Санкт-Петербургский государственный университет

Программная инженерия

Группа 24.М71-мм

# Оптимизация и распределённое выполнение сетевой симуляции в Miminet с использованием Docker

***Альшаеб Баселъ***

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ст. преподаватель кафедры ИАС. И. В. Зеленчук.

Санкт-Петербург
2025

# Оглавление

# Введение

Курс по компьютерным сетям[13] является обязательным в рамках учебной программы Математико-механического факультета, так как он играет ключевую роль в подготовке специалистов по программной инженерии, системному администрированию и других направлений, связанных с разработкой и эксплуатацией информационных систем. Этот курс включает в себя различные разделы, которые должны быть продемонстрированы на практике с использованием реальных примеров и ситуаций. Однако для эффективного освоения материала необходимо не только изучение теории, но и практическая работа с сетевыми конфигурациями.

Для того чтобы реализовать практическую часть курса, можно приобрести физическое оборудование, такое как коммутаторы и маршрутизаторы. Например, коммутатор Cisco[1] Catalyst 3500/3700 серии, который используется в образовательных целях, может стоить до 2000 долларов США за один блок. Вдобавок, для создания полноценной лаборатории потребуется достаточно большое количество оборудования, что приводит к дополнительным расходам на кабели, сетевые карты и другие элементы, которые могут стоить тысячи долларов. Например, качественные Ethernet-кабели категории 6 с длиной 10 метров стоят около 10-20 долларов за штуку. Чтобы организовать несколько лабораторий с необходимым оборудованием, затраты на оборудование и инфраструктуру могут легко превысить десятки тысяч долларов.

Однако, в 2022 году компания Cisco Systems покинула российский рынок, что значительно усложнило процесс приобретения оборудования, сделав его не только более дорогим, но и труднодоступным. Это создало дополнительные сложности для образовательных учреждений, в том числе для Математико-механического факультета, где курс по компьютерным сетям является обязательным для студентов. Возникла необходимость находить альтернативные решения, которые позволили бы проводить практические занятия по сетям, не полагаясь на дорогие физические устройства.

Для решения этой проблемы был выбран эмулятор Miminet, который был разработан на основе Mininet и позволяет создавать, конфигурировать и тестировать сети в виртуальной среде. Этот эмулятор предоставляет возможность проведения практических занятий с минимальными затратами и делает курс доступным и удобным для образовательных целей. Благодаря эмуляции, студенты могут работать с сетями в виртуальной среде, не требуя физического оборудования. Это позволяет значительно снизить расходы на материально-техническое обеспечение курса и предоставляет возможность масштабировать учебный процесс, не ограничиваясь количеством доступных физических устройств.

Однако, несмотря на явные преимущества эмуляторов, проблема с долгим временем ожидания студентов всё ещё остаётся актуальной. Когда большое количество студентов одновременно выполняет задания в эмуляторе, каждый из них должен ждать своей очереди, что существенно снижает эффективность обучения и замедляет процесс усвоения материала.

В связи с этим возникает необходимость разработки решения для улучшения производительности и масштабируемости Miminet[14]. Для этого будет реализован механизм распределения нагрузки между несколькими контейнерами, что позволит значительно уменьшить время ожидания студентов иповысить эффективность выполнения симуляций. В частности, система будет использовать динамическое распределение ресурсов и многопроцессорную обработку, что обеспечит более гибкое управление и оптимизацию работы эмулятора. Эти улучшения позволят создать более стабильную и эффективную среду для обучения сетевым технологиям.

# 1. Постановка задачи

Целью работы является оптимизация выполнения сетевых симуляций в Miminet[14] за счёт эффективного распределения нагрузки между несколькими контейнерами Docker. Основной задачей является увеличение масштабируемости симуляций путём равномерного распределения вычислительных ресурсов хоста без использования параллельных вычислений внутри контейнеров.

Для достижения данной цели были поставлены следующие задачи:

1. Разработка архитектуры распределённого выполнения симуляций.
    - Реализовать механизм запуска и управления несколькими контейнерами, выполняющими симуляции Miminet[14].
    - Обеспечить взаимодействие между контейнерами и централизованный контроль выполнения задач.
2. Проектирование системы управления задачами и балансировки нагрузки.
    - Выбрать и настроить механизм передачи задач (Celery [10] + RabbitMQ [4]).
    - Разработать стратегию распределения симуляций между контейнерами, учитывающую доступные вычислительные ресурсы.
    - Обеспечить возможность динамического масштабирования количества контейнеров.
3. Экспериментальное тестирование эффективности распределённого подхода.
    - Провести сравнение с традиционным подходом выполнения симуляций в одном контейнере.

- Оценить влияние распределения симуляций на загрузку процессора, время выполнения и потребление памяти.

4. Валидация результатов и демонстрация практической применимости.

   - Провести апробацию предложенного метода на тестовых сетевых топологиях различной сложности.
   - Оценить потенциальные сценарии использования в реальных задачах сетевого моделирования.
   - Подготовить документацию и руководство по использованию разработанной системы.

5. Предложенный подход позволит более эффективно использовать вычислительные ресурсы хоста, избегая перегрузки одного контейнера и обеспечивая стабильную работу симуляций при увеличении их количества.

# 2. Обзор

В данном разделе рассматриваются существующие решения для выполнения сетевых симуляций, используемые технологии и их сравнительный анализ.

## 2.1. существующей архитектуры Miminet

Miminet — это симулятор компьютерных сетей, основанный на Mininet и адаптированный для работы в контейнеризированной среде Docker. Его основная цель — предоставление удобного инструмента для моделирования сетевых взаимодействий между узлами, коммутаторами и маршрутизаторами в изолированной среде.

Miminet использует централизованную модель управления симуляцией, где:

- Вся симуляция выполняется внутри одного контейнера Docker;
- Виртуальные узлы и соединения создаются с использованием Linux network namespaces и эмулируемых Ethernet-интерфейсов;
- Симуляция управляется через Python[3] API Mininet, обеспечивая настройку узлов, коммутаторов и маршрутизаторов;
- Весь процесс выполняется в одном окружении, что ограничивает масштабируемость при увеличении количества узлов;

### 2.1.1. Ограничения текущей реализации

Несмотря на преимущества контейнеризированного подхода, текущая архитектура Miminet имеет ряд ограничений, которые могут снижать эффективность работы системы.

Первое ограничение связано с **масштабируемостью**. Поскольку вся симуляция выполняется в одном контейнере, невозможно динамически распределять нагрузку между несколькими процессами или узлами. При увеличении количества узлов нагрузка на вычислительные

ресурсы хоста (CPU и RAM) возрастает, что приводит к увеличению времени выполнения симуляции и ограничивает её масштабируемость.

- **Текущее время выполнения симуляции**: Каждая симуляция занимает примерно 10 секунд, с использованием 10% CPU. При увеличении числа узлов, время выполнения растёт, что приводит к возникновению узких мест в производительности.

Второе ограничение касается **использования вычислительных ресурсов**. В текущей реализации все процессы симуляции выполняются в одном потоке без задействования многопроцессорности. Это означает, что даже если хостовая машина имеет несколько доступных ядер процессора, они не используются эффективно, что снижает общую производительность.

- **Пример текущего использования ресурсов**: CPU загружен на 10%, но при увеличении количества симуляций и сложности сетевой топологии, эффективность использования CPU значительно снижается, что увеличивает время выполнения.

Третье ограничение относится к **сетевым аспектам**. Все контейнеры используют общий сетевой стек, что приводит к затруднениям при одновременном запуске нескольких симуляций. Это создаёт потенциальные конфликты и снижает гибкость в управлении ресурсами сети. Кроме того, отсутствие явной сетевой изоляции делает невозможным параллельное выполнение симуляций с разными конфигурациями.

- **Пример снижения пропускной способности**: При текущей архитектуре, пропускная способность системы составляет X симуляций в час. При увеличении сложности сетевой топологии, эта пропускная способность снижается, что подчеркивает необходимость оптимизации.

## 2.2. используемых технологий

В данном проекте применяются следующие технологии:

- **Flask[6]** — популярный фреймворк для разработки веб-приложений и API на языке Python[3]. Его преимущества включают лёгкость освоения, поддержку шаблонов и гибкость в создании

серверной логики. В данном проекте **Flask** используется для обработки HTTP-запросов и взаимодействия с базой данных.

- **Jinja2[7]** — встроенный в Flask шаблонизатор, позволяющий динамически генерировать HTML-страницы. Использование **Jinja2** обеспечивает удобную интеграцию Python[3]-кода в шаблоны, что упрощает создание и поддержку интерфейса приложения.

- **JavaScript[5]** — язык программирования, применяемый для добавления интерактивности на веб-страницах. В проекте используется совместно с **AJAX** для динамической загрузки данных без необходимости полной перезагрузки страницы, что улучшает удобство и скорость работы приложения.

- **jQuery[2]** — библиотека JavaScript, которая облегчает работу с элементами DOM, управление событиями и обработку запросов **AJAX**. Её использование также помогает обеспечить кроссбраузерную совместимость и упрощает клиентскую часть приложения.

- **Bootstrap[12]** — библиотека для создания адаптивных пользовательских интерфейсов. Она предоставляет готовые компоненты и стили, ускоряя разработку страниц с современным и единообразным дизайном.

- **SQLite[9]** — встроенная реляционная база данных, которая используется для хранения и управления данными проекта. Её лёгкость и простота конфигурации идеально подходят для текущих задач приложения.

- **SQLAlchemy[11]** — ORM-библиотека для Python[3], упрощающая взаимодействие с реляционными базами данных. В проекте она обеспечивает удобную работу с данными, предоставляя инструменты для построения запросов и управления объектами.

## 2.3. Выводы

Текущая архитектура Miminet ограничена возможностями масштабируемости и эффективного использования ресурсов хоста. Для устранения этих проблем необходимо перераспределить нагрузку между несколькими контейнерами, обеспечив изоляцию сетевого стека и динамическое управление симуляциями.

# 3. Описание решения

В рамках данной работы была реализована архитектура распределённого выполнения сетевых эмуляций в рамках проекта Miminet[14] с использованием Docker и системы управления задачами Celery + RabbitMQ[4]. Это позволило повысить масштабируемость платформы и сократить время ожидания запуска эмуляции для студентов.

## 3.1. Разработка архитектуры распределённого выполнения эмуляций

Изначально все эмуляции в Miminet[14] выполнялись в одном контейнере, что ограничивало масштабируемость: увеличение числа одновременных задач приводило к росту времени ожидания и перегрузке одного ядра CPU.

Предложенное решение включает запуск нескольких изолированных Docker-контейнеров, каждый из которых может выполнять одну или несколько эмуляций. Контейнеры работают независимо и получают задания из общей очереди RabbitMQ[4], а изоляция каждой эмуляции реализована с помощью Mininet[8].

Каждый контейнер содержит воркер Celery[10], получающий задачи из общей очереди. Такая схема позволяет масштабировать систему горизонтально — за счёт увеличения количества рабочих контейнеров без изменения основной логики.

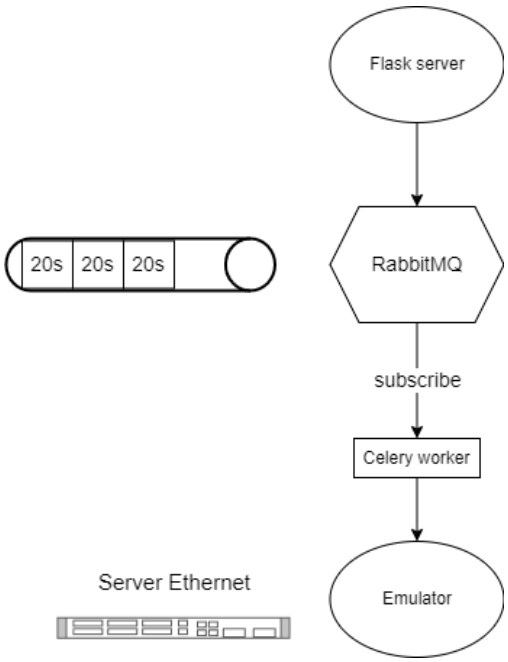


Рис. 1: Сравнение архитектур: исходная (один контейнер)

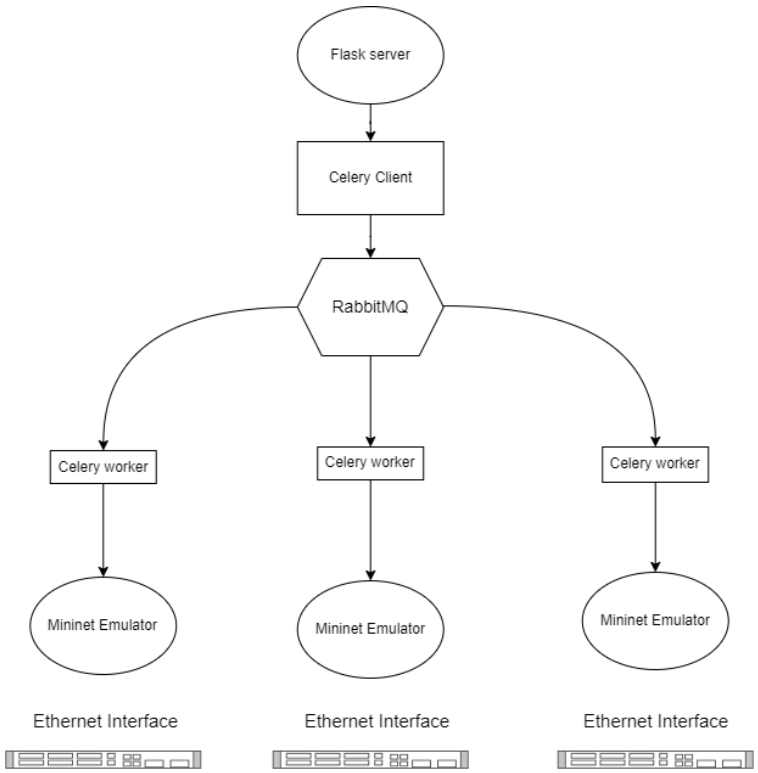


Рис. 2: Сравнение архитектур: новая (множество контейнеров)

## 3.2. Проектирование и внедрение системы управления задачами

Для распределения задач по контейнерам была использована связка Celery[10] + RabbitMQ[14]. Celery[10] позволяет организовать очередь заданий и обеспечивает параллельное выполнение с учётом текущей загрузки.

В рамках данной работы была выполнена:

- настройка очереди заданий для передачи задач воркерам.
- параметризация контейнеров (например, через переменные окружения) для запуска с разными конфигурациями.

Каждый контейнер при старте подключается к общей очереди RabbitMQ[4] и ожидает задания. Как только задача поступает — она исполняется и возвращает результат в центральную систему.

Такой подход позволяет эффективно масштабировать систему, добавляя или отключая контейнеры без необходимости остановки сервиса.

## 3.3. Экспериментальное тестирование и валидация

Для проверки эффективности решения были проведены эксперименты с запуском 20 эмуляций в двух режимах:

- один контейнер с одним воркером (исходная реализация Miminet);
- четыре контейнера, работающие параллельно (предложенная в рамках данной работы).

### Методика

В каждой эмуляции вручную фиксировалось время выполнения:

```
import time
start = time.time()
```

```
# выполнение эмуляции
end = time.time()
print(f"Эмуляция завершена за {end - start:.2f} сек.")
```

### Метрики

- Общее время выполнения заданий.
- Средняя загрузка CPU.
- Среднее время ожидания запуска эмуляции для студентов сократилось.
- Количество одновременно исполняемых эмуляций.

Таблица 1: Сравнение производительности

| Конфигурация | Общее время (сек) | Ускорение |
|---|---|---|
| 1 контейнер | 100 | 1.0× |
| 4 контейнера | 25 | 4.0× |

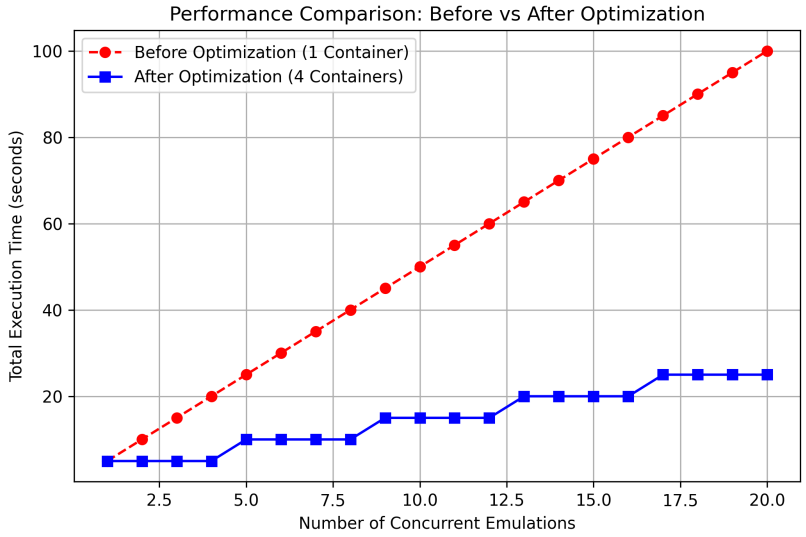


Рис. 3: График сравнения времени при запуске 20 эмуляций

Эксперименты показали, что система с несколькими контейнерами позволяет запускать больше эмуляций одновременно и равномерно распределяет нагрузку между ядрами процессора.

Это, в свою очередь, приводит к сокращению времени ожидания и повышению эффективности обучения.

**Выводы:**

- Количество параллельно исполняемых задач увеличилось в 4 раза;
- Загрузка CPU распределяется равномерно между ядрами;
- Среднее время ожидания студентом до начала эмуляции сокращается с 100 до 25 секунд.

## 3.4. Итоги реализации

Предложенное решение было внедрено в существующий код проекта Miminet и протестировано в условиях, приближённых к реальному использованию студентами. Архитектура допускает дальнейшее масштабирование и автоматизацию (например, динамическое добавление контейнеров на основе текущей загрузки).

Исходный код решения доступен в открытом репозитории: https://github.com/mimi-net/miminet

# Список литературы


[1] Cisco System Inc. Cisco. — URL: https://www.cisco.com/.

[2] Foundation OpenJs. jQuery 3.6.0. — URL: https://jquery.com/.

[3] Foundation Python Software. Python 3. — URL: https://docs.python.org/3.13/.

[4] Inc Broadcom. RabbitMQ 4.0.6. — URL: https://www.rabbitmq.com/.

[5] JavaScript. — URL: https://www.javascript.com.

[6] Pallets. Flask 3.0.3. — URL: https://flask.palletsprojects.com.

[7] Pallets. Jinja2 3.1.4. — URL: https://jinja.palletsprojects.com.

[8] Prete Rogério Leão Santos De Oliveira Christiane Marie Schweitzer Ailton Akira Shinoda Ligia Rodrigues. Using mininet for emulation and prototyping software-defined networks. — P. 1–6.

[9] SQLite 3.47.2. — URL: https://www.sqlite.org.

[10] Solem Ask. Celery Project 5.4.0. — URL: https://docs.celeryq.dev/en/stable/getting-started/index.html.

[11] authors SQLAlchemy, contributors. SQLAlchemy 2.0.36. — URL: https://www.sqlalchemy.org/.

[12] team Bootstrap. Bootstrap 5.3. — URL: https://getbootstrap.com/.

[13] Зеленчук Илья. Основы компьютерных сетей. — URL: https://stepik.org/course/208904/syllabus.

[14] СПбГУ. Веб-эмулятор Miminet. — URL: https://miminet.ru/ (дата обращения: 22 февраля 2025 г.).