Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій

_____Юрій КРАВЧЕНКО
«_____» _____ ___2022 року

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Мережеві та інтернет технологій»

на тему:

СИСТЕМА СИМУЛЯЦІЇ МАРШРУТИЗАТОРА ДЛЯ ТЕСТУВАННЯ РОЗРОБЛЕНИХ ФУНКЦІЙ

Юрій Агафонов
(ім'я та ПРІЗВИЩЕ) (підпис)

Виконав: студент групи МІТ -41

к.т.н., Старкова Олена Володимирівна
(науковий ступень, вчене звання, ім'я та ПРІЗВИЩЕ)

Керівник: доцент кафедри мережевих та інтернет технологій
(посада)

(підпис)

Київ - 2022

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій

_____Юрій КРАВЧЕНКО

« » 2022

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ

Здобувачу вищої освіти Агафонову Юрію Владиславовичу
(прізвище, ім'я, по батькові)

1. Тема роботи:
Система симуляції маршрутизатора для тестування розроблених функцій

затверджена на засіданні кафедри МІТ «24» грудня 2021 р. протокол №8

2. Термін здачі закінченої роботи «30» травня 2022 р.

3.Вихідні дані до проекту (роботи)

Протоколи маршрутизації, технології опису та дослідження процесів маршрутизації

4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 35-40 стор.)

Вступ

1. Огляд підходів до симуляції мережевих пристроїв.

1.1 Розвиток мережевих пристроїв.

1.2 Функції мережевих пристроїв.

1.3 SDN як рішення проблем сучасних мереж.

1.4 Концепція тестового середовища.

2. Склад симуляції.

3. Випробування роботи симуляції.

3.1. Опис топології.

3.2. Опис взаємодії між маршрутизаторами.

3.3. Опис маршрутизації.

3.4 Маршрутизація трафіку.

Висновки

Дата видачі завдання

Керівник роботи Старкова Олена Володимирівна
(підпис) (посада, прізвище, ім'я, по батькові)

Завдання прийняв до виконання
Агафонов Юрій Владиславович
(підпис)

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	02.05.2022	
2	Розділ 1	10.05.2022	
3	Розділ 2	15.05.2022	
4	Розділ 3	20.05.2022	
5	Доповідь та слайди	25.05.2022	
6	Пояснювальна записка	30.05.2022	

Здобувач вищої освіти _____ Агафонов Юрій
(підпис)

Керівник _____ Старкова Олена
(підпис)

РЕФЕРАТ

Пояснювальна записка: 38 с., 36 рис., 23 джерела.

Об’єкт дослідження: симуляція роботи будь якої кількості маршрутизаторів на одному пристрої.

Мета роботи (проекту): розробка зручного засобу для впровадження нових функцій маршрутизатора та розробки нових мережевих протоколів.

Методи дослідження: при дослідженні були використані метод аналізу, методи класифікації, методи імітаційного моделювання.

Актуальність: у зв’язку зі стрімким розвитком мережевих технологій по всьому світу впровадження нових функцій, що дозволять прискорити та покращити роботу мережі завжди в пріоритеті.

Загальна характеристика роботи: робота присвячена актуальній темі розширення та удосконалення існуючого функціоналу мережевих пристроїв, що в майбутньому забезпечить загальне підвищення якості сервісів, що надаються мережею. А також розроблена система симуляції дозволить проводити більш глибокі експериментальні дослідження різноманітних технологічних рішень, що актуально з наукової точки зору.

Ключові слова: мережевий пристрій, протокол динамічної маршрутизації, тестове середовище, Python, SDN, симуляція мережевих функцій.

ЗМІСТ

ВСТУП

Мережеві технології інтенсивно розвиваються. Це пов’язано з широким використанням Інтернету у різних сферах діяльності та дозвілля людини. Традиційно на мережевих пристроях для виконання відповідного функціоналу реалізовувались стандартні (максимум розроблені виробником обладнання) протоколи та технології.[1] Але з часом виявлялися їх недоліки, невідповідність сучасним викликам до мереж (зростанню кількості користувачів, сервісів, пристроїв та каналів зв’язку, вимог до мереж).[2] Також з’явились нові тенденції розвитку мереж у бік впровадження концепції програмованих мереж SDN та віртуалізації мережевих функцій. Таким чином постала задача удосконалення існуючих та створення принципово нових протоколів та технологій, які зможуть цілком задовільнити потреби перспективних інфокомунікаційних мереж та систем.[3][4]

Отже метою роботи є розробка середовища (системи симуляції) для реалізації різних мережевих функцій та протокольних рішень. В подальшому це дозволить проводити експериментальні дослідження, а в перспективі реалізовувати в контексті SDN розроблені технології та протоколи.[5][6]

1. ОГЛЯД ПІДХОДІВ ДО СИМУЛЯЦІЇ МЕРЕЖЕВИХ ПРИСТРОЇВ

1.1 Розвиток мережевих пристроїв

У ті часи, коли мережева технологія була ще в зародковому стані, кожен одержувач в мережі отримував усі передачі даних, і один реальний одержувач визначав, що дані призначені йому, а інші просто відкидали дані.

На сьогоднішній день зрозуміло, що така схема в сучасних умовах недовіри є небезпечною і неефективною. Наприклад ви не можете надсилати дані двосторонньої автентифікації на всі пристрої в мережі.

Технічно вищезгадана схема була в епоху хабів, які не були інтелектуальними мережевими пристроями. Вони спрямовували трафік від одного інтерфейсу до всіх інших.

Оскільки мережева технологія ставала більш складною, розумні пристрої змогли визначити правильного одержувача та відповідно маршрутизувати трафік до місця призначення. Це стосується комутаторів і маршрутизаторів, які зараз зазвичай об’єднані в один пристрій.

1.2 Функції мережевих пристроїв

Існує маса функцій, які виконують мережеві пристрої. Наприклад, роутер виконує багато завдань:

- Підключення локальних мереж до глобальної мережі.
- Сегментація мережі на окремі широкомовні домени, що підвищує безпеку, продуктивність і керованість таких мереж.
- Пошук найкращого маршруту для доставки пакетів по мережі шляхом складання таблиць маршрутизації. Протоколи динамічної

маршрутизації різних типів використовуються для пошуку найкращого маршруту до пункту призначення за різними параметрами.

- Для створення легкодоступної мережевої інфраструктури маршрутизатори можуть служити різними серверами, такими як сервер DNS або DHCP.
- Маршрутизатори можуть встановлювати зашифровані тунелі для безпечної передачі даних у віддалену мережу (VPN).
- Брандмауер і система запобігання вторгненням (IPS).

1.3 SDN як рішення проблем сучасних мереж

На даний момент існують дві тенденції розвитку мереж:

- Оптимізація існуючих технологій, протоколів, механізмів, алгоритмів відповідно до вимог, які пред'являються до сучасних мереж;
- Автоматизація управління мережевими пристроями;
- Віртуалізація мережевих функцій.

Одним із найперспективніших методів, які можуть допомогти вирішити проблеми розвитку сучасних мереж, є модель програмно-визначеної мережі (SDN), яка передбачає поділ функцій передачі трафіку та функцій управління, включаючи контроль як самого трафіку, так і пристроїв, які передають його. Відповідно до концепції SDN, вся логіка управління розташована в контролерах, які здатні контролювати роботу всієї мережі за допомогою спеціальних протоколів (наприклад, OpenFlow), які працюють за концепцією потоків і можуть виконувати з ними різні дії (дозволити , заборонити, перенаправити, редагувати поля в пакетах тощо). Перевагами програмно-визначеної мережі є централізоване управління, спрощення обслуговування мережі та модернізація.

Віртуалізація мережевих функцій (NFV) — це архітектурна структура, створена Європейським інститутом телекомунікаційних стандартів (ETSI), яка визначає стандарти для відокремлення мережевих функцій від власних апаратних

пристроїв і забезпечення їх запуску в програмному забезпеченні на стандартних серверах x86. Деякі переваги NFV подібні до переваг віртуалізації серверів і хмарних середовищ :

- Зменшення капітальних (capex) та операційних витрат (operex) за рахунок зниження витрат на обладнання та ефективності використання простору, електроенергії та охолодження.
- Швидший час виходу на ринок (TTM), оскільки віртуальні машини та контейнери легше розгорнути, ніж обладнання.
- Покращена рентабельність інвестицій (ROI) від нових послуг.
- Можливість збільшення/зменшення та зменшення/збільшення ємності за потребою (еластичність).
- Відкритість до ринку віртуальних пристроїв і постачальників чистого програмного забезпечення.
- Можливості тестувати та впроваджувати нові інноваційні послуги віртуально та з меншим ризиком.

Структура архітектури NFV, розроблена ETSI, показана на рис.1.1.

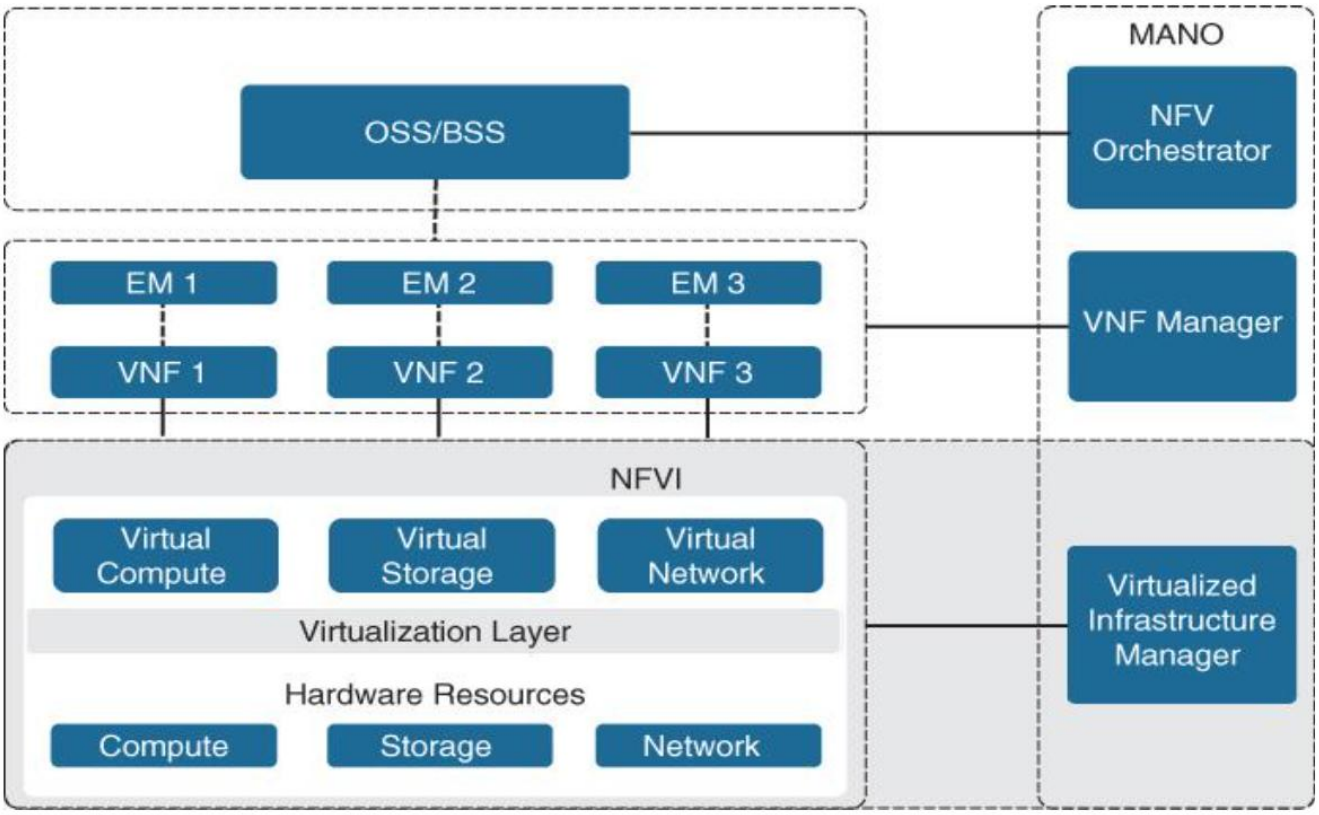

Рис. 1.1 Архітектурний каркас ETSI NFV.

1.4 Концепція тестового середовища

Не існує простого тестового середовища для створення нових протоколів динамічної маршрутизації. Існує кілька проектів моделювання мережі на звичайному комп'ютері, але вони не передбачають значних змін протоколу і завантажують систему речами, які не потрібні при тестуванні маршрутизації, наприклад, другий рівень. Хоча в основному вже існуючі проекти використовують контейнеризацію або повинні бути розгорнуті на різних фізичних пристроях або віртуальних системах за допомогою віртуалізації.

Тому було вирішено створити просту систему виключно для тестування протоколів динамічної маршрутизації. Система працює на одному комп'ютері і дозволяє швидко і легко створювати і модифікувати різні протоколи

маршрутизації з їх подальшою перевіркою, а також в подальшому впроваджувати розроблені методи маршрутизації, які є ефективнішими за існуючі.

2. СКЛАД СИМУЛЯЦІЇ

В роботі за основу був взятий класичний мережевий пристрій -- маршрутизатор. Основні функції маршрутизатора можна розділити на дві області: створення мережевої карти та пересилання пакетів між мережами. Щоб побудувати мережеву карту, маршрутизатори, як правило, використовують статичну маршрутизацію, або динамічні протоколи маршрутизації. За допомогою протоколів динамічної маршрутизації маршрутизатори дозволять іншим мережевим пристроям знати не тільки про топологію мережі, але й зміни змін у ній.[7] Статична маршрутизація не буде адаптуватися до мережевих змін. Обидві моделі виконують завдання побудувати мережеву карту у вигляді таблиці маршрутизації.

Під час процесу маршрутизації маршрутизатори розглядають кілька альтернативних шляхів для того, щоб дістатися до одного пункту призначення. Ці альтернативи є результатом надмірності вбудованого в більшість мережевих проектів. Вам потрібні кілька шляхів, тому, що якщо якийсь шлях передачі даних вийде з ладу, інші альтернативи стануть доступними.

Для розробки системи симуляції використовувались:

- Мова програмування Python
- Модулі socket, ipaddress, threading, json, tabulate для Python

Симуляція роботи маршрутизаторів складається з багатьох об’єктів, що взаємодіють між собою.[8]

Об'єкти симуляції:

— Об'єкт **Simulation** – це об'єкт необхідний для зберігання об'єктів маршрутизаторів. Він може запускати всі маршрутизатори разом використовуючи метод *start_routers* та виводити таблицю інтерфейсів з статусами та іншою інформацією про об'єкти інтерфейсів *print(simulation)*

```python
class Simulation:

    def __init__(self):
        self.routers = {}

    def add_router(self, router_name, router_interfaces):
        self.routers[router_name] = Router(router_name, router_interfaces)

    def start_routers(self):
        for router in self.routers:
            self.routers[router].start()

    def __str__(self):
        data = {"Router": [], "Number": [], "Port": [], "Broadcast": [], "IP": [], "Status": []}
        for router in self.routers:
            for interface in self.routers[router].interfaces:
                data["Router"].append(interface.hostname)
                data["Number"].append(interface.number)
                data["Port"].append(interface.local_port)
                data["Broadcast"].append(interface.broadcast_port)
                data["IP"].append(interface.get_ip())
                data["Status"].append(interface.status)

        return tabulate(data, headers='keys', tablefmt='grid')
```

Рис. 2.1 Клас Simulation

— Об'єкт **Router** – це основний об'єкт симуляції. Він зберігає інтерфейси, що необхідні для обміну інформацією між маршрутизаторами. Об’єкти інтерфейсів потрібно передавати при ініціалізації (створенні об’єкту маршрутизатору).

Об'єкт **Router** також зберігає таблицю маршрутизації й інші данні про маршрутизатор.

```python
class Router:

    def __init__(self, hostname, interfaces):

        self.hostname = hostname

        self.interfaces = interfaces

        self.ip_list = []

        for interface in self.interfaces:
            self.ip_list.append(str(interface.get_ip()))
            interface.hostname = self.hostname
            interface.routing_function = self.__parse_interface_data

        self.__create_broadcast()

        self.data_packet = {"src_ip": "ip", "dst_ip": "ip", "data": "message"}
        self.broadcast_packet = {"src_ip": "ip", "src_port": "port", "data": "message"}

        self.threads = {}

        # self.routing_table = {"network": {"gateway": "0.0.0.0", "interface": 0, "metric": 20}}
        self.routing_table = {}

        self.dynamic_protocol = RIP(router=self)
```

Рис. 2.2 Клас Router

Об'єкт **Router** має такі відкриті методи, як:

- *set_protocol*, що необхідний для встановлення об’єкту протоколу динамічної маршрутизації, що буде працювати на маршрутизаторі.

```python
    def set_protocol(self, protocol):
        if not isinstance(protocol, RoutingProtocol):
            print("You must inherit 'RoutingProtocol' class")
            return 0

        print(f"{self.hostname} now has {protocol.name}")
```

Рис. 2.3 Метод set_protocol

- *has_ip*, що необхідний для перевірки наявності у маршрутизатору певної ip-адреси, що була передана до методу.

```
def has_ip(self, message_ip):
    for interface in self.interfaces:
        if interface.get_ip() == message_ip:
            return True
    return False
```

Рис. 2.4 Метод has_ip

- *message_to_interface*, що необхідний для того, щоб відправити повідомлення з певного інтерфейсу.[9] Потрібно передати номер інтерфейсу з якого повідомлення буде відправлено та саме повідомлення.

```
def message_to_interface(self, message, interface_number, src_ip=None, dst_ip=None):
    connection = self.interfaces[interface_number].get_conn()
    conn = connection["connection"]
    packet = self.data_packet
    packet["data"] = message
    if src_ip:
        packet["src_ip"] = src_ip
    else:
        packet["src_ip"] = str(self.interfaces[interface_number].get_ip())
    if dst_ip:
        packet["dst_ip"] = dst_ip
    else:
        packet["dst_ip"] = str(connection["remote_ip"])
    packet = json.dumps(packet)
    conn.send(bytes(packet, "utf-8"))

def message_to_ip(self, message, ip):
    for interface in self.interfaces:
        if ip == interface.connection["remote_ip"]:
            self.message_to_interface(message, interface.number)
            return 0

    interface_number = self.__find_route(ip)
    self.message_to_interface(message,
                              interface_number,
                              str(self.interfaces[interface_number].get_ip()),
                              ip)
```

Рис. 2.5 Метод message_to_interface

- *message_to_ip*, що необхідний для того, щоб відправити повідомлення на певну ip-адресу. Потрібно передати ip-адресу та саме повідомлення для відправки. Тоді маршрутизатор спробує знайти шлях для відправлення у підключених інтерфейсах або таблиці маршрутизації.

```
def message_to_ip(self, message, ip):
    for interface in self.interfaces:
        if ip == interface.connection["remote_ip"]:
            self.message_to_interface(message, interface.number)
            return 0

    interface_number = self.__find_route(ip)
    self.message_to_interface(message,
                              interface_number,
                              str(self.interfaces[interface_number].get_ip()),
                              ip)
```

Рис. 2.6 Метод message_to_ip

- *message_to_broadcast*, що необхідний для того, щоб відправити повідомлення на широкомовний порт. Це повідомлення отримають всі інтерфейси з такою самою широкомовною адресою.

```
def message_to_broadcast(self, message, interface="All"):
    def send_data(interface):
        sock = interface.broadcast_socket
        port = interface.broadcast_port
        packet = self.broadcast_packet
        packet["src_ip"] = str(interface.get_ip())
        packet["src_port"] = interface.local_port
        packet["data"] = message
        packet = json.dumps(packet)
        sock.sendto(bytes(packet, "utf-8"), ('<broadcast>', port))

    if interface != "All":
        send_data(self.interfaces[interface])
    else:
        for interface in self.interfaces:
            send_data(interface)
```

Рис. 2.7 Метод message_to_broadcast

- *set_default_route*, що необхідний для встановлення маршруту за замовчуванням. Потрібно передати ip-адресу та номер інтерфейсу з якого є доступ до мережі та метрику. Метрика необхідна для знаходження найшвидшого способу відправки повідомлення.

```
def set_default_route(self, gateway, interface, metric=20):
    self.routing_table['0.0.0.0/0'] = {"gateway": gateway,
                                       "interface": interface,
                                       "metric": metric}
```

Рис. 2.8 Метод set_default_route

- *add_route*, що необхідний для встановлення маршруту до конкретної мережі.[10][11] Потрібно передати ip-адресу та маску підмережі для віддаленої мережі, а також ip-адресу й номер інтерфейсу з якого є доступ до віддаленої мережі та метрику.

```
def add_route(self, network, netmask, gateway, interface, metric):
    self.routing_table[f'{network}/{netmask}'] = {"gateway": gateway,
                                                  "interface": interface,
                                                  "metric": metric}
```

Рис. 2.9 Метод add_route

- *start*, що необхідний для запуску маршрутизатору у роботу. Після виконання цієї команди маршрутизатор починає прослуховувати широкомовний порт та протокол динамічної маршрутизації починають свою роботу.

```
def start(self):
    self.__listen_broadcast()
    self.dynamic_protocol.start()
    print(f"Router {self.hostname} have started")
```

Рис. 2.10 Метод start

— Об'єкт **Interface** це інтерфейс маршрутизатора, що може взаємодіяти з іншими інтерфейсами для обміну данними.

```
class Interface:

    def __init__(self, number, local_port, broadcast_port, interface):
        self.number = number
        self.local_port = local_port
        self.broadcast_port = broadcast_port
        self.interface = interface
        self.status = "Unused"

        self.hostname = None
        self.routing_function = None
        self.broadcast_socket = None
        self.listener = None

        self.connection = {}
```

Рис. 2.11 Клас Interface

Має наступні методи:

- *get_ip*, що повертає ip-адресу інтерфейсу.

```
    def get_ip(self):
        return self.interface.ip
```

Рис. 2.12 Метод get_ip

- *get_conn*, що повертає об'єкт зв'язку інтерфейсу з іншим інтерфейсом.

```
    def get_conn(self):
        return self.connection
```

Рис. 2.13 Метод get_conn

- *connect_to_router*, що починає ряд дій для встановлення з’єднання з іншим інтерфейсом.

```python
def connect_to_router(self, r_ip):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('localhost', self.local_port))
    sock.listen(1)
    self.connection = {"remote_ip": r_ip, "connection": sock}
    waiter = threading.Thread(target=self.wait_connection)
    waiter.start()
    self.listener = waiter
```

Рис. 2.14 Метод connect_to_router

- *wait_connection*, що починає очікувати відповідь від іншого інтерфейсу для підтвердження встановлення з'єднання.

```python
def wait_connection(self):
    connection = self.connection
    sock = connection["connection"]
    conn, address = sock.accept()
    print(f"Router {self.hostname} CONNECTED to {connection['remote_ip']}")
    self.connection["connection"] = conn
    # print(conn) #
    self.status = "Connected"
    listener = threading.Thread(target=self.listen_conn)
    listener.start()
    self.listener = listener
```

Рис. 2.15 Метод wait_connection

- *accept_connection*, що підтверджує підключення до іншого інтерфейсу.

```python
def accept_connection(self, r_port, ip):
    connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    connection.connect(('localhost', r_port))
    self.connection = {"remote_ip": ip, "remote_port": r_port, "connection": connection}
    # print(connection) #
    self.status = "Connected"
    listener = threading.Thread(target=self.listen_conn)
    listener.start()
    self.listener = listener
```

Рис. 2.16 Метод accept_connection

- *listen_conn*, що чекає створення зв'язку з іншим інтерфейсом.

```
def listen_conn(self):
    connection = self.connection["connection"]
    while True:
        message = connection.recv(1024)
        message = message.decode("utf-8")
        if not message:
            break

        self.routing_function(message, self)
```

Рис. 2.17 Метод listen_conn

Для утворення зв'язку між двома інтерфейсами потрібно:

1. Викликати метод інтерфейсу *connect_to_router* на першому інтерфейсі й передати ip-адресу віддаленого інтерфейсу до якого підключення створюється.[12]

Після цього перший інтерфейс буде очікувати створення зв'язку між двома інтерфейсами.

2. Викликати *accept_connection* на другому інтерфейсі, до якого підключається перший інтерфейс та передати адресу першого інтерфейсу, що хоче підключитись та його порт для з’єднання інтерфейсів (*local_port*).[13]

Після цього інтерфейс буде очікувати створення зв'язку між двома інтерфейсами.

— Клас **RoutingProtocol** – це абстрактний клас для створення протоколів динамічної маршрутизації, задає основні методи, що обов'язково повинні бути в усіх протоколах динамічної маршрутизації.

```python
class RoutingProtocol:

    def __init__(self, name):
        self.name = name

    @abstractmethod
    def start(self):
        pass

    @abstractmethod
    def new_message(self, message, interface):
        pass
```

Рис. 2.18 Клас RoutingProtocol

— Об'єкт **RIP** – це об'єкт, створений для демонстрації створення та роботи протоколів динамічної маршрутизації з маршрутизаторами у симуляції.

```python
class RIP(RoutingProtocol):

    def __init__(self, router):
        RoutingProtocol.__init__(self, "RIP")
        self.router = router
        self.rip_packet = {'type': 'RIP', 'routing_table': None}
```

Рис. 2.19 *Клас RIP*

Має наступні методи:

- *new_message*, що необхідний для обробки отриманого повідомлення, що адресоване протоколу динамічної маршрутизації. В нашому випадку це повідомлення з таблицею маршрутизації іншого маршрутизатора.

```
def new_message(self, message, interface):
    routing_table = message['routing_table']
    for network in routing_table:
        metric = routing_table[network]["metric"]
        if network not in self.router.routing_table:
            print(1)
            self.router.routing_table[network] = {
                                    "gateway": interface.get_ip(),
                                    "interface": interface.number,
                                    "metric": metric}
            print(f"Update Routing Table {self.router.hostname} "
                  f"From RIP/ {self.router.routing_table}")
        elif metric < self.router.routing_table[network]["metric"]:
            print(2)
            self.router.routing_table[network] = {
                                    "gateway": interface.get_ip(),
                                    "interface": interface.number,
                                    "metric": metric}
            print(f"Update Routing Table {self.router.hostname} "
                  f"From RIP/ {self.router.routing_table}")
```

Рис. 2.20 Метод new_message

- *get_routing_table_to_send*, що необхідний для обробки таблиці маршрутизації маршрутизатору для її подальшого відправлення на інші маршрутизатори.

```
def get_routing_table_to_send(self):
    r_table = self.router.routing_table
    for network in r_table:
        r_table[network]['metric'] = r_table[network]['metric']+1
        r_table[network]['interface'] = None
        r_table[network]['gateway'] = None
    return r_table
```

Рис. 2.21 Метод get_routing_table_to_send

- send_route, що необхідний для відправки підготовленої методом *get_routing_table_to_send* таблиці маршрутизації з усіх інтерфейсів маршрутизатору (тобто усім сусідам у мережі).

```
def send_route(self):
    for interface in self.router.interfaces:
        packet = self.rip_packet
        packet['routing_table'] = self.get_routing_table_to_send()
        if packet['routing_table']:
            self.router.message_to_interface(packet, interface.number)
```

Рис. 2.22 *Метод* send_route

- *runner*, що необхідний для відправлення повідомлень кожні n секунд.

```
def runner(self):
    while True:
        time.sleep(5)
        self.send_route()
```

Рис. 2.23 Метод runner

- *start*, що необхідний для запуску протоколу в роботу. Після виконання запускається цикл методом *runner*, що кожні n секунд відправляє таблицю маршрутизації усім сусідам маршрутизатору у мережі.

```
def start(self):
    runner = threading.Thread(target=self.runner, )
    runner.start()
```

Рис. 2.24 Метод start

3. ВИПРОБУВАННЯ РОБОТИ СИМУЛЯЦІЇ

3.1 Опис топології

Для початку роботи симуляції потрібно створити всі необхідні об’єкти.

Спочатку створити списки з об’єктів інтерфейсу, для їх створення необхідно передати такі параметри, як:

- Номер інтерфейсу.
- Порт інтерфейсу.
- Широкомовний порт інтерфейсу. Для простоти побудови мережі можна використовувати широкомовний порт для утворення підключень між інтерфейсами.[15] Для цього у двох інтерфейсів що ми хочемо підключити один до одного повинен бути однаковий унікальний для цих інтерфейсів широкомовний порт.
- Об’єкт **IPv4Interface** для зберігання ip-адреси

```
sim = Simulation()

r1_interfaces = [Interface(0, 9101, 9910, IPv4Interface('10.1.1.1/24')),
                 Interface(1, 9102, 9920, IPv4Interface('10.1.2.1/24')),
                 ]

r2_interfaces = [Interface(0, 9201, 9910, IPv4Interface('10.2.1.1/24')),
                 ]

r3_interfaces = [Interface(0, 9301, 9920, IPv4Interface('10.3.1.1/24')),
                 Interface(1, 9302, 9930, IPv4Interface('10.3.2.1/24')),
                 ]

r4_interfaces = [Interface(0, 9401, 9930, IPv4Interface('10.4.1.1/24')),
                 ]
```

Рис. 3.1 Створення списків інтерфейсів

Далі викликати метод об’єкту симуляції, що створить та додасть до симуляції маршрутизатори. Для цього потрібно передати назву маршрутизатору та створені в минулому кроці інтерфейси.[16][17] Маршрутизаторів можна створити скільки

завгодно, але потрібно продумати створення інтерфейсів, бо симуляція буде правильно працювати лише при правильно створених параметрах, коли ніякі данні крім широкомовного порта не повторюються між інтерфейсами.

```
sim.add_router("router1", r1_interfaces)
sim.add_router("router2", r2_interfaces)
sim.add_router("router3", r3_interfaces)
sim.add_router("router4", r4_interfaces)
```

Рис. 3.2 Створення об’єктів маршрутизаторів

Після цього запускаємо всі маршрутизатори й викликаємо у всіх маршрутизаторів метод *message_to_broadcast("Hello, lets connect")*. Це необхідно для автоматичного підключення інтерфейсів з однаковим широкомовним портом один до одного.

```
sim.start_routers()

time.sleep(1)

sim.routers["router1"].message_to_broadcast("Hello, lets connect")
time.sleep(1)
sim.routers["router2"].message_to_broadcast("Hello, lets connect")
time.sleep(1)
sim.routers["router3"].message_to_broadcast("Hello, lets connect")
time.sleep(1)
print(sim)
```

Рис. 3.3 Запуск та автоматичне підключення всіх маршрутизаторів

Це утворює наступну топологію:

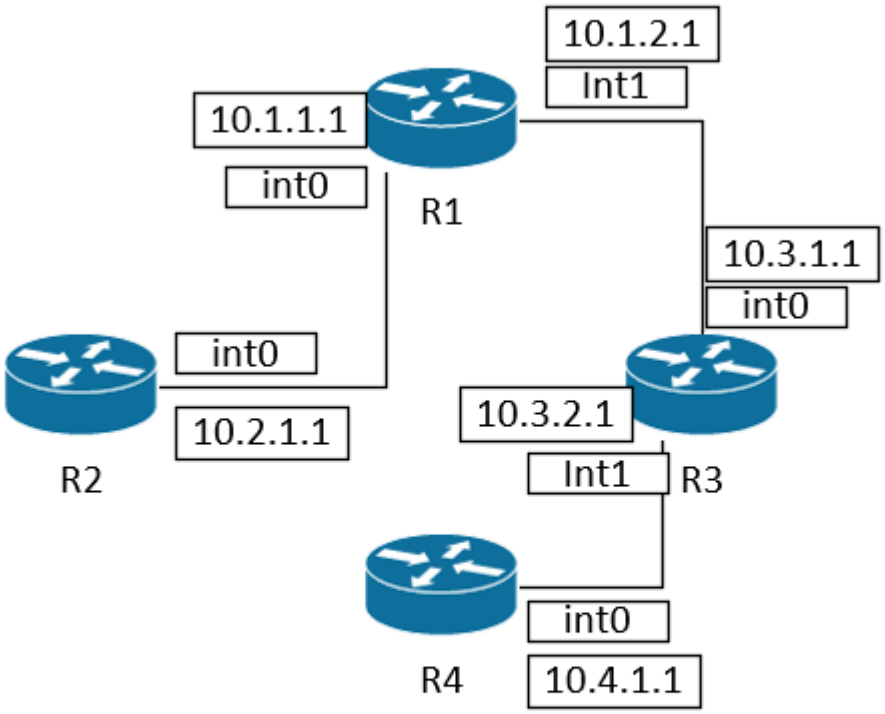

Рис. 3.4 Топологія, створена у симуляції

Створення топології було проведено завдяки наступним широкомовним портам

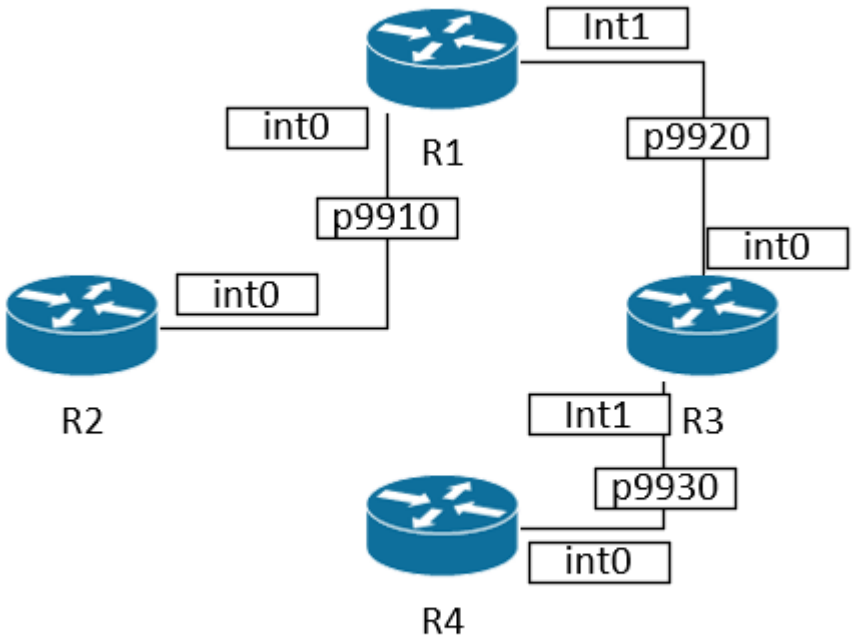

Рис. 3.5 Широкомовні порти при створенні топології

3.2 Опис взаємодії між маршрутизаторами

Для обміну інформацією між маршрутизаторами у симуляції використані такі об’єкти, як сокети. [18]

Сокет – це програмний інтерфейс для забезпечення обміну даними між процесами. Процеси при такому обміні можуть виконуватися як на одному пристрої, так і на різних, пов'язаних між собою мережею.[19][20]

Для роботи сокетів потрібні мережеві порти, тому в симуляції потрібно їх задавати для кожного інтерфейсу.[21]

Сокети у симуляції використані замість другого рівня моделі OSI й дозволяють створити між двома програмними об’єктами сокет - з’єднання й обмінюватись даними між цими програмними об’єктами.

В об’єкті інтерфейсу існують широкомовні сокети, що прослуховують та передають данні на один порт.

При створенні з’єднання між двома інтерфейсами ми створюємо між портами цих інтерфейсів сокет - з’єднання, що дозволяє відправити данні з будь якого з цих двох інтерфейсів й отримати їх на іншому інтерфейсі.

Процес підключення інтерфейсів у сокет - з’єднання може виглядати наступним чином:

1. Спочатку у нас є декілька маршрутизаторів, деякі інтерфейси яких мають однаковий широкомовний порт. Як на прикладі з наступного малюнку.

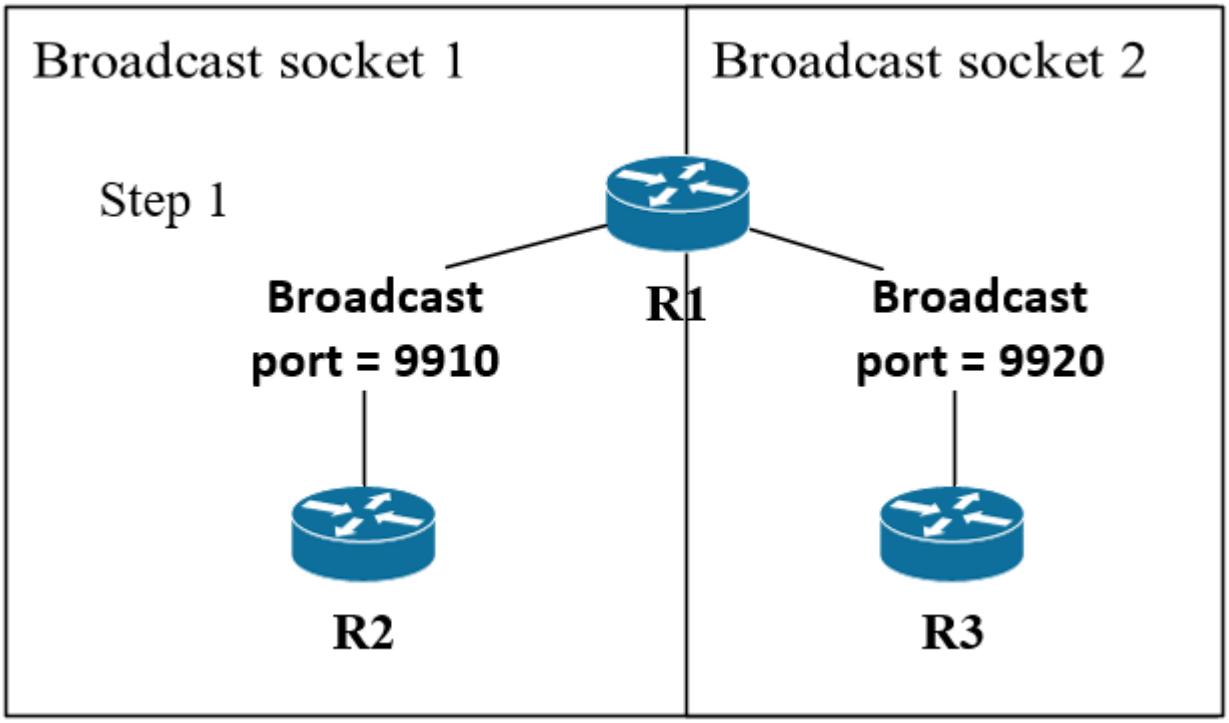

Рис. 3.6 Стан перед початком утворення сокет - з’єднання

2. Далі один з інтерфейсів відправляє на широкомовний сокет повідомлення про те, що він хоче утворити з’єднання.

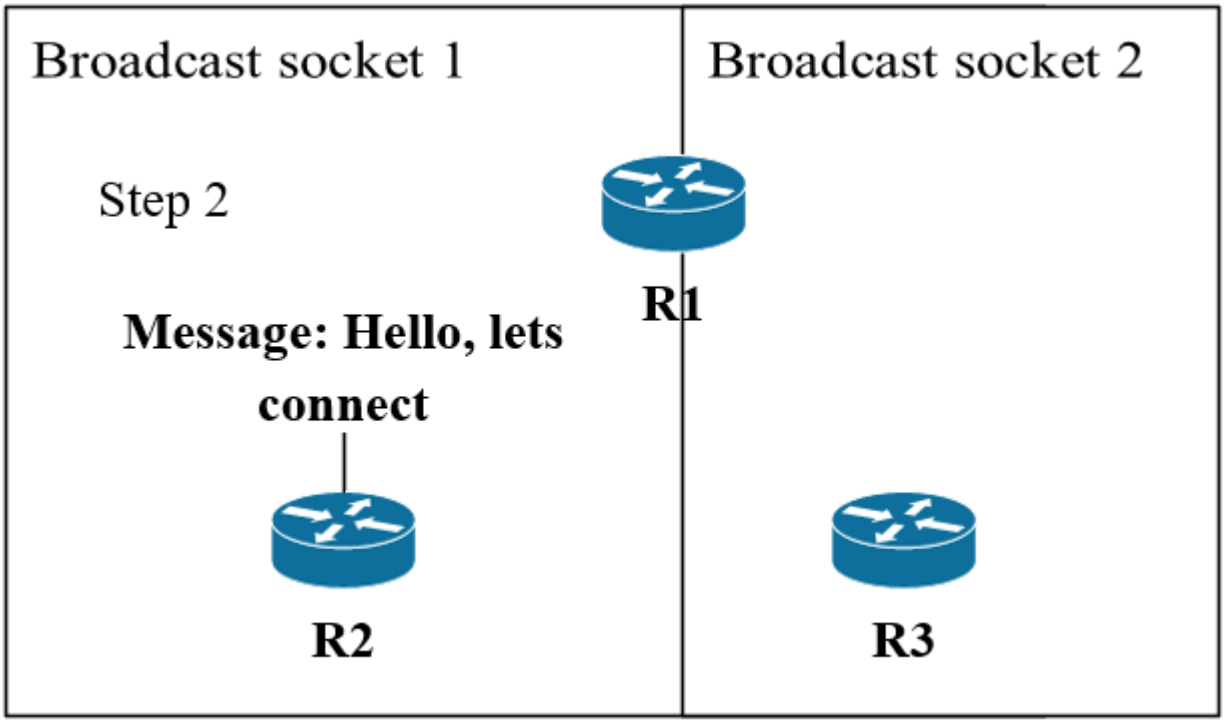

Рис. 3.7 Утворення сокет - з’єднання крок 2

3. Далі інший інтерфейс отримує це повідомлення та якщо він вільний, тобто ще не поєднаний з іншим інтерфейсом він відповідає згодою.

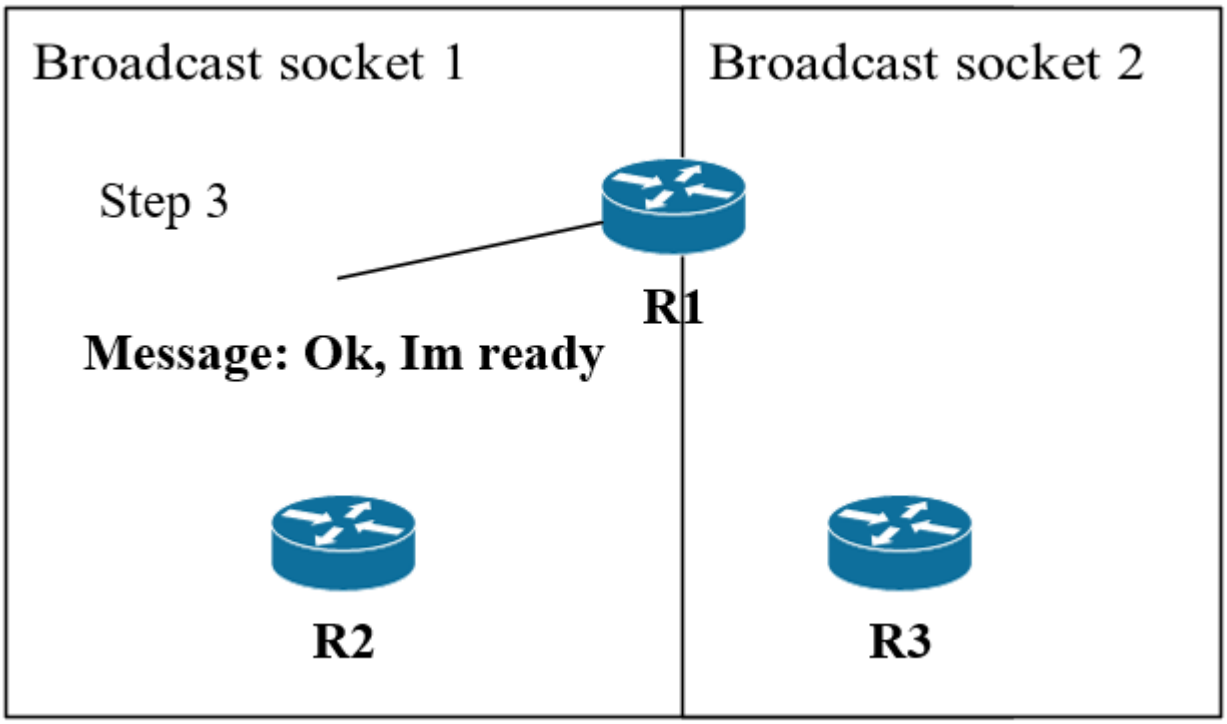

Рис. 3.8 Утворення сокет - з’єднання крок 3

4. Після цього створюється сокет - з’єднання на кожному інтерфейсі. Відтепер ці інтерфейси зв’язані один з одним й можуть обмінюватись даними через створене сокет - з’єднання.

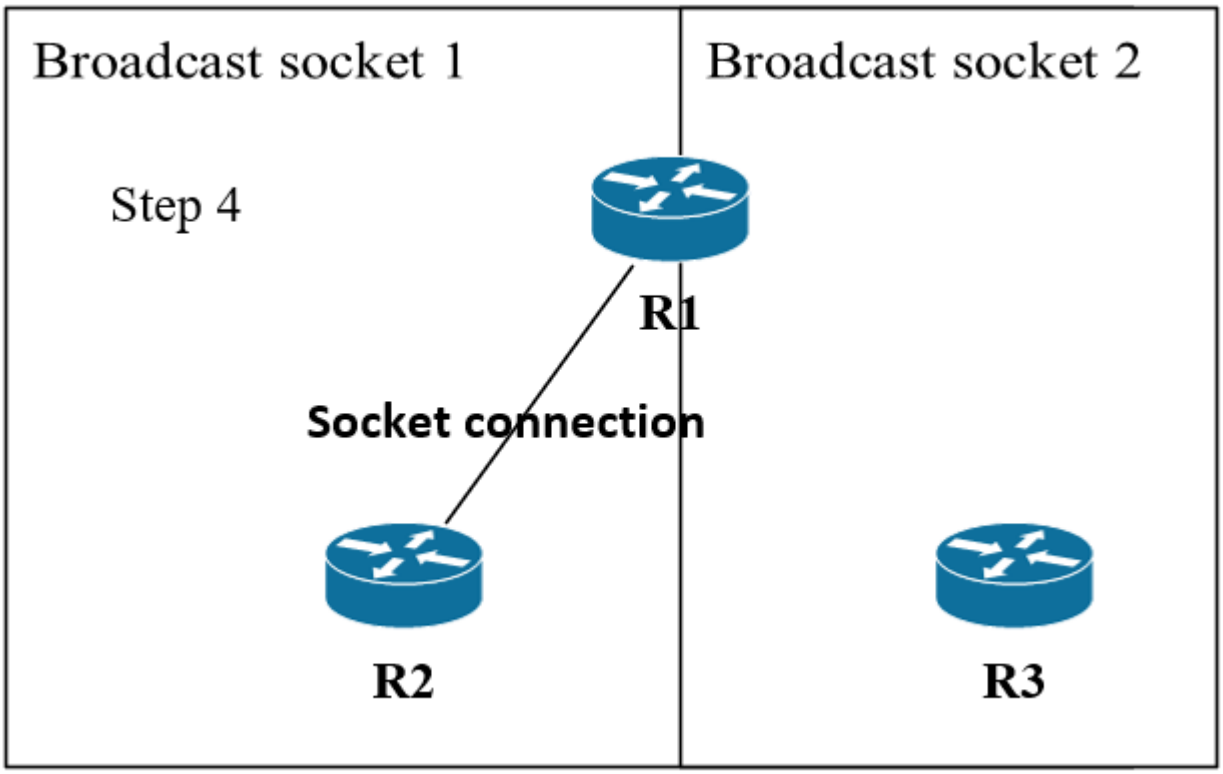

Рис. 3.9 Утворене сокет - з’єднання.

Всі данні що відправляються інкапсулюються у аналог пакетів, що виглядають наступним чином:

- Пакет для звичайного повідомлення через сокет - з’єднання виглядає наступним чином:

data_packet = {"src_ip": "ip", "dst_ip": "ip", "data": "message"}

- Пакет для повідомлення, що буде відправлено на широкомовний порт виглядає наступним чином:

broadcast_packet = {"src_ip": "ip", "src_port": "port", "data": "message"}

Також у протоколах динамічної маршрутизації є можуть бути свої шаблони пакетів для інкапсуляції даних. У симуляції є протокол динамічної маршрутизації для прикладу – RIP. В цьому протоколі данні упаковуються у такий пакет:

rip_packet = {'type': 'RIP', 'routing_table': None}

3.3 Опис маршрутизації

При встановленому з’єднанні між двома інтерфейсами можна відправити повідомлення з конкретного інтерфейсу, тоді це повідомлення буде отримано на іншому інтерфейсі й за необхідності оброблене.[22]

Крім того можна відправити повідомлення на конкретну ip-адресу, тоді маршрутизатор спочатку спробує знайти ip-адресу призначення у списку інтерфейсів, що напряму підключені до інтерфейсів маршрутизатора.

При відправленні даних на ip-адресу маршрутизатор буде куруватись наступною логікою дій:

1. Якщо інтерфейс з ip-адресою призначення напряму підключений до маршрутизатора, він відправить повідомлення з свого інтерфейсу, що підключений до віддаленого інтерфейсу призначення.
2. Якщо маршрутизатор не знайде у списку інтерфейсів, що напряму підключені до інтерфейсів маршрутизатора ip-адресу призначення, він спробує знайти мережу призначення у своїй таблиці маршрутизації.
3. Якщо мережа в якій знаходиться ip-адреса призначення є у таблиці маршрутизації, маршрутизатор відправить повідомлення з порту, номер якого є у таблиці маршрутизації.
4. Якщо ніякою мережі, де міг би знаходитись одержувач немає у таблиці маршрутизації, маршрутизатор повідомить про те, що не може знайти шлях для відправлення пакету.

При отримані даних, які не призначені маршрутизатору, тобто ip-адреса отримувача у пакеті не співпадає з ip-адресою у таблиці ip-адрес всіх інтерфейсів маршрутизатора, дані будуть перепаковані й відправлені за допомогою методу відправки даних на ip-адресу й послідовність дій буде та сама.

В маршрутизаторі існують наступні бази даних (таблиці) для зберігання важливої для роботи маршрутизатора інформації:

1. Таблиця ip-адрес всіх інтерфейсів маршрутизатора. Вона заповнюється автоматично в момент ініціалізації маршрутизатору.

```
self.ip_list = []

for interface in self.interfaces:
    self.ip_list.append(str(interface.get_ip()))
    interface.hostname = self.hostname
    interface.routing_function = self.__parse_interface_data
```

Рис. 3.10 Таблиця ip-адрес маршрутизатора

2. Таблиця потоків обробки. Це не відноситься до саме функцій маршрутизатору, але для збереження всіх запущених у маршрутизаторі циклів існує така таблиця.

3. Таблиця маршрутизації. Найголовніша база даних, звідки маршрутизатор знаходить спосіб дістатись інших пристроїв у мережі.

```
# self.routing_table = {"network": {"gateway": "0.0.0.0", "interface": 0, "metric": 20}}
self.routing_table = {}
```

Рис. 3.11 Таблиця маршрутизації

3.4 Маршрутизація трафіку

Для того щоб вирішити куди відправляти пакети маршрутизатор використовує таблицю маршрутизації. Алгоритм дій для вибору маршруту був розглянутий у попередньому пункті.[23]

Для заповнення такої важливої бази даних як таблиця маршрутизації існує два способи:

1. Статична маршрутизація

Для заповнення таблиці маршрутизації в маршрутизатора викликаються такі методи, як:

- *set_default_route* у який потрібно передати ip-адресу та номер інтерфейсу через який маршрутизатор можливо може отримати доступ до будь якої віддаленої мережі та метрику. Метрику краще передавати більшу ніж звичайна, щоб маршрутизатор спочатку намагався відправити дані у конкретну мережу, якщо така є у таблиці маршрутизації.
- *add_route* у який необхідно передати ip-адресу та маску підмережі для віддаленої мережі, а також ip-адресу й номер інтерфейсу з якого є доступ до віддаленої мережі та метрику.

2. Динамічна маршрутизація за допомогою протоколу динамічної маршрутизації.

Основною метою створення додатку є тестування на основі симуляції нових протоколів динамічної маршрутизації.

На разі не існує додатків, що дозволяли би писати свої алгоритми динамічної маршрутизації на мові програмування Python.

Для підключення свого коду як протоколу динамічної маршрутизації до симуляції мають бути виконані наступні умови:

- Створений клас динамічної маршрутизації має наслідувати абстрактний клас **RoutingProtocol**.

- У створеному класі мають бути два методи, що абстрактно задаються у класі **RoutingProtocol**.

Крім цих умов можна робити все, що в теорії покращить заповнення таблиці маршрутизації.

Можна брати створений клас **RIP** як приклад та на його основі писати свої протоколи.

ВИСНОВКИ

Емуляція мережевих пристроїв та їх функцій дає можливість розробляти та модифікувати власні стандарти, що дає нам більшу гнучкість при проектуванні корпоративних та глобальних мереж.[24] У цій роботі створено метод розробки нових та модифікації існуючих мережевих функцій та протестоване середовище для тестування самостійно створених протоколів динамічної маршрутизації.

В роботі описано повну послідовність роботи симуляції, що дозволяє тестувати своїх протоколи динамічної маршрутизації, написані на мові програмування Python, на будь якій мережі з будь якою кількістю маршрутизаторів та зав’язків між ними.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] Network routing: algorithms, protocols, and architectures / Medhi D., Ramasamy K. San Francisco: Kaufmann Publishers is an imprint of Elsevier, 2007.-824 p.

[2] ISO/IEC 10589 Information technology — Telecommunications and information exchange between systems — Intermediate System to Intermediate System intra-domain routing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473), Second edition 2002-11-15.

[3] RFC 2328 OSPF Version 2, April 1998.

[4] Software-defined networking (SDN): a survey. [Електронний ресурс] - Режим доступу: https://onlinelibrary.wiley.com/doi/epdf/10.1002/sec.1737

[5] Network Functions Virtualisation (NFV). [Електронний ресурс] - Режим доступу: https://www.etsi.org/technologies/nfv

[6] Use Containerlab to emulate open-source routers. [Електронний ресурс] - Режим доступу:
https://www.brianlinkletter.com/2021/05/use-containerlab-to-emulate-open-source-routers/

[7] Creating a simple router simulation using Python and sockets. [Електронний ресурс] - Режим доступу:
https://medium.com/swlh/creating-a-simple-router-simulation-using-python-and-sockets-d6017b441c09

[8] Exploring the Functions of Routing. [Електронний ресурс] - Режим доступу: https://www.learncisco.net/courses/icnd-1/lan-connections/functions-of-routing.html

[9] Use Containerlab to emulate open-source routers. [Електронний ресурс] - Режим доступу: https://www.brianlinkletter.com/2021/05/use-containerlab-to-emulate-open-source-routers/

[10] Kravchenko, Y., Dakhno, N., Leshchenko, O., Tolstokorova, A. “Machine learning algorithms for predicting the results of COVID-19 coronavirus infection”, International conference Information Technology and Interactions, IT&I-2020, CEUR Workshop Proceedings, 2021, 2845, pp. 371–381.

[11] Kravchenko, Y., Afanasyeva, O., Tyshchenko, M., Mykus, S. “Intellectualisation of Decision Support Systems For Computer Networks: Production-Logical F-Inference”, International conference Information Technology and Interactions, IT&I-2020, CEUR Workshop Proceedings, 2021, 2845, pp. 117–126.

[12] Mashkov, V.Task allocation among agents of restricted alliance. In Proc. of the 8th IASTED International Conference on Intelligent Systems and Control, ISC, pp. 13-18, 2005.

[13] Mashkov, V. Restricted alliance and coalitions formation. In Proc. of IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2004, pp. 329-332, 2004.

[14] Yudin, O., Suprun, O., Ziubina, R.,Buchyk, S., Frolov, O., Barannik, N. Efficiency Assessment of the Steganographic Coding Method with Indirect Integration of Critical Information: Proceeding of the International Conference on Advanced Trends in Information Theory (ATIT 2019), Kyiv, Ukraine, pp.36-40.

[15] Savchenko, V., Akhramovych, V., Tushych, A., Sribna, I., Vlasov, I. Analysis of Social Network Parameters and the Likelihood of its Constraction. International Journal of Emerging Trends in Engineering Research. Volume 8 No. 2. 2020. pp. 271–276.

[16] Ivanova, D., Starkova, O. and Herasymenko, K. Realization of the Remote Power Management System Based on the Concept of Internet of Things. In Proceedings of the IEEE Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T), 2016, Kharkov: IEEE Ukraine Section, pp. 96-98.

[17] Polianytsia, A., Starkova, O. and Herasymenko, K. Survey of Hardware IoT platforms. In Proceedings of the IEEE 4th International Scientific-Practical

Conference Problems of Infocommunications Science and Technology, (PIC S and T 2017), Kharkov: IEEE Ukraine Section, 2017, p. 369-371.

[18] Starkova, O., Herasymenko, K. and Babailova, Y. Remote Control Systems of Household Appliances. In Proceedings of the IEEE 4th International Scientific-Practical Conference Problems of Infocommunications Science and Technology, (PIC S and T 2017), Kharkov: IEEE Ukraine Section, 2017, pp. 585-588.

[19] Pliushch, O.G. "Gradient Signal Processing Algorithm for Adaptive Antenna Arrays Obviating Reference Signal Presence," presented at the IEEE International Scientific-Practical Conference PIC S&T, Kyiv, Ukraine, October 8–11, 2019, p. 190.

[20] Dudnik, A., Daria, P., Kobylchuk, M., Domkiv, T., Dahno, N., Leshchenko, O. (2020, November). Intrusion and Fire Detection Method by Wireless Sensor Network. In 2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (ATIT)pp. 211-215.

[21] Leshchenko, O., Trush, O., Dahno, N., Dudnik, A., Kazintseva, K., & Kovalenko, O. (2020, November). Methods for Predicting Adjustments to the Rates of Modern "Digital Money". In 2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (ATIT), pp. 222-226).

[22] V. Bondarenko, "Subjective-probability approach to design an expert system for assessment of states of complex systems in conditions of non-regular destructive influences", 2019 IEEE International Conference on Advanced Trends in Information Theory. 18.12.2019-20.12.2019. Kiev Ukraine. pp. 183-186.

[23] K.Park, K.Lee, S.Park, H.Lee, "Telecommunication node clustering with node compatibility and network survivability requirements" Management Science, vol. 46(3), 2000, pp.363-374.