Санкт-Петербургский государственный университет

Программная инженерия

Группа 24.М71-мм

Оптимизация и распределённое выполнение сетевой симуляции в Miminet с использованием Docker

Альшаеб Басель

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ст. преподаватель кафедры ИАС, к. ф.-м. н. И. В. Зеленчук.

Санкт-Петербург
2025

Оглавление

Введение

Обучение компьютерным сетям играет ключевую роль в подготовке специалистов по программной инженерии, системному администрированию и других направлений, связанных с разработкой и эксплуатацией информационных систем. Для эффективного освоения этой области необходимо не только изучение теории, но и практическая работа с сетевыми конфигурациями. Однако использование физического оборудования для обучения зачастую связано с высокими затратами и сложностью организации.

Для решения этой проблемы многие университеты используют эмуляторы, которые позволяют моделировать сетевые взаимодействия и проводить практические занятия без необходимости в реальном оборудовании. Одним из таких инструментов является Miminet[5], разработанный на базе Mininet[3]. Miminet[5] позволяет создавать, конфигурировать и тестировать сети в виртуальной среде, что делает его доступным и удобным для образовательных целей.

Несмотря на явные преимущества использования эмуляторов, в Miminet[5] существует проблема с долгим временем ожидания, которая ограничивает эффективность обучения. Когда большое количество студентов одновременно выполняет задания в эмуляторе, каждый из них должен ждать своей очереди, что существенно замедляет процесс обучения. Эта проблема усугубляется ограничением вычислительных ресурсов, так как текущая архитектура эмулятора использует один контейнер для всех симуляций, что приводит к перегрузке системы.

В связи с этим возникает необходимость разработки решения для улучшения производительности и масштабируемости Miminet[5]. 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 улуч-

шения позволят создать более стабильную и эффективную среду для обучения сетевым технологиям.

1. Постановка задачи

Целью работы является оптимизация выполнения сетевых симуляций в Miminet[5] 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[5];
    - Обеспечить взаимодействие между контейнерами и централизованный контроль выполнения задач;
2. Проектирование системы управления задачами и балансировки нагрузки;
    - Выбрать и настроить механизм передачи задач (Celery [4] + RabbitMQ [2]);
    - 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[1] 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** — популярный фреймворк для разработки веб-приложений и API на языке Python[1]. 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** — встроенный в Flask шаблонизатор, позволяющий динамически генерировать HTML-страницы. Использование **Jinja2** обеспечивает удобную интеграцию Python[1]-кода в шаблоны, что упрощает создание и поддержку интерфейса приложения.

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

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

- **SQLAlchemy** — ORM-библиотека для Python[1], 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.

Список литературы

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

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

[3] 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.

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

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