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: Зокиров Кодиржон
Группа: 5040102/40201

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

2024 г.

1. Постановка задачи

Разработать и реализовать **протокол динамической маршрутизации OSPF (Open Shortest Path First)**. Необходимо создать **модель сети из взаимодействующих маршрутизаторов**, которые автоматически определяют **кратчайший путь** для передачи сообщений между узлами на основе актуальной топологии и метрик каналов.
Требуется исследовать поведение протокола в следующих топологиях:

- **Линейная**
- **Кольцевая**
- **Звёздная**

Дополнительно: проанализировать **процесс реконвергенции таблиц маршрутизации** при **случайных (стохастических) разрывах каналов связи**, включая скорость обновления маршрутов и устойчивость системы к сбоям.

2. Теория

Для построения **кратчайшего пути** между маршрутизаторами каждый узел должен обладать **полной информацией о топологии сети**. Эта информация формируется с помощью **выделенного маршрутизатора (Designated Router, DR)**, который собирает и агрегирует данные о связности сети.
Процесс обмена информацией происходит в несколько этапов:

1. **Обнаружение соседей** Каждый маршрутизатор периодически рассылает **HELLO-пакеты** по всем своим интерфейсам. При получении HELLO от другого узла устанавливается **двунаправленное соседство (adjacency)**.
2. **Сбор данных о локальных связях** После подтверждения соседства маршрутизаторы обмениваются **LSA-пакетами (Link State Advertisements)**, содержащими:
    - идентификаторы соседей;
    - метрики (стоимости) каналов;
    - состояние интерфейсов.
3. **Передача данных выделенному маршрутизатору** Все маршрутизаторы направляют полученные LSA **выделенному маршрутизатору (DR)**, который выступает в роли **централизованного сборщика топологии**.
4. **Формирование и распространение глобальной топологии** DR объединяет все LSA в **единую базу данных состояния связей (LSDB)** и рассылает её всем участникам области OSPF.
5. **Вычисление кратчайших путей** Каждый маршрутизатор, получив **актуальную LSDB**, применяет **алгоритм Дейкстры** для построения **SPF-дерева (Shortest Path First)** с корнем в себе. На основе этого дерева формируется **таблица маршрутизации** с указанием **следующего перехода (next hop)** до каждого достижимого узла.

Таким образом, **децентрализованное обнаружение соседей** и **централизованная координация через DR** обеспечивают **сходимость топологии** и **оптимальную маршрутизацию** в динамически изменяющейся сети.

3. Реализация

Для **моделирования маршрутизаторов** в системе запускается **отдельный процесс** (multiprocessing.Process) на каждый узел сети. Такой подход гарантирует **полную изоляцию состояний** и позволяет симулировать **реальное параллельное выполнение** протокола OSPF.

Каналы связи реализованы через **двунаправленную пару классов**:

- LinkOutput — **выходной интерфейс**, через который маршрутизатор **отправляет пакеты** (HELLO, LSA, данные) в канал;
- LinkInput — **входной интерфейс**, принимающий пакеты **для маршрутизатора-получателя**.

Каждое соединение, определённое в предыдущей лабораторной работе через объект Connection, преобразуется в **пару** (LinkOutput ↔ LinkInput), обеспечивая **направленную передачу** в обе стороны.

Внутри каждого экземпляра LinkOutput и LinkInput создаётся **отдельный поток** (threading.Thread). Эти потоки отвечают за:

- **очередь отправки/приёма**,
- **таймеры подтверждений**,
- **повторную передачу** по протоколу надёжной доставки.

По умолчанию используется протокол Selective Repeat, но пользователь может переключиться на **Go-Back-N** через параметр запуска.

Управление параллелизмом:

- **Процессы и межпроцессные очереди** — модуль multiprocessing
- **Потоки и синхронизация внутри канала** — модуль threading

Такая архитектура позволяет **точно воспроизвести поведение реальной сети**, включая **асинхронную передачу**, **конкуренцию за канал** и **независимую обработку событий** на каждом узле.

Каждый пакет в системе — это объект класса Message, состоящий из четырёх полей:

- **ID отправителя** — уникальный идентификатор маршрутизатора, который сформировал сообщение;
- **ID получателя** — адрес конечного узла (остаётся **незаполненным** при передаче **HELLO-пакетов**, поскольку на этапе обнаружения соседи ещё не определены);
- **тип пакета** — указывает на его функциональное назначение;
- **содержимое** — сами передаваемые данные.

Перечень возможных типов сообщений включает:

- **HELLO** — используется при рассылке **HELLO-пакетов** для обнаружения соседних маршрутизаторов.
- **GET_NEIGHBORS** — запрос от **выделенного маршрутизатора (DR)** к обычному узлу с требованием передать список его соседей.
- **SET_NEIGHBORS** — применяется в двух сценариях:
  1. **От маршрутизатора к DR** — передача списка **входящих соседей** (кто может присылать ему пакеты);
  2. **От DR к маршрутизатору** — передача списка **исходящих соседей** (кому он может отправлять сообщения).
- **SET_TOPOLOGY** — рассылка **полной топологии сети** от выделенного маршрутизатора всем участникам.
- **DATA** — передача **пользовательских данных**, не относящихся к протоколу OSPF (тестовый трафик).
- **DISCONNECT** — команда от DR к маршрутизатору, имитирующая **разрыв связи** с указанным узлом.

Каждый маршрутизатор в системе реализован как объект класса Router, а выделенный маршрутизатор — как экземпляр класса DesignatedRouter.

Топология сети моделируется в виде ориентированного графа с использованием библиотеки networkx.

Процесс установления соседства проходит следующим образом:

1. Каждый маршрутизатор рассылает **HELLO-пакеты** через **все свои выходные интерфейсы** (LinkOutput). В каждом таком пакете содержится:
   - **время отправки**,
   - **идентификатор интерфейса**, по которому пакет уходит.
2. Маршрутизатор собирает **входящие HELLO-пакеты** через все свои входные интерфейсы (LinkInput). После получения всех ожидаемых сообщений и **дополнительного запроса от выделенного маршрутизатора** (GET_NEIGHBORS), он формирует отчёт.
3. В отчёте, отправляемом **выделенному маршрутизатору**, указывается:
   - список **соседей**, от которых пришли HELLO;
   - **время передачи** (разница между временем получения и отправки);
   - **информация о сопряжении интерфейсов** — какой локальный порт соответствует порту соседа.
4. **Выделенный маршрутизатор (DR):**
   - На основе собранных данных строит **ориентированный граф топологии**, где **вес ребра** = измеренная задержка передачи.
   - Формирует **таблицу соответствия интерфейсов** для каждого маршрутизатора.
   - Рассылает всем узлам:
     - **полную топологию сети** (SET_TOPOLOGY),

- **локальные данные о соседях** (SET_NEIGHBORS).

Важно: На старте работы маршрутизатор **знает только свои физические каналы**, но **не имеет информации об идентификаторах соседних узлов**. Эти данные становятся доступны **только после обмена с DR**.

4. Результаты

Проведено тестирование программы на **трёх различных топологиях сети**. Каждый маршрутизатор, включая **выделенный (DR)**, оснащён **системой логирования**, что позволяет в реальном времени отслеживать все этапы работы: от установления соседства до доставки пакетов.

Для **проверки корректности маршрутизации** используется следующий механизм: при получении **транзитного сообщения** (не предназначенного текущему узлу) маршрутизатор **добавляет свой идентификатор** в специальное поле данных пакета, после чего **пересылает его дальше** по вычисленному маршруту. Это позволяет на конечном узле восстановить **фактический путь следования** и сравнить его с **ожидаемым кратчайшим маршрутом**.

1) Линейная топология

Узлы: [0, 1, 2]

Список номеров соседних узлов: [[1], [0, 2], [1]]

Если к сети подключены все три узла, то кратчайшие пути будут следующими:

0: [[0], [0, 1], [0, 1, 2]]

1: [[1, 0], [1], [1, 2]]

2: [[2, 1, 0], [2, 1], [2]]

Если от сети отключен узел с индексом 0, то кратчайшие пути будут следующими: 0: [[0], [], []]

1: [[], [1], [1, 2]]

2: [[], [2, 1], [2]]

2) Кольцевая топология

Узлы: [0, 1, 2]

Список номеров соседних узлов: [[2, 1], [0, 2], [1, 0]]

Если к сети подключены все три узла, то кратчайшие пути будут следующими:

0: [[0], [0, 1], [0, 2]]

1: [[1, 0], [1], [1, 2]]

2: [[2, 0], [2, 1], [2]]

Если от сети отключен узел с индексом 1, то кратчайшие пути будут следующими:

0: [[0], [], [0, 2]]

1: [[], [1], []]

2: [[2, 0], [], [2]]

3) Звездная топология.

Узлы: [0, 1, 2, 3]

Список номеров соседних узлов: [[1], [0, 2, 3], [1], [1]]

Если к сети подключены все три узла, то кратчайшие пути будут следующими:

0: [[0], [0, 1], [0, 1, 2], [0, 1, 3]]

1: [[1, 0], [1], [1, 2], [1, 3]]

2: [[2, 1, 0], [2, 1], [2], [2, 1, 3]]

3: [[3, 1, 0], [3, 1], [3, 1, 2], [3]]

Если от сети отключен узел с индексом 2, то кратчайшие пути будут следующими:

0: [[0], [0, 1], [], [0, 1, 3]]

1: [[1, 0], [1], [], [1, 3]]

2: [[], [], [2], []]

3: [[3, 1, 0], [3, 1], [], [3]]

Если от сети отключен узел с индексом 1, то кратчайшие пути будут следующими:

0: [[0], [], [], []]

1: [[], [1], [], []]

2: [[], [], [2], []]

3: [[], [], [], [3]]

5. Обсуждение

Реализован протокол OSPF и проведено тестирование работы протокола на различных топологиях: линейная, кольцо, звезда. На основании результатов проведенных тестов можно утверждать о работоспособности протокола на различных топологиях.

6. Приложения

1. Репозиторий с кодом программы и кодом отчёта:
2. interval-and-networks/networks/lab2