МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра многопроцессорных систем и сетей

ЕВСЕЕВ
Матвей Олегович

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ АВТОМАТИЗАЦИИ ПОСТРОЕНИЯ КОМПЬЮТЕРНЫХ СЕТЕЙ

Дипломная работа

Научный руководитель:
кандидат физ.-мат. наук,
доцент С.В. Марков

Допущена к защите
«___»_____2025 г.
Зав. кафедрой многопроцессорных систем и сетей
кандидат физ.-мат. наук, доцент И.Е.Андрушкевич

Минск, 2025

РЕФЕРАТ

Дипломная работа, 39 страниц, 13 рисунков, 2 таблицы, 16 источников, 1 приложение.

Ключевые слова: РАЗРАБОТКА НА PYTHON, КОМПЬЮТЕРНЫЕ СЕТИ, ТОПОЛОГИЯ, АВТОМАТИЗАЦИЯ, ГРАФ.

Объект исследования — возможность автоматизации построения компьютерных сетей.

Цели работы — изучить возможные способы автоматизации построения топологий компьютерных сетей и реализовать соответствующий проект.

Методы исследования — а) теоретические: изучение литературы, посвящённой архитектуре компьютерных сетей и их созданию; б) практические: разработка и анализ соответствующего приложения.

Результатами являются — полноценно функционирующая программа для автоматизации.

Область применения — использование в качестве инструмента, упрощающего рабочий процесс компаний и разработчиков.

РЭФЕРАТ

Дыпломная праца, 39 старонак, 13 малюнкаў, 2 табліцы, 16 крыніц, 1 дадатак.

Ключавыя словы: РАСПРАЦОЎКА НА PYTHON, КАМП'ЮТАРНЫЯ СЕТКІ, ТАПАЛОГІЯ, АЎТАМАТЫЗАЦЫЯ, ГРАФ.

Аб'ект даследавання - магчымасць аўтаматызацыі пабудовы кампутарных сетак.

Мэты працы — вывучыць магчымыя спосабы аўтаматызацыі пабудовы тапалогій кампутарных сетак і рэалізаваць адпаведны праект.

Метады даследавання — а) тэарэтычныя: вывучэнне літаратуры, прысвечанай архітэктуры кампутарных сетак і іх стварэнню; б) практычныя: распрацоўка і аналіз адпаведнага дадатку.

Вынікамі з'яўляюцца — паўнавартасна якая функцыянуе праграма для аўтаматызацыі.

Вобласць ужывання — выкарыстанне ў якасці прылады, які спрашчае працоўны працэс кампаній і распрацоўнікаў.

ABSTRACT

Diploma paper, 39 pages, 13 figures, 2 tables, 16 sources, 1 appendix.

Keywords: PYTHON DEVELOPMENT, COMPUTER NETWORKS, TOPOLOGY, AUTOMATION, GRAPH.

Research object — the possibility of automating the construction of computer networks.

The objectives of the study — to study possible ways to automate the construction of computer network topologies and implement the corresponding project.

Research methods — a) theoretical: studying the literature devoted to the architecture of computer networks and their creation; b) practical: development and analysis of the corresponding application.

The results are — a fully functioning program for automation.

Scope — use as a tool that simplifies the workflow of companies and developers.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современном мире компьютерные сети играют важнейшую роль в обеспечении работы практически всех отраслей экономики, науки, образования и повседневной жизни. Сложность и масштаб сетевой инфраструктуры постоянно растут, что делает задачи проектирования и развертывания таких сетей все более сложными и трудоемкими. От надежности и эффективности компьютерных сетей напрямую зависит производительность предприятий, безопасность данных и удобство работы пользователей.

Процесс проектирования компьютерных сетей часто сопровождается необходимостью выбора оборудования, исследования и разработки логических топологий, а также учёта множества ограничений, что требует значительных ресурсов и времени. Одновременно с этим автоматизация процессов проектирования и моделирования сетей может позволить значительно сократить временные затраты, повысить точность проектных решений и упростить процесс для специалистов.

Исследование данной темы обусловлен актуальностью задачи автоматизации создания компьютерных сетей и возможностью использования в инструментах для сетевых инженеров и администраторов. Целью данной работы является создание программного обеспечения, которое сможет автоматизировать значительную часть рутинных операций, что поможет повысить общую эффективность процесса проектирования. Уникальность подхода заключается в стремлении объединить простоту использования, гибкость настройки и возможности моделирования и тестирования.

ГЛАВА 1 ОСНОВЫ И ПОДХОДЫ К ПОСТРОЕНИЮ КОМПЬЮТЕРНЫХ СЕТЕЙ

1.1 Основные элементы компьютерных сетей

Компьютерная сеть представляет собой совокупность устройств, соединенных между собой для обмена данными. Основные элементы сети включают:

- устройства (узлы),
- программное обеспечение,
- среда передачи данных,
- сетевые протоколы.

Устройства представляют собой важнейший элемент компьютерной сети и подразделяются на:

- клиенты – компьютеры, смартфоны и другие устройства, которые используют ресурсы сети,
- серверы – устройства, предоставляющие ресурсы и услуги (например, файловые серверы, веб-серверы),
- маршрутизаторы (роутеры) – определяют оптимальные пути передачи данных между сетями,
- коммутаторы (свитчи) – соединяют устройства в пределах одной локальной сети и обеспечивают эффективную передачу данных на канальном уровне модели OSI,
- точки доступа (Access Points) – расширяют возможности беспроводного доступа к сети, обеспечивая подключение устройств через Wi-Fi,
- модемы – обеспечивают доступ к интернету через телефонную линию, оптоволокно или мобильные сети,
- файерволы (межсетевые экраны) – осуществляют фильтрацию сетевого трафика, предотвращая несанкционированный доступ и обеспечивая безопасность сети,
- NAS (Network Attached Storage) – специализированные устройства хранения данных, предоставляющие централизованный доступ к файлам для всех устройств сети.

Среда передачи данных может быть проводной и беспроводной:

1. Ethernet (витая пара) являются наиболее распространенным типом проводного соединения, обеспечивающим передачу данных до 100 Гбит/с с использованием стандарта 802.3.

2. Оптоволоконные кабели применяются для передачи данных на большие расстояния с минимальными потерями и чрезвычайно высокой пропускной способностью (до нескольких терабит в секунду).

3. Wi-Fi – беспроводной стандарт семейства IEEE 802.11, позволяющий устройствам подключаться к сети на расстоянии. Современные стандарты Wi-Fi обеспечивают высокие скорости передачи данных и большую пропускную способность.

Из наиболее интересных используемых протоколов маршрутизации и коммутации можно выделить следующие:

- OSPF (Open Shortest Path First),
- BGP (Border Gateway Protocol),
- Static,
- EIGRP,
- RIP (Routing Information Protocol).

1.2 Проектирование компьютерных сетей

При проектировании сети требуется пройти через этапы анализа требований, разработки логической и физической топологии, выбора оборудования и протоколов, а также обеспечение безопасности и отказоустойчивости. При проектировании сетей необходимо учитывать минимальные входные параметры, такие как пропускная способность, задержка, совместимость оборудования и потребности в масштабируемости.

Компьютерные сети можно классифицировать по различным критериям, но основными являются подразделения по масштабу и топологии. Масштаб характеризует количество устройств и охват сети. По масштабу компьютерные сети подразделяются на:

– Локальные сети (LAN), которые охватывают небольшую территорию, например, офис или дом. Их основным преимуществом является высокая скорость передачи данных и низкая задержка.

– Региональные сети (MAN), которые соединяют локальные сети в пределах города или региона.

– Глобальные сети (WAN), которые охватывают большие территории, например, страны или континенты. Примером глобальной сети является Интернет.

Топология компьютерной сети – это способ организации соединений между устройствами в сети. Топология определяет, как устройства связаны друг с другом, как передаются данные и как сеть реагирует на возможные сбои. Важно выбрать правильную топологию сети, потому что она может

влиять на производительность, масштабируемость, надежность и стоимость сети. Основные виды топологий:

1. Шина – простая структура, в которой все устройства подключены к одной линии связи.
2. Звезда – центральный узел подключается ко всем остальным узлам. Отличается высокой надежностью, так как при выходе из строя одного из узлов, данные все равно могут пройти по другому маршруту.
3. Кольцо – узлы соединены в замкнутый контур. Обеспечивает равномерную нагрузку, но является менее отказоустойчивым.
4. Сеть с ячеистой структурой – каждый узел может быть связан с несколькими другими. Обеспечивает высокую производительность и отказоустойчивость.
5. Древовидная – обладает иерархическим устройством, а именно: узлы соединены в виде дерева, где один или несколько центральных узлов играют роль корня, а остальные узлы – ветвей и листьев.
6. Смешанная топология является комбинацией нескольких других топологий.

Сравнение различных топологий по их характеристикам, учитываемых при проектировании, представлено далее на рисунке 1.1.

Топология	**Надежность**	**Стоимость реализации**	**Простота управления**	**Масштабируемость**
Шинная	Низкая	Низкая	Средняя	Ограниченная
Звёздная	Средняя	Средняя	Высокая	Высокая
Кольцевая	Низкая	Низкая	Сложная	Ограниченная
Ячеистая	Очень высокая	Очень высокая	Сложная	Средняя
Древовидная	Средняя	Средняя	Средняя	Высокая
Смешанная	Высокая	Высокая	Сложная	Очень высокая

Таблица 1.1 – Сравнение различных топологий

1.3 Проблемы, требующие решения

Для успешного выполнения поставленной задачи необходимо решить несколько ключевых задач:

1. Реализовать алгоритмы генерации топологий с учетом пользовательских параметров.

2. Обеспечить интеграции с инструментами моделирования.
3. Создать простой и интуитивно понятный интерфейс для пользователя.
4. Добавить возможность изменения частей сгенерированной сети.
5. Разработать методов тестирования надежности и безопасности предложенных решений.
6. Минимизировать время выполнения расчетов, генерации топологии и её тестирования.
7. Сделать сохранение и использование сгенерированной сети удобным.

1.4 Сравнение с аналогами

Аналогичные реализованные решения разделяются на инструменты для эмуляции и симуляции сетей и на инструменты для создания схем и визуализации топологий.

Инструменты для эмуляции и симуляции сетей подразумевают под собой такие решения, как GNS3, Cisco Packet Tracer, EVE-NG, Boson NetSim и Cisco VIRL. Их основное назначение – создание виртуальных сред для тестирования сетевых конфигураций и обучения. Далее приведены некоторые из таких решений:

1) **GNS3**: один из самых востребованных инструментов для эмуляции построения сетевых устройств. Он поддерживает работу с реальными образами операционных систем (например, Cisco IOS), что делает его незаменимым инструментом моделирования для профессионалов и студентов. Недостатки: требует ручной настройки образов, обладает высокими системными требованиями [1].

2) **Cisco Packet Tracer**: инструмент от Cisco, широко используемый в обучении благодаря простоте и поддержке всех ключевых технологий Cisco. Недостатки: ограниченность экосистемой Cisco, не подходит для тестирования мультивендорных решений. Пример интерфейса показан на рисунке далее.

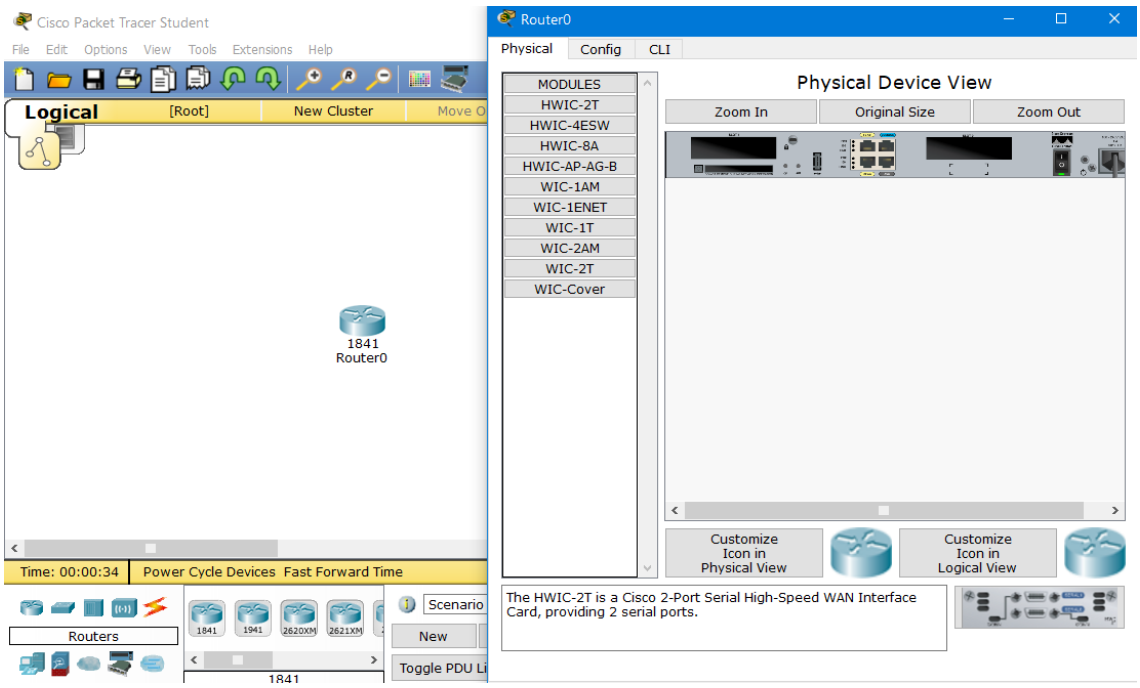


Рисунок 1.1 – Пример интерфейса Cisco Packet Tracer

3) **EVE-NG** (Emulated Virtual Environment): универсальная платформа для тестирования мультивендорных сетей (Cisco, Juniper, Fortinet и др.). Применяется профессионалами для работы с реальными конфигурациями. Недостатки: сложность в настройке по сравнению с GNS3, платная версия для полного функционала [2].

Инструменты для создания схем и визуализации включают Miro, EdrawMax, Microsoft Visio, Lucidchart и SolarWinds Network Topology Mapper. Эти решения ориентированы на документирование и презентацию сетей, а также совместную работу над проектами.

1) **Microsoft Visio**: индустриальный стандарт как для создания сложных диаграмм компьютерных сетей, так и диаграмм для других целей. Недостатки: высокая стоимость лицензии, сложность освоения для новичков [3].

2) **SolarWinds Network Topology Mapper**: уникальное решение, которое автоматически сканирует сеть и строит топологию. Недостатки: дороговизна, предназначенность для крупных корпоративных сетей [4].

3) **Lucidchart**: онлайн-инструмент для совместной работы с широкими возможностями визуализации различных диаграмм. Недостатки: ограниченный функционал в бесплатной версии [5].

4) **EdrawMax**: подходит для профессиональной документации, имеет поддержку широкого спектра диаграмм. Недостатки: сложный интерфейс, требующий времени на освоение [6]. Пример интерфейса показан на рисунке далее.

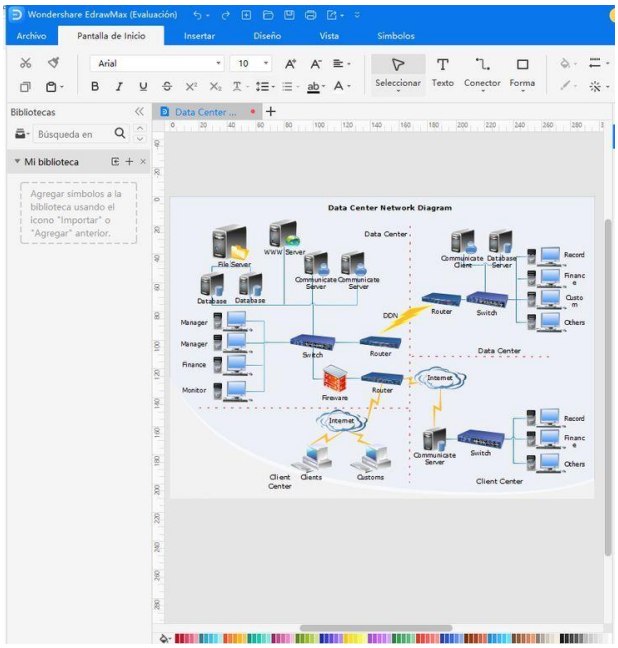


Рисунок 1.2 – Пример интерфейса EdrawMax

5) **Miro**: удобная онлайн-доска для совместной работы и для создания базовых схем, включающих компьютерные сети. Недостатки: ограниченные возможности для сложных технических диаграмм [5].

Каждая из групп инструментов имеет свои сильные и слабые стороны, но в большинстве случаев они фокусируются либо на симуляции работы сети, либо на создании диаграмм, а задачи полной автоматизации проектирования сети остаются слабо охваченными. В отличие от аналогов, разрабатываемая программа будет ориентирована на облегчение процесса проектирования, включая автоматическое предложение оптимальных топологий, учет требований безопасности и возможность предварительного тестирования решений.

ГЛАВА 2 ПЛАНИРОВАНИЕ РЕАЛИЗАЦИИ АВТОМАТИЗИРОВАННОГО ПОСТРОЕНИЯ КОМПЬЮТЕРНЫХ СЕТЕЙ

2.1 Определение необходимых для построения сети данных

При запуске программы пользователь будет задавать определённое количество параметров, достаточных для автоматизированного построения компьютерной сети. При запуске пользователь сможет выбрать два режима: «для компаний» и «для разработчика». В режиме «для компаний» будет необходимо ввести такие параметры как:

- Количество филиалов – позволяет понять, сколько подсетей будет в полной сети.
- Количество пользователей на филиал – позволяет программе в последующем подобрать типы соединений и требуемые устройства.
- Требуемый ценовой сегмент – позволяет пользователю уложиться в требуемый бюджет.
- Требуемый уровень безопасности – предназначен для того, чтобы гарантировать защищенности сети.
- IP-адрес сети – является важным задаваемым параметром, потому что некоторые компании обладают зарезервированным пулом ip-адресов.

В режиме разработчика должна быть возможность вручную построить или изменить конкретную топологию, удовлетворяющую требованиям пользователя.

2.2 Исследование подходов к реализации

Для автоматизации построения сетей необходимо определить архитектуру приложения, обязательные модули и технологии, которые будут использоваться при реализации проекта. Основу приложения могут составить следующие элементы:

1. Модуль проектирования топологии сети для выбора оптимальной топологии сети необходимо описать оптимальные алгоритмы соединения устройств. Эти алгоритмы должны учитывать такие параметры как пропускная способность соединений, количество устройств, отказоустойчивость сети и ценовой сегмент.
2. Способ получения информации об устройствах, составляющих компьютерную сеть.

3. Инструмент тестирования для поиска уязвимостей и создания отчёта о характеристиках после автоматического построения сети.
4. Графический интерфейс для удобства использования пользователем.

2.3 Возможности масштабирования, обновляемости и доработки

При автоматическом проектировании компьютерной сети важно предусмотреть возможности её масштабирования, изменения и последующего её использования. Это позволит повысить гибкость программы и адаптировать её под различные сценарии использования. Планируются следующие функции:

– Поддержка масштабирования: программа должна эффективно работать как с небольшими локальными сетями, так и с более крупными сетями компаний, увеличивая количество устройств, подсетей и сложность топологии без значительного снижения производительности.

– Обновляемость, что означает возможность добавлять новые алгоритмы или конфигурации для работы с новыми типами устройств, конфигурациями и протоколами.

– Сохранение и переиспользование: характеристики программы, позволяющие сохранять построенные конфигурации для повторного использования или последующей работы над ними.

ГЛАВА 3 ВЫБОР ИНСТРУМЕНТОВ ДЛЯ РЕАЛИЗАЦИИ ПРИЛОЖЕНИЯ

3.1 Выбор языка программирования

Для реализации данного проекта был выбран язык программирования Python. Его выбор обусловлен следующими причинами:

1. Простота изучения и использования: данный язык программирования обладает понятным синтаксисом, что делает его доступным.
2. Обширная стандартная библиотека: Python предоставляет широкий набор встроенных модулей, которые упрощают выполнение различных задач, таких как работа с файлами, сетью, визуализацией данных и т.д.
3. Масштабируемость и гибкость: Python подходит как для небольших утилит, так и для крупных приложений.
4. Поддержка сообщества: большое сообщество разработчиков обеспечивает доступ к многочисленным библиотекам и инструментам, а также позволяет быстро находить решения для возникающих проблем.

Python — идеальный язык для сетевых инженеров, предлагающий инструменты, которые ранее были доступны только системным инженерам и разработчикам приложений [14].

3.2 Исследование инструментов для разработки

При создании программного обеспечения для автоматизации проектирования сетевых топологий был проведен тщательный анализ современных инструментальных средств. Критериями выбора послужили следующие факторы:

1. Функциональная полнота - способность инструмента покрывать все этапы разработки.
2. Интеграционные возможности - способность взаимодействовать с другими инструментами.
3. Производительность - эффективность работы с вычислительно сложными задачами.

Всем этим критериям и наличию языка Python соответствует среда разработки PyCharm. Для реализации решения данной проблемы были найдены следующие инструменты разработки и библиотеки, хорошо вписывающиеся в тематику программы:

– *NetworkX*: предоставляет комплексные алгоритмы работы с графами, поддерживает различные метрики анализа сетей, позволяет импортировать и экспортировать графы в различные форматы.

– *Matplotlib*: обеспечивает профессиональную визуализацию сетевых топологий, поддерживает интерактивные элементы отображения, позволяет создавать графики хорошего качества.

– *math*: библиотека для математических вычислений, позволяющая специальным образом располагать узлы графа и считать метрики расстояний и пропускной способности.

– *random*: библиотека для добавления случайных аспектов.

– *json*: библиотека, позволяющая обеспечить сериализацию и десериализацию данных для сохранения и загрузки конфигураций сети.

– *ipaddress*: – библиотека позволяющая работать с ip адресами.

3.3 Обзор библиотеки NetworkX

Библиотека NetworkX была выбрана для работы с графами и сетевыми структурами. Основная задача приложения — автоматизация построения компьютерных сетей, что подразумевает необходимость создания, модификации и визуализации графов. Преимущества использования NetworkX:

- *Мощные инструменты для работы с графами*: NetworkX предоставляет функционал для создания, редактирования и анализа графов, включая расчет различных метрик и характеристик сетей.
- *Гибкость*: библиотека поддерживает как ориентированные, так и неориентированные графы, а также позволяет добавлять пользовательские атрибуты к узлам и рёбрам.
- *Интеграция с другими инструментами*: NetworkX легко интегрируется с такими библиотеками, как Matplotlib (для визуализации) и Pandas (для работы с данными).

NetworkX является лидером среди библиотек для работы с графами в Python, что делает её лучшим выбором для реализации задач, связанных с топологиями сетей.

3.4 Обзор библиотеки Matplotlib

Для визуализации топологий сетей была выбрана библиотека Matplotlib. Её основные преимущества включают:

- *Широкие возможности визуализации*: Matplotlib позволяет строить графики, диаграммы и визуализации любой сложности.

- *Удобство настройки*: библиотека предоставляет гибкие настройки для кастомизации графиков, включая изменение цвета, размера и стилей.

Использование Matplotlib позволяет наглядно представлять топологию сети, что важно для понимания её структуры и анализа.

3.5 Инструменты графического интерфейста

Для создания графического пользовательского интерфейса в проекте используется библиотека Tkinter. Этот инструмент был выбран по следующим причинам:

- *Встроенность в Python*: Tkinter включён в стандартную библиотеку Python, что упрощает его использование и настройку.
- *Простота разработки*: с помощью Tkinter можно быстро создать интерфейс, включающий кнопки, поля ввода, списки и другие элементы управления.
- *Кроссплатформенность*: приложения, созданные с использованием Tkinter, работают на Windows, macOS и Linux без необходимости дополнительных настроек.

Tkinter идеально подходит для небольших проектов, где требуется простой и функциональный интерфейс для взаимодействия с пользователем.

3.6 Использование библиотеки ipaddress

Основные возможности библиотеки *ipaddress*:

- Создание и работа с объектами IP-адресов (IPv4 и IPv6) и подсетей.
- Проверка корректности IP-адресов, сетевых масок и подсетей.
- Генерация последовательностей IP-адресов в заданной подсети.
- Подсчет количества адресов в подсети.
- Сравнение адресов между собой (например, проверка принадлежности IP-адреса к подсети).

В разработанном приложении библиотека *ipaddress* используется для автоматического назначения IP-адресов устройствам и проверки их корректности.

3.6.1 Инициализация пула ip-адресов

С помощью объекта IPv4Network программа создает пул доступных IP-адресов из диапазона 10.0.0.0/8. Первые 1000 адресов из этого пула используются для назначения устройствам.

```
ip_pool = list(ipaddress.IPv4Network("10.0.0.0/8").hosts())[:1000]
```

3.6.2 Автоматическое назначение ip-адресов

Каждый раз, когда создается новое устройство, программа автоматически берет первый свободный адрес из пула.

```
def assign_ip():
    global ip_pool
    if not ip_pool:
      messagebox.showerror("Ошибка", "Свободных IP-адресов больше нет!")
        return None
    return str(ip_pool.pop(0))
```

3.6.3 Проверка корректности ip-адресов

При редактировании устройства пользователь может вручную изменить IP-адрес. Для этого используется функция проверки валидности IP-адреса (через IPv4Address).

```
try:
    ipaddress.IPv4Address(new_ip)
except ipaddress.AddressValueError:
    messagebox.showerror("Ошибка", "Некорректный IP-адрес")
```

Таким образом, библиотека *ipaddress* автоматизирует работу с IP-адресами и предотвращает ошибки, связанные с их некорректным вводом.

3.7 Библиотека NetworkX

Библиотека *NetworkX* используется для построения и изменения сгенерированных топологий.

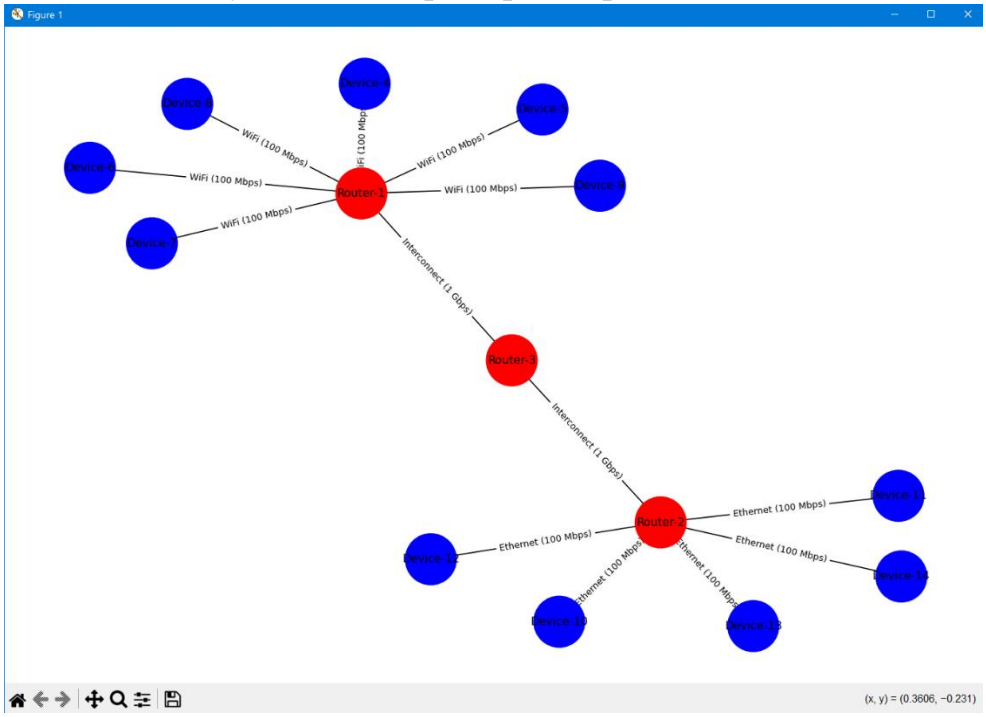
Для каждой топологии создается объект графа Graph, в который добавляются узлы (устройства) и рёбра (соединения между устройствами).

```
graph = nx.Graph()
graph.add_node(device_id, name=router_name, ip=router_ip, device_type="Router")
graph.add_edge(router, device_id, connection_type=connection_type, bandwidth="100 Mbps", latency="1 ms")
```

Узлы графа содержат информацию о названии устройства, его IP-адресе и типе (например, "Router" или "Computer"). Рёбра графа содержат параметры соединений, такие как тип соединения (WiFi, Ethernet и т.д.), пропускная способность и задержка.

На рисунке ниже представлен пример простой сети, построенной с помощью библиотеки *NetworkX*, что позволяет оценить возможный результат.

Рисунок 4.1 – Пример построенной сети



Топологии сохраняются в JSON-формате, где узлы и рёбра графа экспортируются вместе с их атрибутами.

Загрузка топологий из JSON позволяет восстановить графы.

```
for topology in config:
        graph = nx.Graph()
        graph.add_nodes_from([(node, data) for node, data in
topology["nodes"].items()])
        graph.add_edges_from([(u, v, data) for u, v, data in
topology["edges"]])
        network_graphs.append(graph)
```

ГЛАВА 4 СОЗДАНИЕ И ПОДКЛЮЧЕНИЕ БАЗЫ ДАННЫХ

4.1 Выбор СУБД

Использование базы данных для реализации данного проекта входит в список обязательных технологий. На этапе планирования проекта стало очевидно, что для использования различных устройств, более подходящих для сети, наиболее простым образом отлично подойдет использование базы данных с этими устройствами.

В качестве СУБД мною была выбрана реализация PostgreSQL. Этот выбор был сделан в связи с уже имеющимся опытом работы с данной СУБД, а также простотой поиска информации по имеющимся в ней технологиях, связанной с наличием огромного сообщества пользователей и ресурсов, легкостью настройки, а также тем, что данная СУБД является бесплатным вариантом.

Преимуществом данной СУБД является поддержка сложных типов данных, что позволяет хранить и эффективно работать с массивами, текстовыми описаниями и числовыми параметрами. Это особенно важно для учета специфики сетевого оборудования, где каждый элемент обладает уникальным набором характеристик. Возможность создания пользовательских типов данных и функций делает PostgreSQL идеальным выбором в качестве СУБД для данного проекта. Итого, PostgreSQL позволяет:

- Формировать сложные запросы для подбора оборудования на основе комбинаций параметров.
- Обеспечивать масштабируемость базы данных при добавлении новых категорий оборудования или параметров.

В работе был использован Psycopg — это наиболее популярный и надежный адаптер PostgreSQL для языка Python, позволяющий взаимодействовать с СУБД PostgreSQL напрямую из Python-кода. Он предоставляет удобный интерфейс для выполнения SQL-запросов, управления транзакциями и работы с результатами запросов.

4.2 Добавление и заполнение таблиц

Создание БД и заполнение её таблицами с данными было произведено с помощью SQL запросов. Запрос для создания БД:

```
CREATE DATABASE network_design_db WITH OWNER = postgres ENCODING
= 'UTF8';
CREATE TABLE device_types (типы и характеристики);
```

```
INSERT INTO device_types (типы и характеристики) VALUES
(‘устройства’);
CREATE TABLE network_configurations (характеристики
конфигурации);
```

Первый запрос отвечает за создание самой базы данных. В нём указывается название, имя владельца и кодировка. Следующий запрос отвечает за создание таблицы с типами устройств и их характеристиками. Это такие характеристики как: category – категория устройств (роутер, компьютер и тд.), vendor - производитель, model - модель, description – описание, price_range – ценовая категория, security_level – уровень защиты, throughput – пропускная способность, ports – количество портов, wifi_clients – возможное количество WiFi подключений, power_consumption – энергопотребление, interfaces – имеющиеся интерфейсы.

Запрос типа

```
INSERT INTO device_types (category, vendor, model, description,
price_range, security_level, throughput, ports,
wifi_clients, power_consumption, interfaces) VALUES
```

заполняет указанную таблицу данными. Последний из перечисленных запросов создает таблицу для записи туда конфигураций сетей. На рисунке 5.1 можно увидеть заполненную устройствами таблицу БД.

id [PK] inte	category character	vendor characte	model charact	descript text	price_range character v	security_level character var	throughput character va	ports integer	wifi_clients integer	power_consumption character varying (20	interfaces character
1	Router	Cisco	ISR ...	Мар...	high	high	1 Gbps	4	0	60W	{"Giga...
2	Router	Mikro...	RB4...	Мо...	medium	medium	1 Gbps	10	0	24W	{"Giga...
3	Router	TP-Li...	ER6...	Бюд...	low	medium	500 Mb...	5	0	12W	{"Giga...
4	Switch	Cisco	Cat...	Ком...	medium	high	1 Gbps	24	0	30W	{"Fast ...
5	Switch	HP	Pro...	Упр...	high	high	1 Gbps	24	0	35W	{"Giga...
6	Switch	Ubiqu...	US...	Ком...	medium	medium	1 Gbps	24	0	50W	{"Giga...
7	Acces...	Ubiqu...	UAP...	Точ...	medium	high	300 Mb...	1	200	9W	{"Giga...
8	Acces...	TP-Li...	EAP...	Точ...	low	medium	300 Mb...	1	100	12W	{"Giga...
9	Acces...	Cisco	AIR-...	Точ...	high	high	450 Mb...	1	150	15W	{"Giga...
10	Comp...	Dell	Opti...	Офи...	medium	medium	100 Mb...	1	0	65W	{"Giga...
11	Comp...	HP	Elite...	Раб...	high	high	1 Gbps	1	0	85W	{"Giga...
12	Comp...	Lenovo	Thin...	Мин...	low	medium	100 Mb...	1	0	35W	{"Giga...
13	Server	Dell	Pow...	Сер...	high	high	10 Gbps	4	0	500W	{"10G ...
14	Server	HP	Pro...	Сер...	medium	high	1 Gbps	4	0	350W	{"Giga...
15	Firewall	Fortin...	Fort...	UT...	high	high	1 Gbps	6	0	40W	{"Giga...
16	Firewall	pfSen...	SG-...	Бра...	medium	high	1 Gbps	4	0	20W	{"Giga...
17	NAS	Synol...	DS1...	Сет...	high	high	1 Gbps	2	0	100W	{"Giga...
18	NAS	QNAP	TS-4...	Сет...	medium	high	1 Gbps	2	0	70W	{"Giga...

Таблица 5.1 – Заполненная таблица типов устройств

Стоит оговориться, что в процессе работы параметр энергопотребления не был использован, но может быть полезен для проектирования очень больших систем, бесперебойная работа которых может приводить к большому энергопотреблению.

4.3 Подключение базы данных к приложению

Подключение к базе данных в приложении инициализируется в методе initialize_database() главного класса NetworkDesignApp, где параметры подключения задаются в явном виде. Для этого используется библиотека psycopg2, которая предоставляет интерфейс для работы с PostgreSQL. Параметры подключения (имя базы данных, пользователь, пароль, хост и порт) задаются явно в коде:

```
self.db_connection = psycopg2.connect(
    dbname="network_design_db",
    user="postgres",
    password="user",
    host="localhost",
    port="5432")
```

В случае ошибки подключения пользователю предлагается продолжить работу без базы данных или завершить программу, также для обеспечения отказоустойчивости реализована двухуровневая система обработки исключений. При работе с БД используются блоки try-except для обработки возможных ошибок. При отсутствии подключения к БД методы возвращают значения по умолчанию. Пользователь информируется о проблемах, но программа продолжает работу в ограниченном режиме. В критических операциях (например, save_to_db) используется откат изменений при ошибке.

Приложение позволяет сохранять топологии в БД в таблицу network_configurations, включая метаданные (уровень безопасности, количество устройств и т. д.). Для загрузки данных используется диалоговое окно LoadFromDBDialog, которое отображает список доступных конфигураций:

```
cursor.execute("""
    INSERT INTO network_configurations
    (name, description, topology_json, security_level, total_devices,
average_latency, total_bandwidth)
    VALUES (%s, %s, %s, %s, %s, %s, %s)
""", (name, description, json.dumps(config), avg_security, len(graph.n
odes()), avg_latency, total_bw))
```

Приложение поддерживает добавление новых типов устройств через диалоговое окно AddDeviceTypeDialog. Данные сохраняются в

таблицу device_types, которая содержит информацию о категории, производителе, модели, характеристиках и других параметрах.

При завершении работы приложения подключение к БД корректно закрывается в методе on_close():

```
def on_close(self):
      if self.db_connection:
            self.db_connection.close()
      plt.close('all')
      self.root.destroy()
```

ГЛАВА 5 АРХИТЕКТУРА ПРИЛОЖЕНИЯ И ЕЁ РАЗРАБОТКА

5.1 Создание интерфейса пользователя

Программа построена по модульному принципу, где каждый функциональный блок вынесен в отдельные классы и методы. Главное окно программы реализовано в классе NetworkDesignApp, который является центральным управляющим классом графического интерфейса. Это окно создаётся при инициализации программы и содержит логические блоки с элементами управления. Вид графического интерфейса начального окна представлен на рисунке далее.

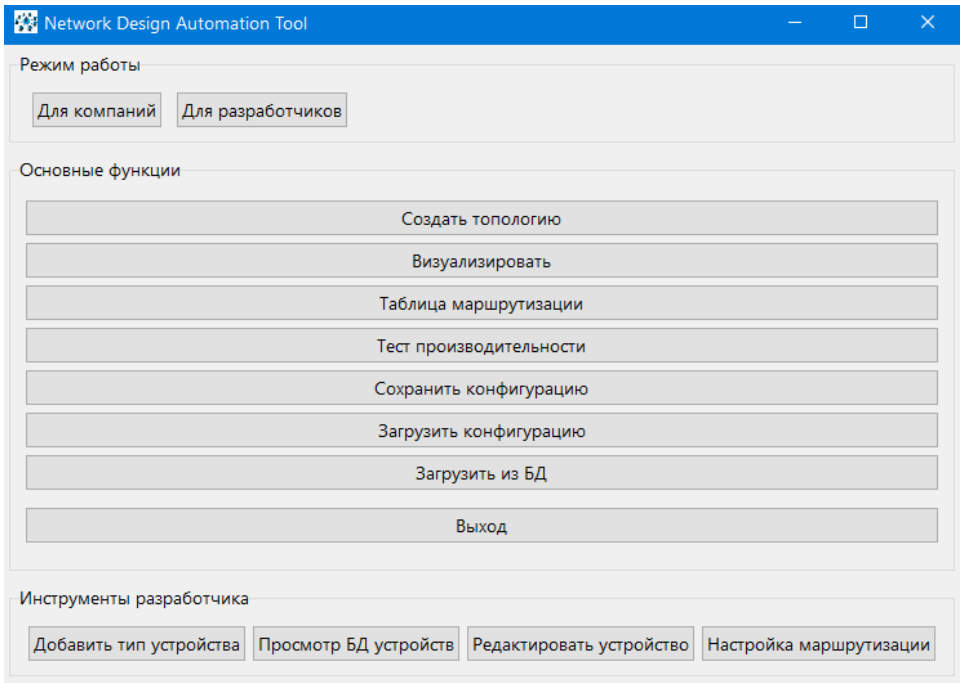


Рисунок 6.1 – Интерфейс, встречающий пользователя

Главное окно создаётся в методе __init__:

```
class NetworkDesignApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Network Design Automation Tool")
        self.root.protocol("WM_DELETE_WINDOW", self.on_close)
       self.ip_pool = []
        self.current_topology_index = None
        self.db_connection = None
        self.current_mode = "company"
        self.initialize_database()
        try:
            self.root.iconbitmap('network.ico')
```

```
except:
    pass
self.setup_ui()
```

В данном методе устанавливается заголовок окна, определяется обработчик закрытия для корректного завершения работы программы. Задаются стандартные параметры приложения, а именно отсутствующий начальный индекс выбранной топологии и режим «компания». Также в этом методе для программы устанавливается иконка network.ico, если такая существует, иначе же этот шаг пропускается.

Различные типы устройств в сети представлены различимыми разноцветными фигурами с помощью matplotlib, также различным типам соединений присваиваются разные вариации прямых разных цветов: сплошная, пунктирная и точечная.

Далее пользователь задаёт входные данные для начала построения сети. Это представлено на следующих рисунках:

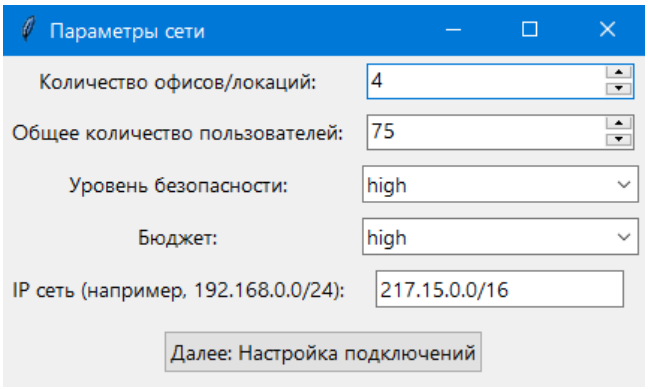


Рисунок 6.2 – Первая часть настройки подключений

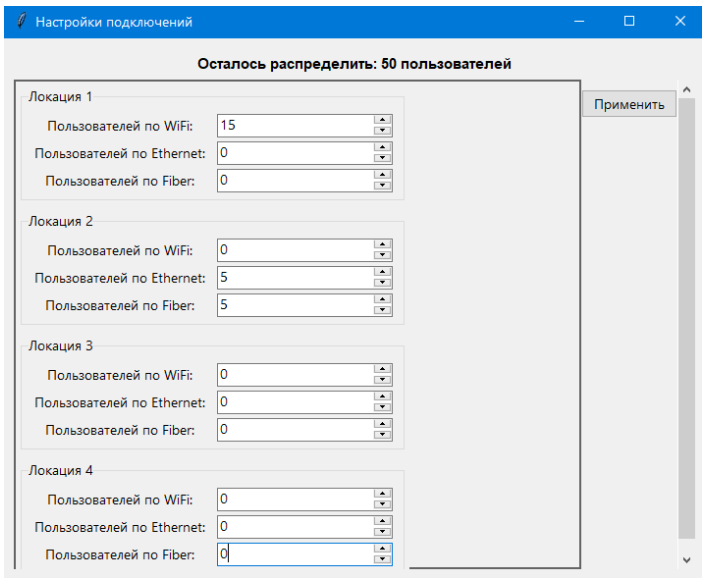


Рисунок 6.3 – Вторая часть настройки подключений

В данных окнах собирается вся необходимая информация. Пользователь сначала задаёт общее количество пользователей, которые могут нагружать сеть, после чего указывает необходимое количество нагрузки для каждого из филиалов. Можно обратить внимание, что для удобства использования динамически изменяется сообщение о том, сколько пользователей не распределено по типам соединений. После получения данных программа может приступить к построению требуемой компьютерной сети.

Ранее в работе рассказывалось о видах различных топологий и их характеристиках, как общих, так и в сравнении с остальными. В качестве основных топологий для реализации я выбрал централизованную, децентрализованную и гибридную топологии. Они являются более общими по сравнению с простейшими, что позволяет использовать их как универсальные. Особенностью централизованной топологии является то, что все подсети соединены одним общим элементом, через который проходит трафик. Децентрализованная топология такого элемента не имеет, а подсети могут быть связаны друг с другом как напрямую, так и через промежуточные устройства. Гибридная топология, в свою очередь, может сочетать обе модели: часть подсетей формирует централизованную структуру, а часть — децентрализованную.

5.2 Алгоритмическое построение топологии

Для централизованной топологии создаётся главный маршрутизатор (ядро сети), к которому подключаются все остальные устройства. Для более высокого уровня безопасности добавляется файрвол между ядром и остальными устройствами. При большем бюджете пользователя добавляется сетевое хранилище NAS. Подсети разных локаций подключаются к центральному узлу по принципу «звезды».

В децентрализованной топологии каждая из локаций имеет свой маршрутизатор, соединенный с другими по принципу минимального связного дерева. Для высокого бюджета добавляются дополнительные соединения между маршрутизаторами (подобно mesh-топологии). Устройства внутри локации, очевидно, подключаются к своему маршрутизатору.

В случае гибридной топологии маршрутизаторы группируются в кластеры, каждый из которых имеет свой главный роутер, подключенный к общему ядру. Внутри кластера используется звездообразная топология, а между кластерами топология типа mesh.

5.2.1 Распределение пользователей

За распределение пользователей по локациям отвечает следующий метод:

```
        def _distribute_users(self, total_users, num_locations,
connection_settings):
            distributed_users = 0
            users_per_location = [0] * num_locations
            for i in range(min(num_locations,
len(connection_settings))):
                loc_users = sum(connection_settings[i].values())
                users_per_location[i] = loc_users
                distributed_users += loc_users
            remaining_users = total_users - distributed_users
            if remaining_users > 0:
                base_users = remaining_users // num_locations
                extra = remaining_users % num_locations
                for i in range(num_locations):
         users_per_location[i] += base_users + (1 if i < extra else 0)
            return users_per_location
```

Этот метод распределяет общее количество пользователей total_users между локациями, учитывая настройки подключений connection_settings для каждой локации (Wi-Fi, Ethernet, Fiber). Сначала учитываются пользователи, явно указанные пользователем, оставшиеся, если такие есть, распределяются равномерно по всем локациям.

5.2.2 Назначение ip-адресов

За присвоение адресов элементам сети отвечает метод assign_ip(network):

```
def assign_ip(self, network):
    if not hasattr(self, 'ip_allocator'):
        self.ip_allocator = ipaddress.ip_network(network).hosts()
    try:
        return str(next(self.ip_allocator))
    except StopIteration:
        messagebox.showerror("Ошибка", "Свободных IP-адресов больше не
т")
        return None
```

В методе создается итератор ip_allocator для хостов в подсети. При каждом вызове метод возвращает следующий свободный адрес.

5.2.3 Размещение устройств по локациям

Метод _add_location_devices отвечает за генерацию и добавление в граф набора сетевых устройств и их взаимных соединений в рамках одной локации. Основываясь на типах подключения пользователей, метод создаёт структуру сетевых узлов, которая обеспечивает связь между пользователями и маршрутизаторами, а также размещает вспомогательные устройства, такие как серверы и точки доступа. В зависимости от количества пользователей, нуждающихся в подключении типа Ethernet, метод рассчитывает количество коммутаторов исходя из того, какой коммутатор был выбран для сети в зависимости от ценового сегмента. Затем в граф добавляются соответствующие связи между коммутаторами и маршрутизаторами. Аналогичный процесс происходит и для подключений по WiFi. Размещаются точки доступа и подключаются к маршрутизатору и устройствам. В зависимости от количества пользователей данный метод также размещает сервера, которые подключаются либо к коммутатору, либо к маршрутизатору, при этом используются каналы с увеличенными пропускными способностями. При добавлении компьютеров проверяется, имеется ли достаточно подключений соответствующих типов, и если это не так, выводится предупреждение о недостатке.

Алгоритм в этом методе позволяет масштабировать инфраструктуру сети в зависимости от пользователей и ресурсов, взаимодействует с другими методами и имеет логические проверки для правильного построения.

5.3 Визуализация построенной сети

В следующих пунктах будут описаны механизмы визуализации сети и используемые алгоритмы.

5.3.1 Алгоритм расстановки устройств

За расстановку устройств по сети отвечает метод _create_ordered_layout(graph). Его задача заключается в том, чтобы графически сеть была доступной к пониманию. Цель метода – это определить координаты (x, y) для каждого узла так, чтобы логически связанные устройства располагались рядом друг к другу, центральные узлы были различимы и не были перекрыты, ребра имеют минимальное количество пересечений между собой. На первых шагах алгоритм определяет тип топологии и на основании этого размещает узлы графа:

- При централизованной топологии главный узел (ядро) располагается в центре, а остальные узлы вокруг него.
- При децентрализованной топологии роутеры расположены по окружности, а узлы произвольно вокруг них.
- Гибридная топология имеет расположение кластеров вокруг своих маршрутизаторов.

Для оптимизации размещения узлов применяется алгоритм пружинной модели (Force-directed graph drawing**)** с помощью метода spring_layout. Смысл этой модели в том, что узлы отталкиваются друг от друга. Это реализуется тем, что ребра действуют как пружины, стягивая связанные узлы. Некоторое количество итераций позволяет найти баланс между читаемостью и минимизацией пересечений.

5.3.2 Визуализация топологий

Для визуализации топологии используется метод def draw_network_graph(self, graph, figure). В данном методе задаются визуальные свойства соединений. Далее создается упорядоченное расположение узлов с помощью метода _create_ordered_layout(graph).

```
            pos = self._create_ordered_layout(graph)
```

Далее отрисовываются соединения с соответствующими свойствами:

```
            for u, v, data in graph.edges(data=True):
                conn_type = data.get('connection_type', 'default')
    style = connection_styles.get(conn_type,
connection_styles['default'])
                color =  connection_colors.get(conn_type,
    connection_colors['default'])
                nx.draw_networkx_edges(graph, pos, edgelist=[(u, v)],
ax=ax,
                  style=style, width=1.5, edge_color=color)
```

Получаем визуальные свойства узлов сети:

```
            node_shapes = []
             node_sizes = []
             node_colors = []
             for node, data in graph.nodes(data=True):
                 device_type = data.get('device_type', 'Unknown')
                 if device_type in self.device_shapes:

node_shapes.append(self.device_shapes[device_type])
                 elif device_type in self.custom_device_shapes:
```

```
node_shapes.append(self.custom_device_shapes[device_type])
                else:
                    shape =
random.choice(list(self.device_shapes.values()))
                    self.custom_device_shapes[device_type] = shape
                    node_shapes.append(shape)
```

Назначаем различным типам устройств размер узла:

```
                if device_type == "Router":
                    node_sizes.append(800)
                elif device_type == "Switch":
                    node_sizes.append(600)
                elif device_type == "Server":
                    node_sizes.append(700)
                elif device_type == "Firewall":
                    node_sizes.append(750)
                elif device_type == "NAS":
                    node_sizes.append(650)
                else:
                    node_sizes.append(500)

       node_colors.append(self.device_colors.get(device_type, 'gray'))
```

Рисуем сами узлы:

```
            for i, (node, shape) in enumerate(zip(graph.nodes(),
node_shapes)):
              nx.draw_networkx_nodes(graph, pos, nodelist=[node],
node_shape=shape, node_size=node_sizes[i],
ax=ax,node_color=node_colors[i])
            node_labels = {n: f"{data.get('name',
'')}\n{data.get('ip', '')}"
            for n, data in graph.nodes(data=True)}
            nx.draw_networkx_labels(graph, pos, labels=node_labels,
             font_size=8, ax=ax, font_color='black')
            edge_labels = {(u, v): f"{d.get('connection_type',
'')}\n{d.get('bandwidth', '')}"
                          for u, v, d in graph.edges(data=True)}
            nx.draw_networkx_edge_labels(graph, pos,
edge_labels=edge_labels, font_size=0, ax=ax)
```

Создаётся легенда для ориентирования между устройствами и связями:

```
            legend_handles = []
            unique_device_types = set(data.get('device_type',
'Unknown')
           for _, data in graph.nodes(data=True))
             for dev_type in unique_device_types:
                 shape = (self.device_shapes.get(dev_type) or
                     self.custom_device_shapes.get(dev_type) or 'o')
                 color = self.device_colors.get(dev_type, 'gray')
                 legend_handles.append(plt.Line2D([0], [0],
marker=shape,color='w',
                  label=dev_type, markerfacecolor=color, markersize=10))
             unique_conn_types = set(d.get('connection_type',
'Ethernet')
             for _, _, d in graph.edges(data=True))
             for conn_type in unique_conn_types:
                 style = connection_styles.get(conn_type, '-')
                 color = connection_colors.get(conn_type, 'black')
                 legend_handles.append(plt.Line2D([0],[1],color=color,
         linestyle=style, linewidth=2, label=f"{conn_type}
connection"))
```

Размещаем легенду в окне отрисовки графа:

```
           ax.legend(handles=legend_handles, bbox_to_anchor=(1.05, 1),
                     loc='upper left', fontsize=8)
             ax.set_facecolor('#f0f0f0')
             figure.set_facecolor('#f0f0f0')
             plt.tight_layout()
```

На следующем рисунке показан пример визуализированной сети (децентрализованная топология, 5 локаций суммарно на 60 устройств, для каждой локации связи внутри подсети различные):

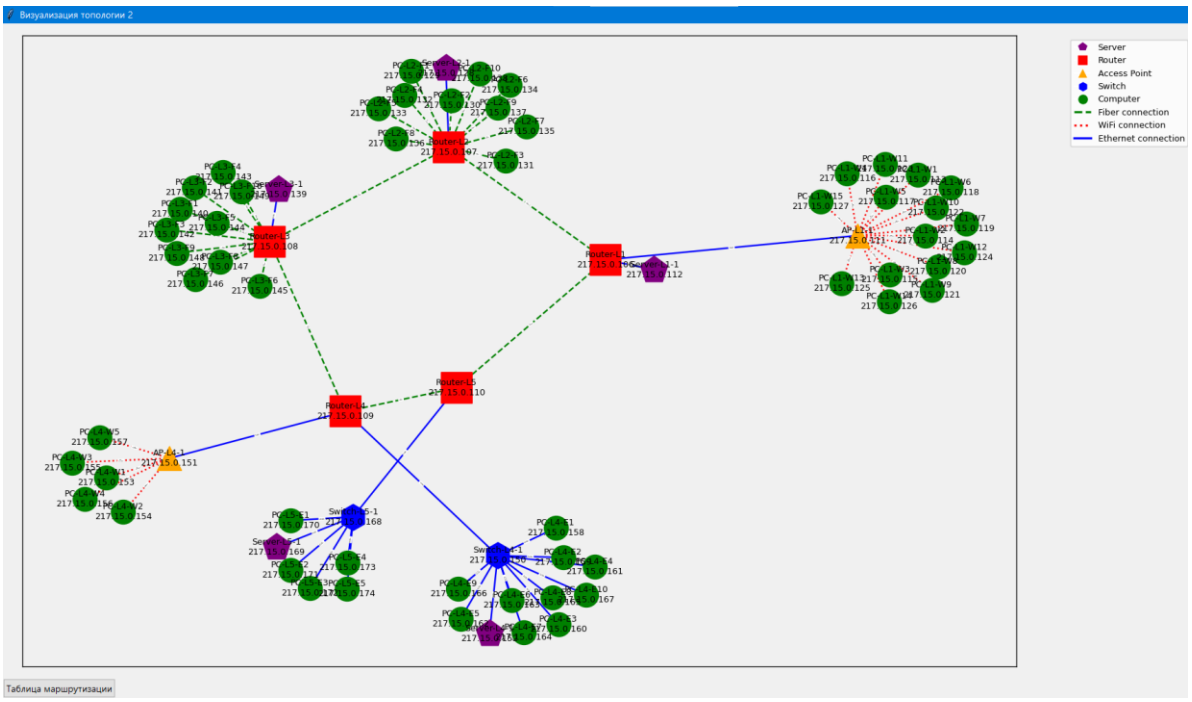


Рисунок 6.4 – Пример построения децентрализованной сети

Примеры визуализаций для централизованной и гибридной топологий представлены в Приложении А.

5.4 Тестирование построенной сети

Для тестирования сети используются различные метрики с помощью метода _calculate_basic_metrics(graph). В результате использования этого метода получаются такие характеристики как общее количество устройств и соединений, распределение устройств по типам, средняя и общая пропускная способность, средняя задержка.

С помощью метода _find_bottlenecks(graph) проверяются соединения с низкой пропускной способностью или перегруженностью.

Метод _simulate_traffic(graph) проверяется нагрузка на каждое соединение и количество проходящих через узел соединений.

Безопасность сети проверяется методом _analyze_security(graph). Данный метод в зависимости от характеристик безопасности устройств и наличия файрволов делает вывод об уровне защищенности.

Данные от этих методов могут быть использованы методом test_network_performance(self). В нем реализовано общее тестирование сети с генерацией отчета в виде html файла. Данные для отчета собираются с помощью

```
report = {
        "topology_type": self._detect_topology_type(graph),
```

```
            "basic_metrics":self._calculate_basic_metrics(graph),
            "traffic_simulation": self._simulate_traffic(graph),
            "security_analysis": self._analyze_security(graph),
            "subnet_analysis": self._analyze_subnets(graph),
            "critical_paths": self._find_critical_paths(graph),
            "recommendations": []
        }
```

Важно заметить, что тестирование проходит только последняя визуализированная пользователем, либо же загруженная из файла топология. На рисунке далее приведен пример такого отчета:

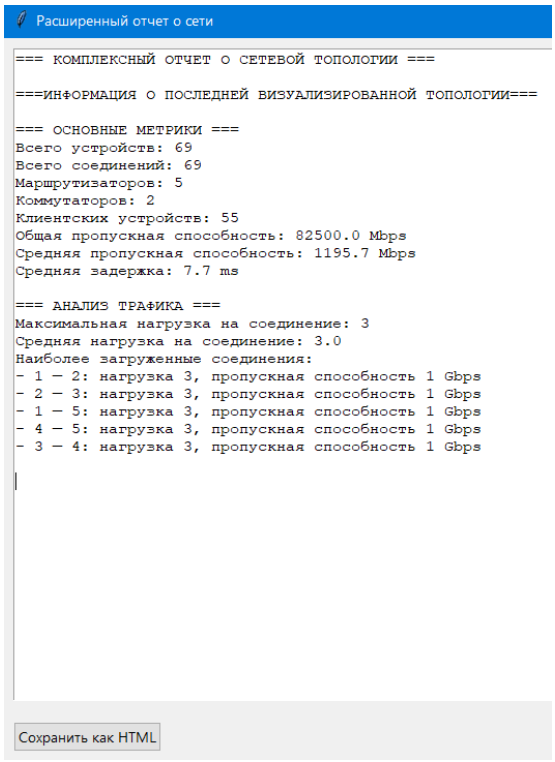


Рисунок 6.5 – Отчёт о сетевой топологии

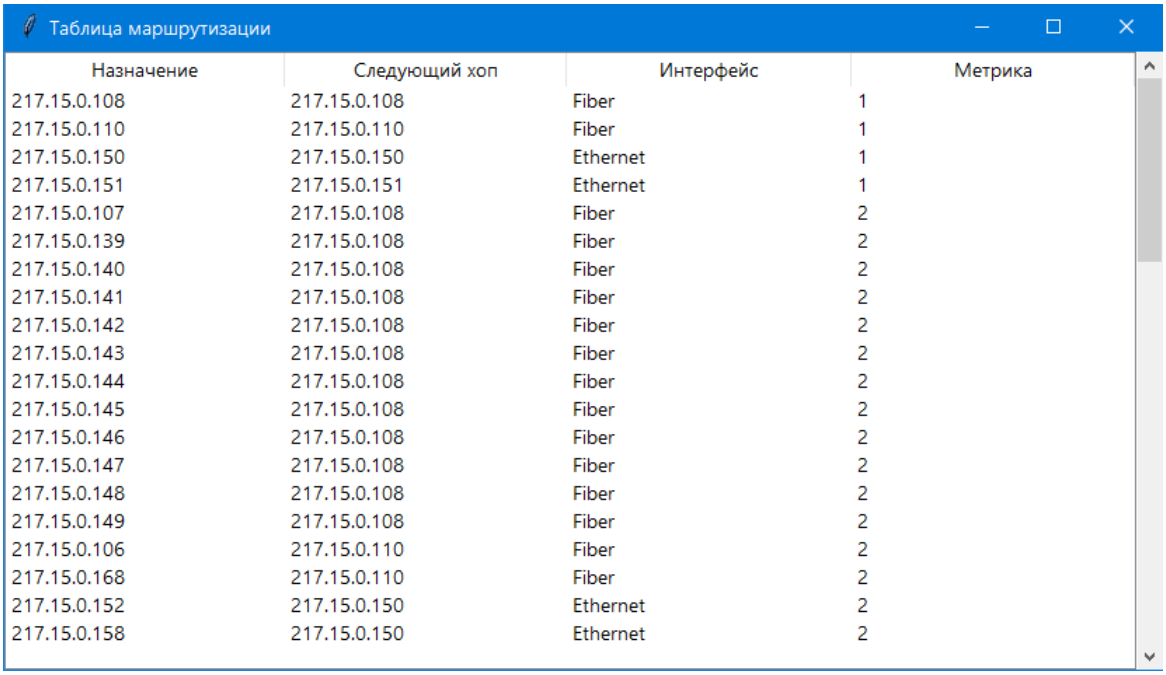
5.5 Дополнительный реализованный функционал

Одним из реализованных и полезных инструментов является построение таблиц маршрутизации. После построения некоторой сети пользователь может не только изучить связи каждого из маршрутизаторов с остальными устройствами, но и сделать это в зависимости от алгоритма маршрутизации. В данной работе были использованы 4 алгоритма маршрутизации:

- Shortest path - определяет оптимальный путь между узлами на основе метрики (например, количество переходов, задержка, пропускная способность).

- OSPF - каждый маршрутизатор строит полную топологию сети, обмениваясь LSA (Link-State Advertisements).
- EIGRP - хранит топологическую таблицу и выбирает маршруты на основе метрик**.** Поддерживает равноправные и неравноправные пути**.**
- Static – стандартная статическая маршрутизация.

Эти алгоритмы в коде реализованы в классе RoutingManager с помощью встроенной функции nx.single_source_dijkstra_path( ). Она используется для каждого из алгоритмов, но имеет разные обрабатываемые параметры, которые и позволяют через изменение подсчета метрики строить таблицу маршрутизации для соответствующих алгоритмов. За визуальную часть таблицы маршрутизации отвечает класс RoutingTableDialog. На рисунке ниже представлен пример таблицы маршрутизации.



Назначение	Следующий хоп	Интерфейс	Метрика
217.15.0.108	217.15.0.108	Fiber	1
217.15.0.110	217.15.0.110	Fiber	1
217.15.0.150	217.15.0.150	Ethernet	1
217.15.0.151	217.15.0.151	Ethernet	1
217.15.0.107	217.15.0.108	Fiber	2
217.15.0.139	217.15.0.108	Fiber	2
217.15.0.140	217.15.0.108	Fiber	2
217.15.0.141	217.15.0.108	Fiber	2
217.15.0.142	217.15.0.108	Fiber	2
217.15.0.143	217.15.0.108	Fiber	2
217.15.0.144	217.15.0.108	Fiber	2
217.15.0.145	217.15.0.108	Fiber	2
217.15.0.146	217.15.0.108	Fiber	2
217.15.0.147	217.15.0.108	Fiber	2
217.15.0.148	217.15.0.108	Fiber	2
217.15.0.149	217.15.0.108	Fiber	2
217.15.0.106	217.15.0.110	Fiber	2
217.15.0.168	217.15.0.110	Fiber	2
217.15.0.152	217.15.0.150	Ethernet	2
217.15.0.158	217.15.0.150	Ethernet	2

Рисунок 6.6 – Таблица маршрутизации

Конфигурации сетей могут быть загружены и сохранены с помощью соответствующих кнопок. В данной работе это реализовано с помощью методов load_configuration( ) и save_configuration( ). Оба метода взаимодействуют с конфигурацией и взаимодействуют с файлами типа JSON.

С помощью метода show_device_types(self) и подключения к БД можно просмотреть все устройства имеющиеся там.

Добавить к ним новое устройство можно с помощью метода add_device_type_dialog( ). С помощью него реализовано открытие диалогового окна добавления устройства, где пользователь должен будет ввести все требуемые характеристики.

Редактирование уже имеющихся устройств в графе реализовано с помощью метода основного класса NetworkDesignApp edit_device_dialog( ).

Метод save_to_db( ) конвертирует граф сети в JSON формат, выполняет запрос INSERT и тем самым сохраняет топологию со всеми её параметрами в базу данных, а load_from_db( ) загружает JSON конфигурации, восстанавливает узлы с их атрибутами и соединения с их параметрами, после чего текущая топология переключается на загруженную с возможностью немедленно её визуализировать.

ЗАКЛЮЧЕНИЕ

В данной дипломной работе было проведено исследование процесса построения и возможностей его автоматизации. Рассмотрены преимущества и недостатки реализаций данной задачи у уже готовых сторонних продуктов. Были рассмотрены и использованы различные библиотеки, позволяющие реализовать данную задачу.

Результатом работы является функционирующее приложение по автоматизации построения компьютерных сетей, с возможностью строить различные типы топологий в зависимости от поставленных условий, проводить их тестирование с генерацией развернутого отчета. Была реализована возможность многократно использовать уже созданные топологии, строить таблицы истинности устройств, образующих созданную сеть, а также взаимодействовать с базой данных устройств.

Результаты данной работы могут быть применены при построении компьютерных сетей, а также могут быть доработаны, например, улучшением имеющихся алгоритмов или использованием нейросетей, которые позволят лучше масштабировать и улучшить уже имеющиеся алгоритмы построения топологий.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы GNS3. Обзор – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/266503/.
2. Установка и использование виртуальной сетевой лаборатории EVE-NG совместно с Ansible – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/323014/.
3. Всё о схематизации с Microsoft Visio – [Электронный ресурс]. – Режим доступа: https://www.lucidchart.com/pages/ru/what-is-visio.
4. SolarWinds Network Topology Mapper – [Электронный ресурс]. – Режим доступа: https://www.uvexplorer.com/comparisons/uvexplorer-vs-solarwinds-network-topology-mapper/.
5. Miro vs. Lucid – [Электронный ресурс]. – Режим доступа: https://miro.com/compare/miro-vs-lucidchart/.
6. Официальный сайт EdrawMax – [Электронный ресурс]. – Режим доступа: https://www.edrawsoft.com/ru/.
7. Таненбаум Э., Уэзеролл Д. Компьютерные сети. 5-е изд. – СПб.: Питер, 2012. – 960 с.: ил.
8. Урбанович, П. П. Компьютерные сети и сетевые технологии : учеб. пособие для студентов технических специальностей / П. П. Урбанович, Д. М. Романенко. – Минск : БГТУ, 2022. – 608 с.
9. Лимончелли Т., Хоган К., Чейлап Системное и сетевое администрирование. Практическое руководство, 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2009. – 944 с.
10. Официальная документация по языку Python– [Электронный ресурс]. – Режим доступа: https://docs.python.org/3.
11. Библиотеки Python – [Электронный ресурс]. – Режим доступа: https://pythonru.com/biblioteki/.
12. Официальная документация по СУБД PostgreSQL – [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/.
13. Топология физический связей – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/850834/.
14. Чоу Э. Python для сетевых инженеров. Автоматизация сети, программирование и DevOps, 3-е издание – СПб.: Питер, 2023. – 528 с.: ил.
15. Лутц М. Изучаем Python, 4-е издание. – СПб.: Символ-Плюс, 2011. – 1280 с.: ил.
16. Tutorial – NetworkX 3.4.2 documentation – [Электронный ресурс] – режим доступа: https://networkx.org/documentation/stable/tutorial.html.

ПРИЛОЖЕНИЕ А

ПРИМЕРЫ ВИЗУАЛИЗАЦИЙ ТОПОЛОГИЙ

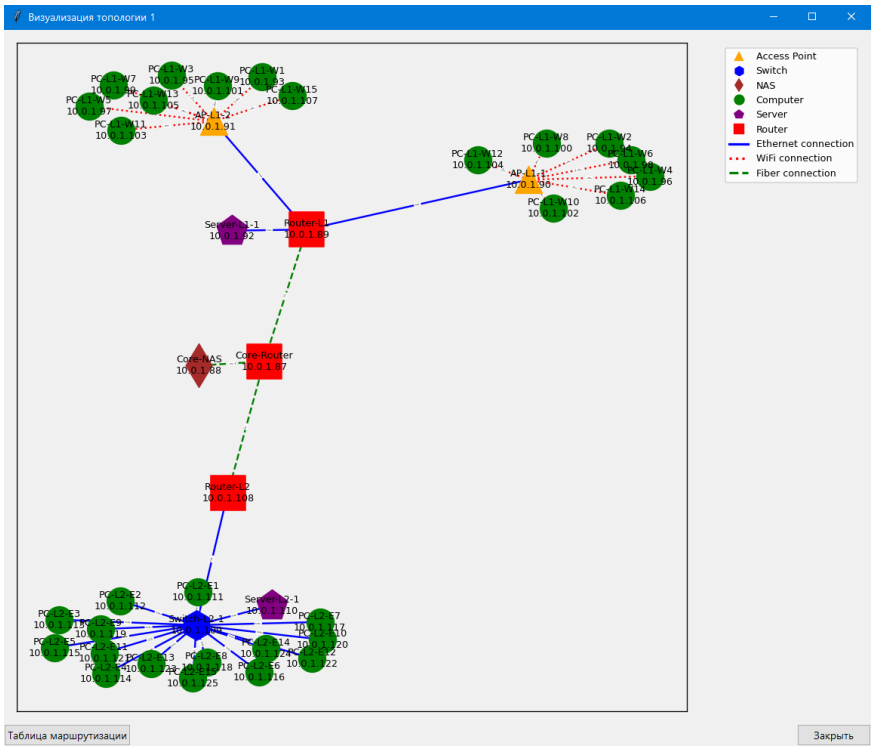


Рисунок А.1 – Пример централизованной топологии

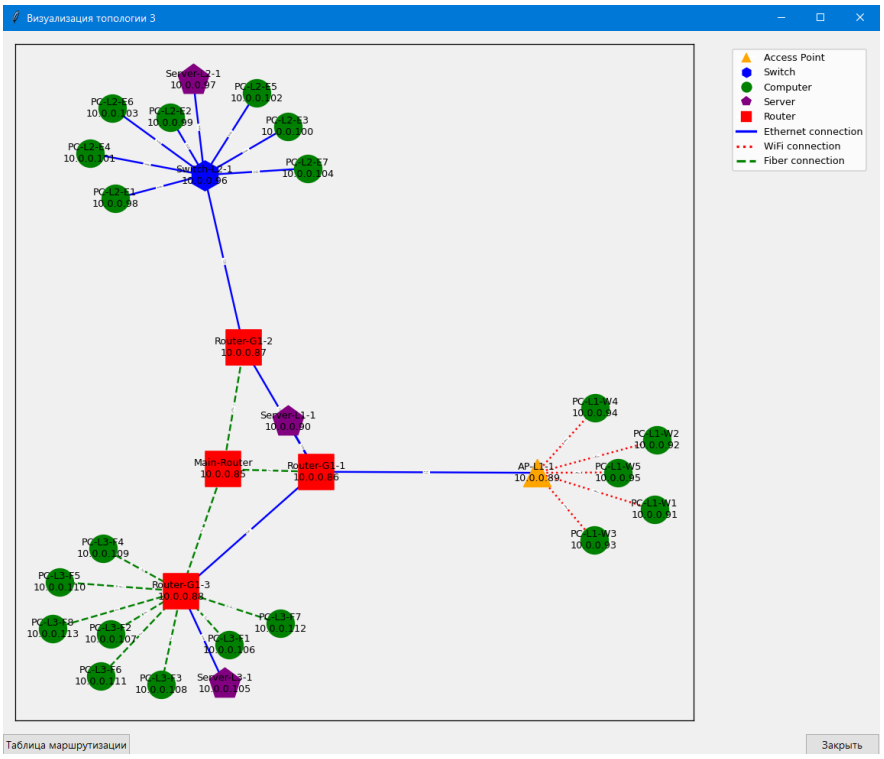


Рисунок А.2 – Пример гибридной топологии