Санкт-Петербургский Политехнический Университет Петра Великого
Физико-Механический институт
Высшая школа прикладной математики и вычислительной физики

Отчет по лабораторной работе: Алгоритмы маршрутизации (OSPF)

Выполнил:
студент гр. 5040102/40201
Стрижкин Д.А.

Проверил:
доцент
Баженов А.Н.

Санкт-Петербург — 2026

1. Введение

В данной лабораторной работе исследуется работа протокола динамической маршрутизации с использованием алгоритма состояния каналов (Link State).

Цель работы:

1. Реализовать симуляцию сети, где каждый маршрутизатор работает как независимый агент.
2. Реализовать механизм автоматического обнаружения топологии сети.
3. Применить алгоритм Дейкстры для построения таблиц маршрутизации.
4. Проверить корректность передачи данных на различных топологиях.

Моделирование проводится на языке **Rust**. Для эмуляции распределенной сети используется многопоточность (`std::thread`), а для обмена сообщениями между маршрутизаторами — каналы (`std::sync::mpsc`).

Исходный код доступен в репозитории: https://github.com/denisstrizhkin/network-labs

2. Детали реализации

2.1. Протокол взаимодействия

Взаимодействие между узлами описывается перечислением `Message`.
Основные типы сообщений:

- **Hello(NodeId)**: Посылается соседям при старте для обнаружения связности.
- **GetNeighbors / SetNeighbors**: Используются выделенным маршрутизатором (DR) для сбора локальной информации от каждого узла.
- **SetTopology**: Рассылка полной карты сети от DR всем участникам.
- **Data**: Пакет данных, содержащий отправителя, получателя и трассировку пути (`path_trace`).

2.2. Алгоритм построения маршрутов

Каждый маршрутизатор хранит полную карту сети (`topology`). При получении обновленной топологии запускается **алгоритм Дейкстры**:

1. Инициализируются расстояния до всех узлов как «бесконечность», до самого себя — 0.

2. Используется очередь с приоритетом (`BinaryHeap`) для выбора ближайшего непосещенного узла.

3. Для каждого соседа текущего узла производится релаксация (обновление) расстояния.

4. Одновременно вычисляется таблица `next_hop`, указывающая, через какого соседа нужно отправить пакет для достижения конкретной цели.

2.3. Модель симуляции

Симуляция проходит в несколько этапов, координируемых функцией `run_simulation`:

1. **Инициализация**: Запускаются потоки-маршрутизаторы и поток-координатор (DR).

2. **Neighbor Discovery**: Узлы обмениваются `Hello` сообщениями.

3. **Topology Synchronization**: Координатор запрашивает списки соседей и рассылает объединенный граф сети.

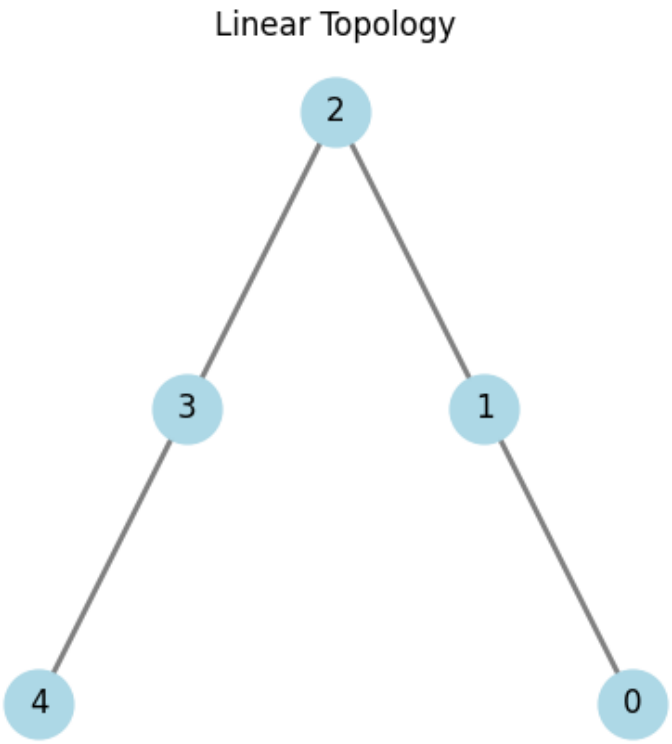
4. **Data Transmission**: После построения маршрутов отправляется тестовое сообщение. Маршрут отслеживается в поле `path_trace`.

3. Результаты моделирования

Было проведено тестирование на трех типовых топологиях. Во всех тестах вес каждого ребра принят равным 1.

3.1. Линейная топология

Топология: `0–1–2–3–4`.
Тест: Отправка сообщения от узла 0 к узлу 4.



Линейная топология

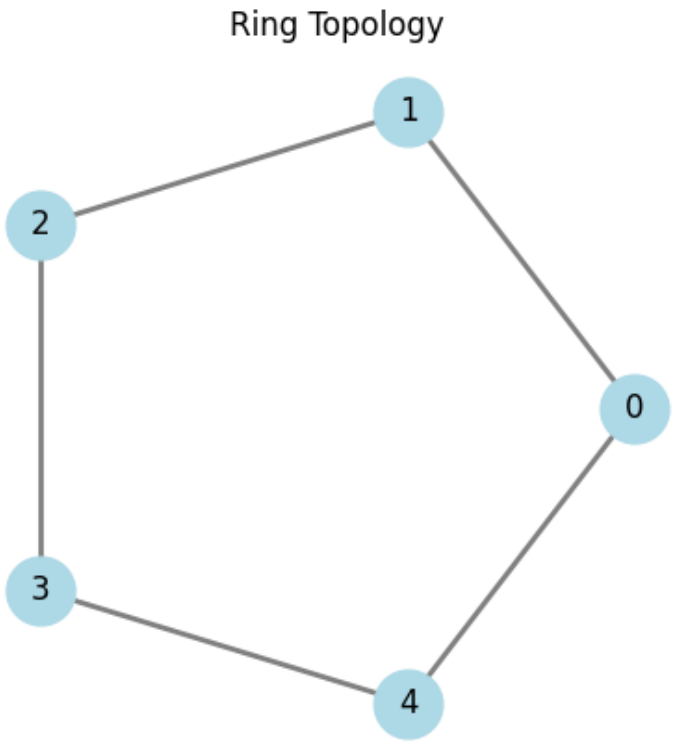
Результат:

```
=== Running Simulation: Linear Topology ===
--- Sending Message from 0 to 4 ---
[Router 4] received message from 0: [0, 1, 2, 3, 4]
```

Маршрут построен корректно, пакет прошел через все промежуточные узлы.

3.2. Кольцевая топология

Топология: `0-1-2-3-4-0`.
Тест: Отправка сообщения от узла 0 к узлу 2.



Кольцевая топология

В данной конфигурации существуют два пути: 1. `0 -> 1 -> 2` (длина 2) 2. `0 -> 4 -> 3 -> 2` (длина 3)

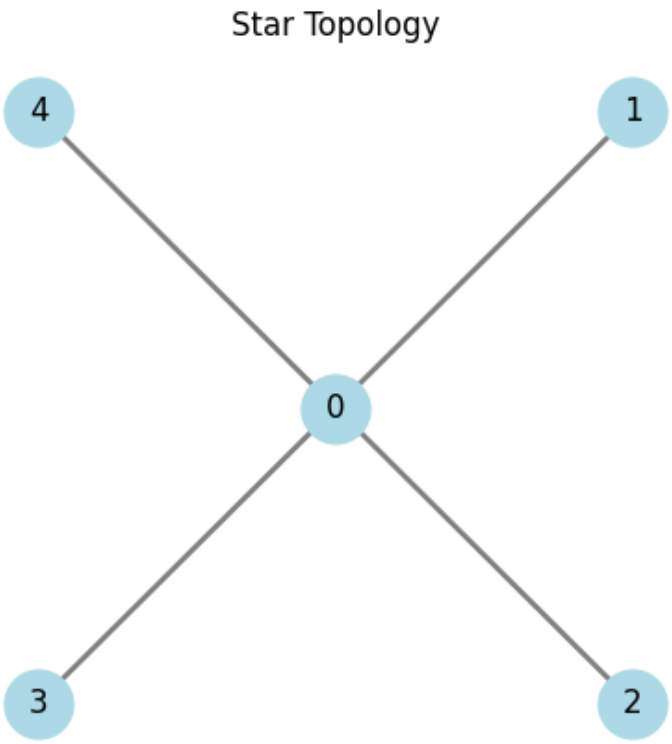
Результат:

```
=== Running Simulation: Ring Topology ===
--- Sending Message from 0 to 2 ---
[Router 2] received message from 0: [0, 1, 2]
```

Алгоритм верно выбрал кратчайший путь через узел 1.

3.3. Топология “Звезда”

Топология: Центр — узел 0, листья — 1, 2, 3, 4.
Тест: Отправка сообщения между листьями (от 4 к 3).



Топология Звезда

Результат:

```
=== Running Simulation: Star Topology ===
--- Sending Message from 4 to 3 ---
[Router 3] received message from 4: [4, 0, 3]
```

Поскольку прямой связи между листьями нет, пакет был маршрутизирован через центральный узел 0.

4. Заключение

В ходе работы была успешно программно реализована модель сети на базе протокола состояния каналов. Реализация на языке Rust с использованием легковесных потоков и каналов позволила наглядно продемонстрировать децентрализованную природу взаимодействия маршрутизаторов. Корректность работы алгоритма Дейкстры подтверждена тестами: во всех случаях выбирались оптимальные маршруты и обеспечивалась доставка данных получателю.