Реализация протокола динамической маршрутизации Open Shortest Path First

Никита Лансков

16 января 2022 г.

Содержание

1 Постановка задачи **2**

2 Реализация **2**

3 Примеры работы программы **3**

3.1 Линейная топология 3

3.2 Топология кольцо 4

3.3 Топология звезда с центром узле с индексом 1 4

4 Результаты **5**

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], [0, 2], [2]]
dr(0): (NEIGHBORS, neighbors(0))
dr(0): (GET_TOPOLOGY)
dr(1): (NEIGHBORS, neighbors(1)))
r(0) :(SET_TOPOLOGY)
dr(1): (GET_TOPOLOGY)
r(0) : (NEIGHBORS: 1, neighbors(1))
dr(2): (NEIGHBORS:neighbors(2)))
r(2) : (NEIGHBORS: 1, neighbors(1)))
r(1) : (SET_TOPOLOGY)
r(0) : (NEIGHBORS: 2, neighbors(2))
r(1) : (NEIGHBORS: 2, neighbors(2))
dr(2): (GET_TOPOLOGY)
r(2) : (SET_TOPOLOGY)
Все 3 узла подключились к сети. Посмотрим на полученные кратчайшие пути:
0: [[0], [0, 1], [0, 1, 2]]
1: [[1, 0], [1], [1, 2]]
2: [[2, 1, 0], [2, 1], [2]]
Предположим, что отключился нулевой узел:
dr(0): (OFF: None)
r(2) : (OFF: 0)
r(1) : (OFF: 0)
Тогда новые кратчайшие пути:
0: [[0], [], []]
1: [[], [1], [1, 2]]
2: [[], [2, 1], [2]]

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

dr(0): (NEIGHBORS: neighbors(0))
dr(0): (GET_TOPOLOGY)
r(1) : (NEIGHBORS: 0, neighbors(0))
r(0) : (SET_TOPOLOGY)
r(2) : (NEIGHBORS: 0, neighbors(0))
Далее восстанавливается связь $1 \to 0$
dr(1): (NEIGHBORS: [0])
r(2) : (NEIGHBORS: 1, [0])
r(0) : (NEIGHBORS: 1, [0])
А кратчайшие пути вернулись в состояние до отключения:
0: [[0], [0, 1], [0, 1, 2]]
1: [[1, 0], [1], [1, 2]]
2: [[2, 1, 0], [2, 1], [2]]

Притом, заметим, что в начале, 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, пришлось разослать её 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 1

nodes: [0, 1, 2, 3]
neighbors: [[1], [0, 2, 3], [1], [1]]
Минимальные пути:
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]]
После отключения центрального узла
0: [[0], [], [], []]

1: [[], [1], [], []]
2: [[], [], [2], []]
3: [[], [], [], [3]]
После отслючения четвертого узла
0: [[0], [0, 1], [0, 1, 2], []]
1: [[1, 0], [1], [1, 2], []]
2: [[2, 1, 0], [2, 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.