УДК 004.942 http://doi.org/10.15407/pp2024.02-03.343

Ю. В. Кравченко, К. В. Герасименко, О. В. Старкова, А. Ю. Булгакова

ТЕХНОЛОГІЯ МАРШРУТИЗАЦІЇ НА ОСНОВІ КОНЦЕПЦІЙ ВІРТУАЛІЗАЦІЇ ТА ПРОГРАМНО-КОНФІГУРОВАНИХ МЕРЕЖ

На сьогоднішній день через значне збільшення кількості користувачів, пристроїв, програм і трафіку перед постачальниками послуг постали нові виклики. Парадигма SDN (програмно-конфігурована мережа) виникла для вирішення деяких із цих нових проблем. SDN спрощує керування мережею та дозволяє автоматизовано конфігурувати мережу на вимогу з оптимальним використанням мережевих ресурсів.

Робота присвячена дослідженню сучасних мереж та виявленню можливостей реалізації віртуалізації мережевих функцій у контексті концепцій SDN та NFV, адже це дві нові технологічні тенденції, які трансформують мережевий менеджмент. Разом вони спрощують надання мережевих ресурсів і забезпечують більшу гнучкість мережі. Цей підхід створює динамічні, гнучкі та масштабовані мережі, які використовують віртуалізовану інфраструктуру сучасних центрів обробки даних.

Ключові слова: Метод маршрутизації, статична маршрутизація, динамічна маршрутизація, NFV, SDN

Yu. V. Kravchenko, K. V. Herasymenko, O. V. Starkova, A. Y. Bulgakova

ROUTING TECHNOLOGY BASED ON VIRTUALIZATION SOFTWARE-DEFINED NETWORKING CONCEPTI

One of the main characteristics of the digital revolution is the acceleration of change. The technologies that have fueled the digital revolution over the past decades are experiencing increasingly rapid innovation cycles. For today, a substantial growth of amount of users, devices, applications and traffic has presented new challenges to service providers. The SDN paradigm emerged to address some of these emerging challenges. SDN simplifies network management and allows automated network configuration on demand with optimal use of network resources.

The work is devoted to the present networks research and the identification of opportunities for the implementation of virtualization of network functions in the context of SDN and NFV concepts. Because these are two new technological trends that are transforming network management. Together, they simplify the provisioning of network resources and provide greater network flexibility.

Keywords: Routing method, static routing, dynamic routing, NFV, SDN

Вступ

Мережеві технології розвиваються досить швидкими темпами. Це пов'язано з широким використанням інформаційних технологій та телекомунікаційних систем та мереж в різних сферах людського життя. Зазвичай для виконання своїх функцій на мережевих пристроях реалізовувалися стандартні протоколи та сервіси [1]. Але з часом виявились їхні недоліки та невідповідність сучасним викликам мереж (зростаюча кількість користувачів, сервісів, пристроїв і каналів зв’язку, вимоги до мереж) [2]. Нові тенденції в розвитку мереж також з'явилися в напрямку реалізації концепції програмованих мереж SDN і віртуалізації мережевих функцій. Тож постало завдання вдосконалення існуючих і створення принципово нових протоколів і технологій, здатних повністю задовольнити потреби перспективних інформаційних комунікаційних мереж і систем [3, 4].

Зростаюче використання віртуалізації мережевих функцій є одним із основних факторів, що сприяють прийняттю SDN. Технології SDN і NFV доповнюють одна одну, при цьому NFV пропонує набагато бі-

льше послуг, які обробляються в SDN. Тоді як NFV зосереджується на вдосконаленні фактичних мережевих служб, які регулюють потоки даних, SDN зосереджується на площині керування [5].

Тому метою роботи було розроблення середовища (системи моделювання) для реалізації різноманітних мережевих функцій і протокольних рішень. У майбутньому це дозволить проводити експериментальні дослідження, а в перспективі впроваджувати розроблені технології та протоколи в контексті SDN [2, 3].

1. Технологія маршрутизації на основі концепцій NFV та SDN

В основу роботи покладено маршрутизатор – традиційний мережевий пристрій. Основні функції маршрутизатора – це вибір оптимальних маршрутів у віддалені мережі, заповнення таблиці маршрутизації та пересилання пакетів між мережами [5].

В процесі маршрутизації маршрутизатори розглядають кілька альтернативних маршрутів для досягнення одного пункту призначення. Ці альтернативи є результатом резервування, вбудованого в більшість проєктів мереж.

Для розроблення системи моделювання були використані:

- мова програмування Python;
- модулі ipaddress, threading, socket, json, tabulate (для Python).

Операція моделювання маршрутизаторів складається з багатьох об'єктів, які взаємодіють один з одним [2].

Об’єкт **Simulation** потрібен для зберігання об’єктів маршрутизатора. Він може запускати всі маршрутизатори разом за допомогою методу *start_routers* і виводити таблицю інтерфейсу зі статусами та іншою інформацією про об’єкти інтерфейсу *print(simulation)*.

```
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. 1. Клас Simulation

Маршрутизатор є основним об'єктом моделювання. У ньому зберігаються інтерфейси, необхідні для обміну інформацією між маршрутизаторами. Маршрутизатор також додає до пам’ті таблицю маршрутизації та інші дані про маршрутизатор.

```
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. Клас Router

Об'єкт **Router** має такі публічні методи: *set_protocol*, необхідний для встановлення об'єкта протоколу динамічної маршрутизації, який буде працювати на маршрутизаторі; *has_ip*, потрібний для перевірки наявності певної IP-адреси в роутері, який передано методу; *message_to_interface*, необхідний для надсилання повідомлення з певного інтерфейсу [6]. Потрібно передати номер інтерфейсу, з якого буде відправлено повідомлення, і саме повідомлення; *message_to_ip*, необхідний для надсилання повідомлення на певну IP-адресу. Слід передати IP-адресу і саме повідомлення, яке буде відправлено. Потім маршрутизатор

спробує знайти шлях для надсилання в підключених інтерфейсах або таблиці маршрутизації; *message_to_broadcast*, який потрібен для надсилання повідомлення на широкомовний порт. Усі інтерфейси з однаковою широкомовною адресою отримають це повідомлення; *set_default_route*, який потрібен для встановлення маршруту за замовчуванням. Необхідно передати IP-адресу і омер інтерфейсу, з якого здійснюється доступ до віддаленої мережі, і метрику; start, який необхідний для запуску маршрутизатора. Після цього маршрутизатор починає слухати широкомовний порт і працювати протокол динамічної маршрутизації.

Об’єкт **Interface** — це інтерфейс маршрутизатора, який може взаємодіяти з іншими інтерфейсами для обміну даними.

```
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. 3. Клас Interface

Він має такі методи: *get_ip*, який повертає IP-адресу інтерфейсу; *get_conn*, який повертає об’єкт з’єднання інтерфейсу з іншим інтерфейсом; *connect_to_router*, який запускає серію дій для встановлення з'єднання з іншим інтерфейсом; wait_connection, який починає чекати відповіді від іншого інтерфейсу для підтвердження встановлення з'єднання; accept_connection, який підтверджує підключення до іншого інтерфейсу; *listen_conn*, який очікує встановлення з'єднання з іншим інтерфейсом.

Для встановлення з'єднання між двома інтерфейсами необхідно:

1. Викликати метод інтерфейсу *connect_to_router* на першому інтерфейсі та передати IP-адресу віддаленого інтерфейсу, до якого створюється з’єднання [7, 8]. Після цього перший інтерфейс очікує на встановлення з'єднання.

номер інтерфейсу, з якого здійснюється доступ до мережі, і метрику. Метрика потрібна, щоб знайти найшвидший спосіб надіслати повідомлення; *add_route*, необхідний для встановлення маршруту до певної мережі [2-6]. Потрібно відправити IP-адресу та маску підмережі для віддаленої мережі, а також IP-адресу та н

2. Викликати метод *accept_connection* на другому інтерфейсі, до якого підключений перший інтерфейс, і передати адресу першого інтерфейсу, який хоче підключитися, і його порт для підключення інтерфейсів *(local_port)* [4-8].

RoutingProtocol – це абстрактний клас для створення протоколів динамічної маршрутизації, він визначає основні методи, які повинні бути присутніми у всіх протоколах динамічної маршрутизації.

RIP – це об’єкт, створений для демонстрації організації та роботи протоколів динамічної маршрутизації в моделюванні. Він має наступні методи: *new_message*, необхідний для обробки отриманого повідомлення, адресованого протоколу динамічної маршрутизації; *get_routing_table_to_send*, який необхідний для обробки таблиці маршрутизації для її подальшої відправки іншим маршрутизаторам; *send_route*, необхідний для відправки таблиці маршрутизації, підготовленої методом *get_routing_table_to_send*, з усіх інтерфейсів маршрутизатора (всім сусідам по мережі); бігун, який потрібен для відправлення повідомлень кожні n секунд; start, необхідний для запуску протоколу для роботи. Після виконання запускається цикл за допомогою методу runner, який надсилає таблицю маршрутизації всім сусідам маршрутизатора в мережі кожні n секунд.

2. Тестування роботи розробленої технології маршрутизації

Для старту симуляції, потрібно створити всі необхідні об'єкти. Щоб створити списки з об’єктів інтерфейсу, потрібно передати такі параметри, як номер інтерфейсу, порт інтерфейсу, порт ширини

смуги інтерфейсу та об’єкт **IPv4Interface** для запису IP-адреси.

Далі потрібно викликати метод об’єкта симуляції, який створить і додасть маршрутизатори до симуляції. Для цього вам потрібно передати назву маршрутизатора та інтерфейсів, створених на попередньому кроці [4-8]. Можна створити скільки завгодно маршрутизаторів, але потрібно подумати про створення інтерфейсів, тому що симуляція буде працювати правильно лише з правильно створеними параметрами, коли жодні дані, крім широкомовного порту, не повторюються між інтерфейсами.

Після цього ми запускаємо всі маршрутизатори і викликаємо метод *message_to_broadcast* на всіх маршрутизаторах. Це необхідно для автоматичного підключення інтерфейсів з одним і тим же широкомовним портом один до одного.

Для обміну інформацією між маршрутизаторами симуляція використовує програмний інтерфейс для забезпечення обміну даними між процесами (сокети) [4-8].

Для роботи сокетів потрібні мережеві порти, тому під час симуляції їх потрібно вказати для кожного інтерфейсу. Сокети використовуються замість другого рівня моделі OSI і дозволяють створювати сокет-з'єднання між двома програмними об'єктами та обмінюватися даними між ними [4-8].

В об’єкті інтерфейсу є широкомовні сокети, які слухають і передають дані на один порт.

Під час створення з’єднання між двома інтерфейсами ми створюємо сокет між портами цих інтерфейсів – з’єднання, яке дозволяє надсилати дані з будь-якого з цих двох інтерфейсів і отримувати їх на іншому інтерфейсі.

Процес підключення інтерфейсів до розетки – підключення може виглядати так:

1. Є кілька маршрутизаторів, деякі з інтерфейсів яких використовують один і той самий широкомовний порт. Як у прикладі з наступного рисунка.

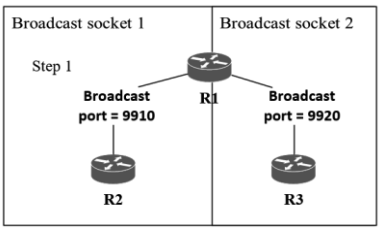

Рис. 4. Стан перед початком утворення сокетів – з’єднань

2. Далі один із інтерфейсів надсилає повідомлення широкомовному сокету про те, що він хоче встановити з’єднання.

3. Далі інший інтерфейс отримує це повідомлення і, якщо він вільний, тобто ще не підключений до іншого інтерфейсу, відповідає згодою.

4. Після цього створюється з’єднання на кожному інтерфейсі (сокет). Відтепер ці інтерфейси з’єднані один з одним і можуть обмінюватися даними через створений сокет – з’єднання.

Усі надіслані дані інкапсулюються в аналогові пакети, які виглядають так:

– Пакет для звичайного повідомлення через сокет - з'єднання виглядає так:

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

– Пакет для широкомовного повідомлення, виглядає так: broadcast_packet = {"src_ip": "ip", "src_port": "port", "data": "message"}

У протоколі динамічної маршрутизації RIP дані упаковані в такий пакет:

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

Коли з'єднання встановлено між двома інтерфейсами, повідомлення може бути надіслано з певного інтерфейсу, потім це повідомлення буде отримано на іншому інтерфейсі та, за необхідності, оброблено [4-8].

На даний момент немає додатків, які б дозволяли писати власні алгоритми динамічної маршрутизації мовою програмування Python. Для підключення власного коду як протоколу динамічної маршрутизації до симуляції, мають бути виконані наступні умови: створений клас динамічної маршрутизації має успадковуватись від абстрактного класу RoutingProtocol; створений клас повинен мати два методи, які абстрактно

визначені в класі RoutingProtocol. Можна взяти створений клас **RIP** як приклад і написати свої протоколи на його основі.

Практичне дослідження розробленої технології маршрутизації

Практичне значення розробленої технології маршрутизації полягає в можливості впровадження результатів моделювання (математичного, імітаційного) за допомогою розробленої технології. Це дозволяє практично реалізувати перспективні методи маршрутизації, не змінюючи реальні протокольні рішення (що мають досить багато недоліків і не відображають особливостей сучасних мереж, тобто є застарілими).

Як приклад моделі маршрутизації можна використати вирішення задачі маршрутизації з метрикою протоколу RIP у формі задачі булевого програмування з використанням засобів пакета Matlab [9].

Постановка задачі маршрутизації виглядає наступним чином. Дано:

- кількість вузлів у мережі *(m)*;
- кількість каналів зв'язку в мережі *(n)*;
-
- вузол-відправник пакетів;
- вузол-одержувач пакетів;
- інтенсивність трафіку *(r)*, що надходить до мережі;
- пропускні здатності каналів зв'язку;
- метрики каналів зв'язку *(f)*.

Необхідно визначити:

- "найкоротший" шлях від вузла-відправника до вузла-одержувача через канал зв'язку модельованої мережі в межах обраної метрики;
- сформувати залежність кількості шляхів, що використовуються в процесі маршрутизації, від щільності трафіку, що надходить до мережі;
- пропускну здатність напрямку зв'язку від вузла-відправника до вузла-одержувача відповідно до обраної моделі маршрутизації.

Опис задачі маршрутизації як задачі булевого програмування, використовуючи метрики протоколу RIP v.1

Кількість каналів зв'язку в мережі (n) визначає розмір вектора x, координати $x_{i,j}$ якого частку трафіку в каналах зв'язку між i-м та j-м вузлами. Розмір метричного вектора f також відповідає кількості каналів зв'язку в мережі (n), а його координати $f_{i,j}$ характеризують метрику каналів зв'язку між i-м та j-м вузлами.

Для реалізації маршрутизації на координати вектора x накладаються наступні обмеження

$$x_{i,j} \in \{0,1\} \quad (i,j = \overline{1,m}\,; \quad i \neq j), \qquad (1)$$

тобто змінна $x_{i,j}$ може приймати тільки два значення:

- 1, якщо трафік проходить через канал (i,j);
- 0, в іншому випадку.

Булеве значення змінної (1) гарантує відсутність відхилень у потоках вздовж мережевого шляху, тобто, весь трафік передається по єдиному шляху.

У першому випадку кожному каналу зв'язку присвоюється метрика, рівна 1 $\left(f_{i,j} = 1,; \quad i,j = \overline{1,m}; \quad i \neq j\right)$, що відповідає знаходженню шляху з мінімальною кількістю ретрансляцій (*fval*) між заданою парою вузлів, як у протоколіv.1.

У процесі розв'язання задачі маршрутизації необхідно забезпечити виконання умов захисту потоку для кожного вузла мережі і для мережі в цілому

$$\begin{cases} \sum_{j:(i,j)} x_{i,j} - \sum_{j:(j,i)} x_{j,i} = 1 \ - \text{ для вузла-відправника} \\ \sum_{j:(i,j)} x_{i,j} - \sum_{j:(j,i)} x_{j,i} = 0 - \text{для транзитних вузлів} \\ \sum_{j:(i,j)} x_{i,j} - \sum_{j:(j,i)} x_{j,i} = -1 \quad - \text{для вузла-одержувача} \end{cases} \qquad (2)$$

Крім умови збереження потоку (2) повинна також виконуватися умова уникнення перевантаження тракту зв'язку

$$r \cdot x_{i,j} \leq c_{i,j} \quad i,j = \overline{1,n}, i \neq j, \quad (3)$$

де $c_{i,j}$ – пропускна здатність каналу зв'язку між i-м та j-м вузлами.

Розв'язання задач маршрутизації в сучасних мережних протоколах, як правило, зводиться до розв'язання задачі пошуку найкоротшого шляху в мережі, яка визначає структуру системи зв'язку. Задача пошуку найкоротшого шляху в мережі формується як задача булевого програмування, для її розв'язання використовувався інструментарій «Optimization Toolbox» па-

кета MatLab, представлений підпрограмою «bintprog».

За змістом рівняння (1), під час розв'язання задачі булевого програмування необхідно мінімізувати цільову функцію, виражену у лінійній формі $\min_x f^t x$ при виконанні ряду умов, представлених у вигляді обмежень рівнянь і нерівностей

$A \cdot x \le b; \quad Aeq \cdot x = beq$, де *f*, *x*, *b*, *beq* – вектори, *A* і *Aeq* – матриці відповідної розмірності.

Середовище MatLab допускає наступні основні синтаксичні варіанти команд виклику підпрограми «bintprog»:

1. $[x, fval] = bitprog(f, A, b)$ – розв'язок $fval = \min_x f^t x$ при $A \cdot x \le b$.

2. $[x, fval] = bitprog(f, A, b, Aeq, beq)$ – розв'язок $fval = \min_x f^t x$ при $Aeq \cdot x = beq$. У випадку, якщо умов у вигляді нерівностей немає, то *A*=[] та *b*=[].

Для опису задачі маршрутизації у формалізмі середовища MatLab умова збереження потоку (2) повинна бути виражена у вигляді векторної матриці: $Aeq \cdot x = beq$. Таким чином матриця *Aeq* має розмірність $m \times n$, а її координати набувають значення $\{-1; 0; 1\}$ наступним чином ($j = \overline{1, m}, \quad i = \overline{1, n}$):

$a_{ji} = 1$, якщо *i*-й канал зв'язку йде від із *j*-го вузла;

$a_{ji} = -1$, якщо *i*-й канал зв'язку входить у *j*-й вузол;

$a_{ji} = 0$, якщо *i*-й канал зв'язку не супроводжує *j*-й вузл.

Розмір вектора *beq* відповідає кількості вузлів у мережі (*m*), а його координати генеруються наступним чином ($j = \overline{1, m}$):

$beq_j = 1$, якщо *i*-й вузол є відправником пакетів;

$beq_j = -1$, якщо *i*-й вузол є одержувачем пакетів;

$beq_j = 0$, якщо *i*-й вузол є ретранслятором.

Умови (3) також повинні бути виражені у вигляді векторної матриці з нерівністю $A \cdot x \le b$.

Приклад метричної задачі маршрутизації для протоколу RIP v.1, складеної та розв'язаної в середовищі Matlab.

Нехай структура мережі та пропускна здатність каналів зв'язку показані на рис.5. Загальне число вузлів у мережі дорівнює п'яти (m = 5), а кількість каналів зв'язку – шести (n = 6). Тоді вузол-відправник пакетів – 1, а вузол-приймач – 5.

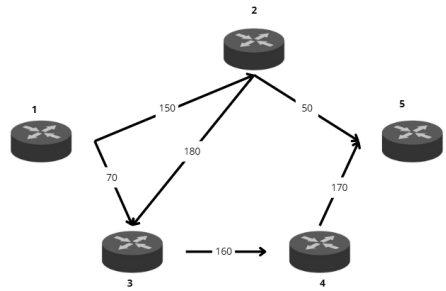

Рис. 5. Приклад змодельованої мережевої структури

Сформуємо вектор *x* і вектор метрик *f*:

$$x = \begin{bmatrix} x_{1,2} \\ x_{1,3} \\ x_{2,3} \\ x_{2,5} \\ x_{3,4} \\ x_{4,5} \end{bmatrix} \quad f = \begin{bmatrix} f_{1,2} \\ f_{1,3} \\ f_{2,3} \\ f_{2,5} \\ f_{3,4} \\ f_{4,5} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Сформулюємо умови збереження потоку вузлів мережі (2):

$$\begin{cases} x_{1,2} + x_{1,3} = 1; \\ -x_{1,2} + x_{2,3} + x_{2,5} = 0; \\ -x_{1,3} - x_{2,3} + x_{3,4} = 0; \\ -x_{3,4} + x_{4,5} = 0; \\ -x_{2,5} - x_{4,5} = -1. \end{cases}$$

Сформуємо матрицю *Aeq* і вектор *beq*:

$$Aeq = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 & 0 & -1 \end{bmatrix}; \quad beq = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}.$$

Сформалізуємо умови запобігання перевантаженню каналів зв'язку (3):

$$\begin{cases} r \cdot x_{1,2} \le c_{1,2}; \\ r \cdot x_{1,3} \le c_{1,3}; \\ r \cdot x_{2,3} \le c_{2,3}; \\ r \cdot x_{2,5} \le c_{2,5}; \\ r \cdot x_{3,4} \le c_{3,4}; \\ r \cdot x_{4,5} \le c_{4,5}. \end{cases}$$

Сформуємо матрицю A і вектор b

$$A = \begin{bmatrix} r & 0 & 0 & 0 & 0 & 0 \\ 0 & r & 0 & 0 & 0 & 0 \\ 0 & 0 & r & 0 & 0 & 0 \\ 0 & 0 & 0 & r & 0 & 0 \\ 0 & 0 & 0 & 0 & r & 0 \\ 0 & 0 & 0 & 0 & 0 & r \end{bmatrix}; \quad b = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}.$$

У наведеному вище прикладі текст програми виглядає так, як показано на рис.6.

Результати розрахунків наводяться у вікні команд (рис.6). Згідно з результатами розрахунків, "довжина" найкоротшого шляху дорівнює 2 (*fval* = 2), а $x_{1,2} = 1$ і $x_{2,5} = 1$, тому можна зробити висновок, що він проходить через вузли 1, 2 та 5 (рис. 7). На рис.7 показано інтенсивність трафіку, що проходить через канал зв'язку.

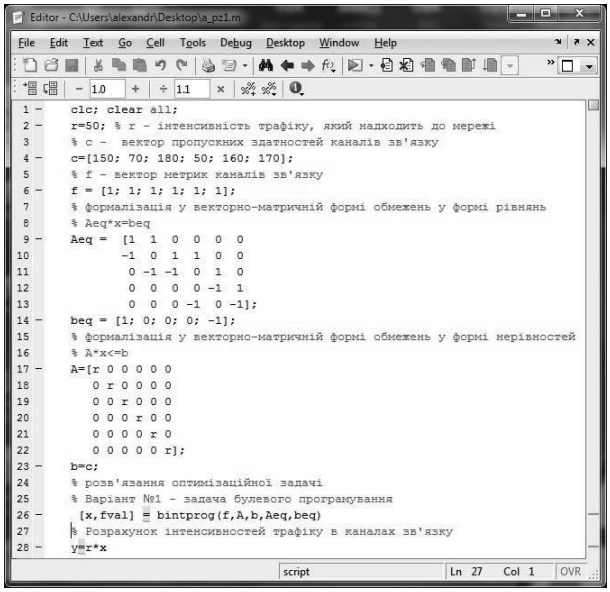

Рис. 6. Текст програми у вікні редагування M-file Editor

В результаті необхідно побудувати графік кількості шляхів (*K*), що використовуються у процесі маршрутизації, як функції від інтенсивності трафіку, що надходить до мережі (*r*). На графіку також необхідно вказати, які шляхи використовуються при якій інтенсивності трафіку. Також варто експериментально визначити пропускну здатність у напрямку зв'язку від вузла-відправника до вузла-одержувача відповідно до реалізованої моделі маршрутизації.

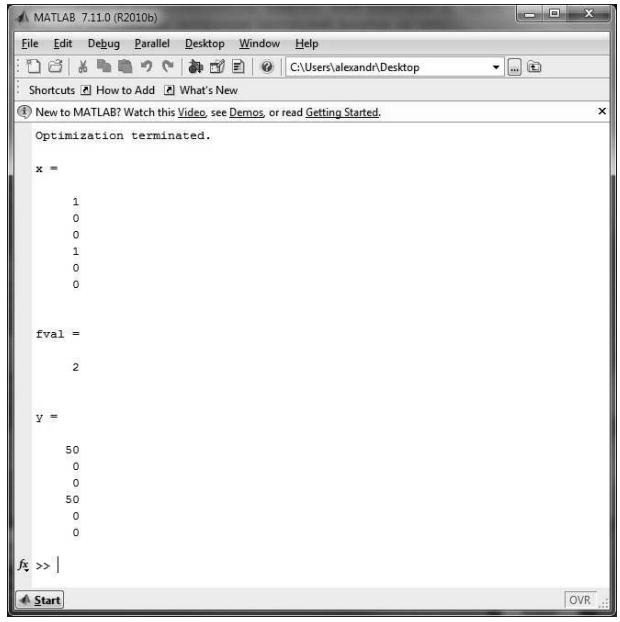

Рис. 7. Результати розрахунків у вікні команд «Comand Window»

Реалізація вказаної моделі маршрутизації за допомогою математичного пакету MatLab та розробленої технології показали аналогічні результати маршрутизації.

Висновки

На завершення можна зробити висновок, що наукові та технічні досягнення в SDN приносять користь як підприємствам центрів обробки даних, так і дослідникам, які пропонують нові мережеві механізми та протоколи, дозволяючи їм проводити більш реалістичні експерименти.

Емуляція мережевих пристроїв та їхніх функцій дозволяє розробляти та змінювати наші власні стандарти, що дає нам більше гнучкості у проєктуванні корпоративних та глобальних мереж. У цій статті було створено метод розробки нових і модифікації існуючих мережевих функцій, а також перевірене середовище для тестування самостійно створених протоколів динамічної маршрутизації.

У статті описана повна послідовність роботи з моделювання, яка дозволяє перевірити протоколи динамічної маршрутизації, написані мовою програмування Python, у будь-якій мережі з будь-якою кількістю маршрутизаторів і з'єднань між ними. Також вказана модель на основі протоколу RIP була реалізована за допомогою

математичного пакету MatLab, що підтвердило результати маршрутизації, отримані за допомогою розробленої технології маршрутизації на основі мови програмування Python.

Література

1. Network routing: algorithms, protocols, and architectures / Medhi D., Ramasamy K. San Francisco: Kaufmann Publishers is an imprint of Elsevier, 2007. – 824 p.
2. Software-defined networking (SDN): a survey. [web source] - Access mode: https://onlinelibrary.wiley.com/doi/epdf/10.1002/sec.1737
3. Network Functions Virtualization (NFV). [web source] - Access mode: https://www.etsi.org/technologies/nfv
4. Creating a simple router simulation using Python and sockets. [web source] - Mode of access: https://medium.com/swlh/creating-a-simple-router-simulation-using-python-and-sockets-d6017b441c09
5. Exploring the Functions of Routing. [web source] - Access mode: https://www.learncisco.net/courses/icnd-1/lan-connections/functions-of-routing.html
6. Use Container lab to emulate open-source routers. [web source] – Access mode: https://www.brianlinkletter.com/2021/05/use-containerlab-to-emulate-open-source-routers/
7. Starkova, O., Herasymenko K., Nikolchev K., Bulgakova A. Virtualization And Programmability In Modern Networks In The Context Of SDN Concept. 2022 IEEE 4rd International Conference on Advanced Trends in Information Theory, ATIT 2022 – Proceedings, pp. 204 - 207.
8. Starkova, O., Herasymenko K., Nikolchev K., Kravchenko O., Bulgakova A. Implementation Of Advanced Routing Methods Based On The SDN Concept And OS Linux. 2022 IEEE 4rd International Conference on Advanced Trends in Information Theory, ATIT 2022 – Proceedings, pp. 244 - 248.
9. М.В. Семеняка, О.В. Лемешко, О.В. Старкова «Методичні вказівки до лабораторних робіт з дисципліни "Системи комутації та розподілу інформації. Частина 2"», Харків: ХНУРЕ, 2014. 92 с.

Одержано: 12.02.2024
Внутрішня рецензія отримана: 19.02.2024
Зовнішня рецензія отримана: 08.03.2024

Про авторів:

[1]*Кравченко Юрій Васильович*,
Доктор технічних наук,
професор
https://orcid.org/0000-0002-0281-4396

[2]*Герасименко Костянтин Васильович*,
Кандидат технічних наук,
https://orcid.org/0000-0002-9545-5272

[3]*Старкова Олена Володимирівна*,
Кандидат технічних наук,
https://orcid.org/0000-0001-8985-2442

[4]*Булгакова Анна Юріївна*,
студентка

Місце роботи авторів:

Київський національний університет
ім. Тараса Шевченка,
факультет інформаційних технологій
Тел. (+38) (044) 481-45-07
E-mail: fit@knu.ua,
Сайт: https://fit.knu.ua/