимо от их местоположения. Телефонная связь является самым распространенным видом оперативной связи и играет важную роль в фирмах, офисах и жизни людей. Сети телефонной связи состоят из множества абонентов, которые могут быть как физическими лицами, так и организациями. Каждый абонент имеет уникальный номер телефона, который используется для идентификации при совершении звонков. Для установления связи абоненты используют телефоны или другие устройства, подключенные к сети телефонной связи [3]. Современные телефоны могут быть проводными или беспроводными.

Центральные офисы (или коммутационные центры) являются ключевыми элементами сети телефонной связи. Они обрабатывают и коммутируют звонки между абонентами, управляют ресурсами сети и обеспечивают качество обслуживания. Для передачи голосовой информации между центральными офисами используются магистральные линии. Эти линии представляют собой высокоскоростные кабели и оптоволоконные линии связи.

Для обмена информацией между различными узлами сети используются специальные протоколы и стандарты, такие как SS7 (Signaling System 7). Они обеспечивают эффективное управление звонками, передачу данных о вызовах и другие функции.

При совершении звонка сеть телефонной связи определяет оптимальный маршрут для доставки голосовой информации от отправителя к получателю. Оптимальный маршрут определяется путём выбора центральных офисов, магистральных линий и других узлов сети.

Для обеспечения связи между различными операторами связи существует процесс интерконнекта [4], который позволяет передавать звонки между различными сетями.

Сети телефонной связи [5] непрерывно контролируют качество обслуживания, чтобы обеспечить стабильное соединение и высокое качество звука для абонентов.

Данное описание применимо к большинству современных телефонных сетей. Среди всех вышеописанных элементов интересовать нас будет интерконнект.

Межоператорский голосовой interconnect [6] – это процесс обмена телефонными звонками между различными операторами связи. То есть, если абонент одного оператора звонит абоненту другого оператора, необходимо установить соединение между их сетями для передачи голосовой информации. Межоператорский голосовой interconnect включает в себя технические и коммерческие аспекты, такие как соглашения о тарифах за использование сети другого оператора и технические стандарты для обеспечения совместимости между различными сетями.

Interconnect Billing [7] занимается расчетом сумм, которые должны быть выплачены и получены от каждого из сетевых операторов, к которым подключается инфраструктура компании, для успешного инициирования и завершения вызова. В данном процессе участвует технология CDR, с помощью которой хранится информация о маршрутизации вызова в виде группы допустимых значений для идентификации оператора и сведений о стране. Рассмотрим концепцию CDR более подробно.

Call Detail Record [8] (записи данных о списаниях) в телекоммуникационной сфере – сервис, обеспечивающий журналирование работы телекоммуникационного оборудования, такого как коммутатор/АТС, IP-АТС, VoIP-шлюз, Виртуальная АТС и т.п.

CDR используется для расчета стоимости телефонных разговоров, оценки рациональности использования трафика, а также для сервисных нужд при настройке оборудования. Для решения этих задач CDR-сервис, как правило, работает с тарификационной или биллинговой программой. Тарификационные и биллинговые программы на основе данных CDR формируют счета на оплату телефонных разговоров, рассчитывают статистику загруженности линий и операторов, статистику активности абонентов и т. п.

Практически любая телефонная станция в том или ином виде предлагает тарификационный сервис, но разные производители могут использовать разные названия. Если сервис включен, то обслуживающий персонал может получить данные о каждом акте коммутации. Степень детализации и вид этой информации зависит от типа телекоммуникационного оборудования (модели изделия, его прошивки) и от его настройки. Как правило, для каждого состоявшегося звонка CDR-информация содержит:

* номера вызывающего и вызываемого абонента;
* время окончания разговора (время по часам АТС);
* длительность разговора;
* другая сопроводительная информация (тип звонка: входящий, исходящий, перевод и т. п., номер транка и номер группы или IP-адреса и порты в VoIP, дата и время начала разговора, длительность дозвона и т. п.).

Таким образом, обработка каждого вызова выглядит следующим образом:

* звонок попадает в сеть от внешнего оператора, который платит компании за маршрутизацию звонка;
* далее вызов проходит через некоторое количество коммутаторов внутри сети;
* каждый коммутатор, получив звонок, формирует CDR запись;
* когда звонок достигает нужного оператора, то он заканчивает путь внутри сети, а компания оплачивает дальнейшую транспортировку звонка.

Чтобы сделать выводы о том, выгодно ли происходит транспортировка звонков внутри сети, компании нужно получить достоверную информацию о каждом звонке: оценить прибыль и затраты по его обработке. Анализируя данные такого вида, а также учитывая архитектуру сети, можно сделать выводы об улучшении транспортировки звонков, затраты на которые в перспективе можно уменьшить.

## **1.2 Проблема восстановления пути звонка**

Чтобы оценить маржинальность по обработке конкретного звонка, требуется узнать, от какого оператора он пришёл к нам в сеть и какому оператору был передан. Для этого необходимо найти первую и последнюю CDR запись, созданную касательно данного звонка. На рисунке 1 продемонстрированы примеры маршрутов звонков.

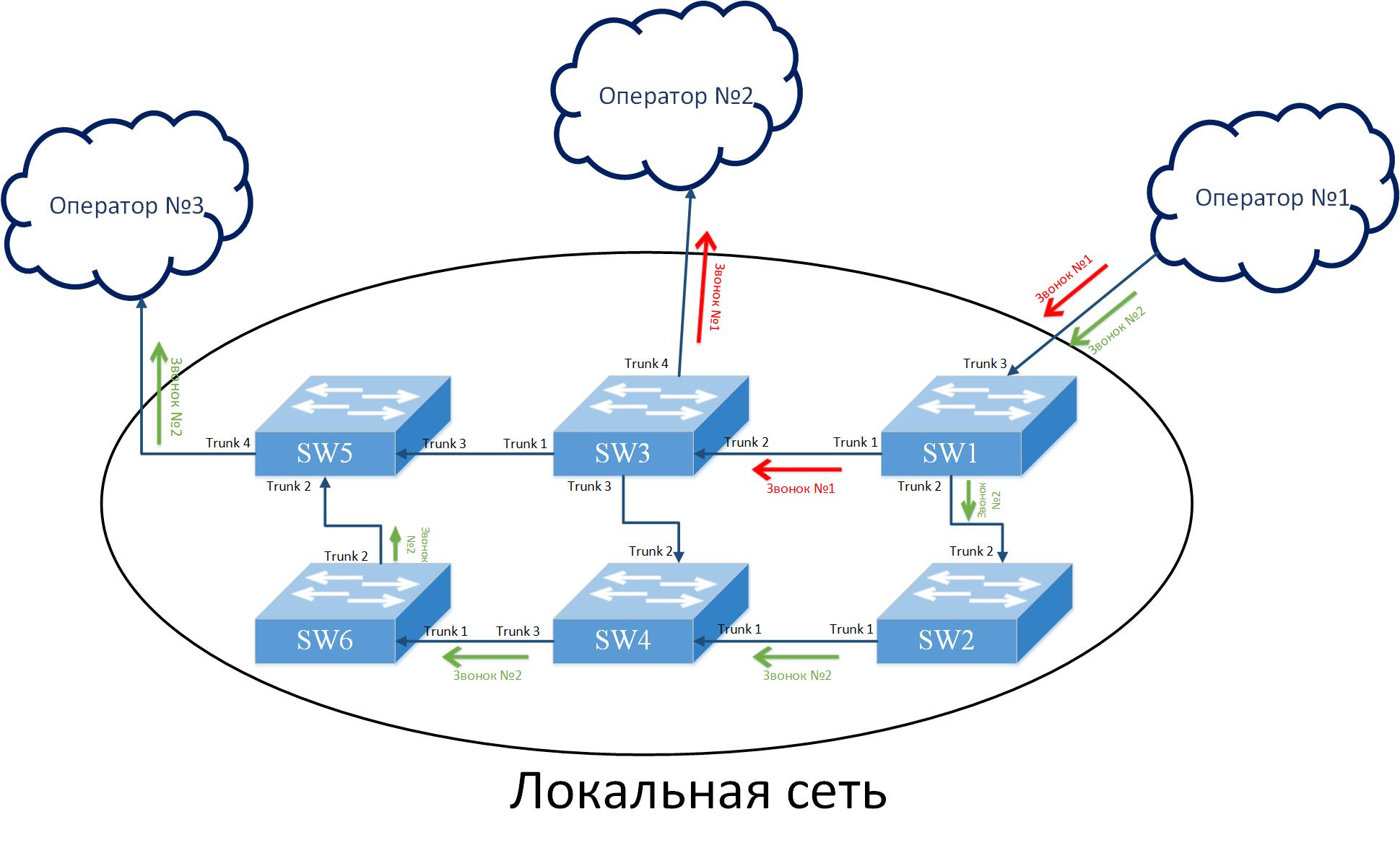


Рисунок 1 – Пример простой схемы сети

Сложность заключается в том, что таблица с CDR записями – это вся информация о звонках, которая имеется у компании. К сожалению, по этим данным не всегда удаётся однозначно определить, где звонок начал путь внутри сети и где он его закончил. Связано это с несколькими причинами:

* отсутствуют сквозные идентификаторы, которые могли бы позволить сопоставить исходящий CDR на одном коммутаторе с входящим CDR на другом коммутаторе;
* есть "погрешности" в данных, выгружаемых коммутаторами: например, часы на коммутаторах не синхронизированы и сопоставить звонок по времени совершения и его длительности напрямую не получится;
* какой-то из коммутаторов может не сформировать CDR запись;
* большое количество событий (примерно, 500 миллионов CDR в сутки).

Учитывая всё вышеперечисленное, проанализировать путь, по которому звонок перемещался внутри сети, становится довольно проблематично. Ни один из известных и доступных алгоритмов не может напрямую выполнить поставленную задачу достаточно оптимально, из-за чего компания рискует получить серьёзные убытки.

## **1.3 Техническое задание**

Задача сводится к нахождению первого и последнего CDR для каждого звонка, которые были созданы в процессе его маршрутизации внутри сети. Перебор всех возможных пар CDR выглядит довольно очевидным решением, однако такой подход не предоставит необходимую информацию для улучшения используемых маршрутов. Попробуем переформулировать постановку задачи: для каждого звонка требуется найти всю цепочку CDR в той последовательности, в которой они были созданы при перемещении звонка внутри сети. Сама цепочка будет восстанавливаться постепенно: для каждой CDR записи будет производиться поиск похожей на неё записи на коммутаторе, на который далее ушёл вызов. Все необходимые данные будем получать из информации о CDR записях, а также из информации об устройстве сети (всех соединениях в ней).

CDR записи, относящиеся к одному звонку и созданные один за другим, имеют схожие параметры. Для корректного сравнения CDR сформулируем некоторые правила:

* в обеих записях поля входящего и исходящего абонентов заполнены одинаково;
* среди всех остальных записей, подряд идущие CDR имеют наименьшее различие по времени создания (после приведения в единый часовой пояс) и данным о длительности звонка.

CDR записи обязательно должны содержать следующую информацию:

* время создания CDR;
* длительность звонка;
* номера входящего и исходящего абонента;
* коммутатор, на котором данная запись была создана;
* входящий транк, по которому звонок пришёл на коммутатор;
* исходящий транк, по которому звонок ушёл с коммутатора на другое устройство в сети или покинул сеть.

Данные об устройстве сети будут сформированы в виде описания всех соединений в ней. Каждая запись – конкретное соединение, сообщающее о том, какие два коммутатора в нём участвуют, кто из них является исходящим, а кто – входящим, транки коммутаторов, задействованных в соединении, а также часовой пояс, в котором ведётся отсчёт времени на исходящем коммутаторе.

# **2 РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## **2.1 Модель организации информации о коммутационной сети и CDR записях. Восстановление пути звонка.**

Сначала нужно подумать, в каком виде хранить данные о CDR записях и коммутационной сети. Решено было использовать для этого две таблицы.

Первая таблица имеет название Switches. С помощью неё опишем нашу сеть: каждая новая запись будет содержать сведения о соединении между двумя коммутаторами. Во второй таблице будет содержаться необходимая информация о всех CDR, её название – CDRs. На рисунке 2 приведена логическая модель описанной базы данных.

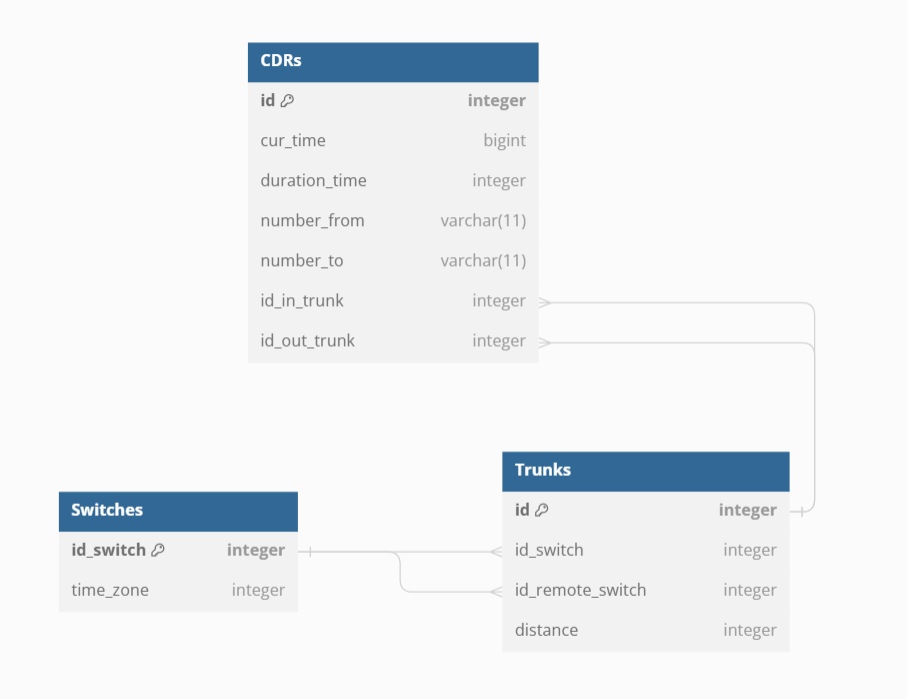


Рисунок 2 – Диаграмма организации СУБД

Чтобы информация о коммутаторной сети была полноценной, нам потребуется хранить в таблице Switches данные о всех соединениях между коммутаторами, а также протяженность каждого соединения и временную зону одного из коммутаторов в соединении (такой коммутатор в соединении всегда определяется однозначно). Данная структура позволит нам при необходимости иметь информацию о всех исходящих соединениях, которые имеются у каждого коммутатора, а также узнать, откуда на конкретный коммутатор пришёл вызов: из внешней сети или от устройства внутренней сети; и куда ушёл вызов с коммутатора: во внешнюю сеть или продолжил путь по нашей сети.

В таблице CDRs будет записываться информация о том, в какое время был создан CDR, на каком коммутаторе, номер исходящего и входящего абонента, длительность звонка и какие транки использовались для получения и передачи звонка на данном коммутаторе.

При генерации данных необходимо вносить некоторую диффузию в CDR записи, чтобы соответствовать реальным условиям, в которых планируется использовать алгоритм. Вызов от коммутатора к коммутатору передаётся не мгновенно, это занимает некоторое время. Поэтому каждый раз при добавлении нового CDR будем смещать время его создания на некоторое значение относительно прошлой записи.

Время создания CDR решено было представлять в формате Unix Timestamp [9]. В нём время исчисляется в секундах, начиная с 1 января 1970 года (для наших целей будем использовать представление в миллисекундах). Такой формат позволяет с лёгкостью сравнивать времена, в которые CDR записи были созданы.

Длительность звонка в реальных условиях также может сместиться. Например, при завершении вызова, сигнал, сообщающий об окончании звонка, доходит до исходящего абонента не сразу, а с некоторой задержкой. Поэтому длительность звонка в записях также необходимо смещать на некоторое значение. Данный параметр, как и время создания CDR записи на коммутаторе, будет записываться в миллисекундах.

В коммутационной сети между устройствами есть некоторое расстояние. При генерации данных будем считать, что звонок всегда идёт по минимальному расстоянию от одного оператора до другого. Звонков за сутки может быть довольно много, и если учесть, что коммутационная сеть имеет непростую архитектуру, то следует вывод, что минимальный путь [10] должен быть вычислен заранее. Также нужно помнить, что устройств, на которых звонки начинают свой путь в нашей сети, может быть несколько, как и устройств, через которые звонки выходят из нашей сети. Исходя из данных условий лучше всего подойдёт алгоритм Флойда-Уоршелла [11], который позволит найти и сохранить кратчайшее расстояние между любыми двумя устройствами в сети за сложность O(), где *n* – количество устройств в сети.

После генерации данных нужно подумать над тем, как найти путь каждого звонка по нашей сети. Пометив с помощью отдельного поля в таблице CDRs первые CDR в маршруте каждого звонка, будем постепенно переходить от коммутатора к коммутатору, каждый раз проверяя, есть ли похожие записи в таблице с той, что имеется в данный момент у нас (изначально имеется первый CDR в пути). Для сравнения будут использоваться следующие метрики: время, в которое CDR был создан на коммутаторе, а также длительность звонка. Чтобы новый CDR был пригодным для продолжения цепочки, он должен соответствовать следующему условию:

(1), где

* – различие времён между CDR записями, в которое они были созданы на коммутаторах;
* – коэффициент учёта различия данных о времени создания;
* – различие в данных о длительности звонка между CDR записями;
* – коэффициент учёта различия данных о длительности звонка;
* – пороговое значение, которое по указанной метрике не должны превышать CDR записи, относящиеся к одному звонку и находящиеся в цепочке друг за другом.

Корректировка значений коэффициентов и позволит учитывать в меньшей или большей мере различия в длительности звонка и времени создания записи.

Корректировка параметра позволит строже или лояльнее относится к подбору CDR записей, пригодных для продолжения цепочки пути.

Перед сравнением параметров с помощью данной метрики необходимо перевести времена CDR к одному часовому поясу (в данной работе используется часовой пояс GMT+0). Для этого нужно воспользоваться таблицей Switches и получить необходимые данные (часовой пояс каждого из коммутаторов, в котором на них ведётся отсчёт времени).

В случае, если на коммутаторе не нашлось CDR, удовлетворяющего условию (1), то это значит, что запись на данном коммутаторе “потерялась”. В таком случае необходимо просмотреть информацию о CDR на всех соседних с ним коммутаторах для поиска записи, которая подойдёт для продолжения цепочки. Современные сети гарантируют, что потери двух подряд CDR одного звонка произойти практически не может, поэтому после выполнения описанных действий мы должны продолжить восстановление цепочки. Если повторный поиск не нашёл подходящей записи, то это означает, что пороговое значение метрики для сравнения двух CDR задано достаточно строго и не позволяет определить путь для некоторой части звонков.

Если запись пропала на коммутаторе, который участвовал в передаче звонка последним, то в таком случае мы не можем оценить убытки компании, так как у нас имеется информация только о перемещениях звонка по локальной сети и о внешнем операторе, от которого был получен звонок.

По окончании поиска пути, по которому перемещался звонок внутри сети, мы будем знать следующее:

* от какого оператора был принят звонок в нашу сеть;
* какие коммутаторы были задействованы при транслировании звонка в локальной сети, а также порядок, в котором они были задействованы;
* какому оператору был передан звонок из нашей сети.

Полученные данные позволят компании посчитать маржинальность по каждому звонку (различие в затратах и доходе за оказанные услуги), судить об эффективности работы сети по транспортировке звонка, а также обнаружить перегруженные и мало задействованные узлы сети.

## **2.2 Стек используемых технологий**

Основным языком программирования, на котором был реализован алгоритм, является C++. Именно этот язык позволяет оперативно производить вычисления, а также взаимодействовать с базами данных без особых усилий. Логика языка позволяет вынести структуры и объявление некоторых функций в отдельный файл, при этом взаимодействовать с ними можно в основном файле.

CMake (Cross-platform make) [12] – кроссплатформенное программное средство автоматизации сборки программного обеспечения из исходного кода. С помощью данного инструмента создаётся файл с названием Makefile [13] на основе библиотек и программ, использованных в работе. Выполнение Makefile файла осуществляется с помощью команды make, что приводит к запуску разработанного программного обеспечения.

POCO (C++ Portable Components) [14] – это коллекция библиотек классов с открытым исходным кодом, которая упрощает и ускоряет разработку сетевых мультиплатформенных приложений на C++. В работе использовался подкласс библиотек Poco/Data/PostgreSQL для взаимодействия с базами данных. Данная технология позволяет обрабатывать SQL запросы внутри среды разработки. Для того, чтобы начать работу с POCO, достаточно развернуть удалённый сервер в контейнере с помощью Docker и подключить данную библиотеку, что крайне удобно.

PostgreSQL [15] – СУБД, с помощью которой осуществляется работа с базами данных. PostgreSQL имеет простой синтаксис команд и инструменты для взаимодействия с данными, из-за чего внесение всех характеристик коммутационной сети и поиск требуемой информации осуществляется крайне легко.

Docker [16; 17] создаёт два контейнера: один используется для хранения баз данных, а второй служит для развёртывания удалённой среды разработки, в которой запускается программное обеспечение. Благодаря Docker можно быстро развёртывать и масштабировать приложения в любой среде.

Для простого взаимодействия с контейнерами был создан docker-compose [18] файл, в котором для каждого контейнера были описаны все необходимые характеристики.

С помощью Microsoft Visio [19] производилась работа по созданию графических иллюстраций для демонстрации устройства сетей связи. В редакторе имеется широкий выбор инструментов для создания схем, что даёт возможность создавать качественные изображения.

Structurizr [20] позволяет создавать диаграммы на основе модели C4 [21] для архитектуры программного обеспечения. Чтобы описать параметры диаграммы, требуется написать небольшую программу, в которой указываются все необходимые характеристики.

Язык Python в работе применялся для отрисовки графиков на этапе тестирования алгоритма. Библиотека Matplotlib [22] имеет простой синтаксис, благодаря которому несложно настраивать параметры графиков.

Microsoft Visual Studio Code [23] – текстовый редактор, в котором происходила разработка программного обеспечения. Библиотека расширений, которую Visual Studio Code предоставляет пользователю, позволяет взаимодействовать с большим количеством различных программ (мною были использованы расширения для работы с Docker и Structurizr) и языками программирования (C++, Java и т.д.), что делает процесс реализации алгоритма крайне удобным.

## **2.3 Алгоритм восстановления пути звонка внутри сети**

Для описания коммутационной сети будем использовать таблицу Switches, в которой будут содержаться следующие поля:

* id – номер соединения в сети;
* id\_switch – номер коммутатора в сети;
* id\_out\_trunk – номер исходящего транка рассматриваемого коммутатора;
* id\_remote\_switch – номер удалённого коммутатора;
* id\_remote\_trunk\_in – номер входящего транка на удалённом коммутаторе;
* time\_zone – временная зона, в которой происходит исчисление времени на рассматриваемом коммутаторе;
* distance – расстояние между текущим и удалённым коммутатором.

Для хранения информации о CDR записях потребуются следующие поля, которые будут храниться в таблице CDRs:

* id – идентификатор записи;
* cur\_time – время, в которое была создана запись на коммутаторе;
* duration\_time – длительность звонка;
* number\_from – номер исходящего абонента;
* number\_to – номер входящего абонента;
* id\_switch – номер коммутатора, на котором был создан CDR;
* id\_in\_trunk – номер транка, по которому звонок пришёл на коммутатор;
* id\_out\_trunk – номер транка, по которому звонок ушёл с коммутатора;
* is\_start – поле для обозначение первых CDR в цепочках звонков.

В начале работы алгоритма требуется настроить подключение к базе данных и создать в ней две таблицы, описанные выше. После этого потребуется описать локальную сеть, которую мы будем использовать для передачи звонков между внешними операторами. Чтобы отличать внешнего оператора от устройства локальной сети, будем обозначать выходные транки узлов всех операторов отрицательным числом. Для каждого внешнего оператора данное число будет уникальным, чтобы по восстановленной цепочке можно было понять, от какого оператора звонок поступил и какому оператору он был передан для дальнейшей маршрутизации.

Затем, используя данные из таблицы Switches задаём граф. Он представляет собой список смежности: для каждого устройства содержится информация о том, на какие коммутаторы может перейти вызов от рассматриваемого. Также нам понадобится двумерный массив, который для каждой пары устройств будет содержать информацию о соединении между ними: входной и выходной транк.

Для определения минимального расстояния между любой парой коммутаторов потребуется создать массив, в который будет внесена информация о протяжённости всех прямых соединений в сети. Расстояния между устройствами, у которых нет прямого соединения, изначально в данном массиве будут равняться условной бесконечности.

Далее происходит вызов функции, которая с помощью алгоритма Флойда-Уоршелла определяет минимальное расстояние между любыми двумя устройствами. Во время каждой итерации алгоритма, в которой мы находим новый кратчайший путь между парой коммутаторов, будем запоминать номер устройства, который стал для этой пары связующим, в отдельный массив parents. Данный массив будет использоваться для формирования кратчайшего пути.

После окончания работы алгоритма нахождения кратчайших путей требуется запомнить полный найденный путь для каждой пары коммутаторов. Выполняется это с помощью рекурсивного алгоритма, который, используя массив parents, разбивается на две ветви: нахождение пути от начальной вершины и связующего звена, и нахождение пути от связующего звена до конечной вершины. Таким образом, постепенно будет собран полный путь, который заносится в массив pathes.

Теперь займёмся транспортировкой звонков. Зададим необходимые параметры для каждого звонка с помощью генератора. Из массива pathes получим список коммутаторов в порядке, в котором нужно их обойти для достижения требуемого оператора. При переходе от устройства к устройству будем делать соответствующие записи в таблице CDRs, не забывая вносить диффузию в параметры (время, в которое CDR был создан на коммутаторе, и длительность звонка) при каждом переходе. Также при указании времени создания CDR необходимо учитывать возможное различие часовых поясов, в которых коммутаторы хранят время. Поле is\_start первого CDR в пути каждого звонка задаётся значением true.

После того, как вся необходимая информация была сформирована и добавлена, начинается алгоритм восстановления цепочек путей для каждого звонка. Будем брать из таблицы CDRs информацию о первом CDR (с помощью поля is\_start) для конкретного звонка и согласно данным о номере транка, по которому перешёл звонок дальше, а также информации о том, какой коммутатор связан с текущим через этот транк, ищем в таблице другие CDR, которые были созданы на следующем коммутаторе, и входящий и исходящий абоненты в которых совпадают с текущим звонком.

Далее требуется понять, какой из найденных CDR подходит нам больше всего для продолжения пути. Воспользуемся метрикой, которая будет совмещать в себе учёт данных о времени создания CDR и длительности звонка. Оба параметра будут учитываться в соответствии с некоторым коэффициентом. Из всех CDR требуется найти тот, различие которого по описанной метрике от текущего CRD будет минимальным. Также это различие должно быть не больше граничного значения, которое задаётся заранее.

Таким образом, найдя CDR, который является наиболее подходящим для продолжения пути звонка, становится текущим и алгоритм переходит на следующую итерацию. В случае, если подходящей записи найти не удалось, алгоритм действует следующим образом: на соседних коммутаторах с тем, на котором мы собирали информацию в начале итерации, осуществляем поиск CDR по таким же параметрам. Различие нового CDR от текущего не должно превышать уже удвоенной метрики, и если такой найдётся, то он становится текущим. Если ни одна запись не подошла под указанные требования, то выводится информация о том, что восстановить путь, используя данное граничное значение метрики при сравнении, для данного звонка не удастся.

Сообщение о том, что путь невозможно восстановить, может также вывестись в случае, когда запись была потеряна на коммутаторе, исходящие соединения которого направлены в сторону внешних операторов. Однозначно сказать, куда пошёл звонок, в данной ситуации невозможно, так как мы обладаем только информацией о том, как двигался звонок внутри нашей сети.

Алгоритм заканчивает восстановление пути звонка в тот момент, когда входящий транк устройства, по которому был передан звонок, равняется отрицательному числу. Это означает, что звонок вышел из нашей сети и цепочка полностью восстановлена. При выводе результата в самом начале указывается оператор, от которого был получен звонок в локальную сеть. Осуществляется это следующим образом: у первого CDR, с которого начался путь звонка по нашей сети, нужно посмотреть значение исходящего транка устройства, с которого звонок пришёл на текущий коммутатор. Если данное значение отрицательное, то выводим информацию о данном узле как о внешнем абоненте.

## **2.4 Исходный код программного обеспечения**

Программа начинается с настройки подключения к базе данных, что отражено на рисунке 3. Для этого необходимо указать параметры подключения: порт, логин, пароль и тип СУБД. В случае успешного подключения необходимо создать переменную сессии, с помощью которой будет осуществляться дальнейшее взаимодействие с базой данных.

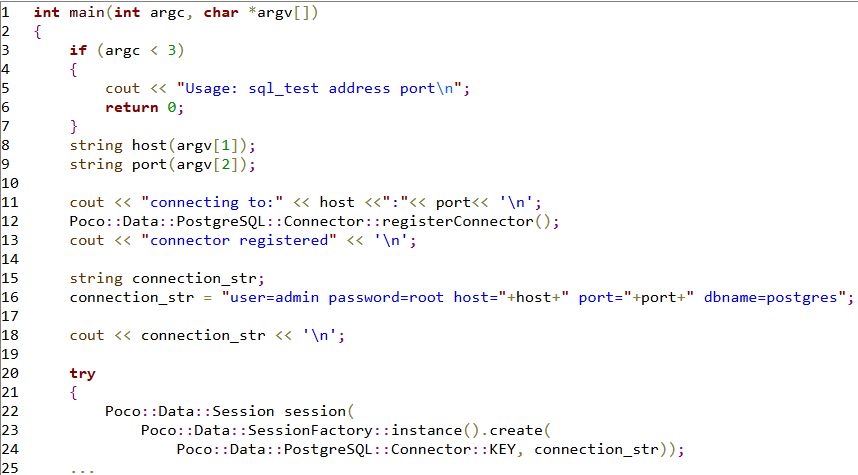


Рисунок 3 – Подключение к базе данных

После подключение к базе данных происходит вызов генератора, в котором создаются таблицы Switches и CDRs и задаются параметры устройства сети. Этот процесс проиллюстрирован на рисунке 4. Важным моментом является то, что устройства, обозначающие внешних операторов, имеют транки с отрицательным значением.



Рисунок 4 – Создание таблиц и обозначение параметров сети

Далее по указанной информации о сети формируются кратчайшие пути, которые будут в дальнейшем использоваться для генерации CDR записей. Как говорилось ранее, для определения кратчайших путей используется алгоритм Флойда-Уоршелла, реализацию которого можно увидеть на рисунке 5. Формирование и сохранение кратчайших путей производится с помощью рекурсивной функции find\_path.

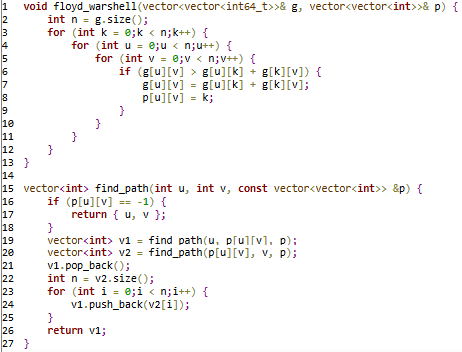


Рисунок 5 – Алгоритм Флойда-Уоршелла и функция

сохранения кратчайших путей

После нахождения кратчайших путей вызывается функция генерации списка звонков, в которой случайным образом [24] определяются все необходимые параметры: номера исходящего и входящего абонентов, длительность звонка, время начала вызова и другое. Так как внешние операторы могут иметь двустороннюю связь с нашей сетью, исходящий и входящий оператор для генерируемого звонка может совпасть. В таком случае необходимо изменить данные о входящем или исходящем операторе в вызове. На рисунке 6 продемонстрирован полный код данной функции.

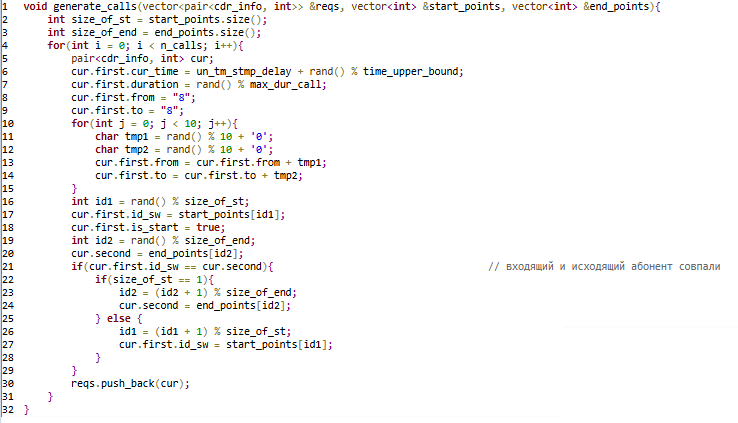


Рисунок 6 – Функция генерации звонков

Затем сгенерированные звонки маршрутизируются по сети. Попадая на коммутатор, в таблице генерируется новый CDR с информацией о пришедшем звонке. С некоторым шансом CDR запись может “потеряться”, но только в том случае, если данный CDR не является первым в пути и предыдущий CDR рассматриваемого звонка не был потерян. Вероятность потери CDR можно варьировать.

Во время работы алгоритма восстановления путей звонков ключевой функцией является нахождение CDR для продолжения цепочки, которая приведена на рисунке 7. В ней среди всех записей на указанном коммутаторе производится поиск тех, которые имеют различие с текущим последним CDR в цепочке не больше порогового значения. Среди всех таких CDR отбирается наиболее подходящий для продолжения пути. Если нужной записи найти не удалось, то в качестве id нового CDR возвращается значение -1.

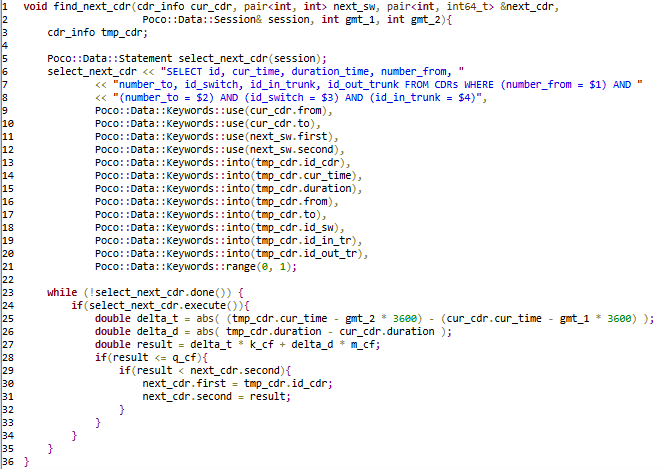


Рисунок 7 – Функция поиска следующего CDR в цепочке

# **3 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ**

## **3.1 Область применения программного обеспечения**

Разработанное программное обеспечение применимо в отрасли связи для оценки маржинальности предоставляемых услуг компанией-оператором по транспортировке звонков внутри своей сети. Оператор собирает информацию о всех CDR записях, созданных в течение определённого промежутка времени (например, в течение дня). Далее накопленные данные вместе с информацией о структуре используемой коммутационной сети оформляются в виде двух таблиц и передаются на вход алгоритму. По окончании работы программы для каждого звонка будет сформирован его полный путь внутри сети, а также сведения о том, от какого оператора звонок был принят в сеть и какому оператору был передан для дальнейшей маршрутизации. Зная, какую сумму необходимо заплатить каждому внешнему оператору за использование услуг по транспортировке звонка, компания легко может вычислить свою прибыль. Аналогично можно будет вычислить и затраты за передачу звонка другому оператору для последующей обработки. Таким образом, станет известно, какую выручку или убыток для компании принесла маршрутизация каждого звонка.

Полученная в ходе работы алгоритма информация поможет компании в принятии следующих решений:

* изменить схемы транспортировки звонков для увеличения маржинальности предоставляемых услуг;
* отказаться от обработки звонков, которые всегда будут являться невыгодными для компании.

Данные о полных маршрутах вызовов также позволят компании улучшить эффективность и скорость обработки звонков. На основе имеющихся сведений несложно будет обнаружить перегруженные участки сети, которые способствуют задержке сетевого трафика и изнашиванию оборудования. Такие участки можно будет разгрузить путём перераспределения звонков по другим маршрутам.

Также программное обеспечение вычисляет процент полностью восстановленных путей звонков, что может использоваться компанией для проверки качества работы коммутационной сети. Определив предельное время, за которое звонок должен обрабатываться между двумя соседними коммутаторами, станет известно, в какой части вызовов соблюдается необходимое требование. Если полученное значение не удовлетворяет компанию, то требуется рассмотреть возможность повышения качества оборудования или внесения корректировок в схему транспортировки звонков.

## **3.2 Демонстрация программного обеспечения**

## На рисунке 8 приведена диаграмма развёртывания (deployment diagram) программного обеспечения, отражающая его архитектуру.

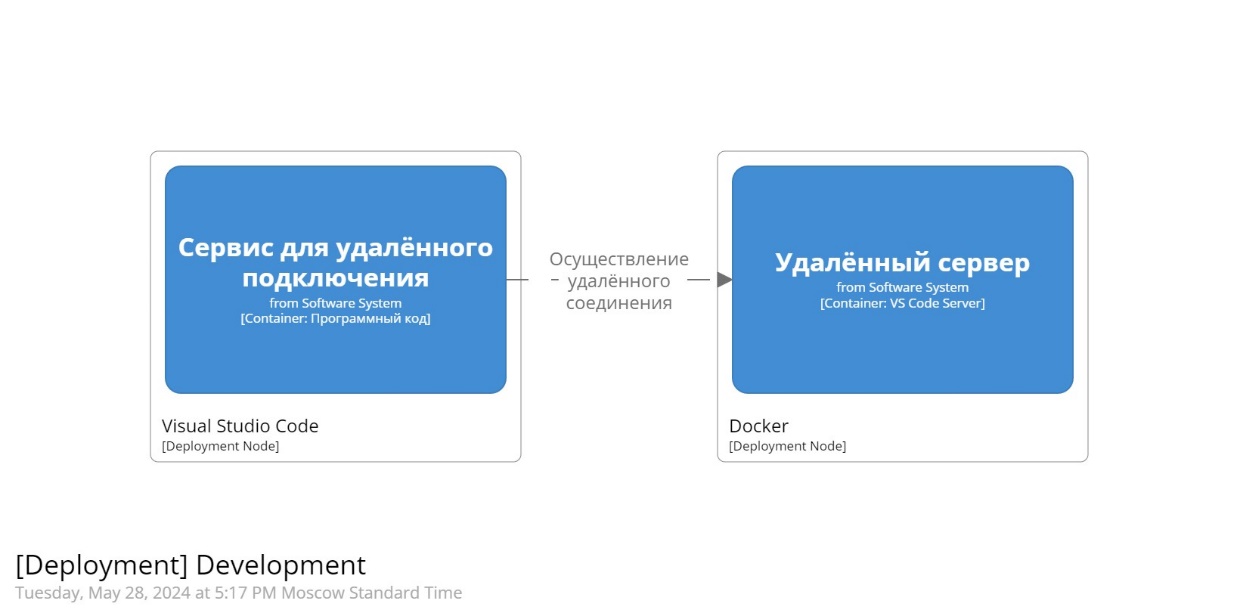


Рисунок 8 – Диаграмма развёртывания программного обеспечения

Также на рисунке 9 продемонстрированы основные элементы программного обеспечения, которые находятся на удалённом сервере.

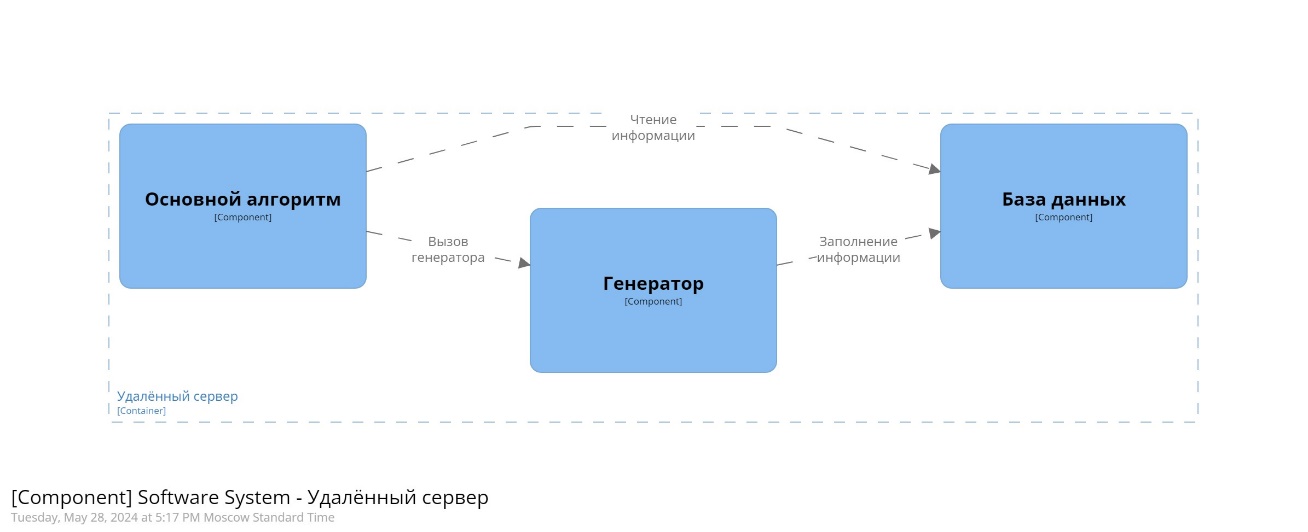


Рисунок 9 – Компоненты удалённого сервера

Перед запуском программного обеспечения необходимо произвести сборку всех его компонентов с помощью CMake. После этого нужно скомпилировать полученный в процессе сборки Makefile. Запуск программы осуществляется с помощью небольшого скрипта, в котором указаны параметры для подключения к базе данных. Весь описанный процесс можно увидеть на рисунке 10.

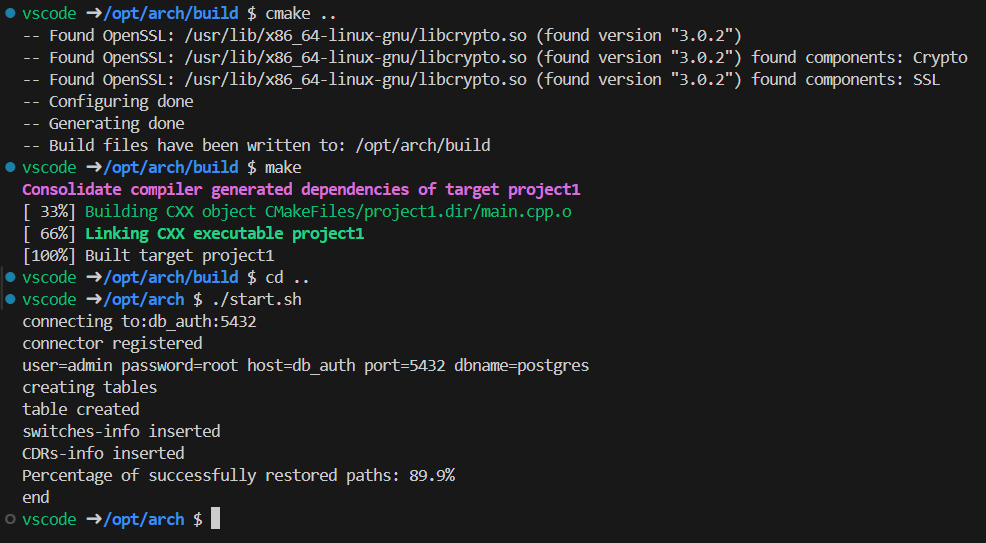


Рисунок 10 – Сборка, компиляция и запуск программного обеспечения

После запуска алгоритма в консоль выводится промежуточная информация: параметры базы данных, текущий этап работы алгоритма, процент успешно восстановленных путей звонков и сообщение об окончании работы алгоритма. Вся информация о результатах восстановления путей вызовов для удобства выводится в отдельный файл. Содержание данного файла проиллюстрировано на рисунке 11.

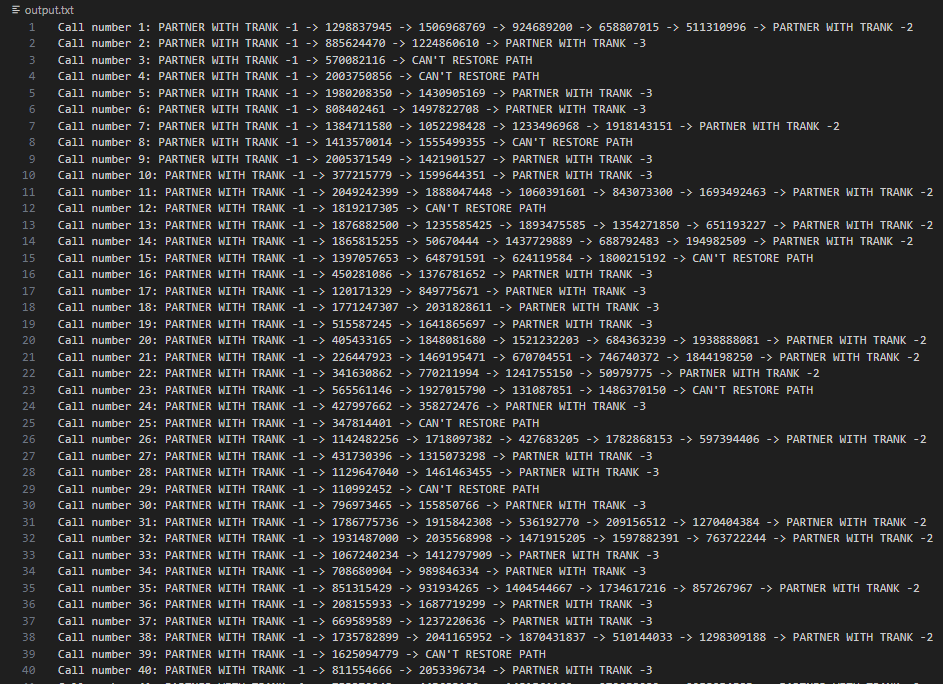


Рисунок 11 – Информация о восстановленных цепочках путей звонков

(первые 40 строк)

В сведениях о восстановленных цепочках для каждого звонка содержится следующие данные:

* номер транка абонента, от которого был получен звонок в нашу сеть; каждый внешний абонент имеет уникальный отрицательный номер транка, с помощью которого он обменивается информацией с нашей сетью, что позволяет однозначно их определять по данному параметру;
* цепочка из id CDR записей, относящихся к рассматриваемому звонку, в порядке их создания;
* номер транка абонента, которому был передан звонок для дальнейшей маршрутизации; в случае, если найти подходящий CDR для продолжения/завершения цепочки не удалось, выводится сообщение “CAN’T RESTORE PASS” [25].

Таблицы Switches и CDRs заполняются в процессе работы генератора всеми необходимыми данными для работы алгоритма.

Данные о структуре сети в генераторе заполняются заранее по уже готовой модели. Важным замечанием является то, что коммутатор с помощью одного транка может иметь ровно одно соединение с другим устройством сети (или внешним оператором). Пример содержания таблицы Switches приведён в таблице 1.

Таблица 1 – Пример содержания таблицы Switches

| **№** | **Id (primary key)** | **Id\_switch** | **Id\_out\_trunk** | **Id\_remote\_trunk\_in** | **Id\_remote\_switch** | **Time\_zone** | **Distanse** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 7 | -1 | 3 | 1 | 0 | 1 |
| 2 | 2 | 1 | 1 | 2 | 3 | 1 | 15 |
| 3 | 3 | 1 | 2 | 1 | 2 | 1 | 7 |
| 4 | 4 | 2 | 2 | 1 | 4 | 1 | 9 |
| 5 | 5 | 3 | 4 | -3 | 8 | 2 | 1 |

Данные о CDR записях заполняются с помощью алгоритма генерации. Значение времени создания в каждой записи указано в миллисекундах; любое из указанных значений не меньше интервала, равного 54 годам (согласно формату Unix Timestamp). Данные о длительности звонка не превышают интервала в 30 минут. Таблица 2 отражает пример содержания таблицы CDRs.

Таблица 2 – Пример содержания таблицы CDRs

| **№** | **Id (primary key)** | **Cur\_time** | **Duration\_time** | **Number\_from** | **Number\_to** | **Id\_switch** | **Id\_in\_trunk** | **Id\_out\_trunk** | **Is\_start** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 7912 | 1719992721308 | 125512 | 8844312606 | 82036581482 | 1 | 3 | 1 | true |
| 2 | 643 | 1719992724910 | 125500 | 8844312606 | 82036581482 | 3 | 2 | 4 | false |
| 3 | 8091 | 1716283869406 | 49912 | 86536745238 | 84754046835 | 1 | 3 | 2 | true |
| 4 | 3 | 1716283869409 | 49883 | 86536745238 | 84754046835 | 2 | 1 | 2 | false |
| 5 | 59120 | 1716293866609 | 49883 | 86536745238 | 84754046835 | 4 | 1 | 3 | false |

Если требуется проанализировать конкретные данные, необходимо убрать вызов генератора из основного файла и заполнить таблицы заранее подготовленными сведениями согласно приведённому формату.

## **3.3 Тесты производительности**

Тестирование проводилось на электронной вычислительной машине со следующими техническими характеристиками и программным обеспечением:

* центральный процессор «Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz»;
* оперативная память: DDR4 8 ГБ, 2400 МГц»;
* твердотельный накопитель: «Samsung SSD 980 1 TB»;
* операционная система «Windows 10 Pro»;
* Microsoft Visual Studio Code, version: 1.89.1;
* Docker version 25.0.3, build 4debf41;
* Docker Compose version v2.24.6-desktop.1;
* PostgreSQL 15.0.

Для тестирования генерировались данные размером соответственно 10, 100, 1000 и 5000 звонков.

В первую очередь производилось исследование степени зависимости восстановления цепочек путей звонков от вероятности потери CDR записи на коммутаторе. Результаты данного исследования приведены на рисунке 12.

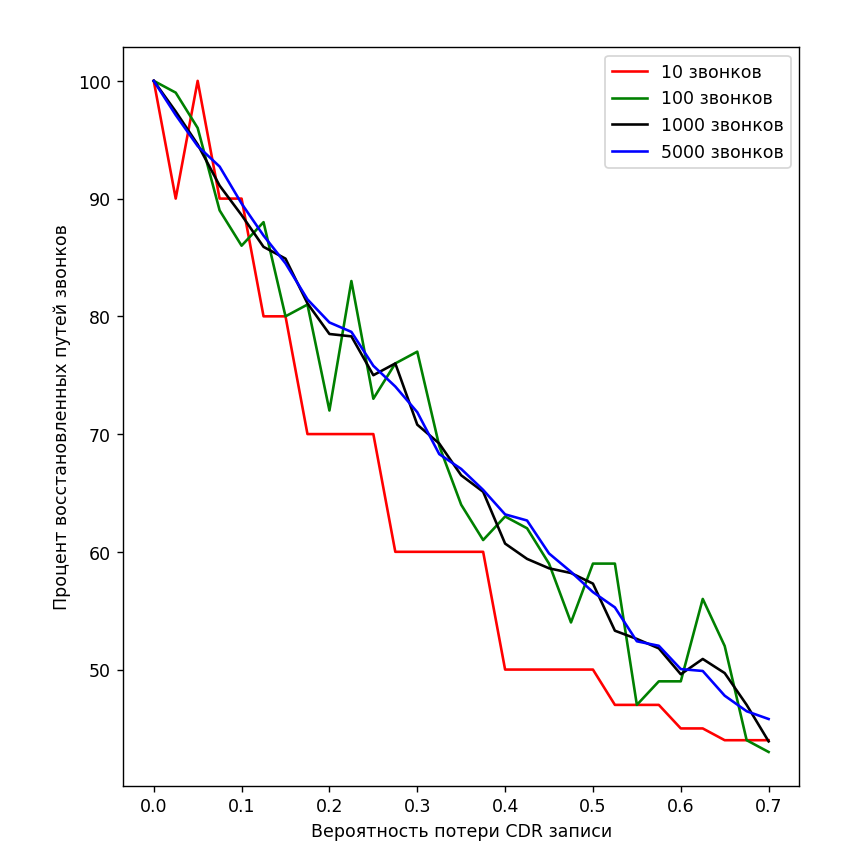


Рисунок 12 – Зависимость процента восстановленных путей звонков от вероятности потери CDR записи

Судя по полученному графику, данная зависимость является практически линейной. Исключение составляют только данные размером 10 и 100 звонков, так как в этих случаях вносимый вклад от результата обработки одного звонка сильно влияет на общий итог.

Затем исследовалась степень зависимости восстановления цепочек звонков от порогового значения метрики учёта данных о длительности звонка и времени создания CDR записи, которая используется при поиске CDR для продолжения цепочки пути. График полученной зависимости можно увидеть на рисунке 13.

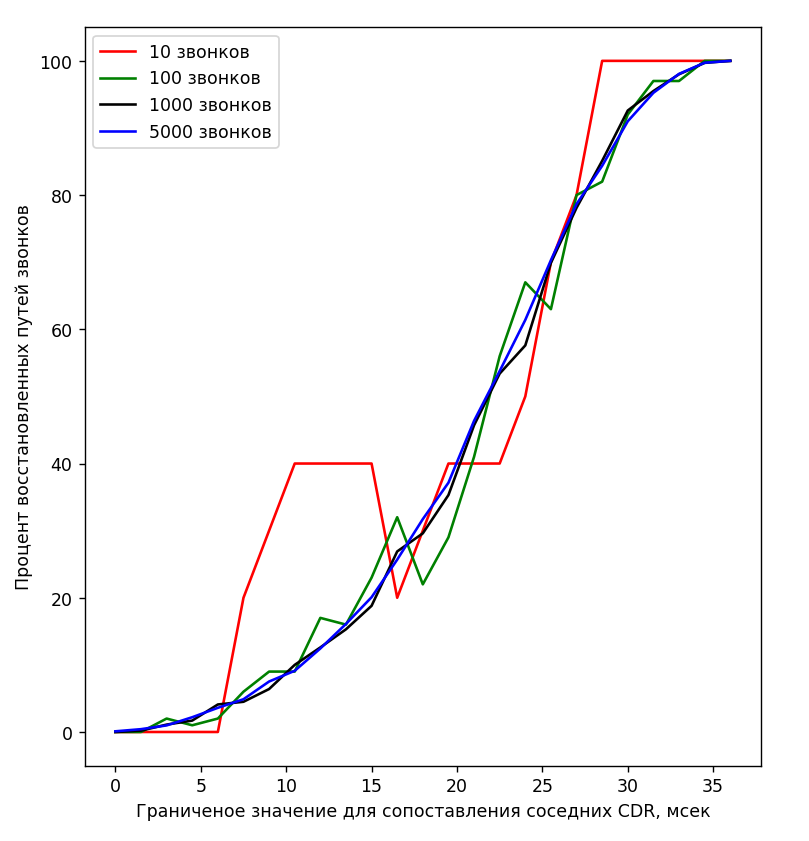
**

Рисунок 13 – Зависимость процента восстановленных путей звонков от

порогового значения метрики для сопоставления соседних CDR

По результатам нетрудно заметить, что самая сильная зависимость между исследуемыми параметрами оказывается в промежутке между 15 и 30 миллисекундами значения метрики. Здесь также, как и на прошлом графике, наблюдаются резкие скачки на данных размером 10 и 100 звонков.

Оба вышеописанных исследования производились независимо друг от друга, фактор влияния граничного значения метрики для сопоставления соседних CDR на итоговый результат в первом замере полностью отсутствует. Аналогично и для второго замера: вероятность потери CDR на коммутаторе во время его проведения равнялась нулю.

Заключительное тестирование программы проводилось с целью выявления зависимости времени работы программного обеспечения [26] от количества звонков в данных, которые подаются на вход алгоритму. Данная зависимость отражена на рисунке 14.

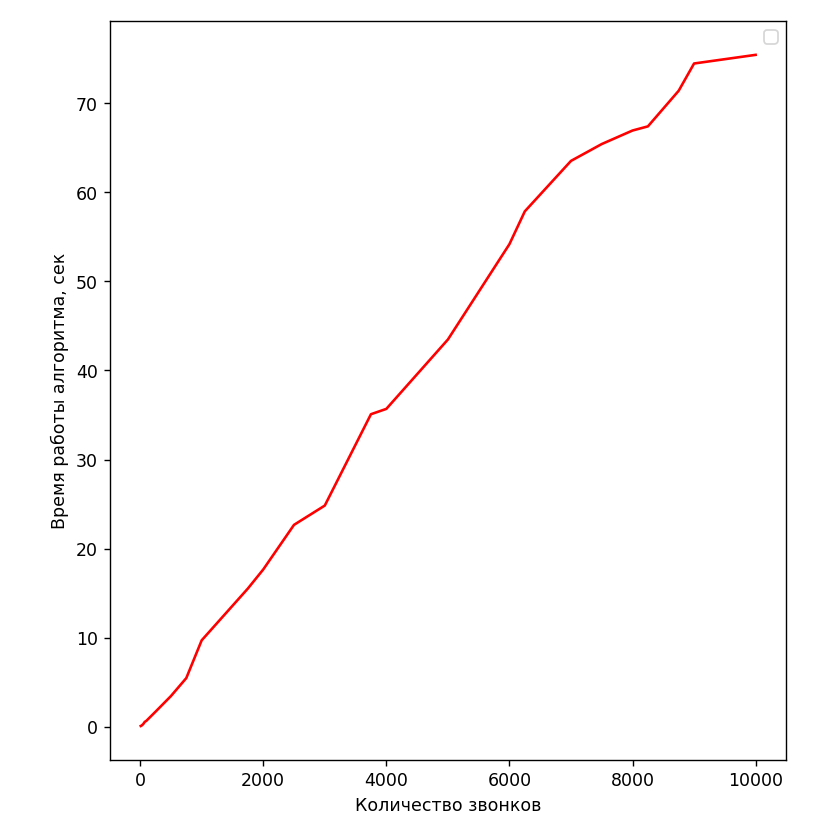


Рисунок 14 – Зависимость времени работы алгоритма от

количества звонков

Таким образом, длительность работы программы довольно сильно возрастает при увеличении объёмов данных. Связано это с тем, что с ростом количества записей в базе данных поиск по ним существенно замедляется [27], а это напрямую сказывается на производительности алгоритма. Для улучшения поиска информации в базе данных можно попробовать отсортировать данные, что позволит искать требуемые записи за логарифмическую сложность.

# **ЗАКЛЮЧЕНИЕ**

Основным результатом, полученными в работе, является программное обеспечение, которое находит информацию о полном пути звонков, переданных внешним оператором в локальную сеть. Были решены следующие задачи:

* настроен генератор данных, с помощью которого тестируется основной алгоритм;
* разработан и реализован оптимальный алгоритм восстановления пути звонков внутри коммутаторной сети;
* произведены тесты производительности алгоритма на разных данных, варьирующихся размером и степенью диффузии, которая в них вносится.

Сложность работы полученного алгоритма по времени складывается из следующих составляющих:

* алгоритм Флойда-Уоршелла и сохранение оптимальных путей в сети: ;
* генерация данных: ;
* восстановление путей звонков: ,

где k - количество звонков, l - количество соединений в сети, m – количество CDR записей, n – количество устройств в сети.

Поиск информации в базе данных занимает большую часть времени работы алгоритма. Для оптимизации времени поиска можно сортировать содержимое таблиц после внесения сведений, что обеспечит логарифмическую сложность для нахождения нужной записи [28].

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Гольдштейн Б.С. Сети связи. - БХВ-Петербург, 2010. - 400 с. (дата обращения 22.03.2024).
2. Гольдштейн Б.С., Пинчук А.В., Суховицкий А.Л. IP-Телефония. - Москва: Издательство «Радио и связь», 2001. - 336 с. (дата обращения 22.03.2024).
3. Гольдштейн Б.С., Гольдштейн А.Б. Softswitch. - БХВ-Санкт-Петербург, 2006. - 368 c. (дата обращения 22.03.2024).
4. Telecommunication Regular Handbook. Module 3 - Interconnection. – URL: [https://www.itu.int/ITU-D/treg/Documentation/Infodev\_handbook/3\_Interc  
   onnection.pdf](https://www.itu.int/ITU-D/treg/Documentation/Infodev_handbook/3_Interconnection.pdf) (дата обращения 23.03.2024).
5. Гольдштейн Б.С., Ехриель И.М., Рерле Р.Д. Интеллектуальные сети. - Москва: Издательство «Радио и связь», 2000. - 502 с. (дата обращения 24.03.2024).
6. Interconnect agreement. – URL: <https://en.wikipedia.org/wiki/Interconnect_agreement> (дата обращения 24.03.2024).
7. Telecom Interconnect Billing. – URL: <https://www.tutorialspoint.com/telecom-billing/interconnect-billing.htm> (дата обращения 25.03.2024).
8. Call Detail Record. – URL: <https://ru.wikipedia.org/wiki/Call_Detail_Record> (дата обращения 25.03.2024).
9. Unix time – URL: <https://en.wikipedia.org/wiki/Unix_time> (дата обращения 01.04.2024).
10. Задача о кратчайшем пути. – URL: [https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0\_%D0%BE\_%D0%BA%D1%80%D0%B0%D1%82%D1%87%D0%B0%D0%B9%D1%88%D0%B5%D0%BC\_%D0%BF%D1%83%D1%82%D0%B8#%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0\_%D0%BE\_%D0%BA%D1%80%D0%B0%D1%82%D1%87%D0%B0%D0%B9%D1%88%D0%B5%D0%BC\_%D0%BF%D1%83%D1%82%D0%B8\_%D0%BC%D0%B5%D0%B6%D0%B4%D1%83\_%D0%B2%D1%81%D0%B5%D0%BC%D0%B8\_%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B8\_%D0%B2%D0%B5%D1%80%D1%88%D0%B8%D0%BD](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D0%BA%D1%80%D0%B0%D1%82%D1%87%D0%B0%D0%B9%D1%88%D0%B5%D0%BC_%D0%BF%D1%83%D1%82%D0%B8#%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D0%BA%D1%80%D0%B0%D1%82%D1%87%D0%B0%D0%B9%D1%88%D0%B5%D0%BC_%D0%BF%D1%83%D1%82%D0%B8_%D0%BC%D0%B5%D0%B6%D0%B4%D1%83_%D0%B2%D1%81%D0%B5%D0%BC%D0%B8_%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B8_%D0%B2%D0%B5%D1%80%D1%88%D) (дата обращения 17.04.2024).
11. Алгоритм Флойда-Уоршелла нахождения кратчайших путей между всеми парами вершин. – URL: <http://www.e-maxx-ru.1gb.ru/algo/floyd_warshall_algorithm> (дата обращения 17.04.2024).
12. CMake. – URL: <https://ru.wikipedia.org/wiki/CMake> (дата обращения 18.04.2024).
13. Makefile. – URL: <https://ru.wikipedia.org/wiki/Makefile> (дата обращения 18.04.2024).
14. POCO C++ Libraries. – URL: <https://docs.pocoproject.org/current/index.html> (дата обращения 20.04.2024).
15. Документация PostgreSQL и Postgres Pro – URL: <https://postgrespro.ru/docs> (дата обращения 20.04.2024).
16. Как работает Docker. – URL: <https://skillbox.ru/media/code/kak-rabotaet-docker-podrobnyy-gayd-ot-tekhlida/> (дата обращения 20.04.2024).
17. Use containerized databases. – URL: <https://docs.docker.com/guides/use-case/databases/> (дата обращения 20.04.2024).
18. Docker Compose Quickstart. – URL: <https://docs.docker.com/compose/gettingstarted/> (дата обращения 21.04.2024).
19. Microsoft Visio. – URL: <https://support.microsoft.com/ru-ru/topic/%D1%83%D1%87%D0%B5%D0%B1%D0%BD%D0%B8%D0%BA-%D0%B4%D0%BB%D1%8F-%D0%BD%D0%B0%D1%87%D0%B8%D0%BD%D0%B0%D1%8E%D1%89%D0%B8%D1%85-%D0%B4%D0%BB%D1%8F-visio-bc1605de-d9f3-4c3a-970c-19876386047c> (дата обращения 22.04.2024).
20. Structurizr. – URL: <https://docs.structurizr.com/> (дата обращения 22.04.2024).
21. The C4 model for visualising software architecture. – URL: <https://c4model.com/> (дата обращения 25.04.2024).
22. Matplotlib: Quick start guide. – URL: <https://matplotlib.org/stable/users/explain/quick_start.html#a-simple-example> (дата обращения 30.04.2024).
23. Visual Studio Code Documentation. – URL: <https://code.visualstudio.com/docs> (дата обращения 20.04.2024).
24. Cppreferece. Library srand(). – URL: <https://en.cppreference.com/w/c/numeric/random/srand> (дата обращения 30.04.2024).
25. Код ответа. – URL: <https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D0%B0> (дата обращения 02.05.2024).
26. Cppreferece. Date and time utilities. – URL: <https://en.cppreference.com/w/cpp/chrono> (дата обращения 02.05.2024).
27. Оценка производительности полнотекстового поиска в PostgreSQL на примерах, приближенных к реальным. – URL: <https://habr.com/ru/companies/bimeister/articles/756704/> (дата обращения 05.05.2024).
28. Целочисленный двоичный поиск. – URL: <https://neerc.ifmo.ru/wiki/index.php?title=%D0%A6%D0%B5%D0%BB%D0%BE%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B4%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA> (дата обращения 05.05.2024).

# **ПРИЛОЖЕНИЕ А Исходный код**

Исходный код разработанного программного обеспечения находится в открытом доступе на Github репозитории, доступ к которому можно получить с помощью следующего QR-кода:

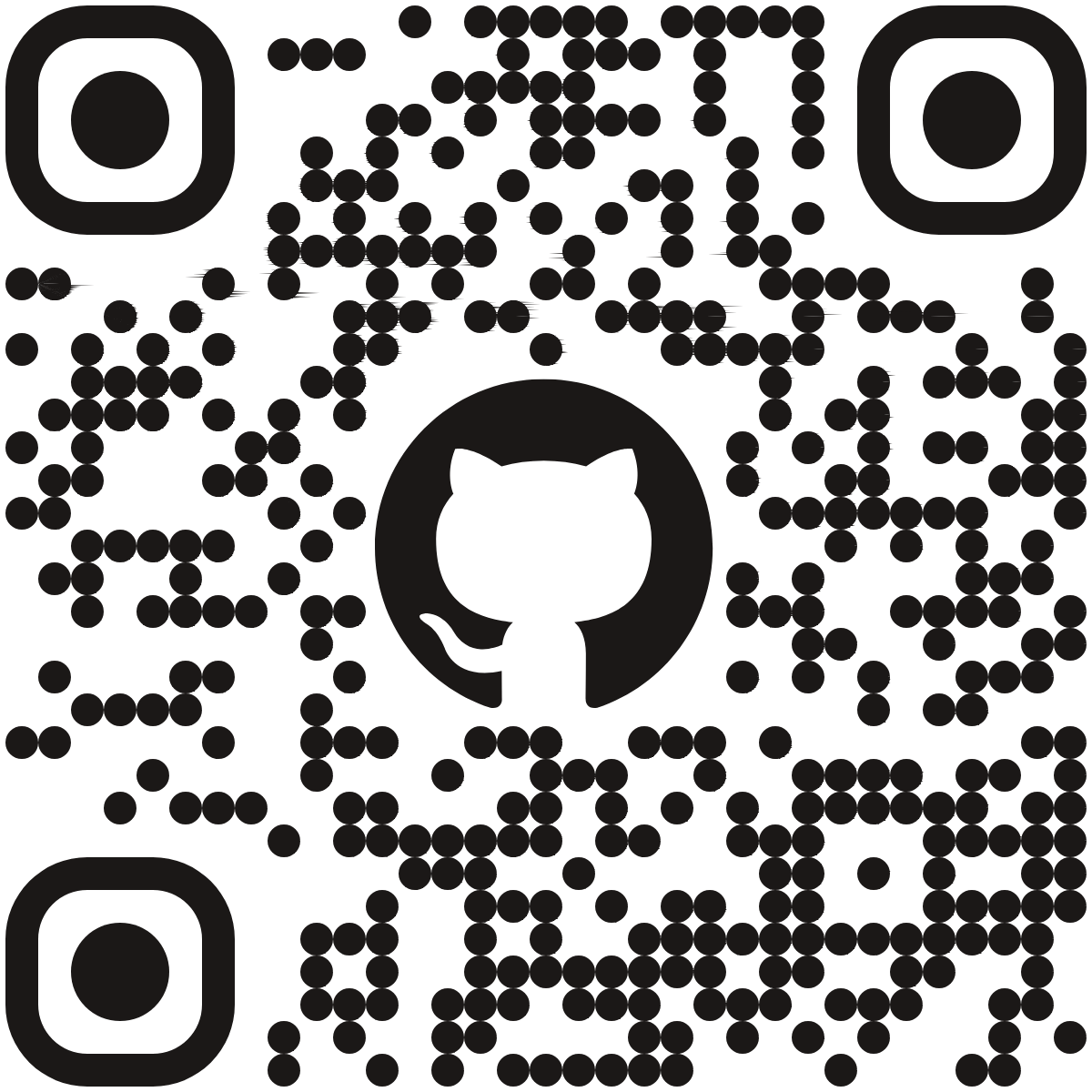


Рисунок А.1 – QR-код на репозиторий