Санкт-Петербургский государственный университет

Кафедра системного программирования

Оптимизация и распределённое выполнение сетевой эмуляции в Miminet с использованием Docker

Альшаеб Басель, группа 24.М71-мм

Научный руководитель: к.ф.-м.н. И. В. Зеленчук

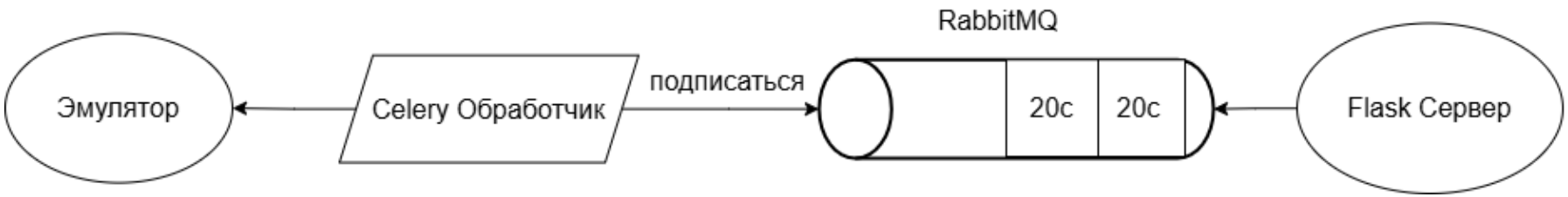
Санкт-Петербург
2025

Введение

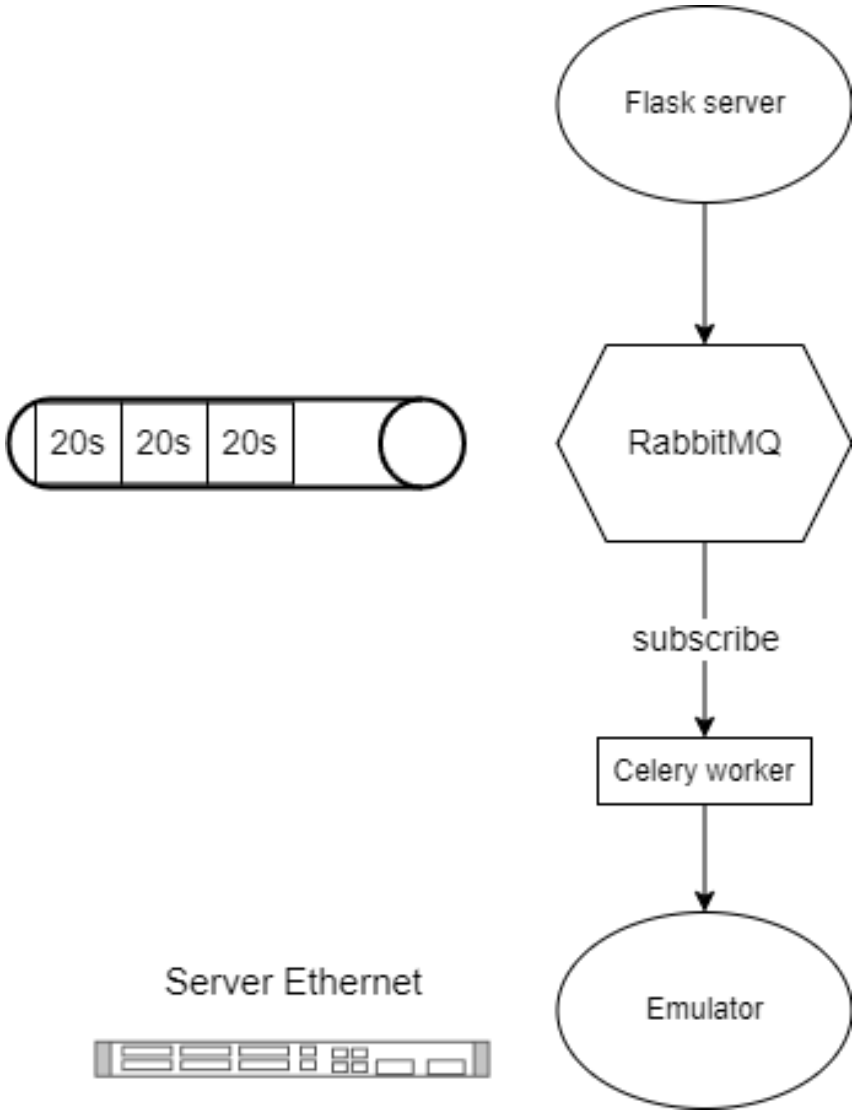
- Практическое обучение компьютерным сетям требует дорогостоящего оборудования, такого как коммутаторы Cisco.
- После ухода Cisco из России возникли сложности с приобретением оборудования.
- Эмуляторы, такие как Cisco Packet Tracer, имеют ограничения и недостатки.
- В качестве альтернативы на нашем факультете используется эмулятор **Miminet**, но при высокой нагрузке время ожидания эмуляции значительно увеличивается.

Обзор существующей архитектуры Miminet

- все эмуляции выполняются в одном контейнере Docker.
- Используются виртуальные узлы и Ethernet-интерфейсы сервера.



Обзор существующей архитектуры Miminet



Ограничения текущей реализации

- **Масштабируемость**: Использование одного контейнера для всех эмуляций ограничивает производительность при увеличении количества узлов.
- **Использование ресурсов**: Недостаточная эффективность при многозадачности и ограниченное использование многопроцессорных ресурсов.
- **Сетевые ограничения**: Ограниченная изоляция сетевых стеков между контейнерами, что снижает гибкость и может приводить к конфликтам.

Постановка задачи

Целью является оптимизация и распределённое выполнение сетевых эмуляций в **Miminet** с использованием Docker для улучшения производительности и масштабируемости системы

Задачи:

- Разработка архитектуры распределённого выполнения эмуляций с использованием нескольких контейнеров Docker.
- Проектирование и внедрение системы управления задачами и балансировки нагрузки для эффективного распределения ресурсов.
- Экспериментальное тестирование предложенного подхода на реальных данных и в реальных образовательных условиях.
- Валидация результатов с целью подтверждения улучшения производительности и сокращения времени ожидания для студентов.

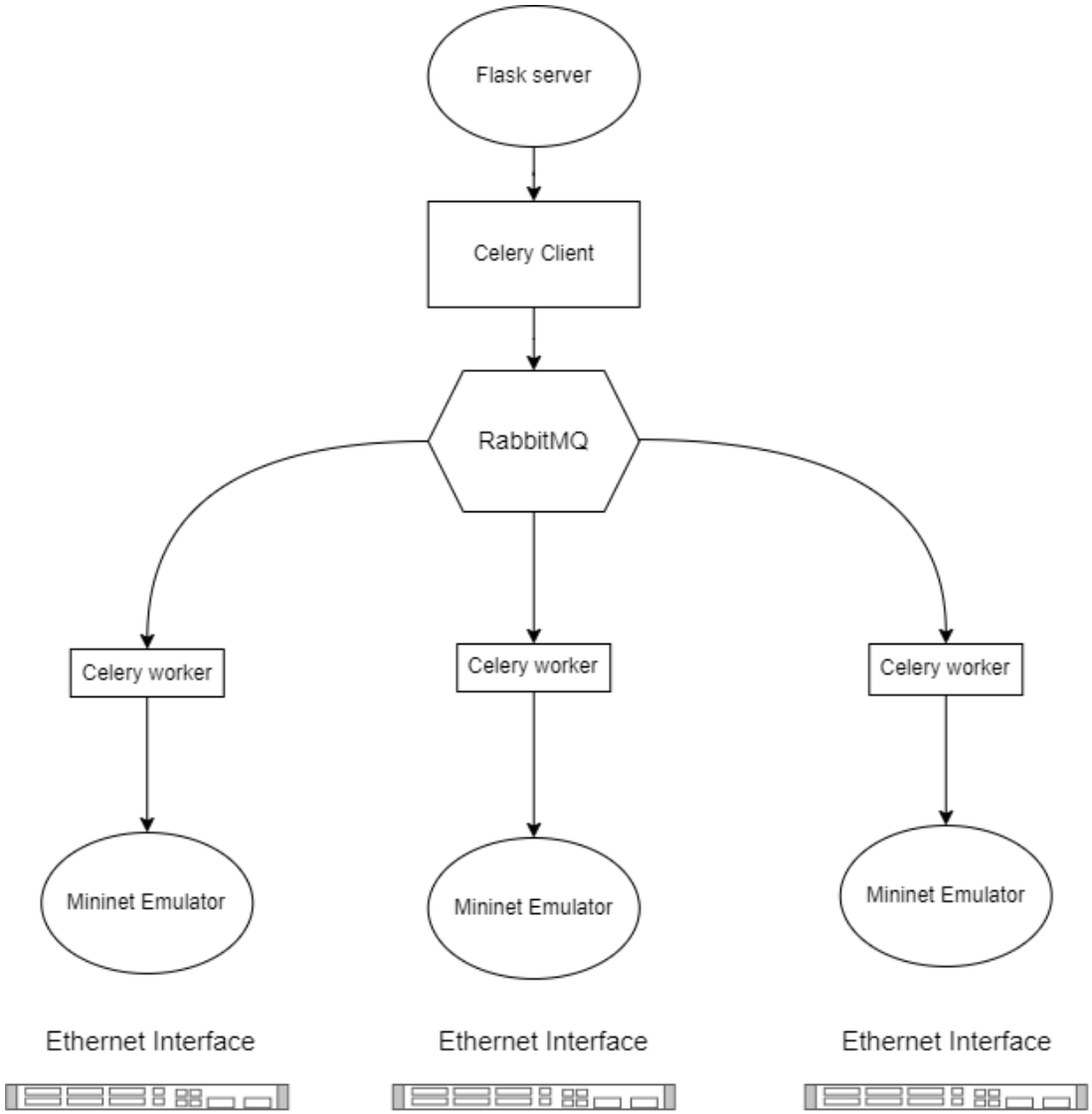
Решение проблемы

- **Распределение нагрузки**: Использование нескольких Docker-контейнеров для распределения вычислительных ресурсов между эмуляциями, что улучшает масштабируемость системы.
- **Многопроцессорная обработка**: Эффективное использование доступных ядер процессора для параллельной обработки задач, что повышает производительность.
- **Динамическое масштабирование**: Автоматическое добавление контейнеров в зависимости от нагрузки, что позволяет снизить время ожидания студентов и увеличить пропускную способность системы.

Архитектура решения

- Система основана на распределённых Docker-контейнерах, где каждый контейнер выполняет отдельную симуляцию.
- **Mininet** используется для изоляции эмуляций, что позволяет эффективно управлять ресурсами.
- **Celery + RabbitMQ** используются для распределения задач и балансировки нагрузки между контейнерами.
- **Многопроцессорность**: каждый контейнер может использовать несколько ядер процессора для параллельного выполнения эмуляций.
- **Динамическое масштабирование**: при увеличении нагрузки автоматически добавляются новые контейнеры для улучшения пропускной способности.

Архитектура решения (визуализация)



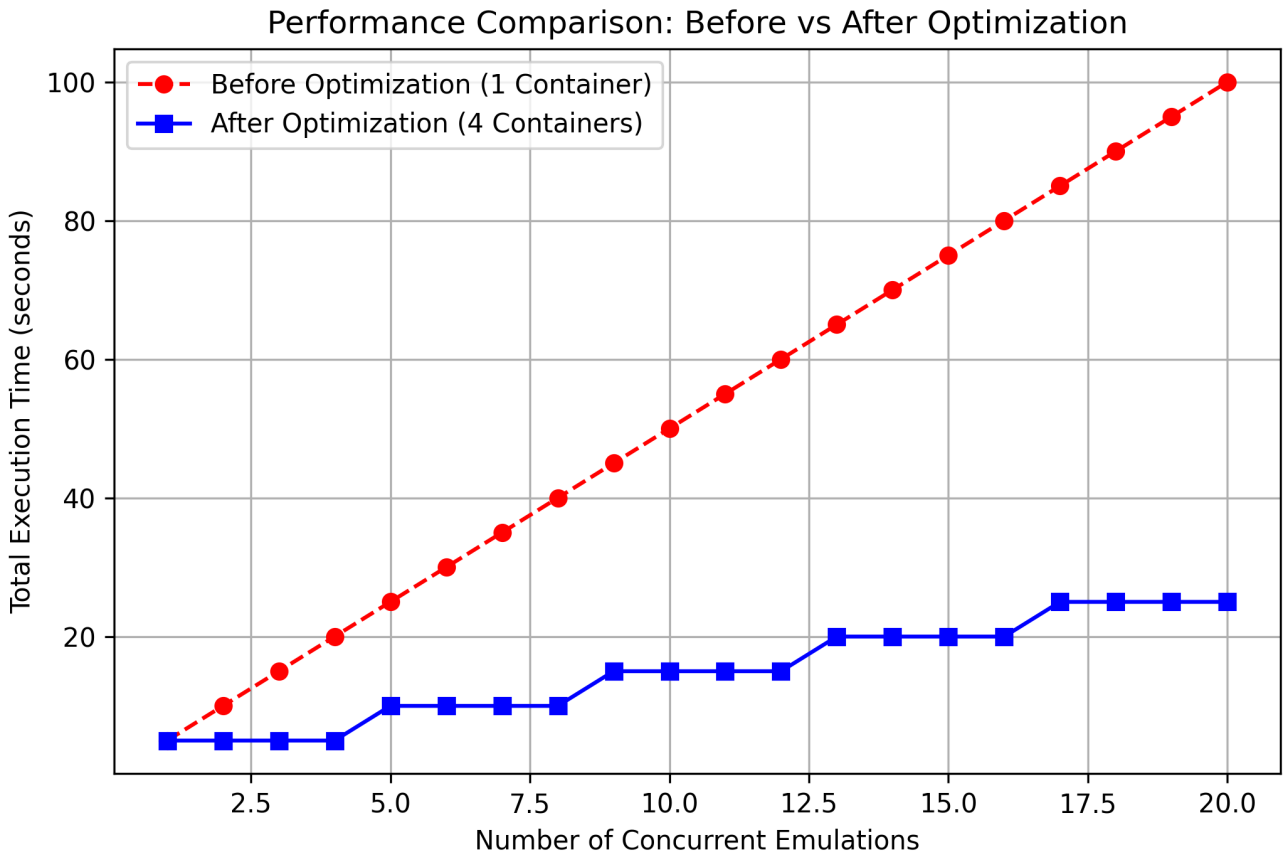
Проектирование решения

- Архитектура решения основана на **распределённом выполнении** эмуляций с использованием нескольких Docker-контейнеров.
- Используется **Celery** и **RabbitMQ** для распределения задач между рабочими контейнерами.
- Количество контейнеров фиксировано, но позволяет выполнять несколько эмуляций одновременно, увеличивая общую пропускную способность.
- Улучшена утилизация процессорных ресурсов за счёт балансировки нагрузки между контейнерами.

Экспериментальное тестирование

- **Цель**: оценить влияние новой архитектуры на производительность и масштабируемость платформы **Miminet**.
- **Методика**:
  - Сравнение двух версий: одноконтейнерной и многоконтейнерной.
  - Запуск серии сетевых эмуляций (20 задач подряд).
  - Время выполнения каждой эмуляции измерялось с помощью встроенного Python-кода:
    - ★ `start = time.time()` до начала задачи;
    - ★ `end = time.time()` после завершения;
    - ★ разница фиксировалась в логах контейнера.
- **Метрики**:
  - Время ожидания запуска задачи.
  - Общее количество параллельно выполняемых задач.
  - Нагрузка на CPU и использование ресурсов.
- **Результаты**:
  - В многоконтейнерной архитектуре удалось запустить до 4 эмуляций одновременно без увеличения времени ожидания.
  - Среднее время ожидания сократилось с **50 секунд** до **12 секунд**.
  - Использование CPU стало более равномерным и эффективным.

Сравнение производительности



Валидация

- Проведены тесты на **производительность и масштабируемость**.
- **Метрики**:
  - Общее количество одновременно выполняемых эмуляций увеличено.
  - Нагрузка на CPU распределяется более равномерно между ядрами, что улучшает общую производительность системы.
  - Среднее время ожидания запуска эмуляции для студентов сократилось.
- **Вывод**: Оптимизированная архитектура позволяет запускать больше эмуляций параллельно, что снижает нагрузку на сервер и сокращает время ожидания студентов.

Выводы

- Предложенный подход значительно улучшает масштабируемость и производительность **Miminet**.
- Распределение нагрузки между контейнерами позволяет эффективно использовать ресурсы и снизить время ожидания.
- Решение делает **Miminet** более удобным и эффективным инструментом для обучения компьютерным сетям.
- В будущем можно расширить функциональность для поддержки ещё более сложных сетевых топологий и сценариев.