# Реализация протокола динамической маршрутизации Open Shortest Path First

Суриков Илья 5040102/10201

24 ноября 2022 г.

## 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:

1. Три вида топологии сети: линейная, кольцо, звезда.
2. Перестройку таблиц достижимости при стохастических разрывах связи.

# 2 Реализация

Система реализована на языке Python. Топология связей роутеров представляется в виде орграфа. Веса всех ребер графа равны единице. Также выделен отдельный роутер (DR - designated router), который находится вне топологии и обеспечивает маршрутизацию сообщений об изменениях в топологии.

В данной работе мы абстрагируемся от типа реализации канала связи. Важно, чтобы сообщения приходили в том же порядке, в каком и отправляются( будем использовать протокол связи Go-Back-N)

Для подключения нового роутера к сети:

1. Роутер устанавливает связь с DR
2. Роутер отправляет DR сообщение с информацией о соседях
3. Роутер запрашивает у DR текущую топологию сети

Designated Router связан со всеми узлами одновременно. Когда DR получает сообщение о подключении или отключении узла, он обновляет свою топологию, после чего отправляет всем узлам сообщения об изменении топологии.

Так как ключевая задача этой работы - протокол маршрутизации, то мы рассмотрим только сообщения имеющие отношения к топологии.

Возможные типы сообщений:

- Для DR:
    1. NEIGHBOURS([neighbours]) - запрос на добавление в топологию новых соседей. Номер узла (чьи соседи) определяется номером отправителя.
    2. GET_TOPOLOGY() - запрос на получение от DR текущей топологии сети
    3. OFF() - сообщение об отключении роутера
- Для Router
    1. NEIGHBOURS(i, $neighbours_i$) - сообщение от DR о необходимости добавления новых соседей для узла i

2. SET_TOPOLOGY(topology) - сообщение от DR с информацией о текущей топологии
3. OFF(i) - сообщение от DR о необходимости исключения узла i из топологии.
4. PRINT_WAYS() - запрос о выводе на печать текущих кратчайших путей до всех узлов. Это сообщение не влияет на топологию.

Так как топология представлена в виде орграфа то роутер при получении сообщения о добавлении новогоо узла проверяет, является ли новый узел его соседом, если да - посылает DR сообщение о добавлении нового соседа (если такого соседства еще нет в топологии).

Все роутеры, в том числе и DR запускаются в отдельных потоках выполнения.

# 3 Примеры работы программы

## 3.1 Линейная топология

Рассмотрим пример работы программы для линейной топологии с тремя узлами.

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

Все 3 узла подключились к сети. Посмотрим на полученные кратчайшие пути:
0: [[0], [0, 1], [0, 2]]
1: [[1, 0], [1], [1, 0, 2]]
2: [[2, 0], [2, 0, 1], [2]]

Предположим, что отключился второй узел:
dr(2): (MsgType.OFF: None)
r(1) : (MsgType.OFF: 2)
r(0) : (MsgType.OFF: 2)

Тогда новые кратчайшие пути:

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

Видим, что нулевой узел ни с кем не связан. Пусть нулевой узел снова восстановил связь:

dr(2): (MsgType.NEIGHBORS: [0])
dr(2): (MsgType.GET_TOPOLOGY: None)
r(0) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [0])
r(1) : (MsgType.NEIGHBORS: 'index': 2, 'neighbors': [0])
r(2) : (MsgType.SET_TOPOLOGY: 0: 1,2; 1: 0; 2: 0)
Далее восстанавливается связь 0 → 2 dr(0): (MsgType.NEIGHBORS: [2])
r(1) : (MsgType.NEIGHBORS: 'index': 0, 'neighbors': [2])
r(2) : (MsgType.NEIGHBORS: 'index': 0, 'neighbors': [2])
А кратчайшие пути вернулись в состояние до отключения:
0: [[0], [0, 1], [0, 2]]
1: [[1, 0], [1], [1, 0, 2]]
2: [[2, 0], [2, 0, 1], [2]]

Обратим внимание, что вначале нулевой узел подключился самом первым. Других узлов в сети ещё не существовали. Потому информация о соседях нулевого узла была отправлена только DR. При повторном подключении второго узла, не пришлось рассылать информацию всем роутерам.

Аналогично можно построить и другие топологии. Отличие будет только в определении соседей для каждого узла:

## 3.2 Топология кольцо

nodes: [0, 1, 2]
neighbors: [[2, 1], [0, 2], [1, 0]]
Минимальные пути:
0: [[0], [0, 1], [0, 2]]
1: [[1, 0], [1], [1, 2]]
2: [[2, 0], [2, 1], [2]]
после отключения 2 узла:
0: [[0], [], [0, 2]]
1: [[], [1], []]
2: [[2, 0], [], [2]]

## 3.3 Топология звезда с центром узле с индексом 2

nodes: [0, 1, 2, 3]
neighbors: [[2], [2], [0, 1, 3], [2]]
Минимальные пути:
0: [[0], [0, 2, 1], [0, 2], [0, 2, 3]]
1: [[1, 2, 0], [1], [1, 2], [1, 2, 3]]
2: [[2, 0], [2, 1], [2], [2, 3]]
3: [[3, 2, 0], [3, 2, 1], [3, 2], [3]]

После отключения третьего узла
0: [[0], [0, 2, 1], [0, 2], []]
1: [[1, 2, 0], [1], [1, 2], []]
2: [[2, 0], [2, 1], [2], []]
3: [[], [], [], [3]]
После отслючения центрального узла
0: [[0], [], [], []]
1: [[], [1], [], []]
1: [[], [], [2], []]
3: [[], [], [], [3]]

# 4 Результаты

Была реализована программа для моделирования протокола динамической маршрутизации OSPF для неограниченого количества взаимодействующих друг с другом маршрутизаторов и стохастическими разрывами соединения.

Данная программа была проверена на трех топологиях, из чего был сделан вывод о ее корректной работе на топологиях: линейная, кольцо, звезда.

# 5 Ссылка на исходный код

https://github.com/Chopikov/comp_networks