Санкт-Петербургский политехнический
университет им. Петра Великого

Физико-механический институт

Высшая школа прикладной математики и вычислительной
физики

отчет по лабораторной работе №2

Реализация протокола динамической маршрутизации Open Shortest Path First

по дисциплине «Компьютерные сети»

Выполнила

студент гр. 5040102/00201 А.Г. Жаворонкова

Преподаватель

к.ф.-м.н., доцент ВШПМиВФ ФМИ А.Н. Баженов

Санкт-Петербург
2022 год

Содержание

1 Постановка задачи

Требуется разработать систему из неограниченного количества взаимодействующих друг с другом маршрутизаторов, которые организуются в сеть и обеспечивают передачу сообщений от каждого маршрутизатора к каждому по кратчайшему пути.
Необходимо рассмотреть три топологии сети: линейная, кольцо, звезда. Также необходимо рассмотреть перестройку таблиц достижимости при стохастических разрывах связи.

2 Теория

OSPF (Open Shortest Path First) – протокол динамической маршрутизации, основанный на технологии отслеживания состояния канала и использующий для нахождения кратчайшего пути алгоритм Дейкстры [2].
Принцип работы протокола заключается в следующем [1]:

1. После включения маршрутизаторов протокол ищет непосредственно подключённых соседей и устанавливает с ними «дружеские» отношения;

2. Затем они обмениваются друг с другом информацией о подключённых и доступных им сетях. То есть они строят карту сети (топологию сети). Данная карта одинакова на всех маршрутизаторах;

3. На основе полученной информации запускается алгоритм SPF(Shortest Path First), который рассчитывает оптимальный маршрут к каждой сети.

Также выбирается выделенный маршрутизатор (designated router, DR), который управляет процессом рассылки LSA в сети. Каждый маршрутизатор сети устанавливает отношения смежности с DR. Информация об изменениях в сети отправляется маршрутизатором, обнаружившим это изменение, на выделенный маршрутизатор, а тот, в свою очередь, отвечает за то, чтобы эта информация была отправлена остальным маршрутизаторам сегмента множественного доступа.

3 Реализация

Язык программирования – Python, среда разработки – PyCharm.
Программа разделена на модули:

- ***Message***
- ***Topology***
  Здесь реализован алгоритм Дейкстры.
- ***Connection***
- ***Router***
  Здесь реализован класс маршрутизаторов и выделенного маршрутизатора.
- ***Network***

Маршрутизаторы связаны с помощью орграфа с единичными весами рёбер. На канальном уровне реализован протокол связи Go-Back-N, но в данной работе внимание уделяется порядку отправки и получения сообщений. Для маршрутизаторов доступны следующие типы сообщений:

- *NEIGHBOURS (i, neighbours(i))* – сообщение от DR о необходимости добавления новых соседей для узла i;
- *SET_ TOPOLOGY (topology)* – сообщение от DR с информацией о текущей топологии;
- *OFF (i)* – сообщение от DR о необходимости отключения узла i от топологии.

Для выделенного маршрутизатора DR доступны следующие типы сообщений:

- *NEIGHBOURS [neighbours]* – запрос на добавление в топологию новых соседей;
- *GET_ TOPOLOGY ()* – запрос на получение от DR информации о текущей топологии;
- *OFF ()* – сообщение об отключении маршрутизатора.

Ссылка на проект с кодом исследования и отчётом:
https://github.com/Zhavoronkova-Alina/Stochastic-models-and-data-analysis

4 Пример работы программы

4.1 Линейная топология

Рассмотрим пример работы программы для линейной топологии на примере сети с 5 узлами. Конфигурация этой сети выглядит следующим образом:

```
"nodes": [0, 1, 2, 3, 4],
"neighbors": [[1], [0, 2], [1, 3], [2, 4], [3]]
```

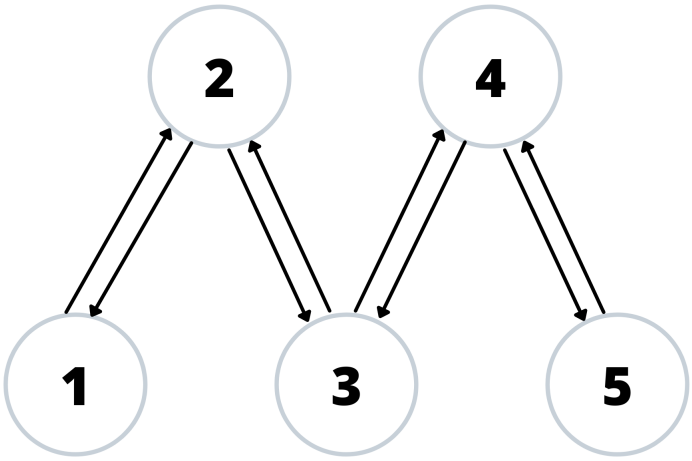

Рис. 1: Конфигурация сети с линейной топологией из пяти маршрутизаторов

Рассмотрим подключение маршрутизаторов к сети:

---

dr(0): (NEIGHBORS: [1])
dr(0): (GET_TOPOLOGY: None)
dr(1): (MsgType.NEIGHBORS: [0, 2])
r(1) : (MsgType.NEIGHBORS: 'index': 0, 'neighbors': [1])
dr(1): (MsgType.GET_TOPOLOGY: None)
dr(2): (MsgType.NEIGHBORS: [1, 3])
dr(3): (MsgType.NEIGHBORS: [2, 4])
dr(4): (MsgType.NEIGHBORS: [3])
r(4) : (MsgType.NEIGHBORS: 'index': 1, 'neighbors': [0, 2])
r(4) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [1, 3])
r(4) : (MsgType.NEIGHBORS: 'index': 3, 'neighbors': [2, 4])
r(1) : (MsgType.SET_TOPOLOGY)
r(1) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [1, 3])
r(1) : (MsgType.NEIGHBORS: 'index': 3, 'neighbors': [2, 4])
r(3) : (MsgType.NEIGHBORS: 'index': 1, 'neighbors': [0, 2])
r(2) : (MsgType.NEIGHBORS: 'index': 1, 'neighbors': [0, 2])
r(0) : (MsgType.SET_TOPOLOGY)
r(3) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [1, 3])
dr(1): (MsgType.NEIGHBORS: [0])
dr(2): (MsgType.GET_TOPOLOGY)
dr(3): (MsgType.GET_TOPOLOGY)
dr(4): (MsgType.GET_TOPOLOGY)
r(2) : (MsgType.NEIGHBORS: 'index': 3, 'neighbors': [2, 4])
r(2) : (MsgType.NEIGHBORS: 'index': 4, 'neighbors': [3])
r(0) : (MsgType.NEIGHBORS: 'index': 1, 'neighbors': [0, 2])
r(1) : (MsgType.NEIGHBORS: 'index': 4, 'neighbors': [3])
r(4) : (MsgType.NEIGHBORS: 'index': 1, 'neighbors': [0])
r(3) : (MsgType.NEIGHBORS: 'index': 4, 'neighbors': [3])
r(3) : (MsgType.NEIGHBORS: 'index': 1, 'neighbors': [0])
r(3) : (MsgType.SET_TOPOLOGY)
dr(2): (MsgType.NEIGHBORS: [1])
dr(3): (MsgType.NEIGHBORS: [2])
dr(4): (MsgType.NEIGHBORS: [3])
dr(2): (MsgType.NEIGHBORS: [3])
dr(3): (MsgType.NEIGHBORS: [4])
r(2) : (MsgType.NEIGHBORS: 'index': 1, 'neighbors': [0])
r(2) : (MsgType.SET_TOPOLOGY)
r(4) : (MsgType.SET_TOPOLOGY)
r(4) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [1])
r(4) : (MsgType.NEIGHBORS: 'index': 3, 'neighbors': [2])
r(4) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [3])
r(4) : (MsgType.NEIGHBORS: 'index': 3, 'neighbors': [4])
dr(2): (MsgType.NEIGHBORS: [1])
r(0) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [1, 3])
r(0) : (MsgType.NEIGHBORS: 'index': 3, 'neighbors': [2, 4])

r(0) : (MsgType.NEIGHBORS: 'index': 4, 'neighbors': [3])
r(0) : (MsgType.NEIGHBORS: 'index': 1, 'neighbors': [0])
r(0) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [1])
r(0) : (MsgType.NEIGHBORS: 'index': 3, 'neighbors': [2])
r(4) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [1])
r(0) : (MsgType.NEIGHBORS: 'index': 4, 'neighbors': [3])
r(1) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [1])
r(3) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [1])
r(3) : (MsgType.NEIGHBORS: 'index': 4, 'neighbors': [3])
r(3) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [3])
r(3) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [1])
r(1) : (MsgType.NEIGHBORS: 'index': 3, 'neighbors': [2])
r(1) : (MsgType.NEIGHBORS: 'index': 4, 'neighbors': [3])
r(1) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [3])
r(1) : (MsgType.NEIGHBORS: 'index': 3, 'neighbors': [4])
r(1) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [1])
r(2) : (MsgType.NEIGHBORS: 'index': 3, 'neighbors': [2])
r(2) : (MsgType.NEIGHBORS: 'index': 4, 'neighbors': [3])
r(0) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [3])
r(0) : (MsgType.NEIGHBORS: 'index': 3, 'neighbors': [4])
r(0) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [1])
r(2) : (MsgType.NEIGHBORS: 'index': 3, 'neighbors': [4])

В результате установлены следующие кратчайшие пути:
0: [[0], [0, 1], [0, 1, 2], [0, 1, 2, 3], [0, 1, 2, 3, 4]]
1: [[1, 0], [1], [1, 2], [1, 2, 3], [1, 2, 3, 4]]
2: [[2, 1, 0], [2, 1], [2], [2, 3], [2, 3, 4]]
3: [[3, 2, 1, 0], [3, 2, 1], [3, 2], [3], [3, 4]]
4: [[4, 3, 2, 1, 0], [4, 3, 2, 1], [4, 3, 2], [4, 3], [4]]

Отключим нулевой узел. Тогда получим следующие кратчайшие пути:
0: [[0], [], [], [], []]
1: [[], [1], [1, 2], [1, 2, 3], [1, 2, 3, 4]]
2: [[], [2, 1], [2], [2, 3], [2, 3, 4]]
3: [[], [3, 2, 1], [3, 2], [3], [3, 4]]
4: [[], [4, 3, 2, 1], [4, 3, 2], [4, 3], [4]]
Затем если восстановить нулевой узел, все кратчайшие пути возвращаются в состояние до отключения. При этом заметим, что нулевой узел подключился самым первым при начале работы. Поэтому информация об его соседях была в распоряжении только DR. При повторном подключении пришлось разослать информацию о соседях всем маршрутизаторам.

4.2 Топология «кольцо»

Определим следующие связи между маршрутизаторами:

```
"nodes": [0, 1, 2, 3, 4],
"neighbors": [[4, 1], [0, 2], [1, 3], [2, 4], [3, 0]]
```

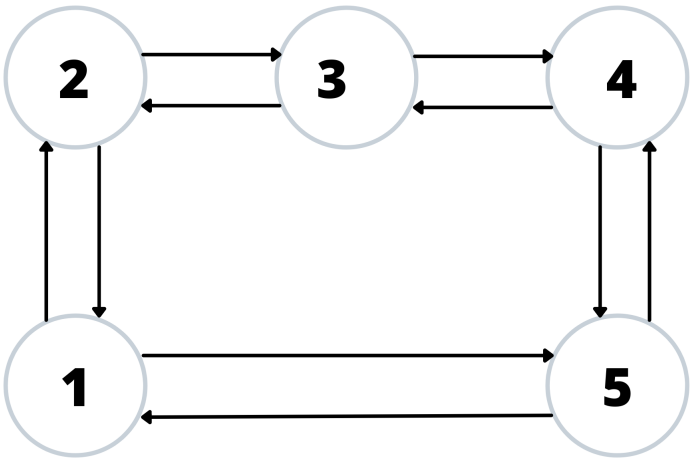

Рис. 2: Конфигурация сети с кольцевой топологией из пяти маршрутизаторов

В результате работы программы были установлены следующие кратчайшие пути:
0: [[0], [0, 1], [0, 1, 2], [0, 4, 3], [0, 4]]
1: [[1, 0], [1], [1, 2], [1, 2, 3], [1, 0, 4]]
2: [[2, 1, 0], [2, 1], [2], [2, 3], [2, 3, 4]]
3: [[3, 4, 0], [3, 2, 1], [3, 2], [3], [3, 4]]
4: [[4, 0], [4, 0, 1], [4, 3, 2], [4, 3], [4]]

Отключим третий узел. Тогда получим следующие кратчайшие пути:
0: [[0], [0, 1], [0, 1, 2], [], []]
1: [[], [1], [1, 2], [], []]
2: [[], [], [2], [], []]
3: [[], [], [], [3], []]
4: [[4, 0], [4, 0, 1], [4, 0, 1, 2], [], [4]]
Затем если восстановить третий узел, все кратчайшие пути возвращаются в состояние до отключения.

4.3 Топология «звезда»

Определим следующие связи между маршрутизаторами:

```
"nodes": [0, 1, 2, 3, 4],
"neighbors": [[2], [2], [0, 1, 3, 4], [2], [2]]
```

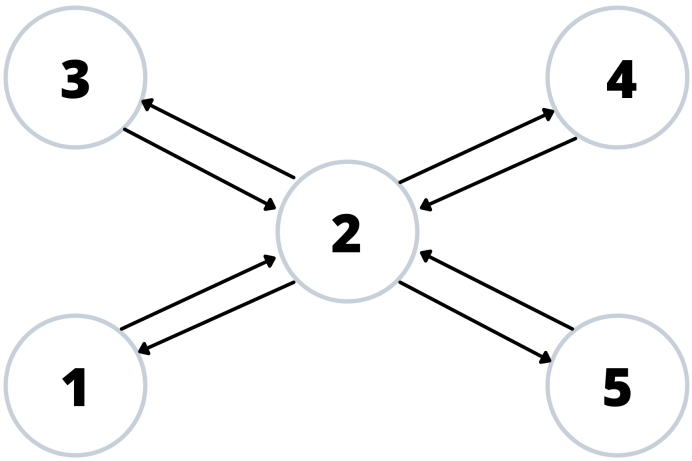

Рис. 3: Конфигурация сети со звёздной топологией из пяти маршрутизаторов

В результате работы программы были установлены следующие кратчайшие пути:
0: [[0], [0, 2, 1], [0, 2], [0, 2, 3], [0, 2, 4]]
1: [[1, 2, 0], [1], [1, 2], [1, 2, 3], [1, 2, 4]]
2: [[2, 0], [2, 1], [2], [2, 3], [2, 4]]
3: [[3, 2, 0], [3, 2, 1], [3, 2], [3], [3, 2, 4]]
4: [[4, 2, 0], [4, 2, 1], [4, 2], [4, 2, 3], [4]]

Отключим второй узел. Тогда получим следующие кратчайшие пути:
0: [[0], [], [], [], []]
1: [[], [1], [], [], []]
2: [[], [], [2], [], []]
3: [[], [], [], [3], []]
4: [[], [], [], [], [4]]
То есть не осталось никаких связей.
Вместо центрального отключился третий узел:
0: [[0], [0, 2, 1], [0, 2], [], [0, 2, 4]]
1: [[1, 2, 0], [1], [1, 2], [], [1, 2, 4]]
2: [[2, 0], [2, 1], [2], [], [2, 4]]
3: [[], [], [], [3], []]
4: [[4, 2, 0], [4, 2, 1], [4, 2], [], [4]]

5 Заключение

Была реализована программа для моделирования протокола динамической маршрутизации OSPF для неограниченного количества взаимодействующих друг с другом маршрутизаторов и стохастическими разрывами соединения. Данная программа была проверена на трёх топологиях. Программа работает корректно.

Список литературы

[1] OSPF [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/OSPF (20.02.2022).

[2] Алгоритм Дейкстры [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Алгоритм_Дейкстры (20.02.2022).