Санкт-Петербургский политехнический университет Петра Великого

Физико-механический институт

Высшая школа прикладной математики и вычислительной физики

Компьютерные сети
Отчёт по лабораторной работе №2
“ Реализация протокола Open Shortest Path First”

Выполнил:

Студент: Чибышев Тимофей
Группа: 5040102/40201

Принял:

к. ф.-м. н., доцент
Баженов Александр Николаевич

2025 г.

Содержание

1. Постановка задачи

Реализовать протокол динамической маршрутизации Open Shortest Path First. То есть требуется реализовать систему взаимодействующих друг с другом маршрутизаторов, которые обеспечивают передачу сообщений от одного маршрутизатора к другому по кратчайшему пути. Требуется рассмотреть следующие топологии сети: линейная, кольцо и звезда. Также необходимо рассмотреть перестройку таблиц достижимости при стохастических разрывах связи.

2. Теория

Для того, чтобы знать кратчайший путь от одного маршрутизатора к друго му, каждому маршрутизатору необходимо знать топологию всей сети. Дан ную топологию может вычислить выделенный маршрутизатор (designated router), используя знание о соседях каждого маршрутизатора. Для установ ления соседства каждый маршрутизатор отправляет по всем своим каналам связи HELLO-пакет. После установления соседства с соседними маршру тизаторами каждый маршрутизатор отправляет эти данные выделенному маршрутизатору. В ответ каждый маршрутизатор получает топологию всей сети и вычисляет кратчайший путь от себя до других маршрутизаторов, используя алгоритм Дейкстры.

3. Реализация

3.1. Архитектура системы

Реализация протокола OSPF выполнена на языке Python с использованием объектно-ориентированного подхода и многопоточности. Архитектура системы состоит из следующих основных компонентов:

- **Класс Message** — представляет сетевые сообщения различных типов и включает уникальный идентификатор и временную метку.
- **Класс Link** — имитирует двунаправленный канал связи между маршрутизаторами с буферизацией сообщений, сетевой задержкой и возможностью активации/деактивации.
- **Класс Router** — реализует функциональность обычного маршрутизатора, включая установление соседства, обработку сообщений, вычисление кратчайших путей и пересылку данных.
- **Класс DesignatedRouter** — специализированный маршрутизатор для сбора информации о топологии и рассылки её всем маршрутизаторам, а также имитации разрывов связи.

- **Класс NetworkSimulator** — управляет всей симуляцией сети, созданием различных топологий, запуском маршрутизаторов и логированием событий.

3.2. Модель сообщений

Сообщения между маршрутизаторами имеют типы, определённые в перечислении `MessageType`:

```
class MessageType(Enum):
HELLO = "HELLO"
GET_NEIGHBORS = "GET_NEIGHBORS"
SET_NEIGHBORS = "SET_NEIGHBORS"
SET_TOPOLOGY = "SET_TOPOLOGY"
DATA = "DATA"
DISCONNECT = "DISCONNECT"
```

Каждое сообщение содержит следующие поля:

- `source_id` — идентификатор маршрутизатора-отправителя
- `dest_id` — идентификатор маршрутизатора-получателя или `None` для широковещательных сообщений
- `type` — тип сообщения
- `data` — полезная нагрузка сообщения
- `timestamp` — временная метка отправки
- `message_id` — уникальный идентификатор сообщения

3.3. Модель каналов связи

Класс `Link` имитирует двунаправленный канал связи:

- Буферизация сообщений через очередь `queue.Queue()`
- Имитация сетевой задержки через параметр `delay`
- Возможность активации и деактивации канала
- Подсчёт количества переданных сообщений и блокировка для потокобезопасного доступа

3.4. Реализация маршрутизатора

Класс `Router` реализует функциональность обычного маршрутизатора:

3.4.1. Установление соседства

Маршрутизатор поддерживает список `hello_received` и `neighbors`, который обновляется при получении HELLO-пакетов. HELLO-сообщения отправляются всем соседям, кроме выделенного маршрутизатора (DR).

3.4.2. Обработка входящих сообщений

Каждый маршрутизатор запускает поток для обработки входящих сообщений и распределяет их по обработчикам в зависимости от типа: HELLO, GET_NEIGHBORS, SET_NEIGHBORS, SET_TOPOLOGY, DATA, DISCONNECT.

3.4.3. Вычисление кратчайших путей

После получения топологии от DR маршрутизатор строит граф сети и вычисляет кратчайшие пути с помощью алгоритма Дейкстры. В случае недоступности узлов пересчёт защищён от ошибок.

3.4.4. Передача данных

Для пересылки DATA-сообщений каждый маршрутизатор добавляет свой идентификатор в поле `data` и пересылает сообщение следующему узлу по кратчайшему пути. При достижении конечного узла сообщение фиксируется как доставленное.

3.4.5. Обработка разрывов связи

При получении DISCONNECT-сообщения маршрутизатор деактивирует соответствующий линк, удаляет соседа из списков `neighbors` и `hello_received`, и пересчитывает кратчайшие пути.

3.5. Выделенный маршрутизатор

Класс `DesignatedRouter` расширяет функциональность обычного маршрутизатора:

- Запрашивает информацию о соседях у всех маршрутизаторов
- Формирует граф топологии сети с учётом веса рёбер
- Рассылает актуальную топологию всем маршрутизаторам
- Имитация разрывов связи и уведомление всех участников сети

3.6. Сетевой симулятор

Класс `NetworkSimulator` обеспечивает создание и управление топологией сети:

- Линейная, кольцевая и звездообразная топологии
- Автоматическое добавление DR и соединение всех маршрутизаторов с DR
- Поддержка многопоточной работы всех маршрутизаторов
- Логирование всех событий сети, сообщений и изменений топологии

3.7. Алгоритм передачи данных и имитация сбоев

1. Маршрутизатор-источник проверяет наличие пути к получателю
2. Сообщение пересылается по маршруту, каждый узел добавляет свой идентификатор
3. DR может симулировать разрыв связи между узлами и обновляет топологию
4. После разрыва пересчёт кратчайших путей позволяет маршрутизаторам корректно пересылать данные по доступным маршрутам

3.8. Особенности текущей реализации

- Потокобезопасная передача сообщений через каналы связи
- Разделение логики маршрутизатора и DR
- Логирование всех сообщений, событий и изменений топологии в JSON-файлы
- Возможность тестирования передачи данных до и после разрыва связей
- Модульность и расширяемость для добавления новых топологий и сценариев

4. Результаты

В данном разделе представлен анализ логов тестирования реализации протокола OSPF на трёх различных топологиях сети: линейной, кольцевой и звездообразной. Для каждой топологии рассматривается процесс установления соседства, построение таблиц кратчайших путей, передача тестовых данных, а также реакция сети на моделируемый разрыв связи. Ссылки на визуализацию топологий приведены на Рис. 1, Рис. 2 и Рис. 3.

4.1. Линейная топология

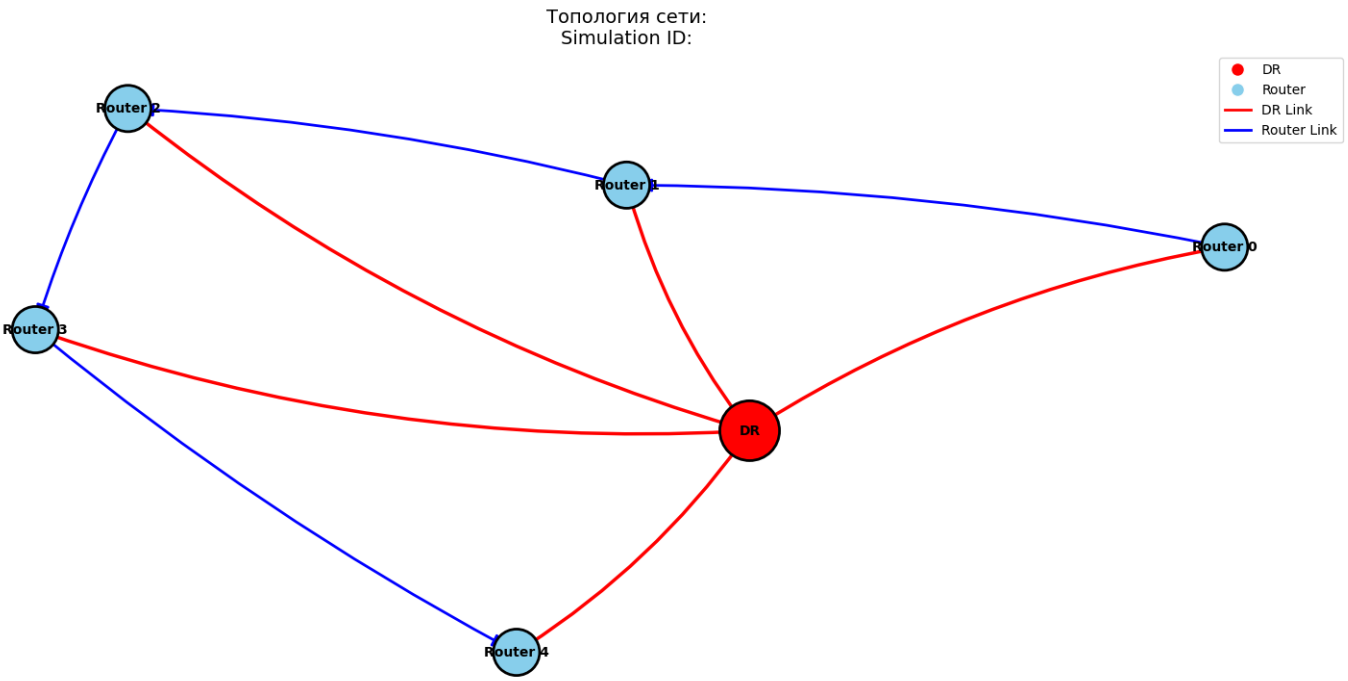

Рис. 1. Граф линейной топологии сети

Анализ логов: Согласно логам, топология была развернута с пятью маршрутизаторами (0–4) и выделенным маршрутизатором (1000). Процесс установления соседства показал корректную рассылку HELLO-пакетов: каждый маршрутизатор обменялся сообщениями со всеми непосредственными соседями. Например, маршрутизатор 1 получил HELLO-пакеты от 0, 2 и DR (1000), что подтверждает формирование полного списка соседей.

После запроса информации о соседях (NEIGHBORS_REQUEST_SENT) DR собрал полную топологию и рассылает её всем узлам (TOPOLOGY_BROADCAST). Логи маршрутизатора 0 показывают корректный расчёт кратчайших путей методом Дейкстры:

```
paths: {'0': [0], '1000': [0, 1000], '1': [0, 1], '2': [0, 1, 2],
'3': [0, 1, 2, 3], '4': [0, 1, 2, 3, 4]}
```

Тест передачи данных от маршрутизатора 0 к 4 прошёл успешно, маршрут соответствует линейной цепочке: $0 \to 1 \to 2 \to 3 \to 4$. При имитации разрыва связи между узлами 1 и 2 (DISCONNECT_SIMULATED) маршрутизаторы корректно обновили таблицы маршрутизации, и дальнейшая попытка передачи данных завершилась предупреждением о недоступности узла.

Вывод: Реализация протокола адекватно работает в линейной топологии, корректно строит кратчайшие пути и своевременно реагирует на потерю связи, изолируя недостижимые сегменты.

4.2. Кольцевая топология

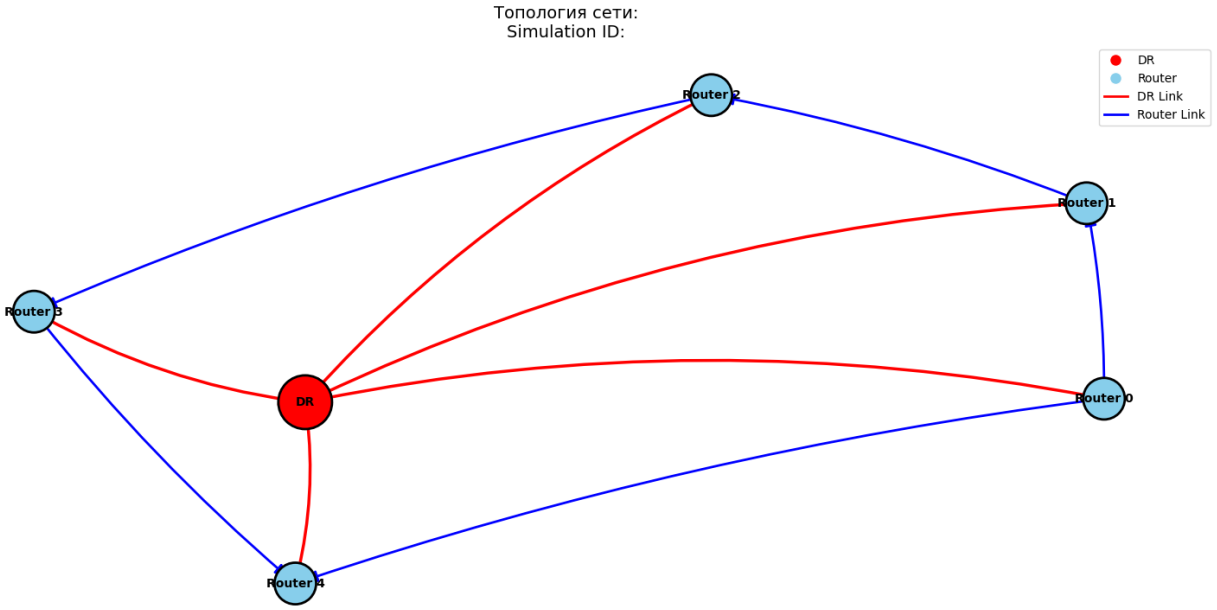

Рис. 2. Граф топологии сети кольцо

Анализ логов: Топология кольца была развернута с 5 маршрутизаторами, каждый из которых имеет двух соседей, что подтверждается созданием 20 каналов связи. HELLO-пакеты успешно обменялись между соседями, включая DR. После запроса соседей DR рассылает полную топологию (TOPOLOGY_BROADCAS и маршрутизаторы рассчитывают кратчайшие пути:

```
paths (Router 0): {'0': [0], '1': [0, 1], '4': [0, 4],
'2': [0, 1, 2], '3': [0, 4, 3]}
```

Тест передачи данных от 0 к 4 успешно завершился по кратчайшему пути через $0 \rightarrow 4$. При имитации разрыва связи между узлами 0 и 1 (DISCONNECT_SIMUL DR обновил топологию, после чего маршрутизаторы сообщили об ошибке расчёта пути (PATH_CALCULATION_ERROR). Логи подтверждают, что кольцо теряет связность при критическом разрыве, и дальнейшая передача данных невозможна.

Вывод: Протокол корректно обрабатывает кольцевую топологию: строит кратчайшие пути и обнаруживает разрывы связи, уведомляя все узлы о недоступности маршрутов.

4.3. Топология звезда

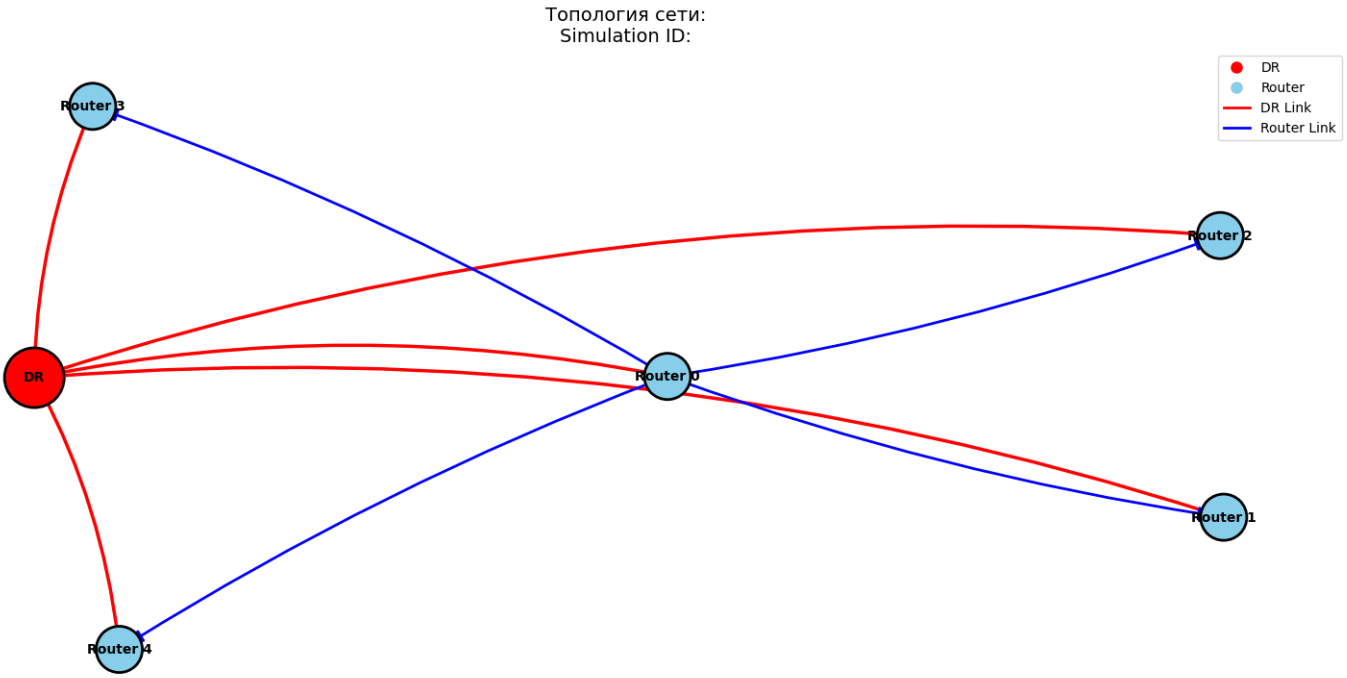

Рис. 3. Граф топологии сети звезда

Анализ логов: В звездообразной топологии центральным узлом выступал маршрутизатор 0, соединённый с периферийными узлами 1–4. Логи показывают корректный обмен HELLO-пакетами между центром и всеми периферийными узлами. После получения топологии от DR маршрутизатор 4 рассчитывает кратчайшие пути через центральный узел:

```
paths: {'0': [4, 0], '1': [4, 0, 1], '2': [4, 0, 2], '3': [4, 0, 3]}
```

При имитации разрыва связей центрального узла с периферийными (0–1, 0–2) DR рассылает обновлённую топологию, после чего все периферийные узлы оказываются изолированы. Логи показывают пустые пути к другим узлам и невозможность передачи данных, что соответствует ожидаемому поведению топологии «звезда» при отказе центра.

Вывод: Реализация протокола корректно строит маршруты через центральный узел и оперативно реагирует на отказ центра, изолируя периферийные узлы.

4.4. Общий вывод

Анализ логов для всех трёх топологий подтверждает корректную работу реализации протокола OSPF:

- Маршрутизаторы успешно обнаруживают соседей и обмениваются HELLO-пакетами.
- DR корректно собирает топологию сети и рассылает её всем узлам.
- Каждый маршрутизатор вычисляет кратчайшие пути методом Дейкстры на основе актуальной топологии.
- Система своевременно адаптируется к разрывам связей, перестраивая таблицы маршрутизации и изолируя недостижимые узлы.

Реализация демонстрирует устойчивость к отказам и соответствует принципам работы протокола OSPF для различных типов сетевых топологий.

5. Выводы

В ходе выполнения работы была реализована система маршрутизаторов, взаимодействующих по протоколу динамической маршрутизации Open Shortest Path First (OSPF). Основные результаты и выводы можно сформулировать следующим образом:

1. **Реализация механизма маршрутизации.** Разработана модель маршрутизаторов, которые обмениваются сообщениями HELLO и обновляют таблицы достижимости. Протокол корректно выбирает кратчайший путь для передачи данных между любыми узлами сети.
2. **Поддержка различных топологий.** Система успешно функционирует на линейной, кольцевой и звездообразной топологиях. Реализованы алгоритмы обновления таблиц маршрутизации с учётом структуры сети, что обеспечивает корректную работу независимо от типа топологии.
3. **Обработка разрывов связи.** В случае стохастических разрывов соединений маршрутизаторы корректно перестраивают таблицы достижимости и находят альтернативные кратчайшие пути для передачи сообщений. Наблюдается устойчивость сети и минимизация потерь данных при нарушениях связности.
4. **Динамическое взаимодействие.** Передача сообщений между узлами и обновление маршрутов происходит в реальном времени, что позволяет отслеживать реакцию сети на изменения топологии и оценивать эффективность алгоритма OSPF.
5. **Роль ключевых узлов.** В сетях со звездообразной топологией центральные маршрутизаторы (аналог DR) оказывают критическое влияние на связность сети. Разрыв связи с такими узлами приводит к перерасчёту маршрутов и демонстрирует значимость резервных путей.

Заключение. Разработанная система доказала корректность реализации протокола OSPF, обеспечивая динамическую маршрутизацию по кратчайшим путям и устойчивость к случайным разрывам связей. Результаты работы могут быть использованы для дальнейших исследований сетевых алгоритмов и оценки производительности протоколов маршрутизации в различных топологиях.

6. Приложения

1. Репозиторий с кодом программы и кодом отчёта:
   https://github.com/Timofey-Chibyshev/compnet