**Методическое пособие по проекту robots-fog-mini-sim**

**Моделирование задержек в распределённой системе «Умный дом»**

**Введение**

**Цель лабораторной работы:** понять принципы работы распределённых систем, научиться моделировать и анализировать задержки передачи данных в IoT-системах на примере «умного дома».

**Распределённая система** — это совокупность нескольких устройств или компьютеров, работающих вместе через сеть для достижения общей цели. В нашем случае такой системой является «умный дом» с несколькими компонентами: **робот-датчик**, **робот Fog-обработчик**, **робот-курьер**, **смартфон** и **ноутбук-сервер**.

С помощью симуляции мы сможем:

* Проследить путь данных от датчика до сервера
* Измерить **сквозную задержку** (end-to-end delay)
* Проанализировать **буфер смартфона**
* Экспериментировать с параметрами системы

**Необходимые навыки:** базовое знакомство с Python, понимание принципов сетевого взаимодействия.

**2. Архитектура системы и подготовка к работе**

**2.1. Компоненты системы**

Наша распределённая система состоит из пяти основных компонентов:

* **Робот-датчик** — собирает данные из окружающей среды (температура, движение, свет)
* **Робот Fog-обработчик** — обрабатывает данные локально, рядом с датчиком
* **Робот-курьер** — доставляет обработанные данные дальше
* **Смартфон** — принимает сообщения и складывает их в буфер
* **Ноутбук-сервер** — центральный узел для хранения и анализа данных

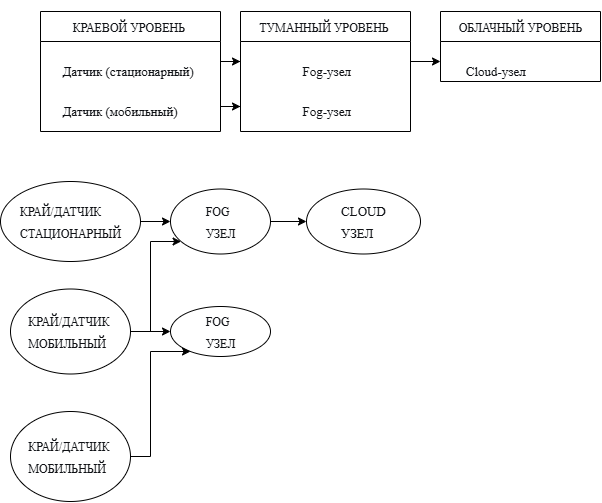


Рисунок 1 - Схема архитектуры системы

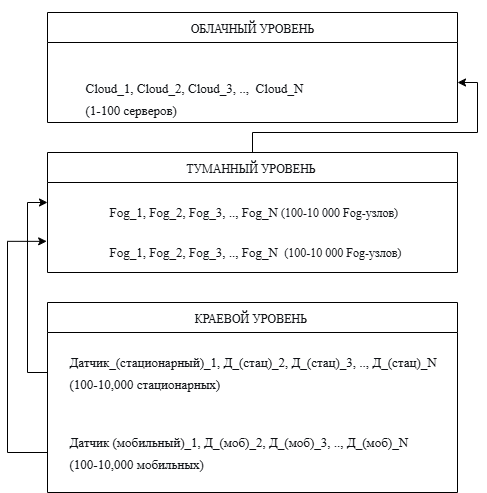


Рисунок 2 – Архитектура эталонной системы

**2.2. Подготовка рабочей среды**

1. **Скачайте проект** с GitHub репозитория robots-fog-mini-sim  
   [GitHub - annfridreksen/Robots-fog-mini-sim: Modelling of robots-fog-mini-sim](https://github.com/annfridreksen/Robots-fog-mini-sim)

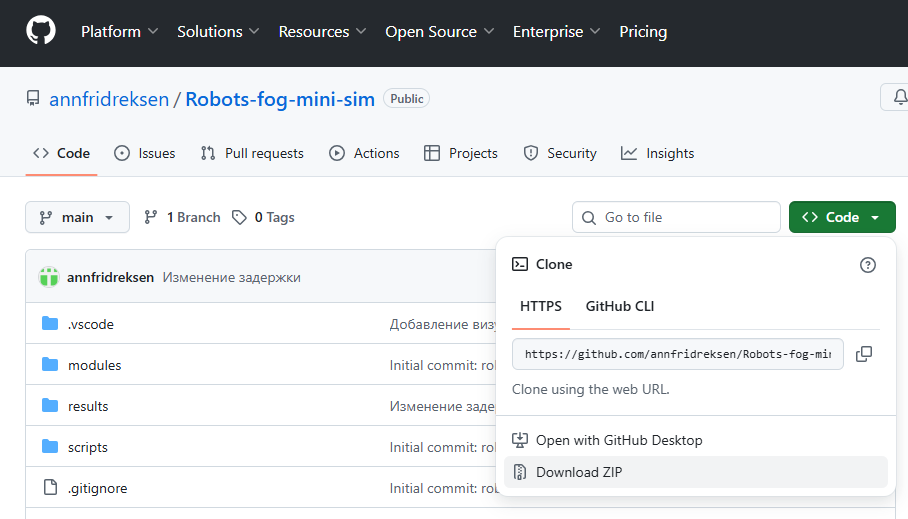


Рисунок 2 – Скачивание репозитория в zip

1. **Установите зависимости:**

>>> pip install -r requirements.txt

1. **Основные файлы проекта:**

* *viz\_phone\_pipeline\_bilingual.py* — главный файл симуляции
* *viz\_phone\_pipeline.py* — упрощённая версия
* *system\_diagram.png* — схема системы

**3. Основы моделирования задержек**

**3.1. Типы задержек в системе**

В распределённой системе возникают следующие типы задержек:

* **Сетевая задержка** — время передачи данных между узлами
* **Задержка обработки** — время обработки данных на каждом узле
* **Задержка очереди** — время ожидания в буфере смартфона

**3.2. Сквозная задержка (End-to-End Delay)**

**Сквозная задержка** — это сумма всех задержек на пути от источника (датчика) до конечного получателя (сервера или пользователя). Формула расчета:

Сквозная\_задержка = Задержка\_датчика + Задержка\_Fog + Задержка\_курьера + Очередь\_смартфона

**4. Практическая работа: Анализ задержек**

**4.1. Запуск базовой симуляции**

1. Откройте файл *viz\_phone\_pipeline\_bilingual.py*
2. Запустите симуляцию:

>>>python viz\_phone\_pipeline\_bilingual.py

1. Проанализируйте полученные графики:

* График сквозной задержки по задачам
* График динамики буфера смартфона

**4.2. Анализ кода моделирования**

Изучите функцию *simulate()* в основном файле:

def simulate(n\_tasks=30, seed=7):

random.seed(seed)

*# Задержки для каждого компонента (в миллисекундах)*

sensor = [random.randint(20, 60) for \_ in range(n\_tasks)]

fog = [random.randint(30, 80) for \_ in range(n\_tasks)]

courier = [random.randint(10, 40) for \_ in range(n\_tasks)]

*# Расчёт сквозной задержки*

latencies = [s + f + c for s, f, c in zip(sensor, fog, courier)]

*# Моделирование буфера смартфона*

read\_interval\_ms = 120

time = 0

buffer\_sizes = []

buf = 0

*# ... (логика обработки очереди)*

**Ключевые параметры:**

* n\_tasks — количество сообщений для симуляции
* read\_interval\_ms — интервал чтения сообщений смартфоном
* Диапазоны задержек для каждого компонента

**5. Эксперименты с параметрами системы**

**5.1. Эксперимент 1: Оптимизация Fog-узла**

**Цель:** определить влияние задержки Fog-обработчика на общую производительность.

**Шаги выполнения:**

1. Найдите в коде строку с задержкой Fog:

fog = [random.randint(30, 80) for \_ in range(n\_tasks)]

1. Измените параметры на более оптимизированные:

fog = [random.randint(10, 40) for \_ in range(n\_tasks)]

1. Запустите симуляцию и запишите результаты:

* Средняя сквозная задержка
* 95-й перцентиль задержки
* Максимальный размер буфера

1. Сравните с исходными показателями

**Ожидаемый результат:** уменьшение сквозной задержки на 25-30%.



Рисунок 3 – Оптимизированная сквозная задержка Датчика

**5.2. Эксперимент 2: Настройка буфера смартфона**

**Цель:** исследовать влияние частоты чтения сообщений на размер буфера.

**Шаги выполнения:**

1. Найдите параметр *read\_interval\_ms* в функции *simulate()*
2. Измените значение:

* Для ускорения обработки: *read\_interval\_ms = 60*
* Для замедления обработки: *read\_interval\_ms = 200*

1. Запустите симуляцию для каждого случая
2. Проанализируйте график буфера смартфона

**Ожидаемый результат:** при уменьшении интервала чтения буфер будет расти медленнее или чаще опустошаться.

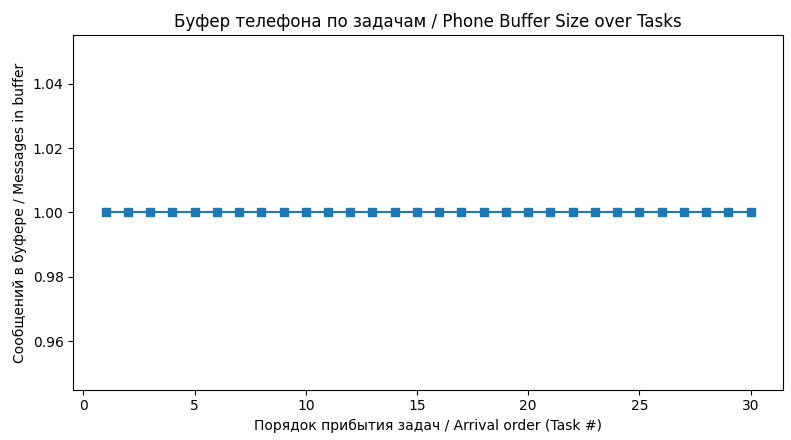
****

Рисунок 4 – Оптимальное значение для буфера телефона по задачам

**5.3. Эксперимент 3: Анализ джиттера**

**Цель:** понять природу variations в результатах симуляции.

**Шаги выполнения:**

1. Запустите симуляцию 5 раз с одинаковыми параметрами
2. Запишите среднюю задержку для каждого запуска
3. Рассчитайте стандартное отклонение
4. Проанализируйте разброс значений

**Ожидаемый результат:** разброс результатов между запусками: ±5-10%

**6. Работа с эталонной моделью распределённой системы**

**6.1. Знакомство с расширенной моделью**

Для более детального изучения распределённых систем используйте файл viz\_cloud\_fog\_edge\_pipeline.py — эталонную модель системы "Край → Туман → Облако". Эта модель позволяет:

* Моделировать реалистичную архитектуру IoT-системы
* Анализировать задержки на всех уровнях системы
* Исследовать динамику очередей на Fog-узлах
* Сравнивать производительность разных типов устройств

**6.2. Подготовка к работе**

1. **Добавьте файл в проект:** поместите viz\_cloud\_fog\_edge\_pipeline.py в папку с проектом robots-fog-mini-sim
2. **Запустите эталонную модель:**

>>>python viz\_cloud\_fog\_edge\_pipeline.py

**6.3. Структура эталонной модели**

Изучите основные компоненты модели:

* DistributedSystemSimulator — класс для создания распределённой системы
* simulate\_ethernet\_architecture\_custom() — функция симуляции архитектуры
* analyze\_performance() — анализ метрик производительности
* plot\_comprehensive\_results() — визуализация результатов

**6.4. Настройка параметров системы**

Для экспериментов с эталонной моделью найдите функцию simulate\_custom\_config() и измените параметры:

def simulate\_custom\_config():

"""Функция для быстрой настройки конфигурации системы"""

*# НАСТРОЙКА ПАРАМЕТРОВ СИСТЕМЫ*

CONFIG = {

'edge\_devices': 300, *# ↦ Количество краевых устройств (100-10000)*

'fog\_nodes': 25, *# ↦ Количество Fog-узлов (100-10000)*

'cloud\_servers': 8, *# ↦ Количество облачных серверов (1-100)*

'tasks': 200, *# ↦ Количество задач для симуляции*

'seed': 42 *# ↦ Seed для воспроизводимости результатов*

}

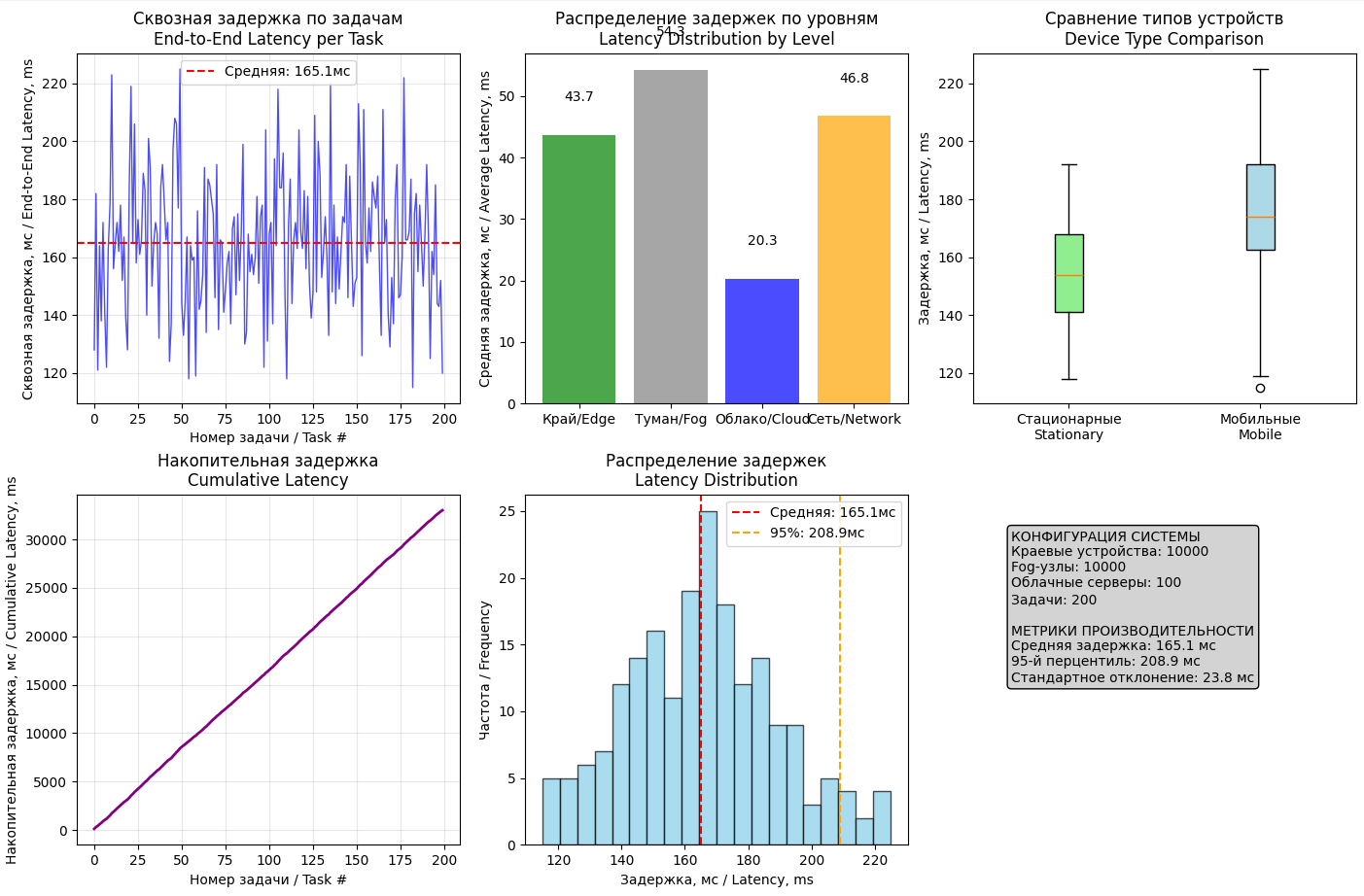


Рисунок 5 – Пример работы эталонной системы

**7. Практические эксперименты с эталонной моделью**

**7.1. Эксперимент 4: Масштабирование системы**

**Цель:** исследовать как изменение количества устройств влияет на производительность системы.

**Шаги выполнения:**

1. Откройте файл viz\_cloud\_fog\_edge\_pipeline.py
2. Найдите функцию simulate\_custom\_config()
3. Измените количество устройств:
   * Малая система: 100 edge, 10 fog, 3 cloud
   * Средняя система: 500 edge, 50 fog, 10 cloud
   * Крупная система: 2000 edge, 200 fog, 20 cloud
4. Запустите симуляцию для каждой конфигурации
5. Сравните показатели:
   * Среднюю сквозную задержку
   * Загрузку Fog-узлов
   * Распределение задержек по уровням

**Ожидаемый результат:** при увеличении масштаба системы задержки могут возрастать из-за нагрузки на Fog-узлы и сеть.

**7.2. Эксперимент 5: Анализ типов устройств**

**Цель:** сравнить производительность стационарных и мобильных устройств.

**Шаги выполнения:**

1. Запустите базовую симуляцию с параметрами по умолчанию
2. Проанализируйте график "Сравнение типов устройств"
3. Обратите внимание на разницу в задержках между стационарными и мобильными устройствами
4. Найдите в коде настройки для разных типов устройств:

python

*# В функции \_init\_edge\_devices():*

if device\_type == "мобильный":

processing\_range = (25, 70) *# Немного выше задержка*

network\_range = (8, 20) *# Менее стабильное соединение*

else:

processing\_range = (20, 60) *# Стабильная задержка*

network\_range = (5, 15) *# Стабильное соединение*

1. Измените параметры мобильных устройств на более оптимистичные и сравните результаты

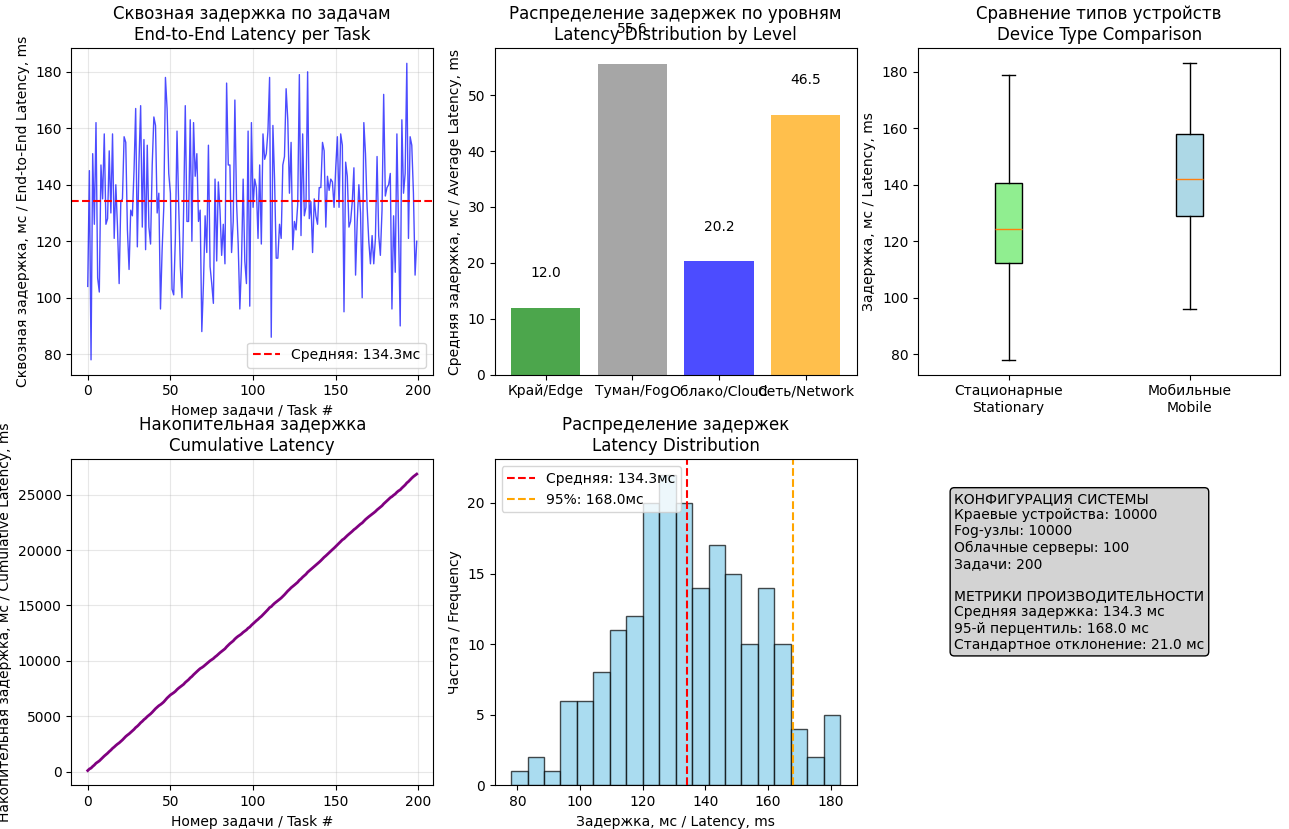


Рисунок 6 – Пример с уменьшенной задержкой

**Ожидаемый результат:** мобильные устройства показывают более высокую задержку из-за менее стабильного соединения.

**7.3. Эксперимент 6: Оптимизация очередей Fog-узлов**

**Цель:** исследовать влияние размера очереди на Fog-узлах на общую производительность.

**Шаги выполнения:**

1. Найдите в коде инициализацию Fog-узлов:

python

*# В функции \_init\_fog\_nodes():*

'queue\_capacity': 50, *# ↦ Измените этот параметр*

1. Протестируйте разные значения: 20, 50, 100, 200
2. Запустите симуляцию для каждого значения
3. Проанализируйте:
   * Задержки в очередях Fog-узлов
   * Количество переполнений очередей
   * Общую сквозную задержку

**Ожидаемый результат:** слишком маленькие очереди приводят к потерям задач, слишком большие — к росту задержек.