НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»
УДК _____

До захисту допущено:
Завідувач кафедри
_____Сергій СТІРЕНКО
«___»_____2023 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Комп'ютерні системи та мережі»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Спосіб балансування навантаження в програмно-конфігурованій мережі за допомогою генетичного алгоритму»

Виконав:
студент VI курсу, групи ІО-11мн
Фризюк Микола Олександрович _____

Керівник:
проф., д.т.н., проф.,
Кулаков Юрій Олексійович _____

Консультант з нормоконтролю:
проф., д.т.н., проф.,
Жабін Валерій Іванович _____

Рецензент:
доц. каф. ІСТ, к.т.н., доц.,
Коган Алла Вікторівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – другий (магістерський)
Спеціальність – 123 «Комп'ютерна інженерія»
Освітньо-наукова програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____ 2023 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Фризюку Миколі Олександровичу

1. Тема дисертації «Спосіб балансування навантаження в програмно-конфігурованій мережі за допомогою генетичного алгоритму», науковий керівник дисертації Кулаков Юрій Олексійович, проф., д.т.н., затверджені наказом по університету від « 20 » березня 2023 р. № 1275-с
2. Термін подання студентом дисертації _____
3. Об’єкт дослідження – рівномірний розподіл навантаження в програмно-конфігурованій мережі
4. Предмет дослідження – генетичний алгоритм для покращення балансування навантаження в програмно-конфігурованій мережі
5. Перелік завдань, які потрібно розробити: дослідити існуючі рішення для балансування навантаження, обрати інструменти для розробки та мову програмування, розробити генетичний алгоритм та застосувати в програмно-конфігурованій мережі, оцінити результати роботи алгоритму.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: загальнотематичні ілюстрації по темі дисертації.

7. Орієнтовний перелік публікацій

8. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	Жабін В.І. д.т.н., професор		
1	Кулаков Ю.О. д.т.н., професор		
2	Кулаков Ю.О. д.т.н., професор		
3	Кулаков Ю.О. д.т.н., професор		
4	Кулаков Ю.О. д.т.н., професор		

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	Затвердження теми роботи	15.01.2023-05.02.2023	
2	Вивчення та аналіз завдання	06.02.2023-12.02.2023	
3	Постановка та формалізація задачі	13.02.2023-19.02.2023	
4	Розробка інформаційного та програмного забезпечення	20.02.2023-26.02.2023	
5	Проведення експериментальних досліджень	27.02.2023-05.03.2023	
6	Оформлення пояснювальної записки	06.03.2023-12.03.2023	
7	Передзахист		
8	Захист		

Студент Микола ФРИЗЮК

Науковий керівник Юрій КУЛАКОВ

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Спосіб балансування навантаження в програмно-конфігурованій мережі за допомогою генетичного алгоритму

студентом: Фризюком Миколою Олександровичем

Робота містить вступ та чотири основні розділи. Загальний обсяг роботи: 90 сторінок основного тексту, 20 ілюстрацій, 9 таблиць та 6 формул. У списку літературних джерел наведено 41 джерело, що було використано для підготовки роботи.

Актуальність. Програмно-конфігурована мережа (SDN з англ. Software-defined Networking) – це нова парадигма в комп'ютерних мережах, яка дозволяє подолати обмеження традиційних мереж. SDN передбачає відділення пристроїв керування від пристроїв передачі даних. Таким чином відбувається централізація керування, де контролер (яким може бути звичайний мережевий пристрій) спілкується та встановлює таблиці потоків на комутаторах за допомогою протоколу OpenFlow (OF). Контролер має всі дані про OF-комутатори та інші мережеві пристрої у своїй мережі, а тому може ефективно визначати маршрути для потоків (flow, у OpenFlow це певний маршрут, що встановлений на OF-комутаторі за яким передаються пакети без участі контролера, а сам flow встановлює контролер). Саме тому для визначення найкращого маршруту потрібен ефективний алгоритм, який виконуватиме розподіл навантаження (балансування навантаження) в мережі з метою мінімізації заторів та затримки передачі, а також дозволить збільшити пропускну здатність. Балансування навантаження необхідне в таких галузях як високопродуктивні розподілені обчислення, обчислення в дата центрах, інфраструктурні хмарні сервіси, аналіз у Big Data, а також набуло застосування в концепції інтернету речей. Саме тому необхідний алгоритм балансування, який матиме властивості: рівномірного розподілу навантаження в мережі, швидкої адаптації до зростання навантаження на

певних вузлах і матиме відносно низьку обчислювальну складність на контролері, для того щоб не вносити додаткової затримки в мережу. Порівняно з аналогічними методами, балансування навантаження з використанням генетичного алгоритму показало приріст продуктивності мережі, зниження затримки та її коливання.

Мета і задачі дослідження. Через те, що сучасні мережі стають все більш популярними, вони також стають все більш складними, зростає складність конфігурації мережі згідно з встановленими правилами. Тому і підтримка мереж та їх обслуговування ускладнюються. Для впровадження QoS (якості сервісів), інженерії трафіку чи балансування навантаження потрібно використовувати низькорівневі команди які залежать від кожного окремого виробника. А оскільки сервіс керування навантаженням життєво-необхідний для ефективного функціонування більшості мереж, то використовується виділене обладнання для цієї задачі. Для боротьби із цими обмеженнями та складністю адаптації мережі, було запропоновано Програмно-Конфігуровану Мережу (SDN). В даній парадигмі контролююча логіка була відокремлена від пристроїв передачі даних, таких як комутатори чи роутери. Це дало те, що комутатори виконують лише роль передачі даних, а контролер має централізоване управління мережею та має інформацію про його комутатори. Таким чином SDN контролер може знаходити найкоротші шляхи, та повідомляти про них OpenFlow (OF) комутатори по даному протоколу. SDN має якісний набір інструментів для керування мережею, але для того щоб робити це ефективно, потребує сервісу балансування навантаження, який буде використовуватись для розподілу робочого навантаження по ідентичним маршрутам з метою зниження затримки та підвищення продуктивності мережі. Метою даної роботи є створення генетичного алгоритму балансування навантаження в SDN, який вирішує такі проблеми: перевантаження або недовикористання з’єднань в мережі (проблема заторів), обмеження у використанні комутаторів (деякі комутатори перевантажені, поки інші простоюють), а також обмеження кількості записів

у таблиці потоків комутаторів (оскільки операція запису в таблицю потоків займає багато часу), що вимірюється як «ціна операції» (кількість доданих або прибраних записів з таблиці). Для вирішення описаних проблем, алгоритм повинен уникати «вузьких місць» в мережі, і також алгоритм має швидко виявляти комутатор або з'єднання яке являється «вузьким місцем» для їх уникнення. Основна задача при пошуку оптимального шляху це знайти баланс між «ціною операції» та довжиною цього шляху.

Методи дослідження. Було використано теоретичний метод для синтезування даної задачі в SDN, як математичної моделі. Враховуючи обмежену кількість комутаторів, щоб оптимізувати використання з'єднань в мережі, обчислюється сума використань з'єднань на певному шляху, що визначається як довжина шляху або «ціна шляху». Для того, щоб обрати найменш навантажений шлях в режимі реального часу, запропонована модель одразу обчислює інтегроване навантаження шляхів отримуючи інформацію від SDN контролера. Таким чином контролер час-від-часу передає інформацію про завантаження кожного шляху балансувальнику навантаження, а останній повертає найкращий шлях назад до контролера.

Також було застосовано емпіричний метод – було проведено експеримент із реалізованим генетичним алгоритмом та використанням фреймоврку Ryu (Ryu - OpenFlow фреймворк для розробки SDN застосунків, os-ken його підтримувана версія) та MiniNet для створення тестової топології. Було проаналізовано результати експерименту та проведено їх демонстрацію на графіках. Серед розглянутих метрик для оцінювання алгоритму були використані: затримка (latency) обходу мережі, джитер (від англ. jitter) – варіація затримки при передачі пакетів тривалим потоком та пропускна здатність мережі (network throughput) – швидкість успішної доставки пакетів у мережі. Отримані результати були отримані за допомогою середніх значень 10 експериментів.

Наукова новизна одержаних результатів. Хоча наукова галузь SDN відносно нова, було проведено багато дослідження на цю тему. Для

вирішення проблеми балансування навантаження в SDN було створено багато алгоритмів, які спираються на певні метрики чи особливості мереж. Дані алгоритми мають як свої недоліки, так і переваги. Наприклад метод балансування з використанням мережних утиліт, які вимірюють затримку та споживання енергії пристроїв, для побудови оптимального шляху. Або інший метод, де використовувався алгоритм із ієрархічною структурою контролюючої площини, тобто багатьма контролерами для розподілення їхніх ресурсів. Ще один метод який був запропонований під назвою QoS-орієнтований Алгоритм Балансування Навантаження, який передбачав аналіз завантаженості сусідніх пристроїв та приблизне обчислення їх пропускної здатності. На відміну від інших, метод балансування навантаження в SDN за допомогою генетичного алгоритму використовує підхід Генетичного Програмування, а оскільки він виконується прямо на обчислювальному обладнанні без будь-якого втручання, виконується швидко. Через свою природу випадкового розподілу, генетичний алгоритм дозволяє уникати локальних оптимумів, і таким чином знаходити оптимальні рішення в складних мережах. Це також дозволяє ефективне маштабування мережі. Отримані результати показують, що реалізований алгоритм має переваги над порівняними у затримці та пропускній здатності.

Практичне значення одержаних результатів. Практичні результати дослідження, тобто результати роботи методу балансування навантаження в SDN за допомогою генетичного алгоритму були отримані з використанням OpenFlow-фреймворку Ryu на топології MiniNet і можуть бути відтворені на реальній топології мережі, наприклад у високонавантаженій мережі датацентру або іншій звичайній мережі з підтримкою SDN. Даний алгоритм створений у вигляді мережевого застосунку на основі фреймворку Ryu, та може бути використаний на SDN-контролері потрібної мережі.

Особистий внесок магістранта. Магістрантом була створена конкретна реалізація методу балансування навантаження в SDN за допомогою генетичного алгоритму із використанням технолігій OpenFlow

фреймворку Ryu та проведено тестування на топології MiniNet, отримані результати тестування та створені графіки і таблиці що відображають ці результати.

Ключові слова

Програмно-Конфігурована Мережа, Генетичний алгоритм, OpenFlow, Ryu, Балансування навантаження, MiniNet

ABSTRACT

for a master's thesis

performed on the topic: Method of load balancing in a software defined network based on a genetic algorithm

by the student: Fryziuk Mykola Oleksandrovych

The work contains an introduction and four main chapters. The total volume of the work: 90 pages of the main text, 20 illustrations, 9 tables and 6 formulas. The list of literary sources includes 41 sources that were used to prepare the work.

Relevance of the research topic. Software-defined networking (SDN) is a new paradigm in computer networks that overcomes the limitations of traditional networks. SDN involves the separation of control devices from data transmission devices. In this way, control is centralized, where a controller (which can be a regular network device as well) communicates and sets flow tables on switches using the OpenFlow (OF) protocol. The controller has all the data about OF-switches and other network devices in its network, and therefore can effectively determine the routes for flows (flows, in OpenFlow it is a certain route set on a OF-switch along which packets are transmitted without the participation of the controller, and the flow itself sets the controller). That is why determining the best route requires an effective algorithm that will perform load distribution (load balancing) in the network in order to minimize congestion and latency, as well as increase throughput. Load balancing is necessary in such fields as high-

performance distributed computing, computing in data centers, infrastructure cloud services, analysis in Big Data, and has also gained application in the concept of the Internet of Things. That is why a balancing algorithm is needed, which will have the following properties: uniform distribution of the load in the network, quick adaptation to the growth of the load on certain nodes, and will have a relatively low computational complexity on the controller, in order not to introduce additional delay into the network. Compared to similar methods, load balancing using a genetic algorithm showed an increase in network performance, a decrease in latency and jitter.

The purpose and objectives of the research. Because modern networks are becoming more and more popular, they are also becoming more complex, increasing the complexity of configuring the network according to established rules. Therefore, the support of networks and their maintenance become more difficult. To implement QoS (quality of services), traffic engineering or load balancing, you need to use low-level commands that depend on each individual manufacturer. And since the load management service is vital for the efficient functioning of most networks, dedicated equipment is used for this task. To combat these limitations and the complexity of network adaptation, Software-defined Networks (SDN) has been proposed. In this paradigm, the control logic was separated from data transmission devices such as switches or routers. This resulted in the fact that the switches perform only the role of data transfer, and the controller has centralized management of the network and has information about its switches. In this way, the SDN controller can find the shortest paths and report them to OpenFlow (OF) switches using this protocol. SDN has a good set of tools for managing the network, but to do it effectively, it needs a load balancing service that will be used to distribute the workload over identical routes in order to reduce latency and improve network performance. The purpose of this work is to create a genetic algorithm for load balancing in SDN, which solves the following problems: overloading or underutilization of connections in the network (congestion problem), limitations in the use of switches (some switches are overloaded while

others are idle), as well as limiting the number of entries in flow tables of switches (since the operation of writing to the flow table takes a long time), which is measured as the "cost of the operation" (the number of entries added or removed from the table of a switch). To solve the described problems, the algorithm must avoid "bottlenecks" in the network, and also the algorithm must quickly detect a switch or connection that is a "bottleneck" in order to avoid them. The main task in finding the optimal path is to find a balance between the "price of the operation" and the length of this path.

Research methods. A theoretical method was used to synthesize this problem in SDN as a mathematical model. Given a limited number of switches, in order to optimize the use of connections in the network, the sum of the connection usages on a given path is calculated, which is defined as the path length or "path cost". In order to choose the least loaded path in real time, the proposed model immediately calculates the integrated load of the paths by receiving information from the SDN controller. In this way, the controller periodically sends information about the load of each path to the load balancer, and the load balancer returns the best path back to the controller. An empirical method was also applied - an experiment was conducted with the implemented genetic algorithm and the use of the Ryu framework (Ryu - OpenFlow framework for developing SDN applications, os-ken is its maintained version) and MiniNet to create a test topology. The results of the experiment were analyzed and their demonstration was carried out on graphs. Among the considered metrics for evaluating the algorithm, the following were used: network bypass latency, jitter (from the English jitter) - delay variation during the transmission of packets in a long stream, and network throughput - the speed of successful packet delivery in the network. The obtained results were obtained using the average values of 10 experiments.

Scientific novelty of the obtained results. Although the scientific field of SDN is relatively new, a lot of research has been done on the topic. To solve the problem of load balancing in SDN, many algorithms have been created that rely on certain metrics or features of networks. These algorithms have both their cons and

pros. For example, a balancing method using network utilities that measure the delay and energy consumption of devices to build an optimal path. Or another method where an algorithm was used with a hierarchical control plane structure, i.e., many controllers to allocate their resources. Another method that was proposed was called the QoS-oriented Load Balancing Algorithm, which involved the analysis of the load of neighboring devices and the approximate calculation of their bandwidth. Unlike others, the genetic algorithm load balancing method in SDN uses a Genetic Programming approach, and since it is performed directly on the computing hardware without any intervention, it is performed quickly. Due to its nature of random distribution, the genetic algorithm allows avoiding local optima, and thus finding optimal solutions in complex networks. It also allows efficient network scaling. The obtained results show that the implemented algorithm has advantages over comparable ones in delay and throughput.

Practical significance of the obtained results. The practical results of the study i.e., the performance results of the genetic algorithm load balancing method in SDN, were obtained using the Ryu OpenFlow framework on the MiniNet topology and can be reproduced on a real network topology, for example, in a highly loaded data center network or other ordinary SDN-enabled network. This algorithm is created as a network application on the Ryu framework platform, and can be used on the SDN controller of the desired network.

Personal contribution of the master's student. The graduate student created a specific implementation of the load balancing method in SDN using a genetic algorithm using OpenFlow technologies of the Ryu framework and conducted testing on the MiniNet topology, obtained test results and created graphs and tables displaying these results.

Keywords

Software-Defined Network, Genetic algorithm, OpenFlow, Ryu, Load Balancing, MiniNet

Пояснювальна записка

до магістерської дисертації

на тему: «Спосіб балансування навантаження в програмно-конфігурованій мережі за допомогою генетичного алгоритму»

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – (англ. application programming interface) Інтерфейс програмування додатків для використання в зовнішніх програмних продуктах

CP – (англ. control plane) Рівень управління мережею в SDN

DFS – (англ. depth first search) Алгоритм пошуку в глибину

DP – (англ. data plane) Рівень передачі даних в SDN

GA – (англ. genetic algorithm) Генетичний алгоритм

LB – (англ. load balancing) Балансування навантаження

MAC-адреса – (англ. Media Access Control) Унікальний ідентифікатор мережевого обладнання.

Mininet – Програмне забезпечення для симуляції мережі та мережевих операцій

North-bound API – Канал передачі OpenFlow повідомлень між мережевим додатком та контролером

OF – (англ. openflow) Мережевий протокол для зв’язку з комутаторами та маршрутизаторами, який реалізує технологію SDN

QoS – (англ. quality of service) Протокол якості обслуговування, інтелектуальний розподіл трафіку в залежності від його типу

RTT – (англ. round trip time) Метрика для оцінювання продуктивності мережі

RYU – Фреймворк для програмування мережевих додатків OpenFlow

SDN – (англ. software-defined network) Програмно-конфігурована мережа

South-bound – Інтерфейс зв’язку рівня управління та рівня передачі даних OF

STP – (англ. spanning tree protocol) Мережевий протокол, який використовується для уникнення циклів топології при передачі пакетів

ЦОД – Центр обробки даних (датацентр)

ВСТУП

Балансування навантаження – це процес розподілу мережевого трафіку між кількома ресурсами, такими як сервери, комутатори або інші мережеві пристрої, для того щоб гарантувати, що жоден ресурс не буде перевантажений трафіком, а інші залишатимуться недостатньо використаними. У програмно-конфігурованих мережах балансування навантаження є важливим механізмом, який допомагає оптимізувати використання мережевих ресурсів і покращити продуктивність мережі.

Проблема балансування навантаження в програмно-конфігурованих мережах виникає через динамічний і складний характер сучасних комп'ютерних мереж. У міру того, як до мережі додається все більше пристроїв і служб, шаблони трафіку та навантаження на мережу можуть значно відрізнятися та швидко змінюватися. Крім того, різні пристрої та служби можуть мати різні вимоги до ресурсів і пропускної здатності, що ускладнює ефективний розподіл трафіку [1].

З метою вирішення даної проблеми, алгоритми балансування навантаження використовуються для розподілу трафіку між ресурсами мережі таким чином, щоб максимізувати використання ресурсів, мінімізувати час відгуку та забезпечити високу доступність і надійність. Однак розробка ефективного алгоритму балансування навантаження для мереж, налаштованих програмним забезпеченням, може бути складним завданням, яке передбачає врахування таких факторів, як топологія мережі, моделі трафіку, використання ресурсів, затримка мережі тощо. Крім того, алгоритми балансування навантаження повинні мати можливість адаптуватися до мінливих умов мережі та забезпечувати ефективне та справедливе використання ресурсів. Для вирішення подібних задач оптимізації природньо пристосовані генетичні алгоритми [2].

Генетичний алгоритм — це популярна техніка оптимізації, яка використовується в інформатиці та ґрунтується на принципах еволюції та природного відбору. Під час балансування навантаження генетичний алгоритм можна використовувати для пошуку оптимальної конфігурації мережевих ресурсів, яка мінімізує перевантаження мережі, максимізує використання ресурсів і зменшує час відповіді.

Підхід генетичного алгоритму до балансування навантаження має кілька переваг, до них входять: здатність обробляти складні, нелінійні зв’язки між мережевими ресурсами та здатність адаптуватися до мінливих умов мережі. Однак він також має певні обмеження, такі як обчислювальна складність для обробки великих популяцій рішень і важкість вибору відповідної функції пристосованості та найкращих параметрів.

В даній роботі будуть розглянуті способи зменшення негативного впливу наведених обмежень на результати застосування генетичного алгоритму в програмно-конфігурованій мережі. А також найкращі практики для застосування генетичного алгоритму на графі, що відображає топологію мережі.

Метою дослідження є розробка ефективного алгоритму балансування навантаження, який може оптимізувати використання мережевих ресурсів, зменшити перевантаження мережі та покращити час відгуку, адаптуючись до мінливих умов мережі. Використання генетичних алгоритмів забезпечує новий і перспективний підхід до балансування навантаження, який може обробляти складні та нелінійні зв’язки між мережевими ресурсами та адаптуватися до мінливих умов мережі.

Загалом, робота має на меті сприяти розробці ефективного і результативного методу балансування навантаження, який може допомогти забезпечити оптимальне використання мережевих ресурсів і підвищити продуктивність програмно-конфігурованої мережі.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ В ОБЛАСТІ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В ПРОГРАМНО-КОНФІГУРОВАНИХ МЕРЕЖАХ

1.1 Програмно-конфігурована мережа

Основне призначення програмно-конфігурованої мережі (SDN) – абстрагувати мережеві додатки від апаратного забезпечення, яке виконує маршрутизацію в мережі. А також ефективне встановлення правил передачі трафіку в мережі за допомогою програмного забезпечення, яке працює на централізованому рівні. Мережні комутатори та маршрутизатори можуть приймати власні рішення. Вони можуть локально керувати своїми таблицями маршрутизації. Управління пересиланням трафіку направляється протоколами спільної панелі керування, тобто контролером.

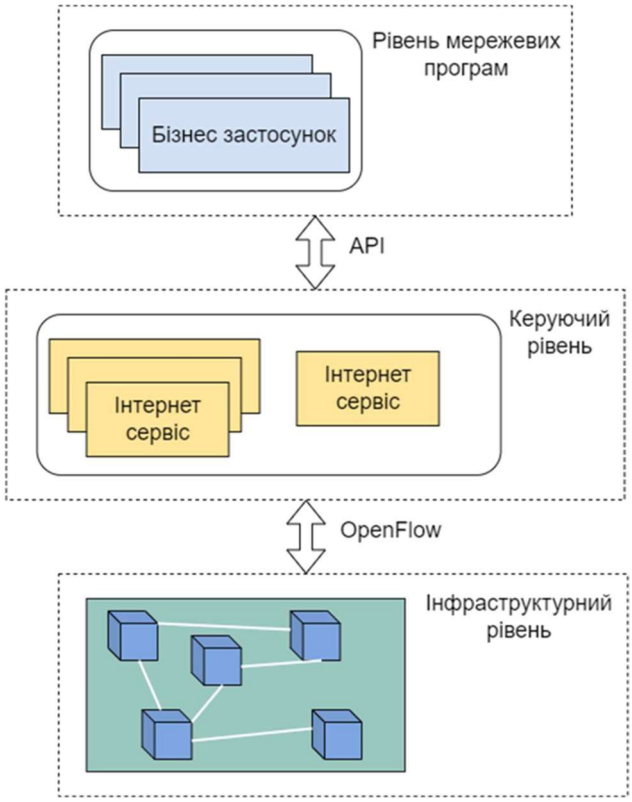


Рис. 1.1 – Програмно-конфігурована мережева (SDN) архітектура

Традиційні мережеві протоколи не адаптуються до різних ситуацій динамічно. Якщо вони повинні працювати разом, мережеві компоненти в домені пересилання повинні відповідати подібним правилам відповідно до загальних стандартів. Керування відокремлене від рівня даних (DP). Натомість в SDN у централізованому контролері присутній рівень управління (CP), який керує та встановлює правила, де в мережі є з'єднання з хостами, і як виглядає топологія цієї мережі, що під'єднується до всіх хостів разом. Центральний контролер, який доступний скрізь, дозволяє мережевим інженерам використовувати ефективні методи маршрутизації, керовані тільки програмним забезпеченням, що працює на ньому.

Архітектура SDN привернула значну увагу завдяки своїй здатності вирішувати недоліки програмування в традиційних мережевих архітектурах, що, у свою чергу, також уможливило прості та швидкі методи мережі. інновації. SDN забезпечує відокремлення рівня даних (DP) від рівня управління (CP) і, крім того, полегшує програмні реалізації складних мережевих програм. Управління менш специфічним і дешевшим апаратним забезпеченням програмними додатками за допомогою стандартизованих інтерфейсів стало реальністю. Більше того, з'явилась можливість динамічного додавання нових функцій до мережі у вигляді додатків для роботи в мережі при досягненні більшої гнучкості [3]. Ця ж концепція відома з операційних систем мобільних телефонів, оскільки програми можна додавати в систему динамічно.

Архітектура SDN зображена на рис. 1.1, який складається з трьох шарів. Рівень даних (DP) — це найнижчий рівень, також відомий як рівень інфраструктури. Цей рівень складається з елементів мережі пересилання. Рівень даних в основному відповідає за пересилання даних, а також моніторинг всієї інформації, що зберігається локально, і збір статистики. Рівень над рівнем даних — це рівень керування (CP), який також відомий як площина керування. Площина пересилання керується та програмується за допомогою площини керування. Інформація, що надається площиною пересилання,

використовується для визначення мережевих операцій і маршрутизації. Він складається з одного або кількох програмних контролерів, які можуть спілкуватися з елементами мережі пересилання за допомогою стандартизованих інтерфейсів, також відомих як southbound інтерфейси.

- Балансування навантаження в мережі

Доступність і адаптивність мережевих серверів і їх додатків, таких як веб-засоби, брандмауер, протокол передачі файлів (FTP), проксі, віртуальна приватна мережа та інші критично важливі сервери, покращується за рахунок балансування мережевого навантаження. Балансування мережевого навантаження також гарантує, що трафік в мережі перенаправляється на всі хости, що залишилися в мережі, якщо будь-який з хостів у кластері несподівано вийде з ладу. Кластер балансування мережевого навантаження можна масштабувати до потрібної кількості серверів. Це допомагає без проблем працювати над критичними операціями [4]. Типова конфігурація балансування мережевого навантаження показана на рис. 1.2.

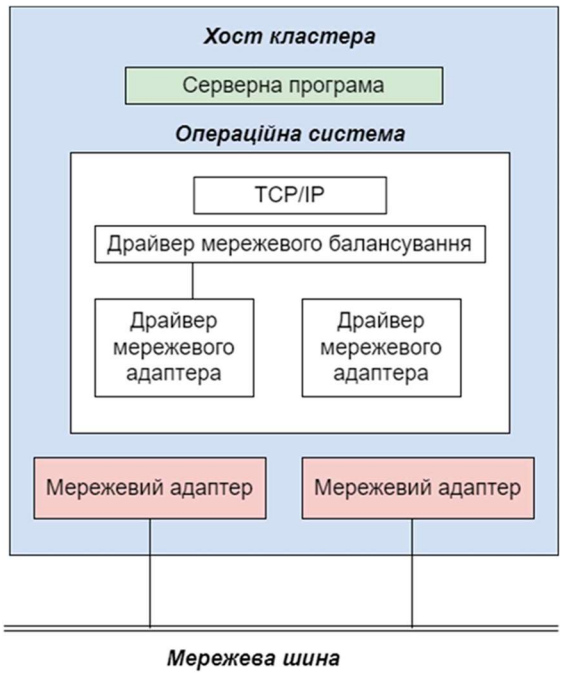


Рис. 1.2 – Типова конфігурація хоста балансування навантаження

1.2 Базові способи балансування навантаження

Існують різноманітні методи і алгоритми, які можна використовувати для балансування навантаження, пов’язаного із запитами доступу, зробленими клієнтами в різних пулах серверів. Зазвичай методи, згадані нижче, використовуються у комбінації, щоб знайти сервер, який найкраще підходить для задоволення нових запитів [5]. Далі розглянемо кілька алгоритмів та способів, які використовуються балансувальниками навантаження [6].

- Round Robin

Метод балансування навантаження Round Robin — це один із найпростіших методів розподілу клієнтських запитів між групою серверів. Переходячи вниз по списку серверів у групі, Round Robin балансувальник навантаження пересилає запит клієнта кожному серверу по черзі. Коли він досягає кінця списку, балансувальник навантаження повертається назад і знову йде вниз по списку (надсилає наступний запит до першого сервера в списку, наступний — до другого сервера і так далі). Основна перевага Round Robin балансування навантаження полягає в тому, що його надзвичайно просто реалізувати. Однак це не завжди призводить до найбільш точного або ефективного розподілу трафіку, тому що багато циклічних балансувальників навантаження припускають, що всі сервери однакові: на даний момент працюють, обробляють однакове навантаження та мають однакову пам’ять і обчислювальну потужність [7].

- Least Connection

Round Robin, а також Weighted Round Robin, не враховують поточний стан завантаження сервера під час розподілу запитів. На відміну від них, метод найменшого підключення (Least Connection) враховує поточне навантаження на сервер. Потім запит надсилається на сервер. який обслуговував найменшу кількість сеансів, які були активні на той час. При використанні методу

Weighted Least Connection, подібно до методу Weighted Round Robin, кожному серверу надається номер. Він використовується балансувальником навантаження під час розподілу запитів сервера. Тоді якщо припустити, що число активних з'єднань однакові на двох серверах, тоді сервер з більшою вагою буде отримувати новий запит [8].

- Агентно-орієнтоване адаптивне балансування навантаження

У пулі серверів кожному серверу призначається агент, який повідомляє балансувальнику навантаження про поточне навантаження. Ця інформація в реальному часі використовується, щоб вирішити, який сервер краще розмістити для обробки запитів. Метод використовується в поєднанні з іншими методами, такими як Weighted Round Robin і Weighted Least Connection.

- Chained Failover (Fixed Weighted)

Цей метод вимагає попередньо визначеного порядку серверів, які потрібно налаштувати в ланцюжку, в якому вони присутні. Першому серверу в ланцюжку надсилаються всі запити. Наступному серверу в ланцюжку надсилаються всі запити тільки, якщо є шанс що не прийматиметься більше запитів, потім третій сервер і так далі.

- Зважений час відгуку (Weighted Response Time)

У цьому методі інформація від серверів про час відповіді постійно оновлюється, щоб перевірити, чи відповідає сервер достатньо швидко в порівнянні з середнім часом відгуку за певний певний період часу.

Наступний запит на доступ до сервера надсилається на даний сервер. Це має гарантувати, що якщо який-небудь сервер вже має справу з великим навантаженням і вже почав повільно відповідати, він не отримає жодних нових запитів. Це дозволяє рівномірно розподіляти навантаження на доступний пул серверів [9].

- Хешування IP відправника

У цьому методі використовується алгоритм, який враховує IP-адреси відправника та клієнта призначення й сервера, які потім об’єднує їх для

створення унікального хеш-ключа. Отриманий ключ потім використовується для розподілу клієнтів на будь-який конкретний сервер. Оскільки ключі завжди можна відновити, у разі розриву сеансу, метод балансування хеш- завантаження вихідного IP може переконатися, що запити від клієнта спрямовуються на той самий сервер, який використовувався раніше. Це корисно, оскільки гарантує, що клієнт може підключитися повернутися до сеансу після відключення, якщо сеанс все ще активний [10].

- Адаптивний Software Defined Networking (SDN)

Цей метод використовує комбінацію знань верхніх мережевих рівнів разом з інформацією про стан мережі на нижніх рівнях. Щоб прийняти рішення про те, як розподілити запити, інформація про дані з мережевих рівнів 4 і 7 поєднується з даними з мережевих рівнів 2 і 3. Це дає змогу отримати інформацію про стан сервера, статус програм, що працюють у мережі, мережеву інфраструктуру, статуси відмов та рівень перевантаженості мережі, щоб їх використовувати у процесі прийняття рішень щодо балансування навантаження [11].

1.3 Керування мережевим трафіком

Керування мережевим трафіком відноситься до процесу перехоплення та аналізу мережевого трафіку, а також спрямування трафіку на оптимальні ресурси на основі пріоритетів. Ключові компоненти, які необхідно відстежувати для кращого керування мережею, включають продуктивність мережі, трафік та безпеку. Інструмент контролю мережевого трафіку використовує такі методи управління, як моніторинг пропускної здатності та продуктивності мережі, моніторинг шаблонів трафіку для виявлення та

запобігання вузьким місцям, аналіз безпеки мережі та оптимізацію для забезпечення оптимального функціонування мережі. Він допомагає максимально підвищити продуктивність і безпеку мережі, запобігаючи перевантаженням мережі та загрозам. Мережі вміщують все більш складний набір даних у трафіку. Визначення типу трафіку допомагає ефективно оптимізувати мережу, основні типи наведені в табл. 1.1.

Таблиця 1.1

Основні типи мережевого трафіку

№	Тип трафіку	Приклад	Проблема	Рішення
1	Burst-трафік	Завантаження FTP, графічного, відеоконтенту	Споживає високу пропускну здатність і сповільнює програми	Встановити обмеження пропускної здатності
2	Інтерактив-ний трафік	SSL-транзакції, миттєві повідомлення, сеанси Telnet	Чутливий до конкуренції за пропускну здатність і призводить до низького часу відгуку в системі	Встановити пріоритет над менш важливим трафіком
3	Трафік, чутливий до затримок	Потокові програми, передача голосу по IP, відеоконференції	Сприйнятливий до конкуренції за пропускну здатність і призводить до низького часу відгуку	Встановити мінімальний і максимальний діапазон пропускної здатності на основі пріоритету

Продовження таблиці 1.1

№	Тип трафіку	Приклад	Проблема	Рішення
4	Трафік не в реальному часі	Електронна пошта, програми пакетної обробки	Споживає пропускну здатність у робочий час	Планувати пропускну здатність у неробочий час

Сучасний інтернет-трафік значно відрізняється від того, який був в Інтернеті за його перших днів. Оскільки трафік експоненційно зростає з постійно зростаючою кількістю веб-сторінок, повнометражними фільмами вмережі, програмними додатками та онлайн-іграми, на сцені виникає потреба в ефективному управлінні мережевим трафіком [12].

Згідно з опитуванням, проведеним на вибірці з близько 2 мільйонів клієнтів широкосмугового доступу, HTTP-трафік споживає 46%, а P2P-трафік – 37% пропускної здатності мережі. Тільки YouTube споживає 20% всього HTTP-трафіку, або більше половини всього потокового HTTP-відео.

1. У звіті Cisco Systems Inc передбачала, що близько 60% всього IP-трафіку споживачів буде створюватися комерційними відеосервісами або IPTV у 2008 році. У ньому також зазначається, що відеосервіси високої чіткості та високошвидкісний широкосмуговий доступ будуть отримувати IP-трафік майже вдвічі кожен раз. 2 роки до 2011 року.
2. Deloitte & Touche у своїй оцінці передбачила, що глобальний трафік перевищить пропускну здатність Інтернету вже в цьому році.

З величезним збільшенням голосових дзвінків в Інтернеті та потоком бізнесу вартістю мільярди доларів через звичайні мережі Інтернет-провайдерів, перевантаження мережі або затор стане все більш поширеним явищем. Це ускладнює життя користувачів в Інтернеті. Через велику кількість серйозних проблем в Інтернеті швидкість завантаження зупиниться, а Інтернет більше не

надаватиме гарантовану якість обслуговування або безпеку, яку вимагають поточні програми.

Процес вимірювання та контролю трафіку на мережевому каналі, щоб уникнути переповнення каналу, називається керуванням пропускною здатністю. Це допомагає підтримувати швидке та безперебійне підключення до Інтернету. Щоб керувати пропускною здатністю, трафік вимірюється, аналізується і визначається причина інтенсивного трафіку. Потім використовується інструмент керування мережевим трафіком, щоб уникнути небажаного трафіку та запланувати використання пропускної здатності [13].

Через різні аспекти лінії зв'язку не досягнуть максимальної пропускної здатності. Використання каналу має бути нижче максимальної теоретичної ємності каналу, щоб можна було забезпечити швидке реагування шляхом усунення черг із вузькими місцями на кінцевих точках пересилання.

Проблеми, які обмежують продуктивність пересилання:

1. Пропускна здатність з'єднання визначається TCP через лавину, доки пакети не будуть скинуті.
2. Вища затримка створюється через черги в маршрутизаторах.
3. Коли мережа досягає своєї пропускної здатності, глобальна синхронізація TCP призводить до втрати пропускної здатності.
4. Інтенсивному трафіку потрібна вільна пропускна здатність
5. Немає належної підтримки явного сповіщення про перевантаження
6. Керування чергою контролюється постачальниками послуг Інтернету

Вимірювання трафіку – аналізатори пакетів переглядають окремі пакети та допомагають відстежувати складні проблеми. Але вони об'ємні і вимагають знання мережевих протоколів. Таким чином, інструменти вимірювання трафіку використовуються для ширшого уявлення про обсяг і тип трафіку в певній мережі. Деякі з інструментів включають:

- Caligare Flow для моніторингу та виявлення мережевих аномалій NetFlow

- Exbander Precision для моніторингу та аналізу
- FireBeast для управління пропускною здатністю та формування трафіку.
- PRTG для моніторингу використання пропускної здатності.
- Розумні мережеві рішення Sandvine для вимірювання та керування мережевим трафіком

Формування трафіку – дія щодо набору пакетів, що накладає на ці пакети додаткову затримку, щоб можна було контролювати трафік у мережі для оптимізації та гарантованої продуктивності, називається формуванням трафіку. Це допомагає контролювати обсяг трафіку, надісланого в певний період. Зазвичай він застосовується на краях мережі для контролю надходження трафіку, але іноді він застосовується у джерелі або елементом мережі.

- Обмеження швидкості – метод контролю швидкості трафіку, який надсилається або приймається через мережевий інтерфейс, називається обмеженням швидкості. Коли трафік менше або дорівнює вказаній швидкості, він надсилається, а коли він перевищує вказану швидкість, він припиняється або затримується. Виконується наступними способами.
- Policing – відкидає зайві пакети і може застосовуватися до будь-якого мережевого протоколу, включаючи IPv6.
- Черга – затримує передачу пакетів і може застосовуватися до будь-якого мережевого протоколу, включаючи IPv6.
- Контроль перевантаження – він маніпулює механізмом перевантаження протоколу і може застосовуватися до TCP.

Переваги використання програмного забезпечення для контролю мережного трафіку:

1. Запобігання збоїв: дає вам змогу бачити дані про продуктивність вашої мережі в режимі реального часу, а також допомагає виявляти та запобігати потенційним проблемам і відключенням.

2. Швидко та ефективно вирішуйте проблеми: полегшує вирішення проблем, виявляючи вузькі місця в пропускній здатності та інші проблеми мережі в режимі реального часу, включаючи нерегулярні зміни поведінки трафіку та зміни конфігурації.
3. Можливість керувати змінами в мережі: допомагає адаптуватися до зростання та змін у вашій мережі, відстежувати їх, щоб виявити ненормальні коливання або загрози, прогнозувати поведінку трафіку та планувати вимоги, щоб забезпечити відповідність вимогам вашої мережі.
4. Можливість визначати загрози безпеці: забезпечує необхідний рівень безпеки за допомогою аналізу поведінки мережі, щоб визначити звичайний сплеск трафіку від загрози безпеці або щось таке серйозне, як вторгнення нульового дня.

Хоча більшість інструментів керування мережевим трафіком у реальному часі включають функції моніторингу пропускної здатності та оптимізації, їхні рішення обмежуються лише цим. Одним з найважливіших аспектів управління мережевим трафіком, який майже завжди не помічають, є мережева безпека та аналіз поведінки. Відсутність надійного рішення безпеки на додаток до моніторингу мережевого трафіку може зробити вашу мережу вразливою для атак. Ідеальний інструмент керування мережевим трафіком повинен:

1. Забезпечувати видимість мережі в режимі реального часу.
2. Аналізувати поведінкові моделі та активно керувати трафіком.
3. Швидко та ефективно діагностувати та усувати несправності.
4. Захищати мережу від атак нульового дня та внутрішніх загроз.

1.4 Огляд існуючих комерційних рішень

На ринку доступно кілька комерційних рішень SDN, які надають різноманітні функції та можливості для керування та контролю мережі. Ось кілька прикладів:

1) Інфраструктура Cisco Application Centric (ACI): Cisco ACI — це програмно-визначене мережеве рішення, яке забезпечує централізоване керування та автоматизацію мережевої інфраструктури.
2) VMware NSX — це мережева віртуалізація та платформа безпеки, яка забезпечує мікросегментацію, розподілений брандмауер і можливості балансування навантаження. Вона дозволяє адміністраторам керувати та захищати мережевий трафік у кількох віртуальних і фізичних середовищах.
3) Juniper Contrail — це SDN-рішення, яке забезпечує централізоване керування та оркестровку мережевих ресурсів. Він включає платформу віртуалізації мережі, яка дозволяє адміністраторам створювати віртуальні мережі та надавати мережеві ресурси на вимогу.
4) NEC ProgrammableFlow — це рішення SDN, яке забезпечує централізований контролер і комутатори, які підтримують OpenFlow. Він включає платформу віртуалізації мережі, яка дозволяє адміністраторам створювати віртуальні мережі та надавати мережеві ресурси на вимогу.

Дані комерційні рішення SDN пропонують різні переваги, такі як підвищена гнучкість мережі, покращена безпека та спрощене керування мережею. Вони також можуть інтегруватися з існуючою мережевою інфраструктурою та забезпечити шлях до автоматизації та оркестровки мережі. Однак вибір відповідного рішення SDN залежить від конкретних потреб і вимог

організації, тому важливо оцінити характеристики та можливості кожного рішення перед прийняттям рішення [14].

- NetFlow Analyzer

Контроль і керування мережевим трафіком мають вирішальне значення, щоб забезпечити максимальну віддачу від мережі. Повне рішення для управління мережевим трафіком, як-от NetFlow Analyzer, значно полегшує розуміння того, що відбувається у мережі. Окрім забезпечення пріоритетності додатків, важливих для бізнесу, запобігаючи тим, що некритичні програми перевантажують пропускну здатність, NetFlow Analyzer пропонує ряд інших переваг, включаючи захист мережі від перебоїв, допомагає швидше виправляти та вирішувати проблеми, передбачати та запобігати можливі вузькі місця, керувати мережі, що розвивається, ефективніше ідентифікувати загрози безпеці і, як наслідок, підвищити ефективність системи [15].

Існує багато причин збоїв або стрибків трафіку у мережі, включаючи людські помилки та загрози безпеці. Відомості про активність мережевого трафіку в режимі реального часу мають важливе значення, щоб допомогти проактивно відстежувати й виявляти будь-які проблеми, які можуть вплинути на продуктивність мережі. Менеджер мережевого трафіку NetFlow Analyzer дає повний огляд мережі з понад 50 настроюваними звітами та графіками. Він відображає чітке уявлення про те, що відбувається у мережі, з різними показниками, як-от використання пропускної спроможності на пристрій і додаток, найпопулярніші користувачі та розмови, затримка та тремтіння. NetFlow Analyzer також допомагає встановлювати порогові сповіщення на основі обсягу, швидкості та використання, а також сповіщає електронною поштою, SMS або квитанцією служби підтримки.

Диспетчер мережевого трафіку, відносно точно прогнозує вузькі місця та інші проблеми, а також дозволяє виправляти проблеми з продуктивністю до того, як вони виникнуть. Для прогнозування тенденцій використання пропускної здатності NetFlow Analyzer використовує такі методи, як

автокореляція, декомпозиція сезонних тенденцій та регресія. Завдяки функції планування пропускної здатності він також дає цілісне уявлення про активність трафіку та тенденції використання пропускної здатності за будь-який вибраний період часу, зростання трафіку та додатків, а також допомагає приймати обґрунтовані рішення щодо планування вимог до пропускної здатності та виявлення будь-яких аномальних зростань трафіку.

Контроль мережевого трафіку передбачає обмеження пропускної здатності певними програмами та користувачами, визначення пріоритетності трафіку та програм і забезпечення певної максимальної або мінімальної пропускної здатності для всіх користувачів. Формування трафіку — це метод керування, який по суті обмежує пропускну здатність, яку споживають некритичні для бізнесу програми та користувачі. Він затримує певні потоки або пакети, щоб забезпечити оптимальне виконання завдань і додатків високого пріоритету.

NetFlow Analyzer використовує різні методи формування трафіку, такі як список контролю доступу (ACL) і політика обслуговування, щоб змінити політику якості обслуговування. Це допомагає забезпечити кращу продуктивність мережі, обмежуючи або блокуючи некритичні IP-адреси та програми, які завантажують пропускну здатність. Cisco CBQoS від NetFlow Analyzer надає можливість бачити шаблони трафіку на основі класів, допомагаючи перевірити політики та їх ефективність.

Наявність інструментів виявлення вторгнень на додаток до кількох брандмауерів для безпеки мережі не завжди є простим або здійсненним варіантом. Модуль розширеної аналітики безпеки NetFlow Analyzer — це інструмент для аналізу безпеки та виявлення аномалій на основі потоку, який допомагає виявити будь-яке вторгнення або атаку, які, можливо, перевищили брандмауер. Він дає оперативні інтелектуальні дані для виявлення широкого спектру внутрішніх і зовнішніх загроз або вторгнень, таких як ботнети, розподілені атаки відмови в обслуговуванні, вторгнення нульового дня та

зонди, які інакше можуть вплинути на загальну продуктивність мережі та привести її до зупинки. NetFlow Analyzer відстежує кожен потік, щоб захистити мережу до того, як загроза переросте в атаку, і пропонує всі ці дані в одному представленні.

- PRTG monitoring

Управління мережевим трафіком за допомогою PRTG використовує інструменти моніторингу мережі та моніторинг пропускної здатності для забезпечення оптимальної роботи мережі. Таким чином, це допомагає максимально підвищити продуктивність і безпеку існуючих мереж. Це також дає змогу ідентифікувати інтенсивні дії в мережі, які можна включити в мережеве планування та стратегії розвитку. Керування мережевим трафіком використовується серед інших методів оптимізації для забезпечення безперебійної роботи IT-інфраструктури.

Керування мережевим трафіком за допомогою PRTG дозволяє уникнути цих проблем:

- Недостатня продуктивність мережі.

  Неправильно налаштовані мережеві пристрої, такі як маршрутизатори або комутатори, можуть швидко поставити під загрозу всю мережу. Зазвичай це призводить до повільної передачі даних або до недоступних мережевих ресурсів. Але програми також можуть знижувати продуктивність мережі.

- Погіршення доступу до Інтернету.

  Погана продуктивність мережі впливає не тільки на трафік даних всередині мережі, а й на доступ до ресурсів в Інтернеті. Часто трапляється, що особливо хмарні додатки реагують дуже повільно або взагалі недоступні.

- Порушення зв'язку між місцезнаходженнями компанії.

  Якщо проблеми виникають у локальній мережі компанії, у багатьох випадках це також впливає на пов’язані місця. Відсутність

доступу до централізовано збережених даних або баз даних і платформ для співпраці зазвичай є неприємним результатом. Але також постраждали працівники, які працюють в дорозі або в своєму домашньому офісі.

У PRTG «датчики» є основними елементами моніторингу. Один датчик зазвичай контролює одне вимірюване значення у мережі, наприклад трафік порту комутатора, навантаження на центральний процесор сервера, вільний простір на диску. У середньому потрібно близько 5-10 датчиків на пристрій або один датчик на порт комутатора.

Відстежуючи мережу та її окремі пристрої, можна швидко отримати уявлення про трафік даних. PRTG ідеально підходить для цього і надає багато датчиків, такі як NetFlow, sFlow або Packet Sniffer для моніторингу. Неважливо, сервер це Windows або обладнання від відомого виробника.

PRTG не тільки відстежує всю мережу та трафік, але й надає звіти відповідно до ваших потреб. Це дозволяє переглядати та аналізувати мережевий трафік протягом більш тривалого періоду часу. Це особливо корисно, коли є великий обсяг трафіку даних поза робочим часом, який інакше залишився б непоміченим.

Тепер, коли отримано інформацію з даних, згенерованих датчиками, і проаналізованих звітів, можна приймати більш обґрунтовані рішення. Наприклад, якщо в певний час спостерігається збільшення трафіку, можна визначити причину та вжити відповідних заходів.

- Azure Traffic Manager

Azure Traffic Manager — це балансувальник навантаження трафіку на основі DNS. Ця служба дозволяє розподіляти трафік до загальнодоступних програм у глобальних регіонах Azure. Traffic Manager також надає загальнодоступним кінцевим точкам високу доступність і швидке реагування.

Менеджер трафіку використовує DNS, щоб спрямовувати клієнтські запити до відповідної кінцевої точки служби на основі методу маршрутизації

трафіку. Менеджер трафіку також забезпечує моніторинг стану для кожної кінцевої точки. Кінцевою точкою може бути будь-яка Інтернет-служба, розміщена всередині або за межами Azure. Traffic Manager надає цілий ряд методів маршрутизації трафіку та параметрів моніторингу кінцевої точки, щоб задовольнити різні потреби додатків і моделі автоматичного перемикання при виникненні збоїв. Менеджер трафіку стійкий до збоїв, у тому числі до збою всього регіону Azure.

Azure надає набір повністю керованих рішень балансування навантаження для різних сценаріїв.

- Якщо потрібно збалансувати навантаження між серверами в регіоні на прикладному рівні, варто використовувати Application Gateway.
- Якщо потрібно оптимізувати глобальну маршрутизацію веб-трафіку та оптимізувати продуктивність і надійність кінцевих користувачів найвищого рівня за допомогою швидкого глобального перемикання збоїв, використовується Front Door.
- Щоб виконати балансування навантаження мережевого рівня, використовується Load Balancer.

Різні сценарії можуть отримати користь від поєднання цих рішень за потреби. Для порівняння параметрів балансування навантаження Azure є огляд параметрів балансування навантаження в Azure.

Traffic Manager пропонує такі функції:

- Підвищення доступності програми:

  Traffic Manager забезпечує високу доступність для ваших критичних програм, відстежуючи ваші кінцеві точки та забезпечуючи автоматичне перемикання збоїв, коли кінцева точка виходить з ладу.
- Підвищення продуктивності програми:

  Azure дозволяє запускати хмарні служби та веб-сайти в центрах обробки даних, розташованих по всьому світу. Менеджер трафіку

може покращити швидкість реагування вашого веб-сайту, спрямовуючи трафік на кінцеву точку з найменшою затримкою.

- Сервісне обслуговування без простоїв:
  Можливість проводити планове технічне обслуговування програм без простоїв. Менеджер трафіку може спрямовувати трафік на альтернативні кінцеві точки, поки триває технічне обслуговування.
- Комбінування гібридних додатків:
  Менеджер трафіку підтримує зовнішні кінцеві точки, що не належать до Azure, що дозволяє використовувати його з гібридною хмарою та локальними розгортаннями, включаючи сценарії «burst-to-cloud», «migrate-to-cloud» та «failover-to-cloud».
- Розподіл трафіку для складних проектів:
  Використовуючи вкладені профілі Traffic Manager, можна об’єднати кілька методів маршрутизації трафіку для створення складних і гнучких правил для масштабування до потреб більших і складніших розгортань проектів.

Хоча комерційні рішення SDN пропонують багато переваг, вони також мають ряд недоліків, порівняно з відкритими та стандартизованими рішеннями.

Ось деякі можливі недоліки комерційних рішень:

1) Вартість. Комерційні рішення SDN можуть бути дорогими, включаючи плату за ліцензії, витрати на апаратне забезпечення та поточні витрати на підтримку та обслуговування. Ці витрати можуть бути значними для невеликих організацій або організацій з обмеженим IT-бюджетом.
2) Складність: рішення SDN можуть бути складними для розгортання та керування, вимагаючи спеціальних навичок і досвіду. Організаціям може знадобитися інвестувати в навчання або найняти додатковий персонал з необхідними навичками для керування та підтримки рішення.

3) Прив’язка до постачальника: комерційні рішення SDN можуть прив’язати організації до певного постачальника чи стеку технологій, ускладнюючи перехід на інше рішення в майбутньому. Це може обмежити гнучкість і вибір постачальника.
4) Взаємодія: рішення SDN можуть бути несумісними з усім мережевим обладнанням і програмним забезпеченням, що вимагає від організацій купувати спеціальне обладнання або програмне забезпечення для підтримки рішення. Це може призвести до блокування постачальника та обмежити взаємодію з існуючою інфраструктурою.
5) Безпека. Рішення SDN можуть створювати нові ризики для безпеки, наприклад потенційні атаки на контролер SDN або використання незахищених протоколів SDN. Організації повинні переконатися, що вони мають відповідні заходи безпеки для захисту своєї мережевої інфраструктури.

Загалом, хоча комерційні рішення SDN можуть запропонувати багато переваг, необхідно ретельно оцінити витрати та недоліки перед їх впровадженням. Вибираючи рішення SDN, важливо враховувати такі фактори, як вартість, складність, прив’язаність до постачальника, сумісність і безпека. Хоча деякі недоліки стосуються також відкритих рішень, проте відкриті рішення мають більше переваг, про які будуть описані в розділі 1.5.

1.5 Огляд існуючих відкритих рішень на основі SDN

Рішення Open SDN — це програмно-визначені мережеві рішення з відкритим вихідним кодом, які забезпечують гнучку та настроювану платформу для створення та керування мережами. Відкриті рішення SDN пропонують кілька переваг, зокрема:

1) Відкритість: рішення Open SDN мають відкритий вихідний код, що означає, що вони безкоштовні для використання та можуть бути налаштовані та змінені будь-ким. Це дозволяє організаціям налаштовувати програмне забезпечення відповідно до своїх конкретних потреб і вимог.
2) Гнучкість: відкриті рішення SDN забезпечують гнучку платформу для побудови та керування мережами. Вони дозволяють адміністраторам програмувати поведінку мережі та автоматизувати завдання керування мережею за допомогою відкритих API.
3) Взаємодія: відкриті рішення SDN розроблені для сумісності з широким спектром мережевого обладнання та програмного забезпечення. Це дозволяє організаціям інтегрувати рішення з наявною інфраструктурою та уникнути прив’язки до постачальника.
4) Підтримка спільноти: рішення Open SDN мають велику та активну спільноту розробників і користувачів, які надають підтримку та роблять внесок у розробку програмного забезпечення. Це може забезпечити доступ до великої кількості знань і ресурсів.
5) Рентабельність: відкриті SDN-рішення зазвичай безкоштовні для використання та не вимагають дорогих ліцензійних платежів. Це може зробити їх економічно ефективною альтернативою комерційним рішенням SDN.

Далі будуть розглянуті існуючі рішення для задачі балансування навантаження в мережі за допомогою OF-контролера програмно-конфігурованої мережі. Перевагами таких рішень, порівняно із комерційними є – універсальність, простота застосування в мережі та відкритість коду [16]. Тут будуть розглянуті рішення які використовують рівень управління (CP), тобто SDN контролер, який за допомогою алгоритму балансування та метрик, зібраних із OF-комутаторів обиратиме оптимальний шлях та встановлюватиме потік (Flow) для того щоб наступні пакети, які надходитимуть від адресата та

матимуть ідентичну адресу призначення, були переслані без участі OF-контролера. В даній роботі використовується саме такий спосіб балансування. Розглянемо кілька основних методів балансування навантаження за допомогою контроллера [17]. Існує кілька методів балансування навантаження SDN, які можна використовувати для розподілу мережевого трафіку між кількома серверами або шляхами [18]. Наприклад, застосування методів розглянутих у розділі 1.2:

1) Round-Robin балансування навантаження — це простий метод, який рівномірно розподіляє мережевий трафік між набором серверів. Метод працює, надсилаючи кожен новий запит на наступний сервер у попередньо визначеному списку серверів.
2) Зважене балансування навантаження призначає вагу кожному серверу на основі його обчислювальної потужності або доступних ресурсів. Потім балансувальник навантаження розподіляє мережевий трафік між серверами пропорційно їх призначеним вагам.
3) Балансування навантаження за найменшими з’єднаннями спрямовує мережевий трафік на сервер із найменшою кількістю активних з’єднань. Цей метод забезпечує рівномірний розподіл мережевого трафіку між серверами та запобігає перевантаженню серверів трафіком.
4) Балансування за допомогою IP-хешування використовує вихідну IP-адресу клієнта, щоб визначити, на який сервер надсилати запит. Цей метод гарантує, що запити від одного клієнта завжди надсилаються на той самий сервер, що може бути корисним для підтримки постійності сеансу.
5) Балансування навантаження на основі вмісту – спрямовує мережевий трафік на певний сервер відповідно до типу запитуваного вмісту. Даний метод може бути корисним для розподілу трафіку на спеціалізовані сервери, оптимізовані для певних типів вмісту, наприклад потокове відео або запити до бази даних.

Розглянуті методи балансування навантаження в SDN можна реалізувати за допомогою різних контролерів SDN і програм балансування навантаження. Вибір методу балансування навантаження залежить від конкретних вимог мережі та програм, що використовуються.

ВИСНОВКИ ДО ПЕРШОГО РОЗДІЛУ

Поки комерційні рішення мають багато переваг, таких як підвищена безпека, автоматизація керування мережею та оркестрування в мережі, вони мають ряд недоліків. Очевидним, є закритість технології, тобто обмежена можливість модифікації і налаштування системи під свої потреби. Також ключовим фактором є взаємодія із різноманітними мережевими пристроями, комерційні рішення можуть просто не підтримувати деякі з них, а закритість коду не дозволяє додавати підтримку власноручно. Вибір відповідного рішення SDN залежить від конкретних потреб і вимог організації, тому важливо оцінити характеристики та можливості кожного рішення перед прийняттям рішення.

Відкриті рішення SDN пропонують кілька переваг перед комерційними рішеннями SDN, зокрема, відкритий вихідний код, що дозволяє налаштовувати програмне забезпечення для конкретних потреб. Гнучка архітектура дозволяє створювати власні мережеві додатки та керувати мережею SDN за допомогою відкритих API. Відкриті рішення SDN зазвичай мають більшу спільноту користувачів, яка пришвидшує процес виявлення багів та вразливостей. Також ці рішення є більш фінансово-ефективними, навіть з врахуванням необхідності залучення розробників для створення додатків.

Загалом відкриті рішення SDN пропонують більшу гнучкість, взаємодію, підтримку спільноти та економічну ефективність порівняно з комерційними рішеннями SDN, що може зробити їх більш ефективним варіантом для організацій, які хочуть реалізувати програмно-конфігуровану мережу.

Були розглянуті базові методи балансування навантаження в мережі та приклади їх застосування в SDN. Більшість балансувальників навантаження використовують саме ці конвенційні методи. Проте вони мають ряд недоліків порівняно з використанням генетичного алгоритму (GA) у SDN для балансування навантаження, зокрема:

1) Традиційні методи балансування можуть спадати до локальних мінімумів. Наприклад при знаходженні завжди найкращого шляху між двома вузлами, певна ділянка мережі буде завжди перевантажена та утворюватимуться вузькі місця.
2) Відсутність або обмеженість адаптації даних методів до змін у мережевому трафіку, відповідно нижча ефективність використання мережевих ресурсів.
3) Маштабованість частини з цих алгоритмів обмежена порівняно з GA.
4) Відсутність коригування параметрів алгоритму до топології мережі та доступності ресурсів.
5) Обмежені можливості з автоматизації балансування навантаження в мережі та необхідність ручного втручання для адаптації.

В цілому, враховуючи перелічені недоліки конвнційних методів, використання GA в SDN для балансування навантаження може забезпечити адаптивність як до топології та і до типів трафіку, автоматизацію й природній розподіл навантаження, що може покращити продуктивність мережі та зменшити необхідність налаштування.

РОЗДІЛ 2
ГЕНЕТИЧНИЙ АЛГОРИТМ ДЛЯ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В ПРОГРАМНО-КОНФІГУРОВАНІЙ МЕРЕЖІ

2.1 Генетичний алгоритм

Генетичний алгоритм (англ. Genetic Algorithm) — це метод розв’язання задач як обмеженої, так і необмеженої оптимізації, який ґрунтується на природному відборі — процесі, який спричиняє біологічну еволюцію. Генетичний алгоритм неодноразово змінює популяцію окремих рішень (в даному випадку кожне рішення – це можливий шлях між двома пристроями в мережі). На кожному кроці генетичний алгоритм вибирає особин із поточної популяції, які стануть батьками, і використовує їх для народження дітей для наступного покоління. Протягом наступних поколінь популяція «еволюціонує» до оптимального рішення. Можна застосувати генетичний алгоритм для розв’язування різноманітних задач оптимізації, які погано підходять для стандартних алгоритмів оптимізації, включаючи задачі, у яких цільова функція є розривною, недиференційованою, стохастичною або сильно нелінійною. Генетичний алгоритм може вирішувати проблеми змішаного цілочисельного програмування, де деякі компоненти обмежені цілими значеннями [19]. Оскільки задача пошуку оптимального шляху має високу просторову складність для непоінформованих алгоритмів, наприклад, алгоритм Дійкстри має складність O(V^2), де V - це кількість вершин графа. А поінформовані алгоритми такі як A-Star потребують надійної евристики (допоміжної функції, яка характеризує наближеність до цільової вершини на кожному етапі алгоритму). Тому потрібен алгоритм, який гарно масштабується, швидко знаходить оптимальний (або близький до оптимального) шлях та має набір гіперпараметрів для гнучкого налаштування та динамічного пристосування до

поточної топології мережі та стану завантаженості зв’язків між вершинами. Генетичний алгоритм, може бути створений таким чином, щоб задовольняти всі перелічені вимоги.

2.2 Генерація початкової популяції

Є два основні методи для ініціалізації початкової популяції генетичного алгоритму:

- Випадкова ініціалізація – заповнення початкової популяції абсолютно випадковими рішеннями.
- Евристична ініціалізація – заповнення початкової популяції за допомогою відомої евристики, специфічної для проблеми.

Згідно з найкращою практикою, всю популяцію не слід ініціалізувати за допомогою евристики, оскільки це може призвести до того, що популяція матиме подібні рішення та дуже малу різноманітність. Було експериментально доведено, що випадкові рішення – це ті, які приводять популяцію до оптимальності [20]. Тому за допомогою евристичної ініціалізації варто просто заповнювати популяцію парою хороших рішень, заповнюючи решту випадковими рішеннями, замість створення популяції з виключно евристичних рішеннь.

Крім того, евристична ініціалізація в деяких випадках впливає лише на початкову придатність популяції, але в кінцевому підсумку саме різноманітність рішень призводить до оптимальності.

Для вирішення проблеми пошуку оптимального шляху в графі, а отже і балансування навантаження, генерація початкової популяції – це один із

найважливіших етапів, а також один із найбільш складних. Для зручності розіб'ємо його на кілька підпунктів:

1. Пошук множини вершин графа, які можуть бути частинами шляху від початкової вершини до вершини призначення. Для цього завдання використовується зворотній алгоритм пошуку в глибину (Reverse DFS), який знаходить всі вершини з яких можливо дістатись до вершини призначення.
2. Наступний крок – створення графа на основі знайдених вершин на 1-му кроці, для подальшого знаходження випадкових шляхів.
3. Генерація простих випадкових шляхів використовуючи рандомізований алгоритм пошуку в глибину (Randomised DFS) на графі створеному в кроці 2.

Таким чином відбувається генерація початкової популяції для генетичного алгоритму. Вона займаї значну частину часу обчислення алгоритму, проте може бути оптимізована для виконання один раз, якщо топологія мережі не змінюватиметься.

2.3 Функція пристосованості та функція селекції

Функція пристосованості (англ. fitness function), визначена як функція, яка приймає рішення задачі як вхідні дані та повертає те, наскільки воно «підходить», наскільки «хорошим» є рішення по відношенню до задачі, що розглядається. В даному випадку функція бере шлях та за допомогою таблиці маршрутизації та вартості зв'язків (матриці суміжності, де ваги це оцінки завантаженості зв'язків) і повертає загальну «ціну шляху». Для алгоритму в даній роботі найкращим вважається шлях із найменшою «ціною».

Розрахунок значення придатності виконується багаторазово в GA, тому має бути достатньо швидким. Повільне обчислення значення придатності може негативно вплинути на GA та зробити його надзвичайно повільним. Тому для функції пристосованості використовується таблиця лише можливих зв'язків між вершинами що розглядаються [21].

У більшості випадків функція пристосованості та цільова функція однакові, оскільки метою є або максимізація, або мінімізація даної цільової функції. Однак для більш складних проблем із кількома цілями та обмеженнями розробник алгоритмів може вибрати іншу функцію пристосованості.

Функція пристосованості повинна мати такі характеристики:

- Вона повинна кількісно вимірювати, наскільки придатним є дане рішення.
- Вона має бути достатньо швидкою для обчислення.

В загальному вигляді функція пристосованості в генетичному алгоритмі для балансування навантаження обчислює суму довжин (це може бути пропускна здатність, затримка, кількість пакетів або інтегральна оцінка даних параметрів мережі) зв'язків у топології мережі. Вона повертає значення довжини певного шляху (хромосоми в термінології генетичного алгоритму).

- Функція селекції

Відбір батьків — це процес знаходження найкращих хромосом, для кросинговеру, для створення нащадків, які потраплять до наступного покоління. Вибір батьків дуже важливий для рівня конвергенції (збіжності) GA, оскільки хороші батьки наближають популяцію до кращих і пристосованіших особин (рішень).

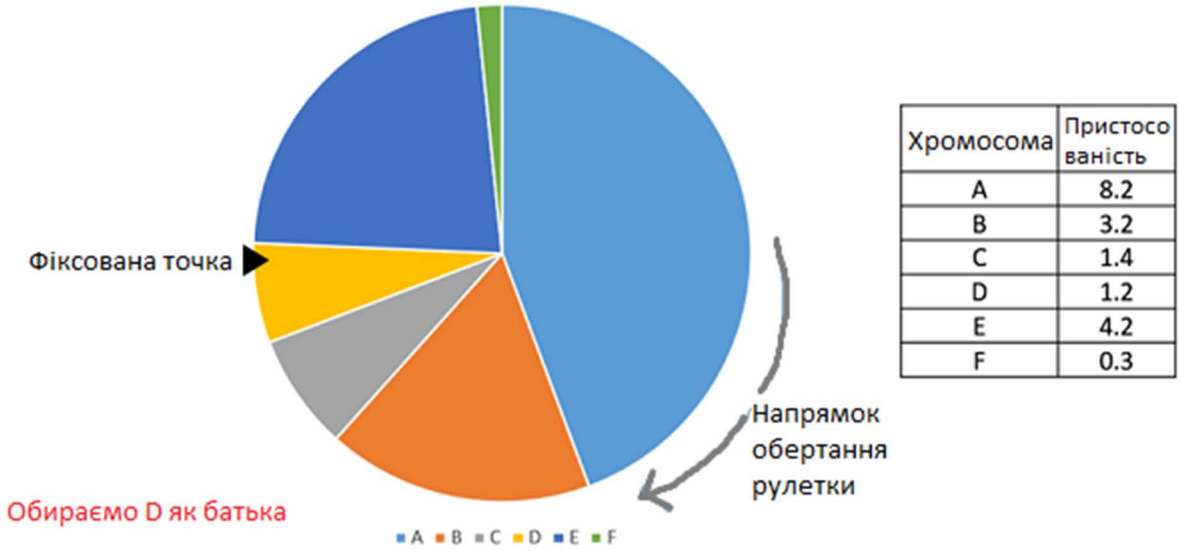
Однак слід підходити до цього обережно, щоб запобігти тому, щоб одне надзвичайно придатне рішення захопило всю популяцію через кілька поколінь, оскільки це призводить до того, що рішення будуть близькі одне до одного в

просторі рішень, що призведе до втрати різноманітності. Підтримання гарної різноманітності в популяції надзвичайно важливо для успіху GA. Таке захоплення всієї сукупності одним надзвичайно придатним рішенням відоме як передчасна конвергенція, це небажаний стан у GA.

- Пропорційний відбір за пристосованістю

Пропорційний відбір за пристосованістю - один з найпопулярніших способів відбору батьків. При цьому кожна особина може стати батьком з імовірністю, пропорційною її пристосованості. Таким чином, обрані особини мають вищі шанси для кросинговеру та передати свої гени наступному поколінню. Таким чином, дана стратегія відбору застосовує відбір до більш придатних індивідуумів у популяції, розвиваючи кращих індивідуумів з часом. Розглянемо кругову діаграму на рис. 2.1. Діаграма ділиться на n секторів, де n — кількість особин у популяції. Кожна особина отримує частину кола, яка пропорційна його значенню фізичної форми.

- Селекція методом рулетки



Хромосома	Пристосованість
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

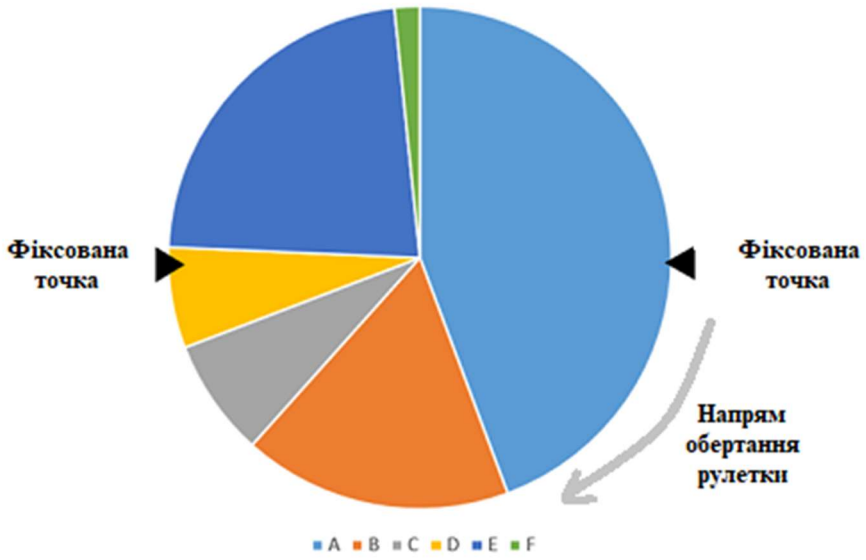
Рис. 2.1 – Метод селекції особин для кросинговеру з використанням рулетки.

Зрозуміло, що більш пристосована особина має більший сектор на діаграмі й, отже, більший шанс зупинитися перед фіксованою точкою, коли колесо обертається. Тому ймовірність вибору особини безпосередньо залежить від її придатності.

Що стосується реалізації, використовуються наступні кроки:

1) Розраховується S, сума пристосованостей всіх особин.
2) Генерується випадкове число між 0 та S.
3) Починаючи з вершини популяції, додаються значення пристосованості особин до часткової суми P, поки P < S.
4) Особина при якій P стала більше S обирається батьком.

- Stochastic Universal Sampling (SUS)



Хромосома	Пристосованість
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Рис. 2.2 – Селекція батьків із популяції з використанням методу Стохастичної Універсальної Вибірки.

Стохастична універсальна вибірка (SUS) дуже схожа на вибір методом рулетки, однак замість однієї фіксованої точки є кілька фіксованих точок, зображення на рис. 2.2. Тому всі батьки вибираються лише за одне обертання колеса. Крім того, такий спосіб підвищує ймовірність вибору особин із високою пристосованістю.

Варто також зазначити, що даний метод не підходить для випадків коли пристосованість має відємне значення. А оскільки в розроблюваному генетичному алгоритмі використовується значення пристосованості – довжина шляху, то для отримання найбільшого значення пристосованості, потрібно розглядати обернені (від'ємні значення довжини шляху). Через це перший метод, рулетки, підходить краще для даної задачі.

- Турнірна селекція

Під час відбору турніром випадковим чином обирається K осіб із популяції та обирається найкращий з них, щоб стати батьком. Той самий процес повторюється для вибору наступного батька. Турнірний вибір, або ж селекція, також надзвичайно популярний у літературі [22], оскільки він може працювати навіть із від'ємними значеннями пристосованості (рис. 2.3).

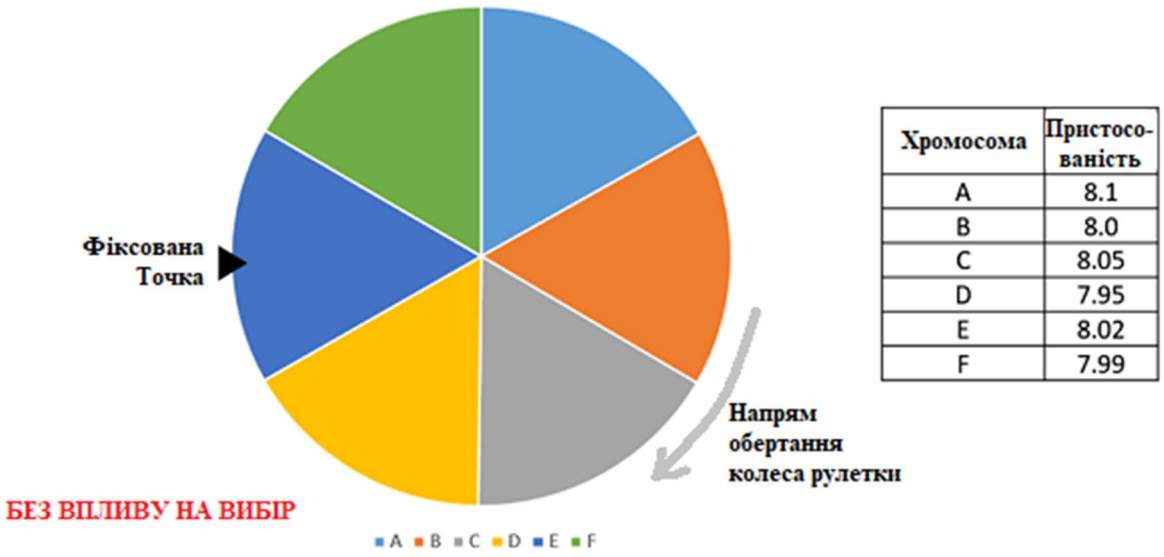
Рис. 2.3 – Турнірна селекція особин для кросинговеру.

- Рангова селекція

Вибір за рангом також працює з від'ємними значеннями пристосованості та здебільшого використовується, коли особини в популяції мають дуже

близькі значення пристосованості (це зазвичай відбувається в кінці прогону). Це призводить до того, що кожна особа має майже рівну частку колеса (як у випадку пропорційного відбору пристосованості) і, отже, кожна особа, незалежно від того, наскільки пристосована відносно одна одної, має приблизно однакову ймовірність бути обраною як батьківська. Це, у свою чергу, призводить до втрати тиску відбору на більш підготовлених осіб, що змушує GA робити поганий вибір батьків у таких ситуаціях. Приклад такої селекції зображує кругова діаграма на рис. 2.4.

Одна з переваг рангового вибору полягає в тому, що він зменшує вплив викидів (крайніх значень) у популяції. Іншими словами, індивідууми з дуже високою або дуже низькою пристосованістю менш імовірно будуть обрані в якості батьків, що може допомогти запобігти передчасному переходу до неоптимального рішення.



Хромосома	Пристосованість
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

Рис. 2.4 – Рангова селекція особин для кросинговеру.

Ще одна перевага рангової селекції полягає в тому, що вона забезпечує баланс між експлуатацією (використанням найкращих особин) та дослідженням (пошук нових більш досконалих особин за допомогою мутації та

кросинговеру). Експлуатація передбачає вибір найбільш придатних індивідуумів, які стануть батьками, тоді як дослідження передбачає вибір менш придатних індивідуумів для дослідження нових областей простору пошуку. Вибір за рангом врівноважує ці дві цілі, гарантуючи, що найбільш підготовлені особи мають вищу ймовірність відбору, водночас дозволяючи деяке дослідження шляхом вибору осіб із нижчими рангами пристосованості.

В цілому ранговий вибір є корисним механізмом відбору в генетичних алгоритмах, оскільки він врівноважує конкуруючі цілі експлуатації та дослідження, а також зменшує вплив викидів у популяції.

2.4 Оператор кросинговеру

Кросинговер (схрещування) – це важливий компонент генетичних алгоритмів, оскільки він дозволяє поєднувати генетичну інформацію з різних варіантів рішень, що може призвести до створення нових і потенційно кращих рішень.

У генетичному алгоритмі генерується сукупність варіантів рішень, які оцінюються на придатність. Потім для відтворення відбираються найпридатніші рішення (хромосоми), а їхня генетична інформація використовується для створення нових варіантів. Схрещування (кросинговер) — це процес поєднання генетичної інформації двох або більше батьківських рішень для створення нового дочірнього рішення.

Метою кросинговеру є введення нової генетичної інформації в популяцію, яка може допомогти досліджувати нові області простору пошуку та уникнути застрягання в локальних оптимумах. Поєднуючи генетичну інформацію з різних батьківських популяцій, кросинговер може створювати

нові рішення, яких немає у вихідній популяції, що потенційно призводить до кращих результатів [23].

Ефективність кросинговеру залежить від генетичного представлення популяції кандидатів. У контексті балансування навантаження SDN генетичне представлення може містити таку інформацію, як ваги, призначені різним серверам, або шляхи маршрутизації, що використовуються для певних потоків трафіку. За допомогою поєднання генетичної інформації таким чином, кросинговер може створювати нові стратегії балансування навантаження, які можуть бути ефективнішими та результативнішими, ніж у вихідній популяції.

Оператор кросинговеру аналогічний розмноженню та біологічному обміну генами. У цьому випадку вибирається двоє батьків, і з використанням їхнього генетичного матеріалу утворюється один чи більше нащадків. Кросинговер зазвичай застосовується в GA з високою ймовірністю – p_c.

У цьому пункті роботи буде описано деякі з найпопулярніших операторів кросинговеру. Слід зазначити, що дані оператори кросинговеру є загальними, і пізніше буде обрана реалізація оператора кросинговеру для конкретної проблеми балансування навантаження методом пошуку оптимального шляху.

- Одноточковий кросинговер

В одноточковому схрещуванні вибирається випадкова точка схрещування, і хвости двох батьків міняються місцями, щоб отримати нових нащадків (рис 2.5).

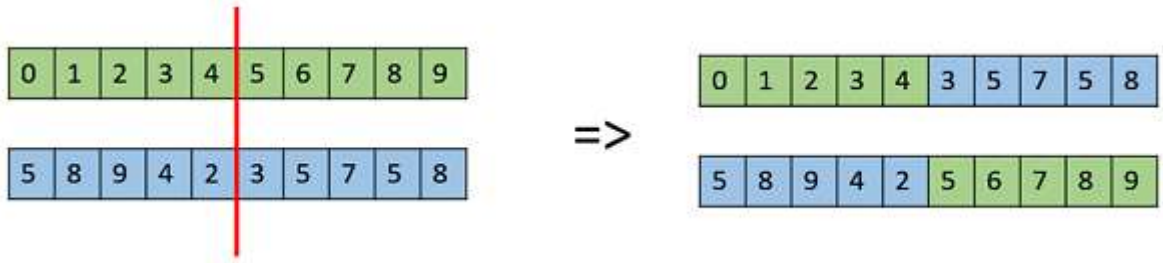


Рис. 2.5 – Звичайний кросинговер в одній точці.

- Багатоточковий кросинговер

Багатоточкове схрещування — це узагальнення одноточкового схрещування, у якому чергуються сегменти, які міняються місцями, щоб отримати нові вихідні хромосоми (рис. 2.6).

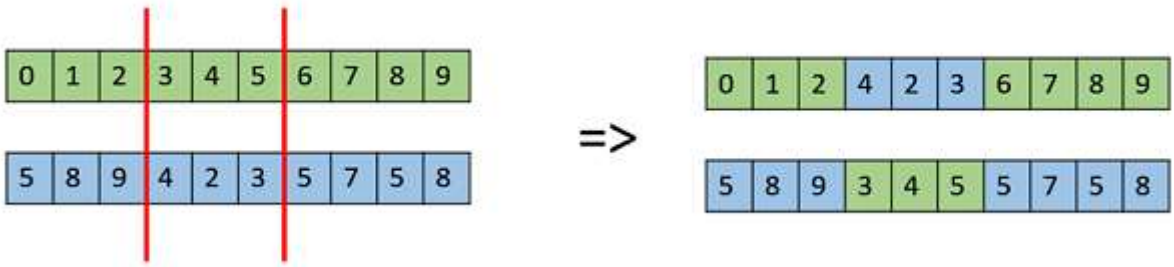


Рис. 2.6 – Багатоточковий кросинговер.

- Рівномірний кросинговер

Під час рівномірного кросинговеру хромосома не ділиться на сегменти, а розглядається кожен ген окремо. У цьому випадку, по суті, підкидається монета за кожен ген, щоб вирішити, чи буде він включений в потомство. Також можна спрямувати монету на одного з батьків, щоб мати більше генетичного матеріалу в потомку від визначеного батька (рис. 2.7).

Рис. 2.7 – Рівномірний кросинговер.

- Застосований алгоритм кросинговеру

При створенні генетичного алгоритму для даної роботи, був використаний модифікований варіант одноточкового кросинговеру. Обираються два шляхи в результаті роботи функції селекції. Потім алгоритм

кросинговеру знаходить спільні вершини в цих двох шляхах, ці вершини і будуть можливими точками для поділу та перестановки частин цих шляхів. В список спільних шляхів не включаються початкова та вершина призначення. Також пропускаються шляхи, які мають менше ніж три вершини. Після знаходження спільних вершин, випадковим чином обирається одна з них та виконується одноточковий кросинговер із двома потомками в результаті.

Загалом кросинговер є важливим механізмом для створення нових рішень-кандидатів у генетичних алгоритмах і може допомогти покращити якість рішень і уникнути застрягання в локальних оптимумах, що необхідно для ефективного балансування навантаження в SDN.

2.5 Оператор мутації

Мутація є важливим компонентом генетичних алгоритмів, які створені для імітації процесу природного відбору та генетичної еволюції. У генетичному алгоритмі генерується популяція варіантів рішень, оцінюється їх придатність, а потім мутується для створення нового набору хромосом. Процес повторюється, поки не буде знайдено хромосома яка володіє задовільним значенням пристосованості [24].

Мутація може бути визначена як невелика випадкова зміна хромосоми для отримання нового рішення. Вона використовується для підтримки та впровадження різноманітності в генетичну популяцію та зазвичай застосовується з низькою ймовірністю – p_m . Якщо ймовірність мутації дуже висока, то GA зводиться до випадкового пошуку.

Мутація – це частина GA, яка пов’язана з «дослідженням» простору пошуку. Мутація є важливою для конвергенції GA, тоді як кросинговер - ні.

У цьому пункті будуть описані деякі з найбільш часто використовуваних операторів мутації. Подібно до операторів кросинговеру, їх існує дуже багато, і для розробки GA може виявитися більш корисною комбінація цих підходів або оператор мутації, пов’язаний із задачею.

- Мутація оберненням біту

У цій мутації перевертання бітів вибирається один або кілька випадкових бітів і вони змінюються на протилежні. Це використовується для GA у двійковому кодуванні (рис. 2.8).

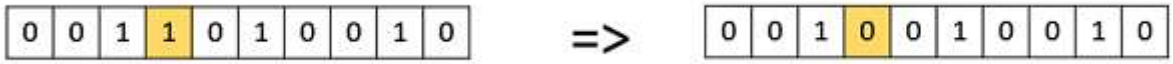


Рис. 2.8 – Мутація оберненням біту.

- Випадкове скидання

Випадкове скидання є розширенням перевертання бітів для цілочисельного представлення. У цьому випадку випадково вибраному гену присвоюється випадкове значення з набору допустимих значень.

- Випадковий обмін

У мутації обміну ми навмання вибираємо дві позиції в хромосомі та міняємо їх значеннями. Це часто зустрічається в кодуваннях на основі перестановки (рис. 2.9).

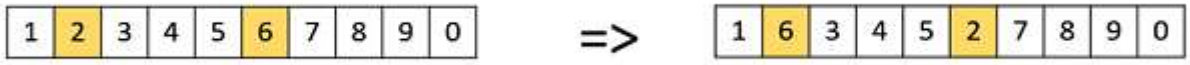


Рис. 2.9 – Мутація з випадковим обміном.

- Scramble-мутація

Scramble мутація також популярна у представленнях з перестановками.

У цьому випадку з усієї хромосоми вибирається підмножина генів, а їхні значення перемішуються або переміщуються випадковим чином (рис. 2.10).

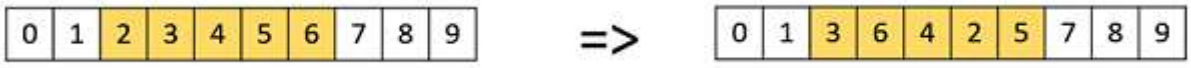


Рис. 2.10 – Scramble мутація.

- Інверсійна мутація

У інверсійній мутації вибирається підмножина генів, як у scramble мутації, але замість того, щоб перетасувати підмножину, весь рядок у підмножині просто інвертується. Даний метод зображено на рис 2.11.

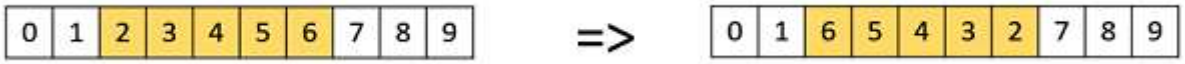


Рис. 2.11 – Інверсійна мутація.

- Обраний метод мутації

Оскільки задача пошуку оптимального шляху на графі більш складна ніж інші тривіальні задачі, вона не може бути описана двійковими хромосомами. Методи скидання чи перевертання генів також не працюють. Для того, щоб при мутації отримувати завжди реальні шляхи, потрібно знати на які ділянки шляху можна заміняти існуючі ділянки в обраному шляху. Для цього використовується матриця суміжності, обчислена на етапі генерації початкової популяції. Алгоритм дій наступний:

1. Випадковим чином вибирається вершина шляху, який був обраний для мутації.
2. Потім в матриці суміжності перевіряється чи є аналогічні вершини, які з’єднують поточну та ту що після наступної.

3. Якщо такі вершини є, то серед них випадково обирається одна наступна вершина після поточної заміняється на обрану. Таким чином відбувається часткова мутація шляху.

2.6 Селекція потомків та умова завершення алгоритму

Політика відбору тих, хто вижив, визначає, яких особин слід викинути, а яких залишити в наступному поколінні. Це має вирішальне значення, оскільки має гарантувати, що більш підготовлені особини не будуть вигнані з популяції, і в той же час у популяції має підтримуватися різноманітність [25].

Деякі GA використовують елітарність. Простіше кажучи, це означає, що поточний найбільш пристосований член популяції завжди передається наступному поколінню. Таким чином, за жодних обставин не можна замінити найпридатнішого члена поточного населення.

Найпростіший спосіб — викинути випадкових членів із вибірки, але такий підхід часто має проблеми з збіжністю, тому широко використовуються наступні стратегії.

- Селекція на основі віку

У віковому відборі немає поняття пристосованості. Такий відбір заснований на припущенні, що кожна особина допускається в популяцію протягом обмеженої кількості поколінь, протягом яких їй дозволяється розмножуватися, після чого її виганяють з популяції, незалежно від того, наскільки добре вона пристосована.

Наприклад на рис. 2.12 зображено приклад виконання вікової селекції, де було обрано двох найстарших особин, не залежно від рівня їх пристосованості, а потім їх було викинуто із популяції. В даному випадку вік — це кількість

поколінь, протягом яких індивід був у популяції. Найстарші члени популяції, тобто P4 і P7, виключаються з популяції, а вік решти учасників збільшується на одиницю.

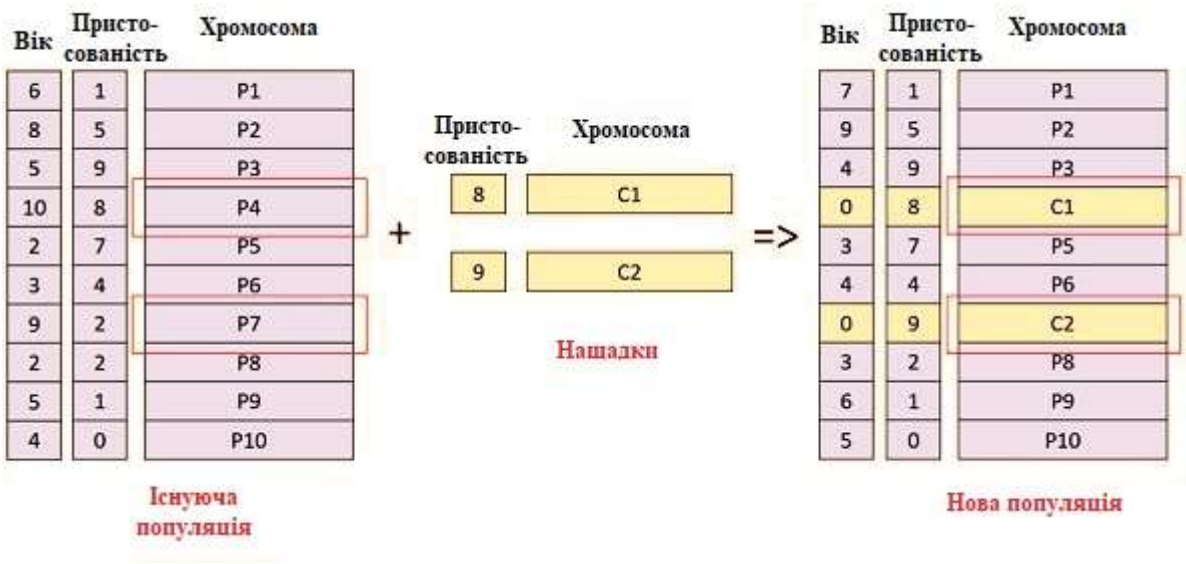


Рис. 2.12 – Селекція нащадків на основі віку.

- Селекція на основі пристосованості

Для виконання селекції на основі пристосованості особин, як правило, замінюють найменш придатних осіб у популяції. Відбір найменш придатних осіб може бути здійснений за допомогою варіації будь-якої політики відбору, описаної раніше – відбір на турнірах, відбір відповідно до фітнесу тощо.

Наприклад, на рисунку 2.13 потомки замінюють найменш придатних осіб із множини від P1 до P10. Слід зазначити те, що оскільки P1 і P9 мають однакове значення придатності, рішення про видалення однієї з цих особин приймається довільно або за додактовими правилами, наприклад випадковим чином, або за віком.

Однією з переваг відбору на основі придатності в генетичних алгоритмах є те, що він дозволяє відбирати особин з вищими показниками придатності, що може збільшити ймовірність створення потомства з ще вищими значеннями

придатності. Інакше кажучи, вибираючи індивідуумів з вищою пристосованістю, алгоритм може «виводити» індивідуумів, які краще підходять для вирішення поточної проблеми, що з часом може призвести до більш ефективних рішень.

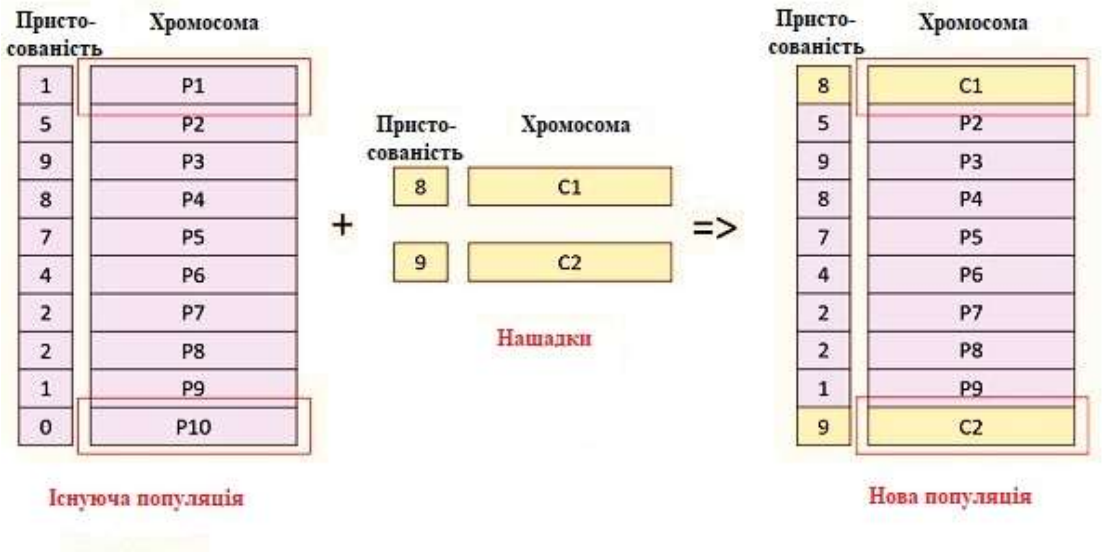


Рис. 2.13 – Селекція нащадків на основі пристосованості.

- Умова завершення алгоритму

Умова завершення генетичного алгоритму важлива для визначення того, коли завершиться виконання GA. Було помічено, що спочатку GA прогресує дуже швидко з кращими рішеннями, які надходять через кожні кілька ітерацій, але це має тенденцію насичуватися на пізніших етапах, коли покращення дуже незначні [26]. Зазвичай потрібна така умова завершення, щоб рішення було близьке до оптимального в кінці циклу.

Зазвичай при розробці генетичного алгоритму застосовується одна або кілька із наступних умов для завершення:

1) Якщо не було покращення популяції протягом K ітерацій.
2) Якщо було досягнуто максимальної кількості поколінь.
3) Коли значення цільової функції досягло певного значення, яке було визначене заздалегідь.

Наприклад, у генетичному алгоритмі вводиться лічильник, який відстежує покоління, для яких не було покращення популяції. Спочатку встановлюється даний лічильник в нуль. Кожного разу, коли не створюються нащадки, які кращі за інших особин у популяції, збільшуємо лічильник на одиницю. Однак, якщо будь-яка з хромосом краща, скидаємо лічильник знову в нуль. Алгоритм завершить роботу, коли лічильник досягне заданого значення.

Як й інші параметри GA, умова завершення також прямо залежить від розв'язуваної задачі, тому при виборі умови, варто враховувати особливості застосування GA в SDN.

- Обраний метод селекції потомків

Для ефективного відбору потомків для вирішення задачі балансування навантаження в SDN за допомогою генетичного алгоритму підходять обидва згадані вище способи. Селекцію в GA можливо проводити як за допомогою відбору на основі віку, так і за допомогою відбору на основі пристосованості.

Оскільки для генетичного алгоритму, що використовується для балансування навантаження, необхідна максимальна швидкість конвергенції (збіжності), то було обрано метод відбору на основі пристосованості. Даний спосіб селекції потребує менше ресурсів та швидше приводить популяцію до оптимальних рішень [27]. Також це було підтверджено в ході експериментів на тестових даних при розробці алгоритму.

- Обрані умови для завершення алгоритму

Як умови для завершення роботи GA, було обрано два критерії:

1) Перевищення максимальної кількості поколінь.
2) Відсутність покращення результатів пристосованості найкращої особини в популяції протягом K поколінь.

Дані умови завершення були обрані оскільки при балансуванні навантаження необхідно враховувати довжину шляху в мережі (наприклад це може бути оцінка загальної затримки комутаторів на шляху або рівень пропускної здатності шляху) для якого максимальна (найкраща) оцінка не

відома, для більшості мереж, через їх гетерогенну структуру. Тому умова досягнення найкращого рішення не може бути використана. А для обмеження максимального часу знаходження оптимальних рішень було введено умови зупинки при досягненні максимальної кількості поколінь та відсутності покращення [28].

2.7 Збір метрик мережі SDN для використання в GA

Для ефективного балансування навантаження та прокладання оптимальних маршрутів генетичний алгоритм повинен отримувати дані про поточний стан мережі. Це може бути загальна оцінка для певних зв'язків між комутаторами, яка включає в себе багато різних параметрів. Або, наприклад, звичайна завантаженість портів комутатора.

Оцінки «довжин» зв'язків графу топології мережі зазвичай містяться в таблиці суміжності даного графа. Така таблиця повинна періодично оновлюватись для забезпечення актуальності інформації про ці зв'язки. Та обов'язкове оновлення при реєстрації змін у топології. Завдяки широкому набору інструментів SDN збір цих параметрів не складний [29]. Деякі загальні параметри, які використовуються балансувальниками навантаження, включають:

1) Пропускна здатність: обсяг доступної пропускної здатності на заданому мережевому шляху може бути ключовим фактором у визначенні найкращого маршруту для мережевого трафіку. Контролери SDN можуть використовувати інформацію про пропускну здатність, щоб вибрати маршрути з найвищою доступною пропускною здатністю.

2) Затримка: затримка або затримка мережевого шляху також може бути важливим фактором у визначенні найкращого маршруту для мережевого трафіку. Контролери SDN можуть використовувати інформацію про затримку, щоб вибрати маршрути з найменшою затримкою, що може допомогти мінімізувати затримку мережі.
3) Перевантаження мережі: перевантаження мережі може виникнути, коли через мережевий шлях надсилається занадто багато трафіку. Контролери SDN можуть відстежувати рівень перевантаженості мережі та вибирати менш перевантажені маршрути, щоб забезпечити безперебійний рух трафіку.
4) Якість обслуговування (QoS): різні типи мережевого трафіку можуть мати різні вимоги до QoS, наприклад програми реального часу, які потребують низької затримки та високої пропускної здатності. Контролери SDN можуть використовувати інформацію QoS для вибору маршрутів, які відповідають конкретним вимогам різних типів мережевого трафіку.
5) Топологія мережі: загальна топологія мережі також може бути ключовим фактором у визначенні найкращих маршрутів для мережевого трафіку. Контролери SDN можуть використовувати інформацію про топологію мережі, щоб вибрати маршрути, які є більш ефективними та мають менше стрибків мережі.

Зазвичай, конкретні параметри мережі, які використовуються для пошуку найкращих маршрутів у SDN, залежатимуть від конкретних вимог до мережі та запущених у ній програм. Контролери SDN можуть використовувати комбінацію цих параметрів для вибору найбільш ефективних і ефективних маршрутів для мережевого трафіку [30].

- Огляд доступних метрик для використання в балансуванні навантаження

Далі розглянемо детальніше показники та критерії, які використовуються для дослідження ефективності балансування навантаження. Показники

використовуються для звичайного балансування навантаження та балансування навантаження на основі генетичного алгоритму. Нижче наведено перелік найбільш популярних показників:

1) Пропускна здатність:

У комп'ютерних мережах пропускна здатність означає кількість даних, які можна передати через мережеве з'єднання протягом певного періоду часу. На пропускну здатність може впливати безліч факторів, зокрема пропускна здатність мережевого з'єднання, затримка з'єднання, швидкість втрати пакетів і кількість активних з'єднань у мережі. Вища пропускна здатність зазвичай вказує на те, що мережеве з'єднання здатне обробляти більше даних, що може бути важливим для програм, які потребують високої швидкості передачі даних, наприклад потокове відео, передача файлів або онлайн-ігри. Формула (1) визначає рівняння для знаходження пропускної здатності.

$$Throughput = \sum_{requestsi}^{n} (timet), \quad (1)$$

де n – кількість успішних запитів;

$timet$ – час виконання запиту.

Для алгоритмів балансування навантаження в SDN пропускна здатність зазвичай використовується в якості основної метрики для оцінки певних зв'язків між вершинами топології.

2) Час відгуку:

Час відгуку означає кількість часу, який потрібен мережевому пристрою або програмі для відповіді на запит від іншого пристрою або програми. Це міра часу затримки між відправленням запиту та отриманням відповіді, нижче наведений її вираз (2).

На час відповіді може впливати безліч факторів, зокрема затримка підключення до мережі, час обробки приймаючого пристрою чи програми та

обсяг даних, що передаються. Менший час відповіді зазвичай свідчить про те, що мережеве обладнання або програма здатні швидше відповідати на запити, що важливо для програм, які вимагають передачі даних у реальному часі, таких наприклад, як відеоконференції, онлайн-ігри або фінансові операції.

$$Response_{time} = abs_{request \sim i}(sub_{time} start_{time}), \quad (2)$$

де sub_{time} – це час за який запит було прийнято;

$start_{time}$ – час за який запит почав оброблятись.

Вимірювання часу відповіді може допомогти виявити вузькі місця в мережі та оптимізувати продуктивність мережі для збільшення швидкості передачі даних та покращення взаємодії з користувачами.

3) Час виконання запиту:

Це базова метрика, яка визначає час за який обробляється запит, її формула (3) наведена нижче:

$$Execution_{time} = request_i(finish_{time} start_{time}), \quad (3)$$

де $finish_{time}$ – час закінчення обробки запиту;

$start_{time}$ – час початку обробки запиту;

$request_i$ – номер поточного запиту.

4) Час зворотного обходу (Round Trip Time, RTT):

Час зворотного обходу — це час, який потрібен пакету для переміщення від джерела до місця призначення й назад. RTT є необхідним елементом вимірювання продуктивності, оскільки це час очікування, поки буде передано підтвердження (ACK), перш ніж сегмент буде повторно відправлено. Якщо

розрахунковий час зворотного обходу нижчий за фактичний час зворотного обходу, то сегменти передаються раніше, ніж автентична фаза, або відповідне підтвердження (ACK) поширюється мережею. Якщо тривалість зворотного обходу занадто велика, тоді тайм-аути тривають довше, ніж необхідно, і тому їх ефективність стає нижчою.

5) Утилізація ресурсів (використання):

Утилізація вимірюється як ефективне використання ресурсіав для обробки запиту, а її мета полягає в тому, щоб максимізувати найважливіший ресурс для обробки запиту, формула (4). Ресурсами можуть бути: процесор, пам’ять, порти комутаторів, мережеві зв’язки чи пропускна здатність.

$$ResourceUtilization = \frac{\sum_{i\sim requesti}^{n} Execution_{time}}{MaxrequestExecution_{time}}, \quad (4)$$

де $Execution_{time}$ – час виконання запиту;

n – кількість запитів.

6) Затримка:

Затримкою в мережі називають час за який пакет переміщується із одної вершини в іншу, в даному випадку вершиною може бути хост, комутатор, маршрутизатор тощо. Є кілька різних типів затримок, таких наприклад, як затримка обробки, затримка комунікації або затримка розповсюдження.

7) Відношення доставлених пакетів:

Відношення кількості пакетів, що були успішно доставлені до загальної кількості відправлених пакетів від джерела до вершини призначення.

8) Наскрізна затримка (від краю до краю):

У комп’ютерних мережах наскрізна затримка означає кількість часу, який потрібен пакету даних для проходження від пристрою-джерела до пристрою призначення, включаючи всі затримки обробки та передачі на цьому шляху.

На наскрізну затримку може впливати безліч факторів, у тому числі фізична відстань між пристроями джерела та призначення, маршрут пакета даних, час обробки проміжних пристроїв, а також затори та затримки в черзі, які виникають у мережі. Формула 5 відображає традиційну EED затримку.

$$EED = TD_{sr} + PD + TD_{dt} + SD, \quad (5)$$

де EED – (англ. end to end delay) затримка від краю до краю;

PD – (англ. propagation delay) затримка розповсюдження;

TD_{sr} – (англ. source transmission delay) затримка відправки із джерела;

TD_{dt} – (англ. destination trans. delay) затримка вершини призначення;

SD – (англ. switch delay) затримка на комутаторі.

Наскрізна затримка є важливим показником для оцінки продуктивності та якості обслуговування мережевого з'єднання, особливо для додатків у реальному часі, таких як відеоконференції чи онлайн-ігри, де затримки можуть мати значний вплив на взаємодію з користувачем. Мережеві адміністратори можуть виміряти та оптимізувати наскрізну затримку, налаштувавши протоколи мережевої маршрутизації, зменшивши втрату пакетів і впровадивши механізми контролю перевантаження.

9) Втрати пакетів:

Кількість пакетів, яка не досягнула пункту призначення, поділена на загальну кількість надісланих пакетів, називається втратою пакетів і може відстежуватися за допомогою програмного забезпечення для тестування продуктивності мережі, мережевих аналізаторів або ручного моніторингу мережі. Щоб зменшити втрату пакетів, можна застосовувати такі стратегії, як протоколи якості обслуговування (QoS), щоб визначити пріоритет важливого трафіку, налаштувати протоколи мережевої маршрутизації для оптимізації потоку трафіку або застосувати механізми виявлення та виправлення помилок для відновлення втрачених пакетів.

10) Споживання енергії:

Кількість енергії яка споживається кожною вершиною (пристроєм) у мережі для виконання запиту, не залежно від того, чи був цей запит виконаний успішно чи ні, називається споживанням енергії. При правильному, ефективному балансуванні навантаження споживання енергії мінімізується.

11) Доступність:

Доступність характеризується як позитивне співвідношення зв'язку контролера і сервера, при якому немає помилок серверів і керування. Вираз (6) відображає формальний запис доступності мережевого ресурсу.

$$Availability = \frac{S_f + Time\_out}{\Sigma C_t}, \tag{6}$$

де S_f – несправність (помилка) сокету;

$Time\_out$ – час протягом якого контролер чекав на повідомлення;

C_t – кількість спроб підключення.

ВИСНОВКИ ДО ДРУГОГО РОЗДІЛУ

Використання генетичного алгоритму (GA) у SDN для балансування навантаження пропонує кілька переваг порівняно з іншими традиційними методами, зокрема:

6) GA може оптимізувати розподіл мережевого трафіку та використання ресурсів, завдяки тому, що може знайти ефективну стратегію балансування навантаження. Це може покращити продуктивність мережі та зменшити ризик перевантаження та кількість вузьких місць.
7) GA можуть адаптуватися до змін у мережевому трафіку та відповідно коригувати стратегії балансування навантаження, забезпечуючи ефективне використання мережевих ресурсів.
8) GA можуть працювати з великомасштабними мережами з багатьма вузлами та потоками трафіку, що робить їх придатними для використання в складних середовищах SDN.
9) GA можна налаштувати відповідно до конкретних вимог мережі та включати різні фактори, такі як топологія мережі, моделі трафіку та доступність ресурсів.
10) GA може автоматизувати завдання балансування навантаження та керування мережею, зменшуючи потребу в ручному втручанні та покращуючи загальну ефективність мережі.

Ефективне використання GA в SDN для балансування навантаження може забезпечити оптимізовані результати, адаптивність, масштабованість, настроюваність і автоматизацію, що може покращити продуктивність мережі та зменшити накладні витрати на керування.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОЇ ЧАСТИНИ

3.1 Вибір технологій для створення і тестування програмної частини

Програмна частина в даній роботі – це контролер SDN, який виконуватиме маршрутизацію і створення потоків на основі генетичного алгоритму. Тому необхідно обрати відповідну мову програмування та фреймворк для створення власного SDN контролера із можливість вбудовування GA , як алгоритму для пошуку оптимального шляху в мережі. Також даний фреймворк повинен забезпечувати можливість збору необхідних для маршрутизації GA метрик. Крім того бажано щоб фреймворк був популярним для порівняння результатів роботи створеного контролера з GA із іншими конкуруючими рішеннями.

Вибір відповідних технологій для створення та тестування контролера SDN важливий також із кількох інших причин: використовувані технології повинні забезпечувати необхідну функціональність для задоволення вимог контролера SDN. Це включає підтримку мов програмування, мережевих протоколів і API. Також технології повинні підтримувати бажаний рівень продуктивності мережі, включаючи низьку затримку та високу пропускну здатність. Технології повинні бути масштабованими для підтримки великомасштабних мереж із багатьма пристроями та користувачами.

Також важливо, щоб вони були сумісні з іншими мережевими пристроями та технологіями, щоб забезпечити повну інтеграцію з існуючою мережевою інфраструктурою. Технології мають бути гнучкими, щоб відповідати змінам у вимогах до мережі та дозволяти налаштування контролера SDN.

Щодо вибору мови програмування, то в даному випадку варто враховувати два ключові фактори: наявність SDN фреймворку, що відповідає

поставленим вимогам та доступний у даній мові програмування; наявність інструментів для розробки ефективного генетичного алгоритму. Враховуючи це було обрано SDN фреймворк Ryu та мову програмування Python.

Ryu — це контролер для програмно-конфігурованої мережі SDN з відкритим кодом, який надає платформу для створення спеціальних мережевих додатків і служб. Він розроблений як гнучкий і модульний, дозволяючи створювати та інтегрувати власні програми керування мережею за допомогою ряду мови програмування Python. Завдяки даній особливості він гарно підходить для інтегрування генетичного алгоритму для балансування навантаження.

Ryu надає набір основних модулів і бібліотек, які дозволяють взаємодіяти з мережевими комутаторами та маршрутизаторами, керувати топологією мережі та впроваджувати мережеві політики та правила. Він також включає RESTful API, який дозволяє його легко інтегрувати з іншими мережевими програмами та службами [31].

Однією з ключових переваг Ryu є його природа з відкритим вихідним кодом, що дозволяє спільно розробляти та співпрацювати над додатками та службами керування мережею. Це робить його популярним вибором для академічних досліджень, мережевих експериментів і розгортання SDN у реальному світі. А також дозволяє знаходити та порівнювати результати різних досліджень та рішень.

Мову програмування Python було обрано через те, що вона має великий набір інструментів для наукових обчислень, зокрема для роботи з графами та генетичними алгоритмами. Також ця мова є популярним вибором для впровадження GA завдяки її читабельності, простоті використання та широкій бібліотеці із рішеннями для обчислень і машинного навчання. Також важливим фактором вибору Python є те, що вона підтримує, окрім фреймворку Ryu, ще платформу для емуляції SDN мереж та тестування SDN контролерів – Mininet.

Mininet — це платформа для емуляції мережі, яка дозволяє створювати віртуальні мережі, які можуть імітувати складні мережеві топології та моделі трафіку. Це пакет програмного забезпечення з відкритим кодом, який працює на одній машині та може емулювати мережу віртуальних хостів, комутаторів, маршрутизаторів і зв'язків між ними. Mininet надає платформу для тестування та розробки мережевих програм та протоколів у контрольованому середовищі без потреби у складному та дорогому фізичному мережевому обладнанні [32].

При використанні Mininet, можна створювати власні топології мережі та симулювати потоки трафіку, щоб перевірити продуктивність і поведінку мережевих програм і протоколів. Платформа крім того надає інтерфейс командного рядка для конфігурації мережевих пристроїв, моніторингу мережевого трафіку й аналізу продуктивності мережі.

Mininet часто використовується в дослідницьких та освітніх установах для вивчення концепцій і протоколів комп'ютерних мереж. Вона також використовується мережевими розробниками для тестування та перевірки нових мережевих програм і технологій перед їх розгортанням у виробничих середовищах. Саме тому дана платформа найкраще підходить для тестування продуктивності контролера SDN.

3.2 Створення GA для пошуку оптимальних шляхів на графі

Генетичний алгоритм (GA) в контексті пошуку оптимального шляху на графі можна використовувати для пошуку найкоротшого шляху між двома вершинами графа, оскільки він імітує природний процес еволюції. Генетичний алгоритм (GA) працює, представляючи потенційні рішення задачі у вигляді випадкових шляхів, відомих як особини початкової популяції (або хромосоми).

У разі пошуку оптимальних маршрутів на графі кожна особина (хромосома) представляє можливий шлях між двома вершинами. Потім генетичний алгоритм (GA) застосовує серію генетичних операторів, наприклад таких як селекція, кросинговер і мутація, щоб створити нову популяцію хромосом із існуючої популяції [33].

Оператор відбору передбачає вибір найкраще підібраних хромосом із поточної популяції на основі їхнього значення пристосованості, яке розраховується на основі їх придатності як вирішення проблеми. Оператор кросинговеру передбачає поєднання пар обраних хромосом для створення нових хромосом нащадків, які успадковують характеристики обох батьків. Оператор мутації передбачає введення випадкових змін у хромосоми для створення нових рішень, які раніше не досліджувалися.

Оскільки генетичний алгоритм генерує нові популяції хромосом, значення придатності кожної хромосоми оцінюються, щоб визначити, чи є вони оптимальним рішенням проблеми, тобто чи не було знайдено найкоротший із шляхів. Процес триває, доки не буде знайдено задовільний шлях або не буде виконано попередньо визначену умову припинення, варіації яких було розглянуто в підрозділі 2.6.

Генетичний алгоритм можна використовувати для пошуку найкоротшого шляху між двома вузлами, представляючи можливі шляхи у вигляді хромосом (особин) і застосовуючи генетичні оператори для створення нових шляхів, доки не буде знайдено оптимальний шлях. Функція пристосування, яка використовується в цьому випадку, базуватиметься на відстані або вартості шляху між вузлами.

Для створення генетичного алгоритму були використані технології згадані в попередньому підрозділі 3.1 та були обрані методи та концепції описані в розділі 2. Загальний генетичний алгоритм виконання пошуку оптимальних шляхів на графі, який використовується в роботі, наведено на рис. 3.1. Даний алгоритм було реалізовано за допомогою обраної мови

програмування – Python, та використано для розробки контролера із метою балансування навантаження в SDN.

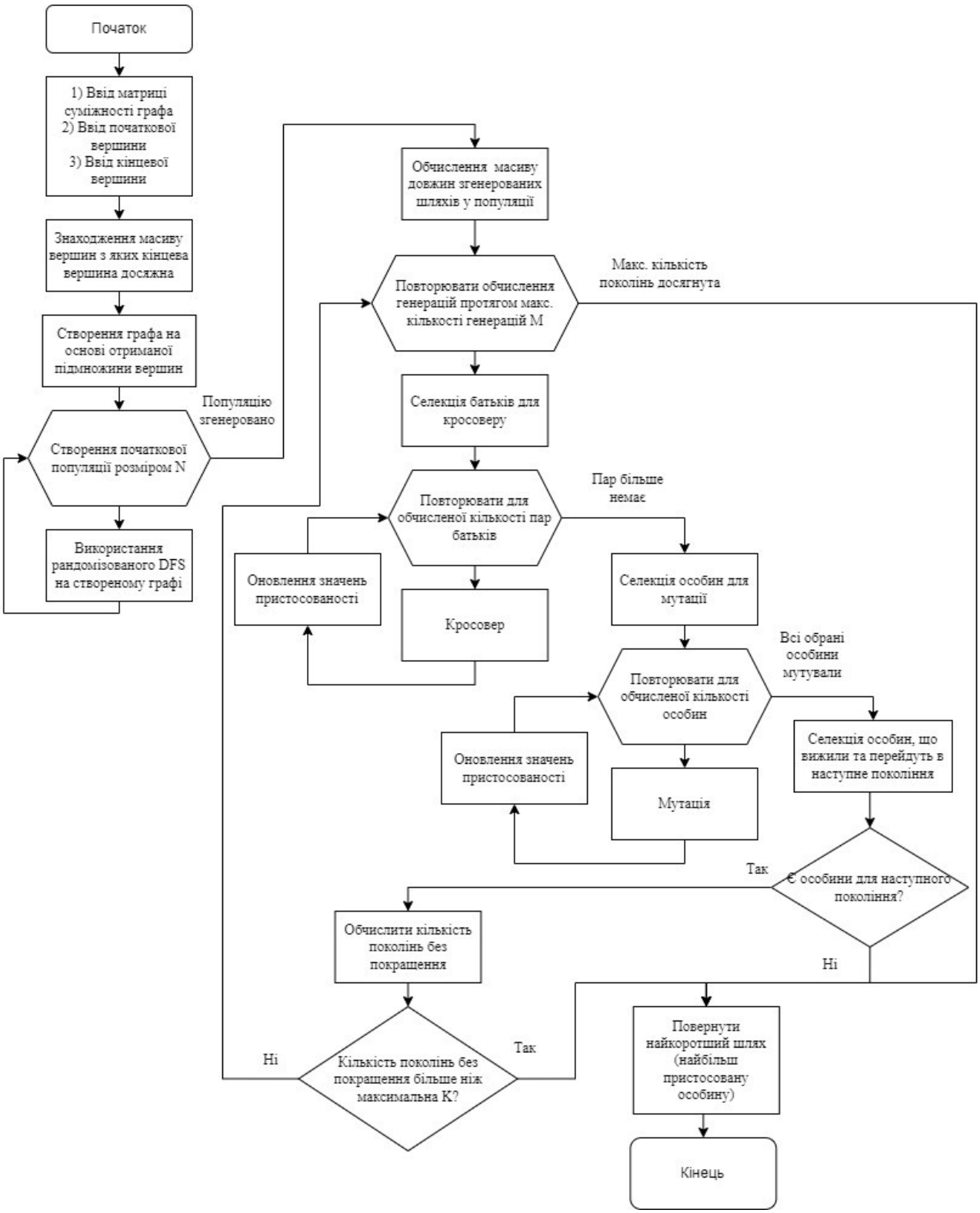


Рис. 3.1 – Генетичний алгоритм пошуку оптимального шляху на графі

Даний алгоритм відображає зв’язок та послідовність виконання генетичних операторів, таких як селекція, кросинговер та мутація. Також на ньому відображена попередня підготовка даних із графа топології для створення графа можливих шляхів із вершини джерела до вершини призначення. За допомогою цього графа і генерується початкова популяція, а також виконуються оператори кросинговеру та мутації.

3.3 Опис програмних компонентів генетичного алгоритму

У цьому підрозділі описані складові програмні функції та структури даних, що формують реалізований генетичний алгоритм. В залежності від призначення ці компоненти розподілені на наступні пункти: ініціалізація, оцінка пристосованості, селекція, розмноження, мутація, заміна та умова завершення. Далі детальніше розглянемо кожен з цих пунктів.

Структури даних, які потрібно передати в функцію генетичного алгоритму, для того, щоб отримати оптимальний шлях. А також параметри роботи генетичного алгоритму, які можуть бути стандартними, але для покращеня продуктивності алгоритму адаптуються відповідно до топології мережі. В таблиці 3.1 наведені структури вхідних даних та опис до них.

Таблиця 3.1

Вхідні дані та параметри GA

№	Аргумент	Тип	Опис
1	graph_data	list[list[int]]	Матриця суміжності графа, яка містить цілочисельні значення довжин ребер графа
2	source	int	Початкова вершина

Продовження Таблиці 3.1

№	Аргумент	Тип	Опис
3	destination	int	Вершина призначення
4	population_size	int	Розмір початкової популяції
5	max_generations_num	int	Обмеження кількості поколінь
6	crossover_prob	float (від 0 до 1)	ймовірність із якою певна пара хромосом із популяції буде обрана для кросинговеру
7	mutation_prob	float (від 0 до 1)	ймовірність того, що певна хромосома буде піддана мутації
8	survival_pct	float (від 0 до 1)	коефіцієнт, що відображає відсоток від популяції, що буде перенесений до наступного покоління
9	max_generations_unimproved	int	максимальна дозволена кількість поколінь, під час яких найкраща особина не покращує свій результат (шлях не скорочується)

- Ініціалізація

Сукупність потенційних шляхів у графі створюється випадковим чином. Кожен потенційний шлях представлено у вигляді масиву унікальних ідентифікаторів вершин, та називається хромосомою або особиною популяції. Ініціалізація виконується лише при запуску та змінах у топології.

Спочатку використовується модифікований обернений алгоритм DFS (англ. Depth First Search), який отримує на вхід матрицю суміжності графа топології мережі та номер вершини призначення, а повертає масив вершин (табл. 3.2 рядок 1), з яких можливо потрапити у вершину призначення. Даний масив є підмножиною вершин графа і використовується для того, щоб створити граф який міститиме лише можливі шляхи із джерела до вершини призначення. За побудову такого «підграфу» відповідає функція, яка створює масиви суміжності (табл. 3.2 рядок 2), використовуючи матрицю суміжності основного графу та масив вершин «підграфу».

Таблиця 3.2

Структури даних генетичного алгоритму

№	Структура	Тип	Опис
1	sub_graph_nodes	list[int]	масив вершин, що входять до обчисленого «підграфу»
2	sub_graph_adj_lists	dict[int, list[int]]	хеш-таблиця масивів суміжності вершин «підграфу»
3	population	list[list[int]]	масив, що містить популяцію особин, тобто шляхів. Кожен шлях являє собою масив номерів вершин, які входять до нього
4	path_lengths	list[float]	масив, який відображає довжини шляхів у популяції, індекс довжини дорівнює номеру відповідного шляху
5	generations_unimproved	int	поточна кількість послідовних поколінь при яких найкраща особина не прогресувала

Продовження таблиці 3.2

№	Структура	Тип	Опис
6	last_best_length	float	найкраще значення довжини шляху із попереднього покоління. Використовується для умови завершення
7	parents_ids	list[int]	порядкові номери шляхів, обраних для виконання оператора кросинговеру
8	mutation_ids	list[int]	порядкові номери шляхів, обраних для виконання оператора мутації
9	survivors_ids	list[int]	масив порядкових номерів шляхів, обраних для збереження в наступному поколінні
10	best_path_id	int	порядковий номер найпристосованішої особини (найкоротшого шляху)

Наступний етап – створення початкової популяції особин (випадкових простих шляхів). Простий шлях графа – це такий шлях, що використовує кожну свою вершину лише один раз та не має циклів. Для вирішення задачі швидкої генерації випадкових шляхів, було створено «підграф» з лише можливих для використання вершин та написано модифікований рандомізований алгоритм DFS. Цей алгоритм отримує на вхід вершини джерела та призначення, а також масив вершин «підграфа» і масиви його суміжності. А на виході отримуємо випадковий простий шлях [34]. Ця процедура повторюється N разів для формування початкової популяції.

На прикінці ініціалізації використовується функція для оцінки пристосованості (тобто довжини) кожного із шляхів у всій популяції. Вона використовує матрицю суміжності, яка періодично оновлюється контролером та містить оцінки для всіх зв’язків графа топології. В таблиці 3 наведено описані структури даних, а в таблиці 3.3 – функції, які реалізують згадані алгоритми.

Таблиця 3.3

Функції, що реалізують алгоритми для ініціалізації GA

Функція	Аргументи	Результат	Опис
reverse_dfs()	destination, graph_data	sub_graph _nodes	модифікований алгоритм DFS, виконує зворотній прохід для знаходження вершин з яких destination доступна
create_sub_graph _adj_lists()	sub_graph_nodes, graph_data	sub_graph _adj_lists	створює хеш-таблицю масивів суміжності для кожної з вершин «підграфа»
randomized_dfs()	source, destination, sub_graph_nodes, sub_graph_adj_lists	випадковий шлях	модифікований алгоритм пошуку в глибину, який випадковим чином обирає кожну наступну вершину шляху
generate_initial _population()	population_size, + аргументи randomized_dfs()	population	застосовує randomized_dfs() для генерації початкової популяції розміром population_size
fitness_all()	population, graph_data	path_lengths	Обчислює довжини шляхів для всієї популяції використовуючи матрицю суміжності

- Оцінка пристосованості

Кожна хромосома в популяції оцінюється на основі функції пристосованості, яка вимірює, наскільки хороший певний шлях (наскільки короткий). Функція пристосованості визначає значення довжини шляху кожній хромосомі, і це значення представляє пристосованість хромосоми. В останньому рядку таблиці 3.3 наведено сигнатуру даної функції та її параметри.

- Селекція:

Метод селекції використовується для вибору найкращих хромосом із популяції на основі їх значень пристосованості. Вибрані хромосоми потім використовуються для створення наступного покоління хромосом. Для розробленого GA була створена універсальна функція селекції selection() (табл.3.4), яка виконує селекцію як для відбору нащадків так і для відбору особин для операторів кросоверингу та мутації.

Таблиця 3.4

Функція selection() та опис її параметрів

№	Параметр	Тип	Опис
1	path_lengths	list[float]	приймає масив довжин (або оцінок) шляхів у популяції
2	remain_pct	float (від 0 до 1)	відсоток особин, які пройдуть відбір, іншими словами ймовірність відбору
3	reverse_prob	bool	якщо встановлено True, то кращою вважається особина, в якої вища оцінка (за замовчуванням False, оскільки подаємо на вхід довжини шляхів, а не їх оцінки)

Продовження таблиці 3.4

№	Параметр	Тип	Опис
4	preserve_best	bool	якщо встановлено True, то найкраща особина із популяції завжди буде обрана, в іншому разі є ймовірність втратити найкращу особину
5	Повертає	list[int]	масив порядкових номерів особин (шляхів) із популяції, які було обрано в наслідок селекції

- Розмноження (кросинговер):

Вибрані хромосоми поєднуються за допомогою оператора кросинговеру для створення нових хромосом потомства. Оператор кросинговеру обмінює генетичну інформацію між двома вибраними хромосомами для створення нового шляху для визначеної селекцією кількості пар.

Функція кросинговеру приймає на вхід два шляхи та повертає кортеж із двох потомків, які ймовірно складатимуться із частин батьків. Дана функція використовує оператор одноточкового кросинговеру, але дещо модифікованого. Перш ніж «розрізати» батьків у випадковій точці та складати частини протилежним чином, функція обчислює масив зі спільних вершин вхідних шляхів, за виключенням початкової та кінцевої вершин. Таким чином отримується масив доступних місць для «розрізу» хромосом, оскільки спільні вершини взаємозамінні в даних шляхах. Потім випадковим чином обирається спільна вершина і виконується кросинговер. В таблиці 3.5 зазначені параметри та вихідні дані функції crossover().

Таблиця 3.5

Функція crossover() та опис її параметрів

Параметр	Тип	Опис
path1	list[int]	приймає шлях 1 для кросинговеру
path2	list[int]	приймає шлях 2 для кросинговеру
Повертає	tuple[list[int], list[int]]	кортеж із двох дочірніх шляхів, якщо масив спільних вершин common_nodes не пустий, інакше повертає початкові шляхи

- Мутація:

Випадкові зміни вносяться в хромосоми потомства за допомогою оператора мутації. Це допомагає ввести новий генетичний матеріал у популяцію та запобігти передчасній конвергенції (збіжності) до неоптимальних рішень. Оператор мутації реалізований за допомогою функції mutation() табл. 3.6. Дана функція працює таким чином, що спочатку перевіряє чи можливо замінити випадково обрану вершину іншою вершиною. Для цього використовується матриця суміжності графа і перевіряються аналогічні ребра для випадково обраної. Потім якщо такі ребра є, серед них випадковим чином обирається одне і в шлях вбудовується нова вершина до якої було приєднане дане ребро. Якщо ж мутація неможлива, то повертається оригінальний шлях.

Таблиця 3.6

Функція mutation() та опис її параметрів

№	Параметр	Тип	Опис
1	path	list[int]	приймає шлях, що вибраний функцією селекції для оператора мутації
2	sub_graph_adj_lists	dict[int, list[int]]	приймає хеш-таблицю масивів суміжності «підграфу»

Продовження Таблиці 3.6

№	Параметр	Тип	Опис
3	Повертає	list[int]	мутований шлях, якщо мутацію можливо виконати, інакше повертає оригінальний шлях

- Заміна:

Хромосоми потомства замінюють найменш придатних членів популяції для формування наступного покоління. Операцію заміни виконує функція selection() із встановленим параметром preserve_best=True для забезпечення принципу елітарності в GA (табл. 5). Також для покращення швидкодії було застосовано зменшення популяції на кожній генерації за рахунок викидання найменш пристосованих особин. Таким чином пришвидшується конвергенція та зменшується час обчислення, хоч і варіація результатів зростає, проте для даної задачі така оптимізація GA необхідна.

- Умова завершення:

Алгоритм продовжує проходити кроки селекції, схрещування, мутації та заміни, доки не буде виконано умову зупинки. В даній реалізації генетичного алгоритму було застосовано три умови завершення. Перша умова це досягнення максимальної кількості поколінь, після чого алгоритм обирає найкращу хромосому та повертає її. Друга умова – зупинка алгоритму, якщо в популяції залишається лише одна особина. Ця умова можлива завдяки імплементованому принципу елітарності, який передбачає збереження найкращої особини в поколінні. А також завдяки оптимізації продуктивності, при якій популяція зменшується на заданий відсоток при переході до наступного покоління. Третя умова це перевищення максимальної кількості поколінь без покращення результатів найкращої особини між поколіннями. Наприклад, якщо 5 поколінь підряд найкращий шлях в кожному з поколінь не

змінювався, то це може бути причиною зупинки алгоритму та повернення даного шляху, як найкращого.

3.4 Програмування контролера SDN із застосуванням GA

Для того, щоб можна було застосувати генетичний алгоритм для побудови оптимальних шляхів OpenFlow (OF) контролером, а отже і забезпечити балансування навантаження, потрібно виконати підготовку. Підготовка заключається в тому, щоб зібрати необхідні дані, які потрібні для знаходження шляху. Серед цих даних ключовими є – OF-комутатор джерела та OF-комутатор призначення. Але існує проблема того, що поки хост призначення не відправив ніяких пакетів у мережу, ми не можемо визначити його адресу, а отже не знаємо до якого комутатора він підключений та через який порт. Для вирішення цієї проблеми використовується flood-алгоритми.

Flood-алгоритм маршрутизації – це метод, за допомогою якого комутатор широкомовно передає пакет на всі свої порти, якщо він не має запису в таблиці маршрутизації для адреси призначення. Але такі дії можуть спричинити негативні наслідки, наприклад, можливо викликати перевантаження мережі та непотрібний трафік, оскільки кожен комутатор у мережі отримуватиме та пересилатиме пакет, навіть якщо пакет не призначений для його сегмента мережі. Це також може призвести до широкомовних штормів, коли велика кількість широкомовних пакетів генерується та передається по всій мережі, що спричиняє проблеми з продуктивністю мережі та потенційні збої мережі. Зазвичай широкомовні шторми виникають через те, що пакети потрапляють у циклічні ділянки мережі та тривалий час блокують пропускну здатність. Проте

Spanning Tree Protocol (STP) вирішує дану проблему, запобігаючи циклам (петлям) у топології мережі, які можуть викликати широкомовні шторми. STP створює топологію без циклів, відключаючи деякі мережеві зв'язки, які не потрібні для підключення. Алгоритм визначає, які зв'язки вимкнути, вибравши кореневий комутатор, який служить точкою відліку для топології мережі. Усі інші комутатори обчислюють найкоротший шлях до кореневого комутатора, а зв'язки, які не знаходяться на цьому шляху, відключаються. Це гарантує, що існує лише один шлях між будь-якими двома комутаторами або пристроями в мережі, і позбавляє безкінечного зациклення широкомовних пакетів [35].

Таким чином, проблема перевантаження мережі та широкомовних штормів, спричинених flood-алгоритмами маршрутизації, вирішується за допомогою STP шляхом створення топології без петель, яка запобігає непотрібному трафіку та забезпечує ефективну доставку пакетів.

У даній роботі алгоритм STP реалізований за допомогою функції контролера - broadcast_stp() та використовується для попередньої маршрутизації пакетів, які не мають вказаного джерела призначення. Коли ж такий пакет досягає хоста призначення, даний хост відправляє пакет у відповідь до хоста джерела. Контролер оновлює таблицю маршрутизації та може використовувати GA для передачі пакету-відповіді в зворотньому напрямку. Перший пакет який передається від певного джерела до певного призначення передається напряму контролером, а потім контролер встановлює потік (Flow) на OF-комутатори, які входять до цього шляху. Потік передбачає двосторонні правила автономної передачі пакетів із даними джерелом та призначенням в заголовках, і повністю автономно [36]. Тобто один раз встановивши запис у таблиці потоків для всіх OF-комутаторів шляху, вони передають відповідний пакет самостійно, без передачі його на OF-контролер.

3.5 Опис програмних компонентів генетичного OF-контролера

Для опису програмних компонентів варто навести життєвий цикл пакета в мережі SDN, яка контролюється реалізованим контролером. Сам генетичний OF-контролер описаний як програмний клас GLBSwitch, який наслідує клас RyuApp із фреймворку Ryu [37]. Він містить основні структури даних, наприклад таблицю маршрутизації, граф топології та дерево STP. Методи даного класу описують правила керування OF-комутаторами для ефективної маршрутизації пакетів у мережі за допомогою генетичного алгоритму та опорного алгоритму STP для визначення адрес [38]. Отже тривіальний шлях пакету в описаній мережі:

1. Пакет надходить на комутатор від вихідного пристрою (хоста).
2. Комутатор перевіряє свою таблицю OpenFlow потоків.
3. Якщо правило передачі пакету із відповідним джерелом та призначенням існує, комутатор пересилає пакет використовуючи правило із таблиці потоків.
4. Якщо відповідного правила немає, комутатор пересилає пакет OF-контролеру.
5. Контролер зберігає запис – адреса джерела/комутатор/порт до таблиці маршрутизації.
6. Якщо адреси призначення немає в таблиці маршрутизації, то виконується алгоритм STP за допомогою методу broadcast_stp().
7. Якщо адреса призначення є, то за допомогою неї із таблиці вибирається комутатор і порт призначення.
8. Виконується GA алгоритм пошуку оптимального шляху, використовуючи номери джерела та призначення.
9. Встановлюються порти для OF-комутаторів знайденого шляху.

10. Контролер встановлює правила передачі пакетів у таблиці OF-комутаторів, що належать знайденому шляху.
11. Виконується GA алгоритм пошуку зворотнього оптимального шляху, використовуючи номери призначення та джерела.
12. Встановлюються порти для OF-комутаторів знайденого зворотнього шляху.
13. Контролер встановлює правила передачі пакетів у таблиці OF-комутаторів, що належать знайденому зворотньому шляху.
14. Контролер напряму передає даний пакет до місця призначення, оскільки пакет не зможе використати щойно створені правила потоків.

Далі будуть описані детальніше структури програмних компонентів класу GLBSwitch. Методи цього класу використовуються для реалізіції етапів зазначених в алгоритмі вище. Першим методом, що виконується при запуску контролера є конструктор класу – __init__(). Конструктор GLBSwitch викликає конструктор батьківського класу RyuApp для коректної ініціалізації базових функцій. Також тут створюються структури даних, такі як хеш-таблиця мак адрес і комутаторів з відповідним портом та дерево STP, які будуть обчислені пізніше.

Основний метод, який отримує вхідні пакети це packet_in_handler() за допомогою декоратора Ryu @set_ev_cls() із параметром ofp_event.EventOFPPacketIn цей метод отримує вхідні повідомлення про пакети, він перевіряє та ігнорує пакети LLDP, що використовуються для низькорівневого виявлення топології та передає інші пакети до головного методу класу – per_flow().

Основний метод, який реалізує алгоритм роботи контролера із пакетами називається per_flow(). Даний метод отримує пакети, що надходять від OF-комутаторів до контролера та обирає який алгоритм застосовувати. Якщо пакет містить достатньо даних і таблиця маршрутизації має запис місця призначення, то використовується генетичний алгоритм для знаходження оптимального

шляху для пакетів від джерела до призначення і назад. Потім встановлюється потік для самостійного пересилання пакетів у мережі OF-комутаторами. Якщо ж інформації не достатньо, або таблиця маршрутизації не містить відповідного запису то використовується алгоритм STP. Загалом per_flow() приймає як параметр тільки OF подію, яка містить повідомлення для контролера, з якого вже отримуються всі необхідні дані, такі наприклад, як адреса джерела та призначення, використовувана версія протоколу OpenFlow, номер комутатора джерела чи порт на який прийшов пакет із повідомлення.

Далі буде розглянуто методи, що реалізують маршрутизацію за допомогою алгоритму STP. Метод, який використовує дерево STP для розповсюдження пакетів називається broadcast_stp(). Він виконує аналогічні функції до методу per_flow() із тою різницею, що не встановлює шляхи, а просто розсилає пакети за допомогою дерева STP. Також даний метод перевіряє чи дерево побудоване та якщо ні, то він викликає метод build_spanning_tree(). Викликаний метод збирає дані про топологію та виконує побудову дерева зв’язків топології за допомогою алгоритму STP. Дані про топологію отримуються завдяки методу get_topology_data(), який за допомогою вбудованих функцій фреймворку Ryu отримує масив комутаторів та масив зв’язків. Функція для отримання масиву комутаторів – ryu.topology.switches.get_switch(), а функція для отримання зв’язків - ryu.topology.switches.get_link(). Потім get_topology_data() перетворює дані масиви в більш зручний формат та повертає їх. Після того як дерево для STP було побудоване, визначаються порти OF-комутатора у топології мережі, які мають завжди ігноруватися згідно з правилами STP. Далі метод розсилає пакет на всі порти окрім заборонених. На цьому робота алгоритму STP завершується, а пакет після проходження певної кількості комутаторів потрапляє до пристрою призначення, який в свою чергу відправляє пакет у відповідь, по якому контролер визначає адресу хоста призначення. Таким чином заповнюється таблиця маршрутизації.

Наступний етап роботи генетичного OF-контролера, за наявності адреси призначення – це знайти оптимальний шлях за допомогою GA. Цю задачу виконує метод find_path(), він відповідає за передачу вхідних даних (матриця суміжності графа топології мережі, початкова та кінцева вершини) до функції генетичного алгоритму, яка була детально описана в підрозділі 3.3. А також відповідає за встановлення портів для знайденого шляху. В таблиці 3.7 описано параметри даного методу.

Таблиця 3.7

Функція find_path() та опис її параметрів

Параметр	Тип	Опис
src_dpid	int	ідентифікатор комутатора джерела
src_port	int	номер порту комутатора джерела на який прийшов пакет
dst_dpid	int	ідентифікатор комутатора призначення
dst_port	int	номер порту комутатора призначення на який потрібно відправити пакет, щоб він потрапив до хоста призначення
self.switch_list	list[int]	масив ідентифікаторів усіх комутаторів у мережі SDN
self.link_list	list[tuple[int, int, {‘port’: int}]]	масив, що містить кортежі із парами ідентифікаторів комутаторів, які мають зв’язок у мережі SDN та номер порту першого комутатора
Повертає	list[tuple[int, int, int]]	шлях – масив кортежів: комутатор, вхідний порт, вихідний порт

Коли оптимальний шлях за допомогою GA було знайдено, контролер повинен встановити записи в таблиці потоків OF-комутаторів мережі SDN. В

реалізованому контролері GLBSwitch це виконано за допомогою методу install_path(), в таблиці 3.8 наведені параметри та опис до них для даного методу. Алгоритм дій даного методу полягає в наступному: із шляху вибереться по черзі комутатор та вхідний/вихідний порт; за допомогою фреймворка Ryu будується правило співставлення пакетів та дія, яку слід виконувати, якщо пакет відповідає правилу; встановлюється потік із заданим правилом та дією. Правило для перевірки пакетів спочатку перевіряє вхідний порт, потім mac-адресу джерела та mac-адресу призначення. Якщо пакет, що надходить на OF-комутатор із встановленим правилом в таблиці потоків, відповідає описаним критеріям, то комутатор зразу виконує встановлену дію, тобто пересилає пакет на порт призначення зі шляху. Ці операції виконуються для кожного комутатора зі шляху і в результаті отримуємо суцільний OpenFlow потік.

Таблиця 3.8 - Функція install_path() та опис її параметрів

Параметр	Тип	Опис
path	list[tuple[int, int, int]]	масив шляхів отриманих за допомогою генетичного алгоритму із методу find_path()
src	mac address	mac-адреса хоста джерела відправленого пакету
dst	mac address	mac-адреса хоста призначення відправленого пакету

Останній метод для завершення алгоритму це – output_packet_port(). Цей метод відповідає за передачу пакету напряму до хоста призначення, під кінець встановлення потоків, оскільки даний пакет ініціалізував створення OpenFlow потоку і не буде переданий по ньому. В таблиці 3.9 наведено опис параметрів методу output_packet_port().

Таблиця 3.9 – Функція output_packet_port() та опис її параметрів

Параметр	Тип	Опис
message	OFPPacketIn	OF-комутатор надсилає отриманий пакет через це повідомлення, а контролер його отримує
datapath	Datapath	екземпляр класу Datapath, що відображає OF-комутатор, який був обраний точкою призначення
port	int	номер порту OF-комутатора на який потрібно надіслати пакет

Таким чином було описано основні методи розробленого класу контролера GLBSwitch. Крім цих методів, ще є методи для перебудови дерева та графу при змінах в топології та очищення таблиці маршрутизації при змінах адрес хостів. Отже було створено повноцінний контролер, який виконує балансування навантаження за допомогою генетичного алгоритму.

ВИСНОВКИ ДО ТРЕТЬОГО РОЗДІЛУ

В результаті виконання третього розділу даної роботи було розроблено генетичний алгоритм для пошуку оптимальних шляхів на графі. Даний алгоритм було програмно реалізовано за допомогою мови Python. Також було створено контролер OpenFlow за допомогою фреймворку Ryu, який використовує даний генетичний алгоритм в якості балансувальника навантаження в мережі SDN.

Технології для створення генетичного алгоритму, такі як селекція, оператори кросинговеру, мутації та інші, були реалізовані з врахуванням їхнього застосування в мережі SDN. Наприклад функція пристосованості (fitness function) може приймати та обчислювати як значення довжин зв'язків графа мережі (де менша довжина – кращий зв'язок), так і значення пропускної здатності зв'язків (де більше значення вважається кращим). Таким чином створений алгоритм може адаптуватися до інформації про мережу, яку надає контролер.

Також був створений OF-контролер для практичного застосування GA для балансування навантаження в мережі і подальшого тестування його результатів. Даний контролер може адаптувати свої та структури даних генетичного алгоритму до змін в мережі. Також він має реалізований алгоритм STP для підтримки роботи генетичного алгоритму, коли вершина призначення не відома і потрібно просто розповсюдити пакет в мережі SDN. За допомогою цього алгоритму визначаються маршрутизатори, які потім використовуються як вершини призначення для побудови оптимальних маршрутів GA.

В наступному розділі буде описано середовище для тестування створеного генетичного контролера SDN. А також результати тестування і порівняння з іншими аналогічними рішеннями.

РОЗДІЛ 4
ЕКСПЕРИМЕНТИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Середовище для тестування генетичного контролера SDN

Для перевірки роботи та продуктивності розробленого генетичного контролера SDN, потрібно обрати топологію і параметри з'єднання, такі наприклад, як пропускна здатність зв'язків. Також потрібно обрати топологію і параметри, що відповідають аналогічним рішенням для порівнюваності результатів. Отже, враховуючи ці критерії, було обрано топологію Fat-Tree (рис. 4.1), яка використовується для тестування багатьох аналогічних рішень та знайшла своє застосування в датацентрах [39].

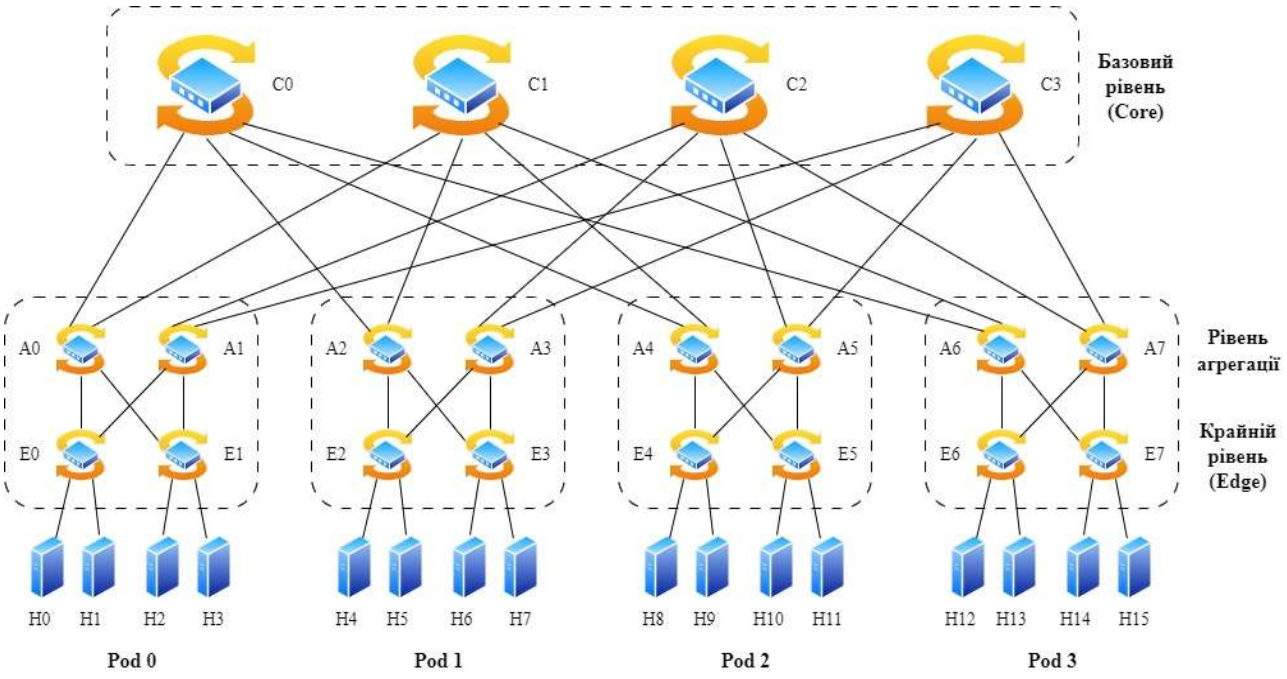


Рис. 4.1 – Топологія Fat-Tree із 20 комутаторів та 16 хостів

Fat-Tree — це популярна топологія мережі, яка використовується в центрах обробки даних (ЦОД) для забезпечення високої пропускної здатності, низької затримки та відмовостійкого з'єднання між серверами, сховищами та

мережевими комутаторами. Свою назву топологія Fat-Tree отримала від свого зовнішнього вигляду у вигляді дерева з основними комутаторами на верхньому ярусі, агрегаційними комутаторами посередині та кінцевими комутаторами внизу. Топологія Fat-Tree є ієрархічною з кількома рівнями комутаторів, де кожен рівень забезпечує окрему функцію.

На базовому рівні топології є один або кілька основних комутаторів, які забезпечують високошвидкісне підключення між різними комутаторами агрегації. Ці комутатори, як правило, високопродуктивні та мають великі буфери, високу пропускну здатність і низьку затримку.

Рівень агрегації складається з кількох комутаторів, які підключаються до основних комутаторів і забезпечують підключення до крайніх комутаторів. Комутатори агрегації агрегують трафік від периферійних комутаторів і пересилають його до основних комутаторів. Кількість комутаторів агрегації в топології Fat-Tree зазвичай визначається кількістю крайніх комутаторів і бажаною пропускною здатністю.

Нижній рівень складається з кількох комутаторів, які підключаються до серверів і пристроїв зберігання в ЦОД. Граничні комутатори забезпечують підключення до кінцевих пристроїв, таких як сервери та пристрої зберігання даних, і пересилають трафік до комутаторів агрегації.

Одна з головних переваг топології Fat-Tree полягає в тому, що вона забезпечує кілька надлишкових шляхів між будь-якими двома пристроями в мережі, що підвищує відмовостійкість мережі та покращує її надійність. Крім того, топологія Fat-Tree забезпечує високий рівень масштабованості, дозволяючи мережі легко розширюватися шляхом додавання більше комутаторів або пристроїв без впливу на продуктивність мережі.

Для тестування балансування навантаження в SDN ця топологія гарно підходить через те, що має такі параметри: регулярність, симетрія, рекурсивна масштабованість, максимальна відмовостійкість, логарифмічний діаметр і бісекційна масштабованість [40]. Тому для тестування створеного генетичного

контролера SDN буде використано топологію Fat-Tree, де сервери (хости) спілкуються з іншими через ієрархічну структуру мережевих комутаторів.

Для задання топології мережі та її параметрів використовуватиметься інструмент віртуалізації мереж – Mininet [41]. Топологія складатиметься із 20 OF-комутаторів, контролера SDN та 16 підключених хостів до крайніх комутаторів. Контролер буде підключений до всіх комутаторів (рис. 17).

Під час симуляції продуктивність мережі буде вимірюватись за допомогою утиліти Iperf кожні 15 секунд в інтервалах по 5 секунд. У підрозділі 2.7 було розглянуто оосновні параметри мережі які можна використати також для тестування. Тому для порівняння будуть використані такі показники: затримка (latency) – це час за який пакет доходить до одержувача і повертається назад у вигляді відповіді; джиттер (jitter) – варіація в затримці (delay) односторонньої подорожі пакета, для цього відправник надсилає тривалий рівномірний потік пакетів, а отримувач вимірює варіацію затримки між ними; пропускна здатність мережі (network throughput) – швидкість передачі успішних повідомлень в мережі.

4.2 Результати тестування

Для внесення навантаження в мережу використовувалась утиліта iperf між парами хостів у мережі, наприклад, h0-h8, h1-h9 і так далі до h7-h15, таким чином при обмеженні пропускної здатності на портах комутаторів у 100Мбіт/с в топології Fat-Tree виникає необхідність вибору маршрутів трафіку для покращення пропускної здатності і завдяки цьому можливо оцінити роботу алгоритмів балансування навантаження в даній мережі. В інтервалах по 15 секунд за допомогою утиліт ping та iperf (udp) вимірювалась затримка мережі й

джиттер, а також за допомогою утиліти iperf по протоколу TCP була виміряна пропускна здатність між крайніми вузлами даної топології h0-h15, протягом 5 секунд. Тестування проводилось протягом 10 хв, а результати зберігались до таблиці. Потім результати вимірів використовувались для побудови графіків для порівняння розробленого способу з існуючими рішеннями та вцілому з відсутністю балансування навантаження.

Далі на рис. 4.2 зображено графік порівняння пропускної здатності мережі з використанням генетичного алгоритму, без балансування та балансування за допомогою алгоритму Дійкстри. Як видно на графіку генетичний алгоритм розподіляє навантаження в мережі найбільш рівномірно, тобто маршрути для проходження трафіку для тестування мережі практично відокремлені, тому пропускна здатність мережі найбільша – приріст складає від 9% до 43% для алгоритму Дійкстри та без балансування відповідно.

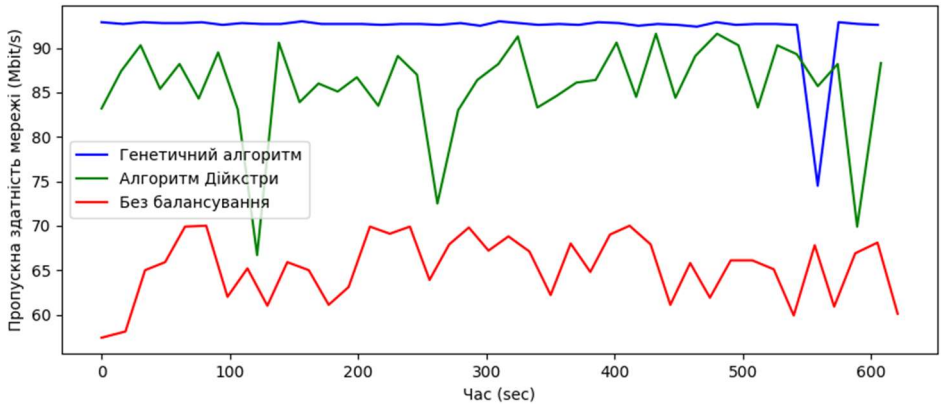


Рис. 4.2 – Порівняння пропускної здатності мережі на топології Fat-Tree (між h0 та h15)

На рис. 4.3 показано графік затримки мережі (RTT) при використанні розробленого генетичного алгоритму, без балансування та балансування за допомогою алгоритму Дійкстри. Балансування навантаження з використанням обох алгоритмів значно знижує затримку в мережі. Також варто зазначити, що генетичний алгоритм знижує затримку в мережі в середньому на 8% порівняно

з використанням алгоритму Дійкстри і на 28% порівняно з відсутністю балансування тому, що генетичний алгоритм виконується швидше для даної топології, та може краще маштабуватись. Крім того параметри GA можливо в подальшому оптимізувати більше для досягнення кращих результатів.

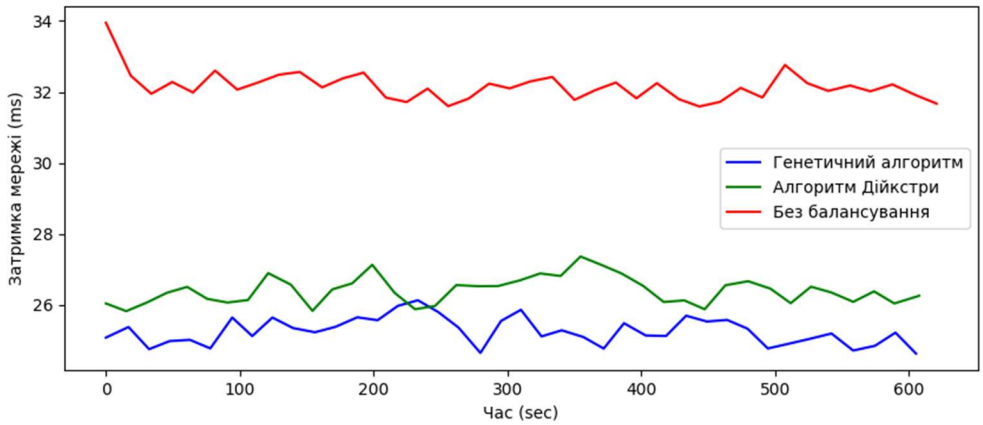


Рис. 4.3 – Порівняння затримки в мережі на топології Fat-Tree (між h0 та h15)

На наступному рис. 4.4 зображено графік джиттеру (jitter) мережі при використанні описаного способу, без балансування та з балансуванням використовуючи алгоритм Дійкстри. Як можна спостерігати при заданих параметрах мережі в 2 мс на кожен зв'язок між двома вузлами, джиттер відрізняється менше ніж затримка, використання GA дає в середньому на 23% кращі показники джиттеру порівняно з відсутністю балансування.

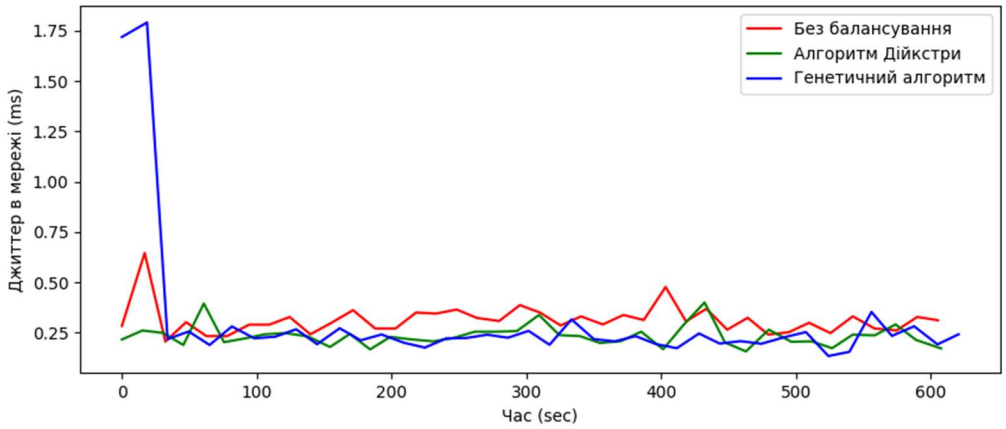


Рис. 4.4 – Порівняння джиттеру в мережі на топології Fat-Tree (між h0 та h15)

ВИСНОВКИ ДО ЧЕТВЕРТОГО РОЗДІЛУ

Результати тестування запропонованого методу балансування навантаження в SDN на основі генетичного алгоритму показали, що підхід був ефективним у зменшенні перевантаження мережі та покращенні продуктивності мережі. Генетичний алгоритм зміг знайти оптимальні маршрути для мережевого трафіку на основі оцінки придатності ключових параметрів мережі, таких як пропускна здатність, час відгуку та наскрізна затримка.

Тестування проводилося з використанням емулятора Mininet і контролера Ryu, що дозволило створити реалістичне мережеве середовище для оцінки запропонованого методу. Результати показали, що генетичний алгоритм зміг значно зменшити перевантаження мережі порівняно з традиційними методами балансування навантаження. Використання генетичного алгоритму дало в середньому на 8-9% кращі результати пропускної здатності та затримки мережі в порівнянні із розглянутими рішеннями та від 23% до 43% кращі результати в порівнянні з відсутністю алгоритму для балансування навантаження в мережі.

Тестування також підкреслило важливість ретельного вибору параметрів для оцінки придатності та стратегій мутації та відбору для генетичного алгоритму. Було виявлено, що використання стратегії на основі рангової селекції і методу одноточкового кросинговеру дало найкращі результати.

На завершення результати тестування запропонованого методу балансування навантаження в SDN на основі генетичного алгоритму продемонстрували, що підхід був ефективним у покращенні продуктивності мережі та зменшенні перевантаження мережі. Тестування дає цінну інформацію про впровадження генетичних алгоритмів у SDN і показує, що цей підхід може революціонізувати спосіб проектування та керування мережами.

ВИСНОВКИ

Запропонований метод балансування навантаження в програмно-конфігурованих мережах (SDN) з використанням генетичного алгоритму пропонує перспективний підхід до вирішення проблеми перевантаження мережі та підвищення продуктивності мережі. Цей метод використовує генетичний алгоритм для пошуку найкращих можливих маршрутів для мережевого трафіку на основі оцінки придатності ключових параметрів мережі, таких як пропускна здатність, час відгуку та наскрізна затримка.

У дослідженні розглянуто попередні роботи з балансування навантаження в SDN і визначено переваги відкритих рішень SDN перед комерційними. Дослідження також окреслило недоліки існуючих методів балансування навантаження та підкреслило переваги використання генетичних алгоритмів для балансування навантаження в SDN.

Реалізація генетичного алгоритму для балансування навантаження в SDN вимагає ретельного розгляду різних факторів, таких як стратегії відбору та мутації, мови програмування та параметри мережі для оцінки придатності. Запропонований метод був оцінений за допомогою Mininet і контролера Ryu і показав багатообіцяючі результати щодо зменшення перевантаження мережі та підвищення продуктивності мережі.

Загалом, метод балансування навантаження в SDN на основі генетичного алгоритму пропонує потужний підхід до оптимізації мережі, що веде до більш ефективних, надійних і масштабованих мереж. Поєднання генетичних алгоритмів і SDN має потенціал революціонізувати спосіб проектування та управління мережами, і це дослідження робить цінний внесок у цю сферу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aly W., Hosny F. Controller adaptive load balancing for SDN networks [Розділ книги] // Eleventh International Conference on Ubiquitous and Future Networks (ICUFN).: IEEE, 2019.

2. Anish G., Mrs. T. Manoranjitham. A study on load balancing techniques in SDN [Журнал] // International Journal of Engineering & Technology. - 2018 р.. - 4 : Т. 7. - сс. 174-177.

3. Bhattarai, Anup. IP-Based Hashing Load Balancer in Software Defined Networking [Звіт].: Pulchowk Campus, 2019.

4. Bottaci, Leonardo, A genetic algorithm fitness function for mutation testing [Збірка доповідей] // Proceedings of the SEMINALL-workshop at the 23rd international conference on software engineering. - Toronto, 2001.

5. Chi Po-Wen and Wang, Ming-Hung and Guo, Jing-Wei. SDN Migration: An Efficient Approach to Integrate OpenFlow Networks with STP-Enabled Networks [Розділ книги] // International Computer Symposium (ICS). - 2016.

6. Gautam S. Different types of Load Balancing Algorithms [Звіт]. - Enjoy Algorithms, 2022 // [Електронний ресурс]. Режим доступу: https://www.enjoyalgorithms.com/blog/types-of-load-balancing-algorithms.

7. Ghaffarinejad A. Comparing a commercial and an SDN-based load balancer in a campus network [Книга]. - Arizona : Arizona State University, 2015.

8. Greenhalgh D. and M., Stephen. Convergence criteria for genetic algorithms [Журнал] // SIAM Journal on Computing. - 2000 р.. - 1 : Т. 80. - сс. 269-282.

9. Hacham S. and D., Norashidah Md and B., Nagaletchumi. Load Balancing in Software-Defined Data Centre With Fat Tree Architecture [Збірка доповідей] // 4th International Conference on Smart Sensors and Application (ICSSA). - 2022.

10. Hinterding R. and Gielewski, H. and P., Thomas C. The Nature of Mutation in Genetic Algorithms. [Розділ книги] // ICGA. - [місце видання невідоме] : Citeseer, 1995.

11. Joshi M. and Hadi, Theyazn H. A review of network traffic analysis and prediction techniques [Журнал] // arXiv preprint arXiv:1507.05722. - 2015 p..

12. Karakus M. and Durresi, Arjan. Quality of service (QoS) in software defined networking (SDN): A survey [Журнал] // Journal of Network and Computer Applications. - 2017 p.. - T. 80. - cc. 200-218.

13. Katoch S. and C., Sumit S. and K., Vijay. A review on genetic algorithm: past, present, and future [Журнал] // Multimedia Tools and Applications. - 2021 p.. - Springer : T. 80. - cc. 8091-8126.

14. Kaur K. and Kaur, Sukhveer and G., V. Least time based weighted load balancing using software defined networking [Збірка доповідей] // Advances in Computing and Data Sciences: First International Conference. - Ghaziabad, 2017.

15. Kavana H.M. and K., V.B. and Madhura, B. and Kamat, N. Load balancing using SDN methodology [Журнал] // Int. J. Eng. Res. Technol. - 2018 p.. - 5 : T. 7. - cc. 206-208.

16. Khalid J. Mohammed M. Selection Methods for Genetic Algorithms [Звіт]. - Rabat : Mohammed V–Agdal University, 2013.

17. Kierstead D. P and DelBalzo, Donald R. A genetic algorithm applied to planning search paths in complicated environments [Журнал] // Military Operations Research. - 2003 p.. - cc. 45-59.

18. Lakhina A. and P., K. Structural analysis of network traffic flows [Розділ книги] // Proceedings of the joint international conference on Measurement and modeling of computer systems. - 2004.

19. Martín S. Jessica C., Ignacio P., Nélida B. On Stopping Criteria for Genetic Algorithms [Розділ книги] // Brazilian Symposium on Artificial Intelligence. : SBIA, 2004.

20. Meena R. C. and B., Mahesh and N., Meenakshi. RYU SDN Controller Testbed for Performance Testing of Source Address Validation Techniques [Збірка доповідей] // 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE). - 2020.

21. Mininet Project Contributors Mininet Walkthrough [Звіт]. - 2022 // [Електронний ресурс]. Режим доступу: http://mininet.org/walkthrough/.

22. Omer Y. Abdulkarim H. and M., Amin B. A. and A., Ashraf G. Performance analysis of round robin load balancing in sdn [Збірка доповідей] // International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE). - 2020.

23. Omran M. A. Alssaheli Z. Zainal Abidin, N. A. Zakaria, Z. Abal A. Software Defined Network based Load Balancing for Network Performance Evaluation [Журнал] // (IJACSA) International Journal of Advanced Computer Science and Applications. - 2022 p.. - 4 : T. 13.

24. Open Networking Foundation Mininet [Звіт]. - 2023 // [Електронний ресурс]. Режим доступу: https://opennetworking.org/mininet/.

25. Open Networking Foundation OpenFlow Switch Specification (v1.3.0) [Книга]. - 2012 [Електронний ресурс]. Режим доступу: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf.

26. Pauline C. Dario G., U. Mueller. Genetic algorithms: A powerful tool for large-scale nonlinear optimization problems [Журнал] // Computers & Geosciences. - 1994 p.. - 7-8 : T. 20. - cc. 1229-1236.

27. Queiroz W. and Capretz, Miriam A.M. and D., Mario. An approach for SDN traffic monitoring based on big data techniques [Журнал] // Journal of Network and Computer Applications. - 2019 p.. - T. 131. - cc. 28-39.

28. Radojevi B. and Z., Mario. Analysis of issues with load balancing algorithms in hosted (cloud) environments [Розділ книги] // Proceedings of the 34th international convention MIPRO. - : IEEE, 2011.

29. Randy L., Haupt S., E. Haupt. Practical Genetic Algorithms [Книга]. - : Copyright © 2004 John Wiley & Sons, Inc., 2003.

30. Razali N., Mohd and G., J. Genetic algorithm performance with different selection strategies in solving TSP [Розділ книги] // Proceedings of the world congress on engineering. - Hong Kong : International Association of Engineers, 2011.

31. Rungta N. and M., Eric G. Generating counter-examples through randomized guided search [Розділ книги] // Model Checking Software: 14th International SPIN Workshop. - Berlin : Springer, 2007.

32. RYU project team RYU SDN Framework Using OpenFlow 1.3 (RyuBook) [Книга]. - 2014 // [Електронний ресурс]. Режим доступу: https://book.ryu-sdn.org/en/Ryubook.pdf.

33. Ryu SDN Framework Community Welcome to RYU the Network Operating System(NOS) [Звіт]. - 2023 // [Електронний ресурс]. Режим доступу: https://ryu.readthedocs.io/en/latest/index.html.

34. Semong T., Maupong T., Anokye S., Kehulakae K., Dimakatso S., Boipelo G., Sarefo S. Intelligent Load Balancing Techniques in Software Defined Networks: A Survey [Звіт]. - 2020 // [Електронний ресурс]. Режим доступу: https://doi.org/10.3390/electronics9071091.

35. Shukla A. and P., Hari M. and M., Deepti. Comparative review of selection techniques in genetic algorithm [Розділ книги] // 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE). - 2015.

36. S. Subramanian, S. Voruganti. Software-Defined Networking (SDN) with OpenStack [Книга]. : Packt Publishing, 2016.

37. Umbarkar A. J. and S., Pranali D. Crossover operators in genetic algorithms: a review. [Журнал] // ICTACT journal on soft computing. - 2015 р.. - 1 : Т. 6.

38. Y. Yun-Shuai, and Chih-Heng K. Genetic algorithm-based routing method for enhanced video delivery over software defined networks [Журнал] // International Journal of Communication Systems 31.1. - 2018 p.. - cc. 33-91.

39. Z. Hong and F., Yaming and C., Jie. LBBSRT: An efficient SDN load balancing scheme based on server response time [Журнал] // Future Generation Computer Systems. - 2018 p.. - T. 80. - cc. 409-416.

40. Гніденко М.П. Вишнівський В.В., Ільїн О.О. Побудова SDN мереж, Державний університет телекомунікацій, 2019 [Книга] // [Електронний ресурс]. Режим доступу: https://dut.edu.ua/uploads/l_1710_34882811.pdf.

41. Климаш М. Бешлей М. Дещинський Ю. Панченко О. Розробка методу балансування навантаження в SDN мережах на основі модифікованого протоколу STP [Журнал] // Комп'ютерні технології друкарства. - 2015 р. - cc. 146-155.