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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

«На правах рукопису»

УДК _____

До захисту допущено

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

«__» _____ 2025 р.

Магістерська дисертація

на здобуття ступеня магістра

**за освітньо-професійною програмою «Системне програмування та спеціалізовані комп'ютерні системи»
зі спеціальності 123 «Комп'ютерна інженерія»
на тему: «Система тестування ефективності алгоритмів балансування навантаження в SDN мережах»**

Виконав:
Студент VI курсу, групи КВ-41мп
Костюков Сергій Васильович _____

Керівник:
доцент кафедри СПіСКС, к.т.н., доцент
Мартинова Оксана Петрівна _____

Консультант з нормоконтролю:
доцент кафедри СПіСКС, к.т.н.
Боярінова Ю.Є. _____

Рецензент:
доцент кафедри ПЗКС ФПМ, к.т.н., доцент
Олещенко Л.М. _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ - 2025 рік

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування та спеціалізовані комп'ютерні системи»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Віталій РОМАНКЕВИЧ

«___» _____ 20__ р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Костюкову Сергію Васильовичу

1. Тема дисертації: «Система тестування ефективності алгоритмів балансування навантаження в SDN мережах», науковий керівник дисертації Мартинова Оксана Петрівна, к.т.н, доцент, доцент кафедри СПіСКС, затверджені наказом по університету від «06» листопада 2025 р. № 4836-с
2. Термін подання студентом дисертації: 09 грудня 2025 р.
3. Об'єкт дослідження: модель системи тестування ефективності алгоритмів балансування навантаження в SDN мережах
4. Перелік завдань, які потрібно зробити:
    1. Провести теоретичний аналіз архітектури програмно-визначених мереж, особливостей передавання трафіку, принципів роботи OpenFlow та ролі SDN-контролера у прийнятті рішень.
    2. Дослідити існуючі SDN-контролери (Ryu, POX, Floodlight, OpenDaylight), оцінити їх придатність для побудови та тестування алгоритмів балансування навантаження.
    3. Виконати аналіз можливостей середовища мережевої емуляції Mininet, оцінити його сумісність з SDN-контролерами та потенціал для моделювання різних мережевих топологій.
    4. Розробити архітектуру системи тестування алгоритмів балансування навантаження, включно з механізмами моделювання мережі,

генерування трафіку, збору телеметрії, обробки метрик та інтеграції алгоритмів.

5. Реалізувати підсистему серверної оркестрації для керування експериментами, взаємодії з Mininet і SDN-контролером, а також інтерфейс для підключення зовнішніх алгоритмів балансування.
6. Розробити клієнтський веб-інтерфейс для конфігурації експериментів, моніторингу мережевого стану та візуалізації телеметрії в режимі реального часу і демонстраційного режиму.
7. Провести експериментальні дослідження роботи алгоритмів балансування навантаження в різних топологіях та сценаріях трафіку, здійснити аналіз отриманих результатів та сформувати узагальнені висновки.

5. Орієнтовний перелік графічного (ілюстративного) матеріалу
    1. Слайд із темою, актуальністю, метою та основними завданнями магістерської роботи.
    2. Схема архітектури програмно-визначуваної мережі (SDN) та принципи передавання трафіку.
    3. Структура та принципи роботи середовища мережевої емуляції Mininet.
    4. Схема встановлення та взаємодії SDN-контролера з комутаторами OpenFlow.
    5. Архітектурна модель розробленої системи тестування та ілюстрації процесу моделювання й візуалізації результатів.
6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «__» _____ 2025 р.

Календарний план

№ з/п	Назва етапів виконання дисертації	Термін виконання етапів дисертації	Примітка
1	Узгодження керівника кваліфікаційної роботи (МД) та його затвердження	15.09.2024	
2	Визначення тематики дослідження магістерської дисертації	15.10.2024	
3	Формулювання об`єкта і предмета дослідження	01.12.2024	

4	Визначення структури магістерської дисертації	01.03.2025	
5	Зміст та вступ до МД. Підготовка тез доповіді для виступу на конференції	20.03.2025	
6	Перший розділ МД з висновками. Підготовка тез доповіді для виступу на конференції	15.04.2025	
7	Другий розділ МД з висновками	15.05.2025	
8	Остаточне формування тематики кваліфікаційної роботи	25.06.2025	
9	Тема магістерської дисертації, заява на ім’я завідувача кафедри з точною назвою МД	10.09.2025	
10	Зміст та вступ до МД	24.09.2025	
11	Реферат до МД (українською мовою) та перший розділ з висновками	08.10.2025	
12	Титульний аркуш, завдання та другий розділ МД з висновками	22.10.2025	
13	Залік з науково-дослідної практики	31.10.2025	
14	Попередній захист магістерської дисертації	28.11.2025	
15	Надання керівнику МД остаточного варіанту дисертації для перевірки на запозичення та збіг/ідентичність/схожість	05.12.2025	
16	Здача всіх документів магістерської дисертації секретарю комісії	16.12.2025	
17	Захисти МД	22.12.2025	

Студент _____ Сергій КОСТЮКОВ

(підпис)

Науковий керівник дисертації _____ Оксана МАРТИНОВА

(підпис)

РЕФЕРАТ

Актуальність теми. Сучасні комп’ютерні мережі характеризуються високою динамікою навантажень, зростанням обсягів трафіку та необхідністю забезпечення стабільної продуктивності сервісів. Традиційні підходи до керування мережею стають недостатньо гнучкими, що зумовило появу нової архітектури - Software Defined Networking (SDN).

SDN дозволяє централізовано керувати мережею через програмний контролер, а отже створює умови для реалізації інтелектуальних алгоритмів балансування навантаження.

Розробка системи тестування ефективності алгоритмів балансування в SDN мережах є актуальною задачею, оскільки забезпечує можливість дослідження, порівняння та оптимізації різних підходів до розподілу навантаження у мережах.

Об’єктом дослідження є процес балансування навантаження у програмно визначених мережах (SDN).

Предметом дослідження є алгоритми балансування навантаження та методи їх тестування у середовищі SDN.

Мета роботи: розроблення тестового середовища для оцінки ефективності різних алгоритмів балансування навантаження у SDN мережах, а також проведення порівняльного аналізу їх продуктивності за заданими метриками.

Для досягнення мети в роботі вирішуються такі задачі:

1. Провести аналіз існуючих архітектур SDN і рішень для балансування навантаження;
2. Розробити архітектуру тестового середовища для моделювання мережі SDN;

3. Реалізувати програмні модулі для підключення різних алгоритмів балансування;

4. Організувати збір та аналітичну обробку метрик ефективності;

5. Провести експериментальне порівняння алгоритмів і визначити оптимальні умови їх застосування.

Наукова новизна полягає в наступному:

1. Запропоновано узагальнену архітектуру системи тестування алгоритмів балансування в SDN середовищах;

2. Розроблено методику оцінювання ефективності балансувальних алгоритмів із використанням симуляційного середовища Mininet та контролера RYU;

3. Запропоновано підхід до адаптивного вибору алгоритму балансування на основі аналізу поточного стану мережі.

Практична цінність отриманих результатів полягає в можливості використання створеної системи для:

- дослідження продуктивності різних балансувальних стратегій;
- навчання студентів і дослідників принципам роботи SDN;
- попереднього тестування алгоритмів перед їх впровадженням у реальних мережах.

Система дозволяє швидко моделювати мережеві сценарії, відтворювати навантаження та отримувати кількісні показники ефективності роботи різних алгоритмів балансування.

Апробація роботи. Основні результати дослідження були представлені та обговорені на студентських науково-практичних конференціях факультету прикладної математики КПІ ім. Ігоря Сікорського та використані у навчальному процесі для демонстрації роботи SDN-контролерів і симуляційних середовищ.

Структура та обсяг роботи. Магістерська дисертація складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків.

У **вступі** подано загальну характеристику теми, обґрунтовано актуальність, сформульовано мету, задачі, об’єкт і предмет дослідження, визначено наукову новизну та практичну цінність.

У **першому розділі** проведено аналіз архітектури SDN, принципів роботи контролерів та огляд існуючих алгоритмів балансування навантаження.

У **другому розділі** наведено обґрунтування вибору напрямку досліджень та наводяться методи вирішення задач та їх порівняльні оцінки, надається увага науковій новизні запропонованого проекту.

У **третьому розділі** розроблено архітектуру тестового середовища, описано моделі мережі, алгоритми балансування та методи збору статистики.

У **третьому розділі** наведено результати тестування, порівняння ефективності алгоритмів та рекомендації щодо їх оптимального використання.

У **висновках** узагальнено результати дослідження та наведено перспективи подальшого розвитку системи, зокрема інтеграції моделей машинного навчання для прогнозування навантаження.

Робота викладена на 88 сторінках, містить 19 рисунків, 4 таблиці, 5 додатки та список із 14 найменувань літературних джерел.

Ключові слова: SDN, балансування навантаження, контролер, Mininet, RYU, тестування, алгоритми, інтелектуальна система.

ABSTRACT

Relevance of the topic. Modern computer networks are characterized by high dynamics of workloads, increasing traffic volumes, and the need to ensure stable service performance. Traditional approaches to network management are becoming insufficiently flexible, which has led to the emergence of a new architecture — Software Defined Networking (SDN).

SDN enables centralized network management through a programmable controller, creating favorable conditions for implementing intelligent load balancing algorithms.

The development of a system for testing the efficiency of load balancing algorithms in SDN networks is an important and relevant task, as it provides the ability to investigate, compare, and optimize various approaches to traffic distribution.

The object of research is the process of load balancing in software-defined networks (SDN).

The subject of research is load balancing algorithms and methods for evaluating their efficiency in an SDN environment.

The purpose of the work is to develop a test environment for assessing the performance of different load balancing algorithms in SDN networks and to conduct a comparative analysis of their efficiency based on defined metrics.

To achieve this purpose, the following tasks are addressed in the work:

1. Analyze existing SDN architectures and load balancing solutions.
2. Develop an architecture for a test environment for SDN network modeling.
3. Implement software modules for integrating various load balancing algorithms.

4. Organize the collection and analytical processing of performance metrics.

5. Conduct an experimental comparison of algorithms and determine the optimal conditions for their application.

The **scientific novelty** of the work lies in the following:

1. A generalized architecture of a system for testing load balancing algorithms in SDN environments has been proposed.

2. A methodology for evaluating load balancing efficiency using the Mininet simulation environment and the RYU controller has been developed.

3. An approach to adaptive selection of a load balancing algorithm based on real-time network state analysis has been introduced.

The practical value of the obtained results lies in the ability to use the developed system for researching the performance of different balancing strategies, training students and researchers in SDN technologies, and preliminary testing of algorithms prior to deployment in real networks.

The system enables rapid modeling of network scenarios, reproduction of workloads, and acquisition of quantitative performance indicators of load balancing algorithms.

Approbation of the work. The main results of the research were presented and discussed at student scientific-practical conferences of the Faculty of Applied Mathematics of Igor Sikorsky Kyiv Polytechnic Institute and were used in the educational process to demonstrate the operation of SDN controllers and simulation environments.

Structure and volume of the work. The master's thesis consists of an introduction, four chapters, conclusions, a list of references, and appendices.

The **introduction** provides the general characteristics of the topic, substantiates its relevance, and formulates the purpose, tasks, object and subject of research, as well as the scientific novelty and practical significance of the work.

The first chapter analyzes the architecture of SDN, the principles of controller operation, and the existing load balancing algorithms.

The **second chapter** presents the justification for the chosen research direction, methods for solving the problems, and their comparative evaluation, with emphasis on the scientific novelty of the proposed solution.

The **third chapter** develops the architecture of the test environment, describes network models, load balancing algorithms, and methods of statistics collection.

The **fourth chapter** presents the results of testing, comparison of algorithm efficiency, and recommendations for their optimal use.

The **conclusions** summarize the research results and outline prospects for further development of the system, including the integration of machine learning models for load prediction.

The thesis is presented on 88 pages and contains 19 figures, 4 tables, 5 appendices, and a list of 14 references.

Keywords: SDN, load balancing, controller, Mininet, RYU, testing, algorithms, intelligent system.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

SDN (Software Defined Networking) — програмно-визначувана мережа, у якій логіка керування відокремлена від рівня пересилання даних та реалізується через централізований контролер.

SDN Controller — центральний елемент SDN-архітектури, що приймає рішення про маршрутизацію, керує потоками даних і взаємодіє з комутаторами через southbound-інтерфейс.

OpenFlow — стандартний протокол взаємодії між SDN-контролером і комутаторами, який дозволяє встановлювати, змінювати та видаляти правила маршрутизації.

Flow Table (FT) — таблиця потоків у комутаторі, що містить правила обробки пакетів, встановлені контролером.

Flow-Mod — OpenFlow-повідомлення, за допомогою якого контролер додає, змінює або видаляє записи в таблицях потоків.

Data Plane (Forwarding Plane) — рівень мережевих пристроїв (комутаторів), що відповідає за пересилання пакетів згідно з встановленими правилами.

Control Plane — рівень, який відповідає за прийняття рішень щодо маршрутизації та управління потоками.

Application Plane — рівень прикладних програм у SDN, які визначають політики управління мережею та взаємодіють з контролером через Northbound API.

Northbound API — інтерфейс, що забезпечує взаємодію прикладних програм із SDN-контролером для формування політик керування.

Southbound API — інтерфейс взаємодії контролера з комутаторами Data Plane, через який передаються правила, телеметрія та повідомлення про події.

QoS (Quality of Service) — набір механізмів, що визначають пріоритети обробки трафіку, пропускну здатність, затримку та втрати пакетів.

Mininet — мережевий емулятор, що дозволяє створювати віртуальні топології з хостами, комутаторами й контролером для тестування SDN.

OVS (Open vSwitch) — віртуальний комутатор, який підтримує OpenFlow та широко використовується у тестових SDN-середовищах.

REST API — архітектурний стиль взаємодії компонентів системи через HTTP-запити з використанням ресурсно-орієнтованої моделі.

WSL (Windows Subsystem for Linux) — середовище виконання Linux-компонентів у Windows, яке використовувалося для запуску Mininet та SDN-контролера.

ВСТУП

Сучасні мережеві технології швидко розвиваються, і зростання обсягів передаваних даних, кількості підключених пристроїв та складність топологій створюють нові виклики для забезпечення ефективної роботи мереж. Традиційні мережеві підходи, де управління трафіком та пересилання даних реалізуються в рамках одного пристрою, стають обмеженням для масштабування, адаптивного управління та автоматизації мережевих процесів.

Software Defined Networking (SDN) пропонує принципово новий підхід до організації мереж, де контрольний рівень відокремлений від рівня пересилання даних і реалізується у вигляді централізованого програмного контролера. Це дозволяє ефективно управляти потоками трафіку, реалізовувати політики якості обслуговування та динамічно змінювати маршрути передачі даних у реальному часі. Однією з ключових задач у SDN є балансування навантаження, що забезпечує рівномірний розподіл трафіку між ресурсами та підвищує продуктивність мережі.

Зважаючи на складність та динамічність сучасних мереж, важливо мати систему тестування алгоритмів балансування, яка дозволяє оцінювати їх ефективність, порівнювати різні підходи та впроваджувати нові методи на основі динамічних даних. Тестування таких алгоритмів у реальних мережах є складним і дорогим, тому для досліджень активно використовуються емулятори мереж, віртуальні комутатори та програмні контролери, що забезпечують гнучке середовище для експериментів.

Метою цієї роботи є розробка системи тестування ефективності алгоритмів балансування навантаження в SDN мережах, яка дозволяє проводити експериментальні дослідження алгоритмів, включаючи як класичні, так і користувацькі реалізації, оцінювати їх продуктивність та вплив на пропускну здатність, затримки та завантаження мережевих вузлів.

Завдання, що вирішуються у роботі, включають:

- Дослідження сучасних технологій SDN та методів балансування трафіку;
- Аналіз існуючих інструментів для тестування SDN-мереж;
- Розробку програмного забезпечення для тестування та моніторингу ефективності алгоритмів балансування;
- Проведення експериментів для оцінки продуктивності різних алгоритмів у контрольованому середовищі.

Результати цієї роботи можуть бути використані для підвищення ефективності роботи мережевих систем, оптимізації ресурсів дата-центрів, впровадження інтелектуальних систем управління та адаптивного балансування трафіку у SDN середовищах.

РОЗДІЛ 1. ОГЛЯД ТЕХНОЛОГІЇ SDN ТА МЕТОДІВ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ

1.1 Поняття та архітектура Software Defined Networking (SDN)

У традиційних комп’ютерних мережах логіка управління трафіком (control plane) та передавання даних (data plane) тісно інтегровані в одному пристрої — маршрутизаторі чи комутаторі. Це ускладнює масштабування, централізоване адміністрування та динамічну адаптацію до змін мережевого навантаження. У відповідь на ці обмеження виникла концепція Software Defined Networking (SDN) — підхід до побудови мереж, у якому управління та контроль відокремлені від функцій пересилання даних і реалізовані у вигляді програмного компонента — SDN-контролера [1].

SDN — це архітектура, що забезпечує централізоване програмне керування мережею, дозволяючи адміністраторам динамічно змінювати політики маршрутизації, пріоритети трафіку, алгоритми балансування та інші параметри без втручання в апаратне забезпечення (див. Рисунок 1.1).

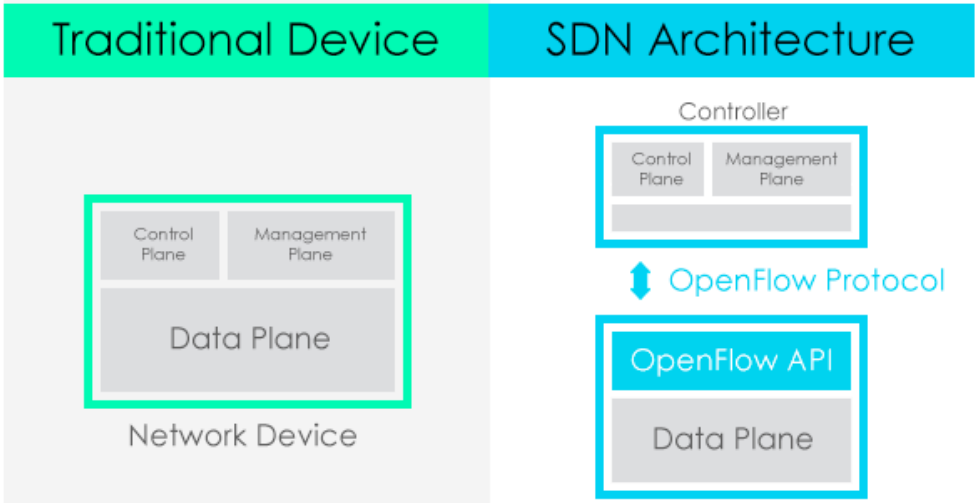

Рисунок 1.1 — Модель традиційної комп'ютерної мережі та SDN

Основна ідея SDN полягає у відокремленні трьох рівнів [2]:

- Рівень прикладних програм (Application Layer) — містить сервіси, які визначають політику управління мережею (наприклад, балансувальники навантаження, системи моніторингу, безпекові модулі тощо). Цей рівень взаємодіє з контролером через Northbound API;
- Контрольний рівень (Control Layer) — ядро SDN-архітектури, яке реалізує централізовану логіку управління мережею. Контролер отримує інформацію про стан мережі від пристроїв нижчого рівня та приймає рішення про маршрутизацію, перенаправлення трафіку чи балансування;
- Рівень даних (Data Plane) — складається з мережевого обладнання (комутаторів, маршрутизаторів, віртуальних вузлів), яке виконує інструкції, отримані від контролера через Southbound API, зазвичай реалізований на базі OpenFlow або аналогічних протоколів (OVSDB, NETCONF, P4Runtime тощо).

Завдяки такому розділенню, SDN надає низку ключових переваг:

- Гнучкість управління — мережеві політики задаються централізовано та можуть змінюватися динамічно через програмні інтерфейси;
- Простота автоматизації — мережа стає «програмованою», що спрощує розгортання сервісів і інтеграцію з DevOps-підходами;
- Масштабованість — контрольний рівень може динамічно адаптувати маршрутизацію та балансування до поточного стану навантаження;
- Зниження вартості — використання відкритих стандартів і програмно керованих пристроїв дозволяє мінімізувати залежність від конкретного виробника обладнання.

Типова реалізація SDN включає SDN-контролер (наприклад, Ryu, ONOS, Floodlight, OpenDaylight), віртуальні або апаратні комутатори, що підтримують OpenFlow (наприклад, Open vSwitch), а також програмні модулі для моніторингу, аналітики та управління політиками [3].

SDN сьогодні активно використовується у дата-центрах, хмарних середовищах, операторських мережах та наукових дослідженнях для створення інтелектуальних систем маршрутизації, динамічного розподілу навантаження та реалізації мережевих політик на основі штучного інтелекту.

1.2. Основні компоненти SDN: контролер, комутатори, прикладний рівень

Архітектура SDN базується на чіткому розділенні функцій мережі між трьома логічними рівнями: рівнем даних (Data Plane), контрольним рівнем (Control Plane) та рівнем прикладних програм (Application Plane) [4]. Така структуризація дозволяє досягти гнучкості, централізованого управління та можливості швидкого впровадження нових сервісів у мережевій інфраструктурі (див. Рисунок 1.2).

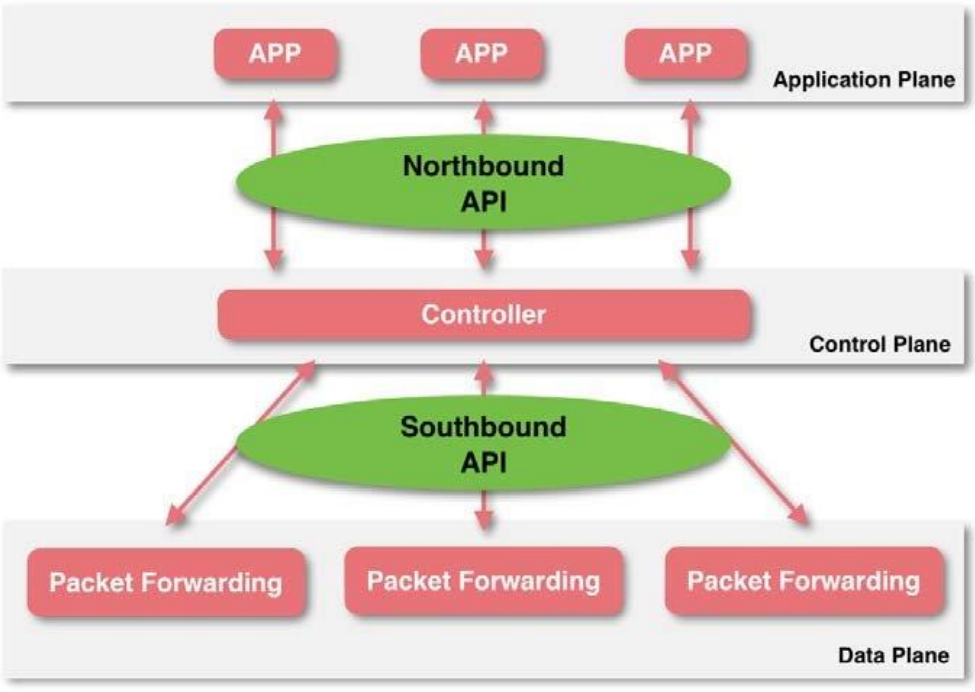

Рисунок 1.2 — Структурна схема компонентів SDN-архітектури з позначенням напрямків взаємодії між рівнями (Northbound та Southbound API)

1.2.1 Контролер (Control Plane)

SDN-контролер — це центральний елемент системи, який виконує роль «операційного мозку» мережі. Він відповідає за:

- Збір інформації про стан мережевих пристроїв;
- Прийняття рішень щодо маршрутизації, перенаправлення або блокування трафіку;
- Динамічну зміну політик балансування навантаження;
- Взаємодію з прикладними сервісами через відкритий Northbound API.

Контролер має повну глобальну картину мережі та може централізовано формувати таблиці потоків для комутаторів.

До найпоширеніших контролерів належать:

- Ryu Controller — легкий Python-контролер із модульною архітектурою, зручний для досліджень і розробки власних алгоритмів балансування [5];
- ONOS (Open Network Operating System) — масштабований контролер для операторських мереж;
- OpenDaylight — промислове рішення з розвиненою системою плагінів;
- Floodlight — Java-контролер із простою інтеграцією через REST API.

Контролер взаємодіє з комутаторами через Southbound API, зазвичай реалізований на базі OpenFlow або інших протоколів управління.

1.2.2 Комутатори (Data Plane)

Комутатори SDN — це апаратні або віртуальні пристрої, які виконують інструкції контролера. Їхнє завдання полягає у фізичній або логічній пересилці пакетів згідно з правилами, що містяться у таблицях потоків (Flow Tables).

Кожен запис у таблиці потоків описує:

- Умову (наприклад, IP-адресу джерела/призначення, MAC-адресу, порт, протокол тощо);
- Дію (переслати, змінити заголовок, скинути, продублювати);
- Пріоритет і таймаут.

Основним прикладом реалізації віртуального комутатора є Open vSwitch (OVS), який підтримує OpenFlow і дозволяє створювати складні топології мереж у середовищах, таких як Mininet.

У рамках SDN ці пристрої не приймають самостійних рішень — вони виконують лише ті дії, які визначив контролер, забезпечуючи повну керованість і передбачуваність мережевої поведінки.

1.2.3 Прикладний рівень (Application Plane)

Прикладний рівень містить сервіси та програми, які визначають політики управління мережею відповідно до бізнес- або дослідницьких цілей.

Це можуть бути:

- Системи балансування навантаження між серверами або каналами;
- Сервіси QoS (Quality of Service) для пріоритизації трафіку;
- Модулі мережевої безпеки (виявлення аномалій, IDS/IPS);
- Аналітичні системи моніторингу стану мережі;
- Інтелектуальні системи, що використовують штучний інтелект або машинне навчання для прогнозування навантаження.

З прикладного рівня до контролера надходять запити через Northbound API (зазвичай REST або gRPC інтерфейси). Контролер, у свою чергу, трансформує ці запити в конкретні інструкції для мережевого обладнання.

Таким чином, SDN-архітектура забезпечує чітке розмежування обов’язків:

- Комутатори — передають пакети;
- Контролер — приймає рішення;
- Прикладний рівень — формує політику управління.

Ця взаємодія створює основу для побудови програмно-керованих, адаптивних і масштабованих мереж, придатних для реалізації інтелектуальних балансувальних систем.

1.3 Протокол OpenFlow та принципи управління трафіком

Одним із ключових елементів архітектури SDN є протокол OpenFlow — стандарт комунікації між контрольним рівнем (SDN-контролером) і рівнем даних (комутаторами). Саме завдяки OpenFlow контролер може надсилати командам комутаторам інструкції про те, як обробляти, маршрутизувати або фільтрувати трафік у мережі (див. Рисунок 1.3).

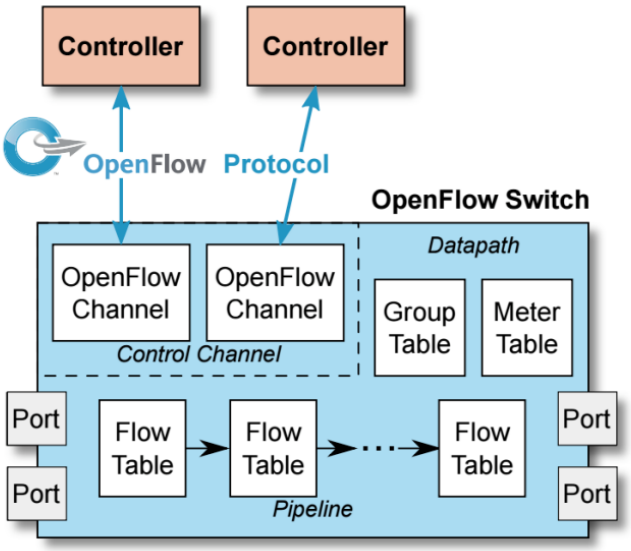

Рисунок 1.3 — Приклад взаємодії контролера з OpenFlow-комутаторами та таблицями потоків

1.3.1 Призначення OpenFlow

Протокол OpenFlow є базовою технологією для реалізації архітектури SDN і відіграє ключову роль у взаємодії між контрольним та площинами даних. Він визначає стандартизований інтерфейс, через який SDN-контролер може програмно керувати поведінкою мережевого обладнання.

OpenFlow дозволяє контролеру:

- Динамічно оновлювати таблиці пересилання (Flow Tables), додаючи, змінюючи або видаляючи правила маршрутизації;
- Отримувати статистику про мережевий трафік, включаючи кількість переданих пакетів, пропускну здатність, час активності потоків і затримки;
- Задавати правила обробки пакетів, що визначають подальшу дію: пересилання, перенаправлення на контролер або модифікацію заголовків.

На відміну від традиційних мереж, де логіка маршрутизації реалізована безпосередньо всередині комутаторів за допомогою вбудованих протоколів, в OpenFlow всі рішення приймає SDN-контролер, а комутатори виступають виконавцями встановлених інструкцій.

1.3.2 Структура OpenFlow-комутатора

Комутатор, що підтримує OpenFlow, складається з таких основних компонентів:

- Flow Table (таблиця потоків) — набір правил, які описують, як обробляти мережеві пакети;
- Secure Channel — захищений канал зв’язку з SDN-контролером (зазвичай через TCP/TLS);
- OpenFlow Protocol Handler — модуль, який приймає команди контролера та повідомляє про події (наприклад, пакети, що не відповідають жодному правилу).

1.3.3 Механізм обробки пакетів

Процес прийняття рішень у комутаторі OpenFlow відбувається у кілька етапів:

- Комутатор отримує вхідний пакет;
- Заголовки пакета порівнюються з умовами у таблиці потоків;
- Якщо знайдено відповідне правило — виконується дія (наприклад, переслати на певний порт, змінити поле заголовка, відкинути);
- Якщо відповідного правила немає — комутатор надсилає Packet-In повідомлення контролеру;
- Контролер приймає рішення і повертає Flow-Mod повідомлення із новим правилом, яке додається до таблиці потоків.

Цей механізм дозволяє контролеру формувати поведінку мережі в реальному часі, базуючись на поточному навантаженні або аналітичних даних.

1.3.4 Типи повідомлень OpenFlow

Комунікація між контролером і комутатором реалізується за допомогою трьох основних категорій повідомлень:

- Controller-to-Switch — команди від контролера (Flow-Mod, Port-Mod, Stats-Request);
- Asynchronous Messages — повідомлення від комутатора (Packet-In, Port-Status, Flow-Removed);
- Symmetric Messages — службові повідомлення для підтримки з’єднання (Hello, Echo, Barrier).

Кожне повідомлення має чітко визначену структуру, що забезпечує сумісність між пристроями різних виробників.

1.3.5 Принципи управління трафіком в SDN через OpenFlow

Використання OpenFlow дозволяє реалізувати централізоване, адаптивне управління трафіком. Контролер може:

- Перенаправляти потоки залежно від поточного завантаження каналів (динамічне балансування);
- Створювати шляхи з урахуванням QoS або безпеки (пріоритети, ізоляція сегментів);
- Аналізувати статистику потоків для прийняття рішень на основі даних (data-driven networking).

При цьому, мережа стає програмованою, тобто адміністратор або система штучного інтелекту може змінювати політику маршрутизації, просто виконавши REST-запит до контролера.

1.4 Класифікація методів балансування навантаження в мережах

Балансування навантаження — це процес рівномірного розподілу вхідного трафіку між кількома ресурсами (сервери, канали, маршрутизатори, комутатори) з метою оптимізації продуктивності, мінімізації затримок і запобігання перевантаженню окремих вузлів. У традиційних мережах цей процес часто обмежується локальною інформацією окремих пристроїв, що знижує ефективність прийняття рішень у складних топологіях.

У контексті SDN мереж, де маршрутизація контролюється централізовано, балансування набуває особливого значення, адже рішення приймаються на рівні контролера, що має глобальне бачення топології, пропускної здатності та стану каналів.

Загальна структура класифікації методів балансування представлена на відповідній схемі (див. Рисунок 1.4), яка систематизує базові підходи та їх особливості [6].

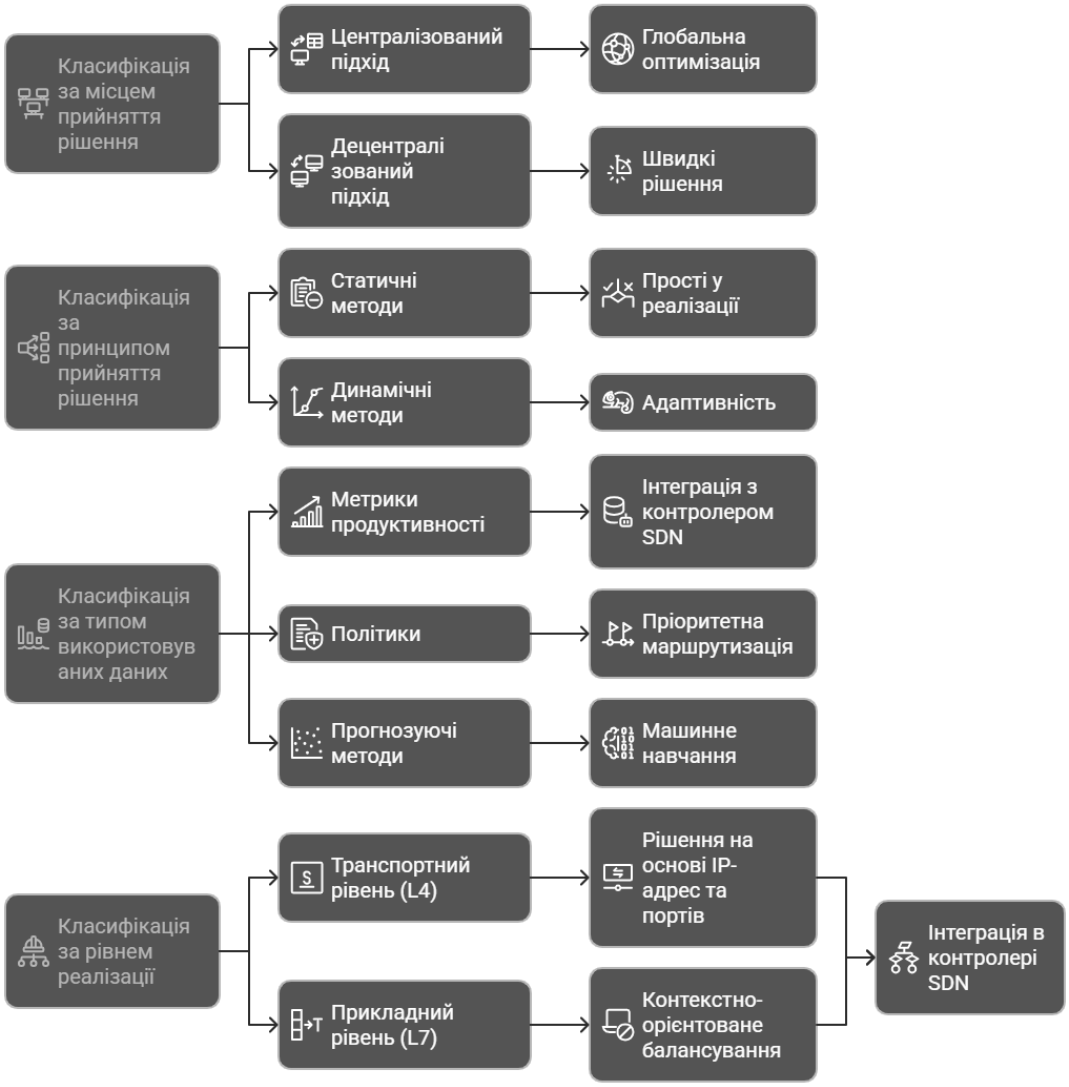

Рисунок 1.4 — Схема класифікації методів балансування навантаження

1.4.1 Класифікація за місцем прийняття рішення

У SDN можна виділити два основні підходи — централізований та децентралізований.

У централізованих системах усі рішення щодо маршрутизації та розподілу навантаження приймає контролер, який аналізує завантаження каналів, активні потоки та пропускну здатність лінків. Така модель забезпечує глобальну

оптимізацію та високу точність керування, однак може спричинити затримки через концентрацію логіки в одному вузлі.

Децентралізовані методи покладаються на локальну інформацію комутаторів. Рішення приймаються швидше, але відсутність повного контексту мережі інколи призводить до неефективного розподілу трафіку або локальних перевантажень.

1.4.2 Класифікація за принципом прийняття рішення

Методи балансування також поділяють на статичні та динамічні. Статичні підходи (наприклад, Round Robin чи маршрутизація на основі хешування) використовують наперед визначені правила, які не змінюються під час роботи системи. Вони прості у реалізації й передбачувані, проте не враховують реальний стан мережі.

На противагу їм, динамічні методи реагують на зміни у навантаженні, аналізуючи такі метрики, як затримка, кількість активних підключень чи середній час відповіді. Приклади: Least Connection, Least Response Time, Adaptive Load Balancing. Ці підходи забезпечують значно кращу адаптивність, однак потребують постійного моніторингу й аналітики.

1.4.3 Класифікація за типом використовуваних даних

Методи можуть спиратися на різні джерела інформації. Найпоширенішими є алгоритми, що використовують метрики продуктивності:

- Завантаження CPU вузлів;
- Використання пропускної здатності;
- Затримка RTT;
- Втрати пакетів.

Такі алгоритми інтегруються з контролером SDN через OpenFlow-статистику.

Інший підхід — методи, що ґрунтуються на політиках. У цьому випадку маршрутизація залежить від типу трафіку або вимог користувача: наприклад, трафік VoIP може оброблятися пріоритетними шляхами.

Сучасна тенденція — використання прогнозуючих (predictive) методів, що застосовують машинне навчання та історичні дані для передбачення майбутнього навантаження.

1.4.4 Класифікація за рівнем реалізації

У традиційних мережах розрізняють балансування на транспортному (L4) та прикладному (L7) рівнях. На рівні L4 рішення приймаються на основі IP-адрес та портів, а L7 враховує контент запитів, що дозволяє реалізовувати контекстно-орієнтоване балансування.

У SDN ці підходи інтегруються у централізованому контролері, який може одночасно аналізувати інформацію від L3 до L7, виконуючи багаторівневу оптимізацію трафіку з урахуванням глобальної топології та станів каналів.

Загалом, класифікація методів балансування навантаження дозволяє вибрати оптимальний підхід залежно від цілей системи: стабільність, швидкодія, адаптивність або інтелектуальне прогнозування. У SDN середовищах це відкриває можливість гнучко змінювати алгоритми через API без зміни фізичної інфраструктури.

1.5 Алгоритми балансування навантаження у SDN

Алгоритми балансування навантаження у SDN є ключовим елементом забезпечення ефективного використання мережевих ресурсів, підвищення пропускної здатності та зниження затримок. Завдяки централізованому

управлінню, SDN надає можливість реалізовувати динамічне та адаптивне балансування трафіку в реальному часі, що значно перевершує традиційні підходи у статичних мережах.

1.5.1 Призначення алгоритмів балансування

Основна мета алгоритмів балансування навантаження (Load Balancing Algorithms) полягає у рівномірному розподілі потоків трафіку між доступними мережевими ресурсами — лінками, комутаторами або серверами. Це дозволяє:

- Уникати перевантаження окремих вузлів;
- Забезпечувати стабільну якість обслуговування (QoS);
- Мінімізувати час відгуку;
- Підвищувати загальну ефективність мережі.

Різноманітні підходи до побудови LBA дозволяють поєднувати статичні, динамічні та інтелектуальні методи оптимізації маршрутів залежно від цілей конкретної системи. Узагальнена структура таких методів подана на відповідній схемі (див. Рисунок 1.5).

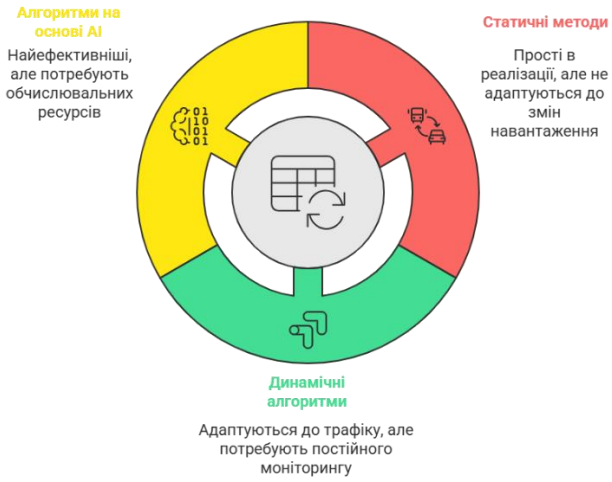

Рисунок 1.5 — Узагальнена схема класифікації методів балансування навантаження

1.5.2 Статичні алгоритми

Статичні методи не враховують поточний стан мережі. Вони базуються на попередньо визначених правилах або політиках маршрутизації.

- Round Robin (RR) — пакети або потоки передаються по черзі між усіма доступними шляхами чи серверами;
- Weighted Round Robin (WRR) — як RR, але з урахуванням ваги каналів (наприклад, різної пропускної здатності);
- Hash-based routing — вибір шляху здійснюється на основі хеш-функції від IP-адрес або портів джерела та призначення.

Перевагою статичних методів є простота реалізації, але вони не адаптуються до змін навантаження, що обмежує ефективність у динамічних SDN-середовищах.

1.5.3 Динамічні алгоритми

Динамічні підходи використовують зворотній зв’язок від мережі, враховуючи параметри трафіку (затримку, пропускну здатність, завантаження портів тощо).

Основні приклади:

- Least Connections (LC) — трафік спрямовується до шляху або сервера з найменшою кількістю активних потоків;
- Least Response Time (LRT) — враховується середній час відгуку кожного сервера;
- Flow-based Adaptive Balancing (FAB) — SDN-контролер динамічно коригує маршрути потоків залежно від завантаженості лінків.

Динамічні методи покращують адаптивність, однак потребують постійного моніторингу стану мережі, що збільшує навантаження на контролер.

1.5.4 Алгоритми на основі машинного навчання (AI-driven)

Сучасний напрям у балансуванні SDN — застосування методів штучного інтелекту (AI) та машинного навчання (ML) для прогнозування стану мережі й прийняття оптимальних рішень у режимі реального часу [7].

Типові підходи:

- Q-Learning / Deep Q-Network (DQN) — агент навчається обирати маршрути з найкращою очікуваною продуктивністю;
- Neural Network-based Load Prediction — моделі прогнозують майбутнє навантаження на вузли, що дозволяє запобігати перевантаженням;
- Genetic Algorithms / Particle Swarm Optimization (PSO) — оптимізація розподілу трафіку через еволюційні або евристичні методи.

Такі алгоритми демонструють найвищу ефективність у складних, масштабних мережах, але потребують додаткових обчислювальних ресурсів та якісних даних для навчання.

1.5.5 Приклади реалізацій алгоритмів у SDN

- Dynamic Load Balancer (DLB) — динамічно змінює маршрути залежно від завантаженості лінків, використовуючи дані з OpenFlow-контролера [8];
- Reinforcement Learning-based Controller — реалізує адаптивне балансування, навчаючись на історичних даних трафіку;
- OpenDaylight Load Balancing Module — модуль для SDN-контролера OpenDaylight, що підтримує політики на основі метрик QoS.

1.5.6 Переваги SDN для балансування навантаження

- Глобальне бачення мережі: контролер володіє повною інформацією про топологію та трафік;
- Гнучкість: можливість швидкої зміни політик маршрутизації;

- Програмованість: реалізація нових алгоритмів без зміни апаратного забезпечення;
- Інтеграція з AI: застосування аналітики та прогнозування.

1.6 Висновки до розділу №1

У першому розділі було розглянуто концепцію Software Defined Networking, її архітектуру та ключові компоненти, що забезпечують централізоване й гнучке управління мережевою інфраструктурою. Було показано, що відокремлення контрольного рівня від рівня даних створює передумови для програмованості мережі та оперативного реагування на зміни трафіку. Протокол OpenFlow, як основний механізм взаємодії між контролером та комутаторами, забезпечує необхідну стандартизацію та дозволяє динамічно змінювати поведінку мережі шляхом оновлення таблиць потоків.

Розглянуті методи та алгоритми балансування навантаження демонструють широкі можливості оптимізації в SDN-мережах — від простих статичних стратегій до динамічних та інтелектуальних підходів, що враховують поточні параметри трафіку й стан обладнання. Застосування таких алгоритмів у середовищі SDN дозволяє покращити розподіл ресурсів, зменшити затримки, підвищити пропускну здатність та забезпечити більш прогнозовану якість обслуговування. Таким чином, у межах розділу було підтверджено, що SDN створює сприятливі умови для побудови ефективних систем балансування навантаження та подальшої інтеграції механізмів штучного інтелекту.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ПРОБЛЕМИ ТЕСТУВАННЯ ТА ОГЛЯД ІНСТРУМЕНТІВ

2.1 Постановка задачі тестування ефективності алгоритмів балансування

Тестування алгоритмів балансування навантаження в SDN-мережах є ключовим етапом оцінки їх придатності до використання в реальних мережевих умовах [9]. Оскільки SDN передбачає централізоване управління трафіком через контролер, а ефективність роботи мережі прямо залежить від коректності прийнятих рішень щодо маршрутизації потоків, виникає потреба у створенні методики, що дозволяє відтворити різні сценарії навантаження й точно виміряти поведінку алгоритмів.

Основною задачею тестування є визначення того, як різні алгоритми балансування розподіляють трафік за умов змінної інтенсивності потоків, неоднорідної топології та можливих перевантажень окремих лінків. Необхідно створити середовище, у якому можливо генерувати контрольоване навантаження, керувати параметрами мережі та отримувати детальні метрики продуктивності. Це забезпечує можливість об’єктивного порівняння поведінки алгоритмів у тотожних умовах.

У рамках поставленої задачі визначається набір ключових параметрів, за якими оцінюється ефективність:

- Пропускна здатність каналів і загальний throughput мережі;
- Середня та максимальна затримка передачі пакетів;
- Рівень втрат пакетів за високих навантажень;
- Рівномірність використання доступних лінків та комутаторів;
- Стабільність роботи алгоритму за умов різких змін інтенсивності трафіку;
- Обчислювальне навантаження на SDN-контролер.

Також важливим аспектом постановки задачі є визначення вимог до програмної інфраструктури. Система тестування має підтримувати можливість автоматизованої генерації топологій, підключення різних контролерів, запуску різноманітних алгоритмів балансування, а також збирання статистики в реальному часі. Окремим елементом є механізм реєстрації результатів експериментів і їх збереження для подальшого аналізу.

Для досягнення коректності тестування важливо забезпечити відтворюваність експериментів. Це передбачає стабільність поведінки емулятора мережі, точний контроль над параметрами трафіку та можливість повторного запуску сценаріїв.

Постановка задачі передбачає також визначення критеріїв, за якими оцінюватиметься успішність алгоритмів: мінімізація затримки, максимізація пропускної здатності, збалансоване використання ресурсів та здатність адаптуватися до змін у режимі реального часу.

2.2 Аналіз існуючих дослідницьких платформ для SDN

Для проведення експериментів із балансуванням навантаження в SDN-мережах важливо обрати відповідне середовище, яке дозволяє точно моделювати поведінку мережевих елементів, контролювати параметри трафіку та забезпечувати інтеграцію з SDN-контролерами [10]. На сьогодні існує декілька платформ, що широко використовуються у дослідницьких цілях. Кожна з них має свої переваги, обмеження та сфери застосування.

2.2.1 Mininet

Mininet є найпоширенішим інструментом для досліджень у галузі SDN. Він дозволяє створювати повноцінні віртуальні топології мереж у межах одного фізичного комп’ютера, використовуючи легкі віртуальні контейнери. У Mininet

хости, комутатори та лінки створюються як мережеві неймспейси й процеси Linux, що забезпечує високу продуктивність та мінімальні затрати ресурсів.

Переваги:

- Повна підтримка OpenFlow та сумісність з популярними SDN-контролерами;
- Можливість запуску складних топологій із сотнями вузлів на одному ПК;
- Швидке розгортання та простота інтеграції зі скриптами тестування;

Недоліки:

- Обмежена точність моделювання апаратних характеристик мережевих пристроїв;
- Затримки та пропускна здатність можуть відрізнятись від реальних мереж через віртуалізацію.

2.2.2 NS-3

NS-3 є потужним мережевим симулятором, який надає можливість детально моделювати поведінку мережевих протоколів на рівні пакетів. На відміну від Mininet, NS-3 не емулює систему у реальному часі, а виконує математичне моделювання.

Переваги:

- Можливість виконання масштабних симуляцій із точним контролем таймінгів;
- Підтримка SDN-модулів (OpenFlow switch model, контролери);
- Можливість інтеграції зі сторонніми модулями для балансування та аналітики.

Недоліки:

- Складність використання у порівнянні з Mininet;
- Відсутність реального виконання мережевих процесів;
- Менша інтерактивність та складніший запуск експериментів.

NS-3 доцільно застосовувати у випадках, коли важлива максимальна точність симуляції, а не реальний час роботи системи.

2.2.3 ONOS Testbed

ONOS Testbed — це середовище, створене розробниками контролера ONOS (Open Network Operating System) для перевірки продуктивності контролера та експериментів у SDN-мережах операторського рівня.

Переваги:

- Підтримка масштабних розподілених SDN-кластерів;
- Підтримка емуляції операторських топологій (ISP-рівень, дата-центри);
- Можливість використання фізичних комутаторів та комбінацій віртуальної і фізичної інфраструктури.

Недоліки:

- Складніші вимоги до обладнання;
- Орієнтованість на ONOS як основний контролер;
- Не така проста інтеграція, як у Mininet.

Це середовище підходить для тестування алгоритмів, орієнтованих на масштабні SDN-мережі та кластеризоване управління.

2.3 Порівняльна характеристика інструментів емуляції та симуляції мереж

Для проведення досліджень алгоритмів балансування навантаження в SDN-мережах дослідники використовують як емулятори, так і симулятори мереж. Вибір конкретного інструменту визначається цілями експерименту, необхідною точністю та масштабом топології. У цій частині проводиться порівняльна характеристика основних платформ (див. Таблиця 2.1), які широко застосовуються у наукових та навчальних проектах: Mininet, NS-3, OMNeT++, GNS3, ONOS Testbed.

Таблиця 2.1 — Порівняння інструментів емуляції та симуляції мереж

Інструмент	**Тип**	**Інтеграція SDN**	**Простота використання**	**Коментарі**
Mininet	Емулятор	Так	Висока	Підходить для прототипування, інтеграції з контролерами, швидких експериментів
NS-3	Симулятор	Обмежена	Середня	Детальне моделювання протоколів, точні симуляції трафіку, повільніші експерименти
OMNeT++	Симулятор	Обмежена	Середня	Гнучка платформа для досліджень, потребує написання власних модулів
GNS3	Емулятор	Частково	Середня	Орієнтований на реальні мережеві пристрої, інтеграція з фізичними комутаторами і маршрутизаторами
ONOS Testbed	Змішаний	Повна	Низька	Підходить для масштабних кластерних експериментів та операторських мереж

Емулятори, такі як Mininet та GNS3, дозволяють запускати реальні SDN-контролери та алгоритми у практично реальному часі, що є ключовою перевагою для тестування ефективності алгоритмів балансування. Вони добре підходять для інтеграційних та функціональних експериментів, але можуть не враховувати низку апаратних обмежень або точність фізичних параметрів.

Симулятори, такі як NS-3 і OMNeT++, забезпечують високу точність моделювання мережевих процесів, дозволяють оцінювати затримки, втрати пакетів та взаємодію протоколів, проте експерименти виконуються не в реальному часі, і їх інтеграція з реальними контролерами обмежена.

ONOS Testbed забезпечує можливість масштабного тестування в кластерних мережах, проте вимагає більш складного апаратного забезпечення та знань для налаштування.

Вибір інструменту залежить від мети дослідження: для тестування алгоритмів балансування у реальному часі з контролерами SDN оптимально використовувати Mininet, тоді як симулятори більше підходять для наукових досліджень протоколів та аналізу точності моделювання мережевих процесів.

2.4. Аналіз існуючих рішень для оцінки ефективності балансувальників

Для тестування та оцінки алгоритмів балансування навантаження в SDN-мережах важливу роль відіграють SDN-контролери, оскільки саме вони відповідають за прийняття рішень щодо маршрутизації потоків трафіку. Існує декілька популярних контролерів, які широко застосовуються в дослідницьких та навчальних проектах: Ryu, POX, Floodlight, OpenDaylight (ODL). Кожен із них має свої особливості, переваги та обмеження, що визначає їх придатність для оцінки ефективності алгоритмів балансування (див. Таблиця 2.2).

Таблиця 2.2 — Порівняння наявних SDN-контролерів

Контролер	**Мова**	**Масштабованість**	**Простота інтеграції**	**Основне призначення**
Ryu	Python	Середня	Висока	Навчальні та лабораторні експерименти, прототипування алгоритмів
POX	Python	Низька	Висока	Навчальні задачі, прості демонстрації
Floodlight	Java	Висока	Середня	Дослідницькі проекти
OpenDaylight	Java	Висока	Середня/Низька	Промислові та операторські сценарії, складні алгоритми

2.4.1. Ryu

Ryu — це легкий, модульний SDN-контролер, написаний на Python, що забезпечує повну підтримку OpenFlow та інших протоколів SDN.

Переваги:

- Простота встановлення та використання;
- Модульна структура, що дозволяє швидко інтегрувати власні алгоритми балансування;
- Можливість роботи з Mininet для швидкого тестування.

Недоліки:

- Менш масштабований у порівнянні з Java-базованими контролерами;
- Обмежена підтримка розподілених кластерів.

Ryu добре підходить для навчальних та лабораторних досліджень, а також для експериментів із прототипування алгоритмів балансування.

2.4.2. POX

POX — це освітній SDN-контролер, написаний на Python, який часто використовується для простих лабораторних завдань та тестування базових алгоритмів OpenFlow.

Переваги:

- Легка вага та простота запуску;
- Добра документація та навчальні приклади;
- Швидке прототипування простих алгоритмів.

Недоліки:

- Обмежена функціональність для складних сценаріїв;
- Відсутність розширених модулів для масштабованого управління трафіком;
- Обмежена підтримка останніх версій OpenFlow.

POX підходить для демонстраційних та навчальних експериментів, але не рекомендується для оцінки алгоритмів у складних або масштабних мережах.

2.4.3. Floodlight

Floodlight — це Java-базований OpenFlow контролер, орієнтований на виробниче використання та дослідницькі проекти, що потребують більшої масштабованості.

Переваги:

- Повна підтримка OpenFlow;
- Наявність багатьох готових модулів для моніторингу, балансування та управління трафіком;
- Висока стабільність при великих навантаженнях.

Недоліки:

- Вища складність налаштування порівняно з Ryu;
- Менш інтегрований із Python-екосистемою, що іноді ускладнює швидке прототипування.

Floodlight підходить для експериментів із масштабними топологіями та тестування алгоритмів, орієнтованих на реальні умови експлуатації.

2.4.4. OpenDaylight (ODL)

OpenDaylight — це модульний, масштабований контролер з відкритим кодом, написаний на Java, що підтримує не лише OpenFlow, а й інші протоколи SDN, включаючи NETCONF, BGP-LS, PCEP.

Переваги:

- Багата екосистема модулів для моніторингу, балансування та інтеграції з іншими мережевими сервісами;
- Застосування у промислових та операторських мережах.

Недоліки:

- Висока складність встановлення та конфігурації;
- Потребує значних обчислювальних ресурсів;
- Менш зручний для швидкого прототипування експериментів у навчальних цілях.

OpenDaylight є оптимальним вибором для досліджень складних алгоритмів балансування та інтеграції AI/ML-рішень, а також для тестування у масштабних та реалістичних сценаріях.

2.5. Недоліки існуючих підходів та необхідність розробки власної системи тестування

Аналіз існуючих платформ та контролерів для тестування SDN-мереж показує, що жодне з готових рішень не забезпечує повний комплекс функціональності, необхідної для ефективної оцінки алгоритмів балансування у всіх дослідницьких сценаріях [11]. Існуючі інструменти мають як сильні, так і слабкі сторони, що створює певні обмеження для наукової роботи та практичного впровадження.

Серед основних недоліків можна виділити:

- Обмежена інтеграція між платформами та контролерами. Наприклад, Mininet добре підходить для швидкого тестування з Ryu або POX, але складніше інтегрувати його з Floodlight чи OpenDaylight, особливо у складних топологіях або кластерних сценаріях.
- Обмежена масштабованість або точність. Емулятори, такі як Mininet, дозволяють створювати великі топології, але точність моделювання затримок, пропускної здатності та апаратних обмежень часто не відповідає реальним мережевим умовам. Симулятори, як NS-3 або OMNeT++, забезпечують високу точність, але не дозволяють тестувати алгоритми у реальному часі та взаємодіяти з контролерами на практичному рівні.
- Недостатня підтримка автоматизації та збору метрик. Більшість готових рішень не надають зручних засобів для автоматичного запуску експериментів, генерації навантаження, інтеграції різних алгоритмів балансування та централізованого збору статистики. Це ускладнює проведення масштабних порівняльних досліджень.

- Складність адаптації під сучасні алгоритми на основі машинного навчання. Існуючі контролери не завжди мають готові інтерфейси для інтеграції з AI/ML-моделями, що обмежує можливість тестування інтелектуальних методів балансування у реальному часі.
- Відсутність гнучкої конфігурації топологій та сценаріїв тестування. Багато платформ орієнтовані на стандартні топології або мають складні процедури створення нестандартних мережевих сценаріїв.

Ці обмеження обґрунтовують необхідність розробки власної системи тестування, яка забезпечить:

- Інтеграцію з різними SDN-контролерами (Ryu, OpenDaylight) у реальному часі;
- Підтримку як статичних, так і динамічних алгоритмів балансування, включно з AI/ML-підходами;
- Автоматизовану генерацію навантаження, запуск експериментів та збір метрик;
- Можливість конфігурації різних топологій і сценаріїв тестування;
- Відтворюваність експериментів та централізоване збереження результатів для подальшого аналізу.

Розробка такої системи дозволяє поєднати переваги емуляторів та симуляторів, забезпечуючи достатню точність, масштабованість і гнучкість, необхідну для комплексного оцінювання ефективності алгоритмів балансування у SDN-мережах.

2.6. Вимоги до системи тестування та критерії оцінки ефективності

Для забезпечення коректного та об’єктивного оцінювання алгоритмів балансування навантаження в SDN-мережах необхідно визначити ключові вимоги до системи тестування, а також метрики ефективності, за якими здійснюватиметься аналіз результатів експериментів.

2.6.1. Вимоги до системи тестування

- Підтримка різних топологій та сценаріїв мережі. Система повинна дозволяти створювати як стандартні, так і довільні топології, включаючи багаторівневі комутаційні структури, різні типи лінків і навантажень.
- Інтеграція з SDN-контролерами. Необхідно забезпечити можливість підключення популярних контролерів (Ryu, POX, Floodlight, OpenDaylight) для тестування алгоритмів у реальному середовищі OpenFlow.
- Можливість використання різних алгоритмів балансування. Система має підтримувати запуск статичних, динамічних та інтелектуальних (AI/ML) алгоритмів балансування з легкою інтеграцією нових методів.
- Автоматизація експериментів. Важливо реалізувати механізми автоматичної генерації навантаження, запуску експериментів і збору даних без ручного втручання.
- Моніторинг та збір статистики. Система повинна забезпечувати збір даних про ключові параметри мережі: затримки, пропускну здатність, втрати пакетів, завантаження лінків та вузлів.
- Відтворюваність експериментів. Кожний тестовий сценарій повинен бути відтворюваним, щоб результати були порівнянними та об’єктивними.
- Масштабованість. Система має дозволяти тестування як невеликих лабораторних топологій, так і більш масштабних сценаріїв із сотнями вузлів і потоків трафіку.
- Зручність аналізу результатів. Потрібна можливість збереження даних у структурованому вигляді та інтеграції з інструментами для побудови графіків, звітів та візуалізації результатів.

2.6.2. Критерії оцінки ефективності алгоритмів балансування

Для порівняння та оцінки алгоритмів необхідно використовувати набір об’єктивних метрик, які відображають роботу мережі під різним навантаженням:

- Затримка (Latency). Час, що проходить пакет від джерела до отримувача. Важливим є як середнє, так і максимальне значення затримки. Низька затримка свідчить про ефективний розподіл трафіку та швидке обслуговування потоків.
- Пропускна здатність (Throughput). Обсяг даних, що передається через мережу за одиницю часу. Показує ефективність використання доступних ресурсів і здатність алгоритму підтримувати високий рівень передачі даних.
- Втрати пакетів (Packet Loss). Частка пакетів, які не досягли призначення через перевантаження лінків або вузлів. Мінімізація втрат є критичною для підтримки якості обслуговування (QoS).
- Використання ресурсів (Resource Utilization). Навантаження на комутатори, сервери та канали. Динамічний балансувальник повинен забезпечувати рівномірне використання ресурсів і уникати «вузьких місць».
- Стабільність роботи алгоритму. Здатність алгоритму адаптуватися до різких змін трафіку без значних коливань затримок чи пропускної здатності.
- Час реакції на зміну навантаження. Визначає, наскільки швидко алгоритм перенаправляє потоки для оптимізації роботи мережі.

2.7 Висновки до розділу №2

У другому розділі були розглянуті сучасні підходи до тестування ефективності алгоритмів балансування навантаження в SDN-мережах, а також існуючі інструменти та контролери, що застосовуються для цих цілей. Було проведено аналіз платформ-емуляторів і симуляторів, таких як Mininet, NS-3, OMNeT++ та ONOS Testbed, з урахуванням їх масштабованості, точності моделювання та інтеграції з SDN-контролерами. Досліджено популярні контролери — Ryu, POX, Floodlight і OpenDaylight, що показало різницю у простоті використання, масштабованості та готовності до інтеграції алгоритмів балансування, включно з інтелектуальними підходами.

Виявлено низку обмежень існуючих рішень, серед яких недостатня інтеграція між платформами та контролерами, обмежена автоматизація експериментів, складність масштабування та адаптації під сучасні AI/ML-алгоритми, а також відсутність гнучкої конфігурації сценаріїв. Це обґрунтовує необхідність розробки власної системи тестування, яка забезпечує автоматизацію, відтворюваність експериментів, централізований збір статистики та підтримку різних типів алгоритмів.

Також визначено ключові вимоги до системи: підтримка різних топологій, інтеграція з контролерами, автоматизація запуску експериментів, моніторинг затримок, пропускної здатності, втрат пакетів і використання ресурсів. Було окреслено критерії оцінки ефективності алгоритмів балансування, що дозволяють об’єктивно порівнювати їх продуктивність, стабільність та адаптивність до змін навантаження.

Таким чином, підготовлено теоретичну та методологічну основу для розробки власної системи тестування алгоритмів балансування в SDN.

РОЗДІЛ 3. ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Загальна архітектура системи

Архітектура розробленої системи тестування побудована як узгоджена багаторівнева платформа, що забезпечує повний цикл моделювання, управління та аналізу експериментів із балансування навантаження в SDN-мережах. Система формує єдине функціональне середовище, у межах якого взаємодіють емулятор мережі, контрольний рівень, серверна логіка оркестрації та користувацький інтерфейс (див. Рисунок 3.1). Кожен із цих рівнів виконує спеціалізовану роль, але всі вони працюють у тісній взаємодії один з одним, забезпечуючи цілісність експериментального процесу.

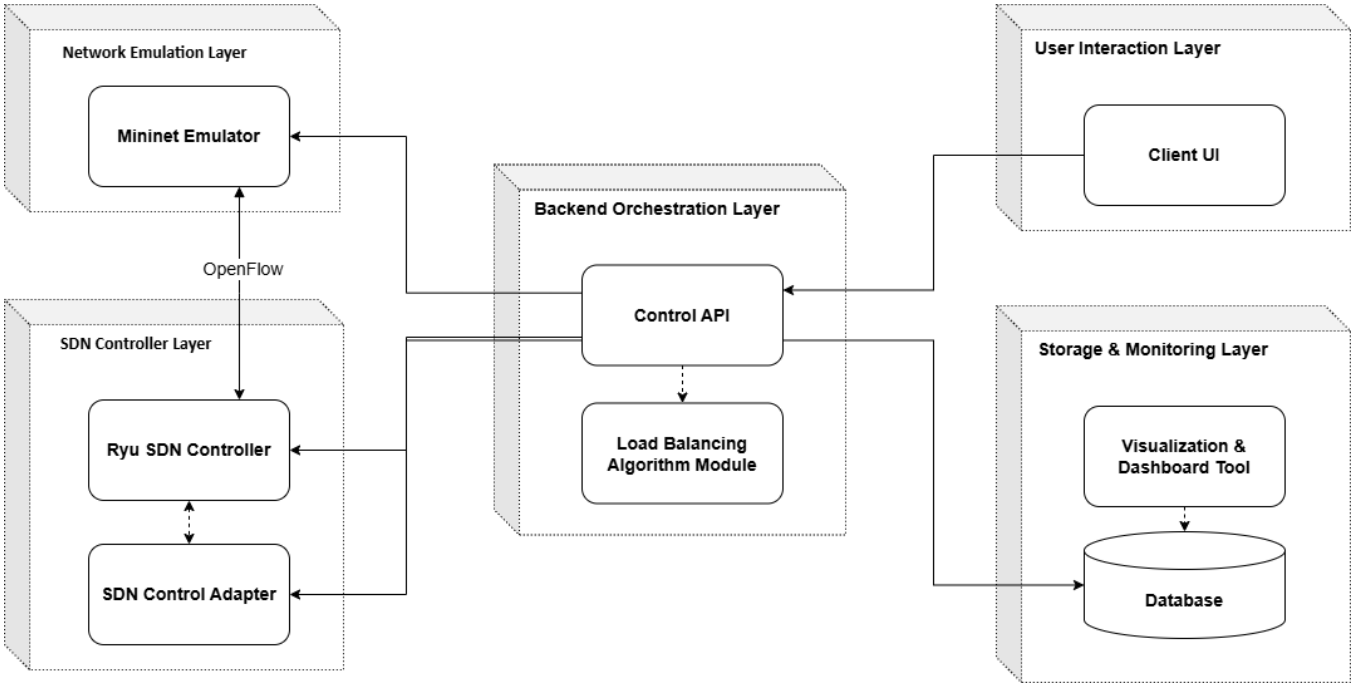

Рисунок 3.1 — Загальна архітектура системи

У фундаменті архітектури знаходиться мережевий рівень моделювання (Network Emulator Layer), який відповідає за створення та виконання віртуальних топологій. У цьому середовищі формуються віртуальні вузли, комутатори та канали зв’язку, що відтворюють поведінку реальної SDN-мережі. Саме на цьому рівні проходить генерація та передавання трафіку, а також виникають події, що надалі впливають на роботу алгоритмів балансування.

Контрольний рівень (SDN Controller Layer) реалізує логіку взаємодії з мережею та забезпечує централізоване управління потоками. Контролер отримує від мережі інформацію про стан каналів, параметри портів та таблиці потоків, надаючи системі узагальнену картину роботи віртуальних комутаторів. У відповідь на отримані дані він приймає та застосовує рішення щодо маршрутизації трафіку. Цей рівень виступає зв'язувальною ланкою між емулятором та серверною частиною, трансформуючи логіку балансування у конкретні правила обробки пакетів.

Центральне місце в архітектурі посідає серверний рівень оркестрації (Backend Orchestration Layer), що координує роботу всієї системи. Він ініціює створення мережевих топологій, керує запуском і зупинкою експериментів, взаємодіє з контрольним рівнем для застосування маршрутних рішень та виконує обробку отриманих телеметричних даних. Саме на цьому рівні реалізовано механізм завантаження, вибору та виконання алгоритмів балансування, які працюють як підключені модулі. Сервер забезпечує необхідну автономність та гнучкість, дозволяючи проводити експерименти з різними конфігураціями мережі та набором алгоритмів без зміни основної логіки системи.

Важливою складовою архітектури є рівень користувацької взаємодії (User Interaction Layer), який надає засоби для налаштування експериментів, керування процесами та аналізу отриманих результатів. Через інтерфейс користувач може визначити структуру мережі, вибрати алгоритм балансування, стежити за перебігом експерименту та переглядати сформовані метрики. Інтерфейс виступає

зовнішнім представленням системи та забезпечує доступ до всіх її можливостей у зручному інтерактивному вигляді.

Окремим функціональним елементом архітектури є підсистема збору та збереження даних (Storage & Monitoring Layer). Вона акумулює статистику, що надходить із контрольного та мережевого рівнів, перетворює її в узагальнений формат та зберігає для подальшого аналізу. Завдяки цьому система підтримує формування історичних даних, порівняння між експериментами та побудову інформаційних панелей, що відображають ключові показники ефективності алгоритмів.

Таким чином, архітектура системи формує цілісну структуру, у межах якої процес тестування реалізується як послідовність взаємодій між рівнями моделювання, контролю, оркестрації, аналізу та користувацького керування. Такий підхід забезпечує гнучкість, масштабованість і відтворюваність експериментів, дозволяючи досліджувати алгоритми балансування навантаження в умовах, наближених до реальних SDN-середовищ.

3.2 Використані технології та середовище розробки

Технологічна платформа системи сформована на основі поєднання інструментів для емуляції мереж, SDN-контролю, серверної оркестрації та клієнтської взаємодії. Усі компоненти інтегруються у спільне середовище, що забезпечує керування життєвим циклом експериментів та оцінку ефективності алгоритмів балансування навантаження.

Базовою частиною середовища моделювання виступає Mininet 2.3.0, що працює у середовищі WSL2 Ubuntu 20.04. Mininet використовується як основний інструмент для побудови віртуальних топологій, створення комутаторів Open vSwitch (версія OVS 2.17+) та запуску трафіку за допомогою ping, iperf і внутрішніх механізмів емуляції каналів. Середовище підтримує специфічні параметри мережевих лінків, такі як пропускна здатність, затримка та ймовірність втрат, що

дозволяє відтворювати різні режими навантаження. Використання WSL2 дає можливість поєднати Linux-інструменти для SDN з основним робочим оточенням розробника на Windows.

Компонент контролю мережі реалізовано на основі Ryu SDN Framework (версія 4.34), що підтримує стандарт OpenFlow 1.3. Ryu працює у Linux-середовищі паралельно з Mininet і відповідає за прийом OpenFlow-подій, обробку FlowStats та PortStats, а також виконання зовнішніх FlowMod інструкцій, які надходять від серверної частини. Для розширення функціональності контролера використовується власний Ryu-модуль, що забезпечує REST-інтерфейси для отримання статистики та керування потоками. Це дозволяє центральному серверу повністю контролювати маршрутизацію всередині емульованої топології.

Серверна частина реалізована на платформі .NET 8, що використовується для організації центрального рівня оркестрації всієї системи. У серверному застосунку функціонують модулі управління топологіями, алгоритмами, трафіком, експериментами та телеметрією. Система побудована за принципами модульності та розширюваності: алгоритми балансування інтегруються у вигляді окремих C#-бібліотек (DLL), що динамічно завантажуються через механізми рефлексії. Це дає можливість розширювати систему новими алгоритмами без внесення змін до основного коду. Сервер виконується у Windows середовищі й забезпечує взаємодію між усіма рівнями через HTTP/REST.

Веб-клієнт створений на Angular 17 як незалежний інтерфейс керування, що працює у браузері та взаємодіє з сервером через REST API. Він забезпечує інструменти для конфігурації мережевих топологій, запуску експериментів, моніторингу у реальному часі та аналізу результатів. Візуалізаційний модуль клієнта дозволяє відображати основні параметри мережі, структуру потоків та хід експерименту, спираючись на отримані від сервера дані.

Система використовує реляційне середовище зберігання даних для фіксації історії експериментів, метрик і структурних описів топологій. Застосування реляційної моделі забезпечує підтримку транзакційності, узгодженості та можливість виконання складних аналітичних запитів. У процесі розробки застосовувалися інструменти логування, тестування API, емуляції запитів та автоматизації сценаріїв взаємодії з Mininet і Ryu.

Уся система розгортається у змішаному середовищі Windows + Linux (WSL2), що дозволяє одночасно використовувати високопродуктивні інструменти серверної розробки та повноцінне лінуксове SDN-середовище. Такий підхід забезпечує сумісність, гнучкість та високу швидкість налаштування експериментальних сценаріїв.

Таким чином, технологічна платформа системи включає повноцінний стек інструментів SDN-емуляції, серверної логіки, модульного розширення та аналітики, що дозволяє проводити різнопланові дослідження алгоритмів балансування навантаження у контрольованих, відтворюваних умовах (див. Рисунок 3.2).

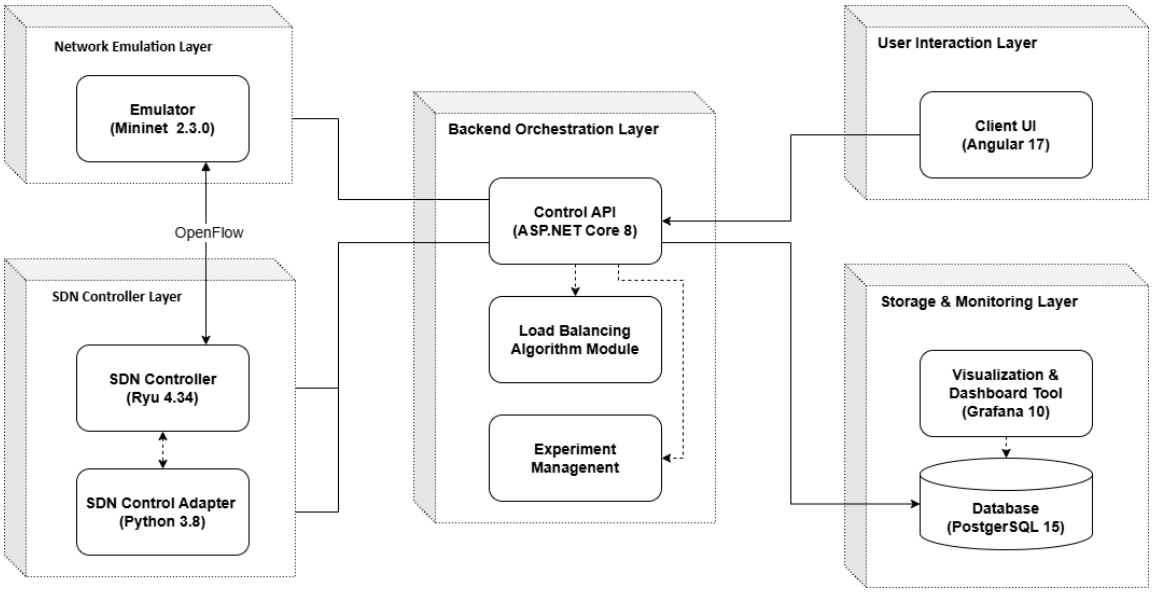

Рисунок 3.2 — Використані у системі технології

3.3 Механізм моделювання мережі

Механізм моделювання мережі у системі реалізований на основі інтеграції емулятора Mininet та SDN-контролера Ryu, між якими забезпечено керований обмін інформацією через серверну частину. Така структура дозволяє створювати віртуальні мережеві середовища з параметрами, що відповідають конкретним експериментальним сценаріям, а також здійснювати централізований контроль за поведінкою потоків у мережі.

Моделювання починається з формування топології, конфігурація якої передається від серверної частини у вигляді структури, що містить тип топології, кількість комутаторів, кількість хостів та параметри каналів зв’язку. На основі цих даних сервер генерує відповідну команду запуску Mininet або Python-сценарій, який автоматично розгортається у Linux-середовищі. Після запуску Mininet створює логічні мережеві компоненти — хости, OpenFlow-комутатори та зв’язки між ними — та під’єднує їх до SDN-контролера через інтерфейс протоколу OpenFlow.

Після встановлення з’єднання всі комутатори повідомляють про свою присутність контролеру Ryu, а той в свою чергу ініціалізує базові правила обробки пакетів, необхідні для подальшого отримання інформації про стан мережі. Контролер працює у режимі централізованого керування: він отримує статистику портів, потоки даних та події PacketIn від комутаторів, а далі передає ці дані на серверний рівень, де вони використовуються для виконання алгоритмів балансування.

Серверна частина зберігає інформацію про активну мережу у власній моделі, яка містить опис топології, структуру вузлів, параметри лінків та актуальний стан мережі. Цей стан формується на основі даних, що регулярно надходять від Ryu, включаючи статистику портів, дані про завантаженість лінків, кількість переданих пакетів і байтів, а також активні потоки, що проходять через комутатори. Таким

чином, мережеве середовище постійно відображається у внутрішній моделі системи, що дозволяє алгоритмам балансування працювати з актуальними даними.

Механізм моделювання також включає підтримку генерації трафіку всередині Mininet. Для цього у віртуальних хостах запускаються стандартні інструменти, такі як iperf3, який використовується для створення довготривалих потоків TCP/UDP, або ping для вимірювання затримок між вузлами. Серверна частина ініціює запуск генераторів, керує їхнім завершенням та фіксує результати вимірювань.

Завершальним елементом механізму моделювання є можливість повного очищення або перезапуску топології. Сервер надсилає команду на зупинку процесу Mininet, видаляє всі тимчасові мережеві конфігурації, після чого емульоване середовище повертається до початкового стану. Це дозволяє виконувати багаторазові експериментальні серії в однакових умовах та забезпечує відтворюваність результатів.

Таким чином, механізм моделювання мережі реалізовано як послідовність взаємодій між емулятором, контролером та серверною логікою, що формує кероване та динамічне середовище для тестування алгоритмів балансування навантаження у програмно-конфігурованих мережах.

3.4 Модуль серверного REST API

Модуль серверного REST API забезпечує централізований доступ до функціональних можливостей системи та реалізує всі операції з управління мережевою топологією, експериментами, трафіком, алгоритмами та телеметрією. Він побудований як набір REST-контролерів, кожен з яких відповідає за окремий домен системи, а взаємодія здійснюється через стандартизовані HTTP-запити у форматі JSON. Структура API документується за допомогою Swagger/OpenAPI, що дозволяє наочно представити доступні методи, параметри та формати відповідей.

3.4.1 TopologyController

Цей контролер відповідає за повний життєвий цикл віртуальної топології, створеної у Mininet, включаючи її ініціалізацію, валідацію, перегляд структури та скидання. Через нього сервер формує топологічну конфігурацію, на основі якої запускається Mininet, та отримує інформацію про створені хости, комутатори й лінки (див. Рисунок 3.3). Контролер виступає основним інтерфейсом для формування мережевих моделей.

Основні запити:

- POST /api/topology/create — створення нової топології за параметрами моделі CreateTopologyModel;
- GET /api/topology/current — отримання опису активної топології.
- DELETE /api/topology/reset — повне очищення та зупинка середовища;
- POST /api/topology/validate — перевірка коректності топологічної конфігурації;
- GET /api/topology/nodes — отримання списку вузлів (host, switch, link);
- GET /api/topology/types — запит підтримуваних типів топологій.

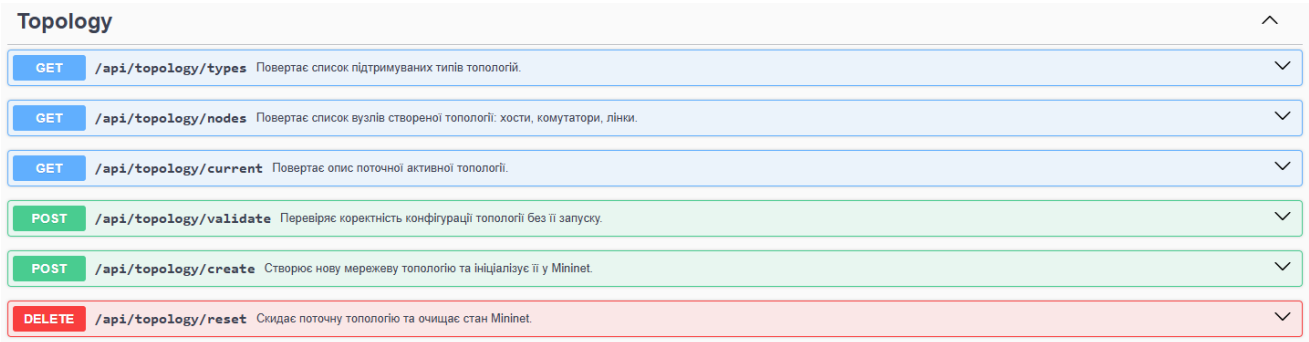

Рисунок 3.3 — Структура контролера Topology

3.4.2 ExperimentController

Контролер координує виконання експериментів, включаючи запуск, зупинку, паузу, відновлення та отримання статусу проведених тестів (див. Рисунок 3.4). Саме через нього API узгоджує роботу між топологією, трафіком, телеметрією та модулями алгоритмів балансування, підтримуючи структурований життєвий цикл експерименту.

Основні запити:

- POST /api/experiment/start — створення нового експерименту із заданими параметрами;
- POST /api/experiment/stop — завершення експерименту та зупинка генераторів трафіку;
- GET /api/experiment/status — поточний стан експерименту (Running/Paused/Stopped);
- GET /api/experiment/history — отримання переліку усіх виконаних експериментів;
- GET /api/experiment/get — отримання детальної інформації конкретного експерименту.

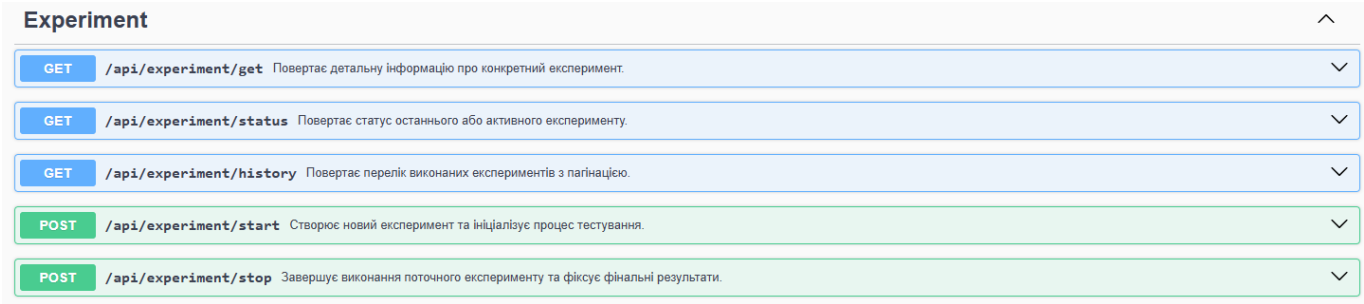

Рисунок 3.4 — Структура контролера Experiment

3.4.3 TrafficController

Контролер забезпечує запуск та завершення генерації навантаження у межах віртуальної мережі. Він може виконувати iperf3-тести, ping-вимірювання, генерувати TCP/UDP-потоки та управляти активними процесами трафіку (див. Рисунок 3.5). Контролер використовується для створення необхідних умов навантаження під час експериментів.

Основні запити:

- POST /api/traffic/start — запуск генератора трафіку між вказаними вузлами;
- POST /api/traffic/stop — завершення виконання активного генератора трафіку;
- GET /api/traffic/active — отримання переліку усіх активних генераторів трафіку;
- POST /api/traffic/ping — виконання ICMP-тесту між вузлами та отримання значення затримки;
- POST /api/traffic/iperf — запуск iperf-тесту пропускної здатності та отримання значень пропускної здатності.

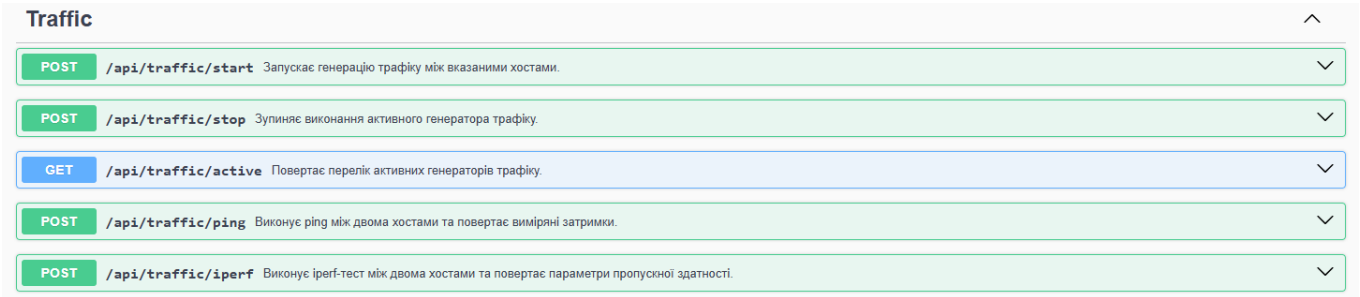

Рисунок 3.5 — Структура контролера Traffic

3.4.4 MetricsController

Цей контролер забезпечує доступ до всіх типів статистичних даних, отриманих під час експериментів — від зібраних Ryu Controller FlowStats/PortStats до виміряних затримок, пропускної здатності та агрегованих метрик (див. Рисунок 3.6). Дані отримуються в режимі запитів або у форматі near-real-time для візуалізаційних панелей у UI.

Основні запити:

- GET /api/metrics/ports — отримання статистики портів комутаторів (PortStats);
- GET /api/metrics/flows — отримання статистики потоків (FlowStats);
- GET /api/metrics/host-latency — отримання значення затримки між хостами;
- GET /api/metrics/throughput — отримання значення пропускної здатності у ході експерименту;
- GET /api/metrics/experiment/{id} — отримання зведеної статистики для заданого експерименту;
- GET /api/metrics/realtime — отримання значень метрик у реальному часі.

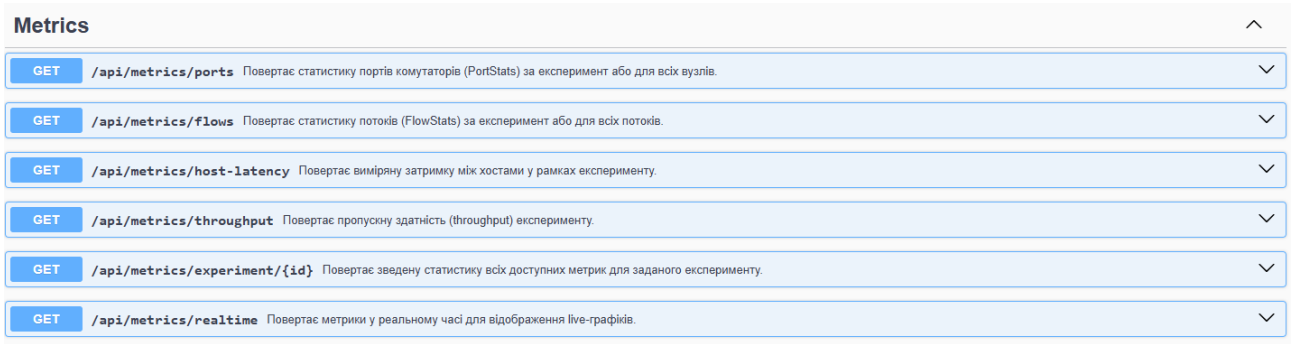

Рисунок 3.6 — Структура контролера Metrics

3.4.5 AlgorithmsController

Контролер відповідає за інтеграцію та виконання алгоритмів балансування, які завантажуються у систему як окремі модулі (DLL або сценарії). Через нього API надає механізми вибору активного алгоритму, завантаження нових реалізацій, перевірки їх структури та перегляду наявної інформації про них (див. Рисунок 3.7).

Основні запити:

- GET /api/algorithms/list — отримання переліку доступних алгоритмів;
- POST /api/algorithms/select — вибір активного алгоритму;
- POST /api/algorithms/upload — завантаження нового алгоритму (DLL) у систему;
- POST /api/algorithms/validate — перевірка алгоритму на сумісність та коректність.

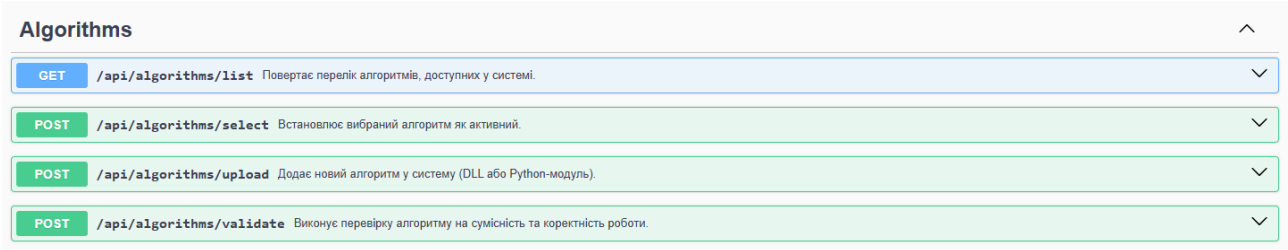

Рисунок 3.7 — Структура контролера Algorithms

3.4.6 ControllerIntegrationController

Цей контролер забезпечує логічний міст між сервером і Ryu Controller, дозволяючи отримувати статистику комутаторів, переглядати Flow Tables та керувати OpenFlow-правилами (див. Рисунок 3.8). Через нього сервер застосовує маршрутні рішення алгоритмів до емульованої мережі.

Основні запити:

- GET /api/controller/flows — отримання активних потоків із контролера;
- GET /api/controller/ports — PortStats для всіх комутаторів;
- POST /api/controller/add-flow — додавання FlowMod правила;
- POST /api/controller/delete-flow — видалення FlowMod правила;
- GET /api/controller/switches — перелік підключених OpenFlow-комутаторів;
- GET /api/controller/events — отримання подій PacketIn/FlowRemoved.

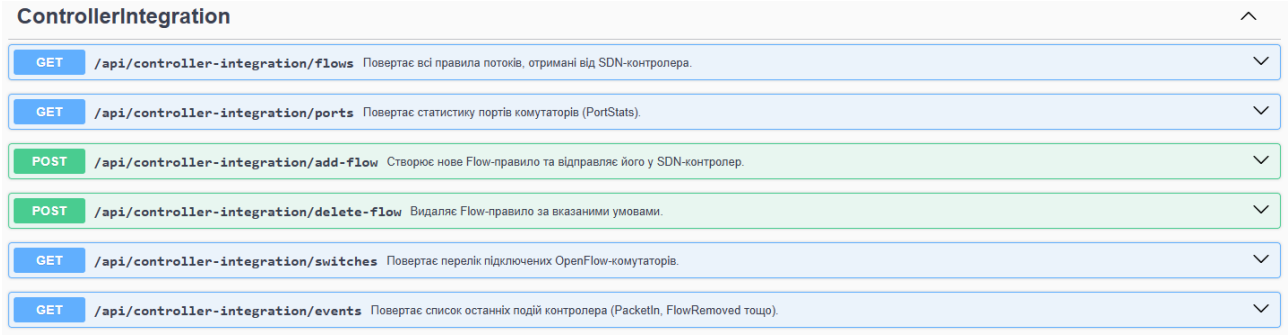

Рисунок 3.8 — Структура контролера ControllerIntegration

3.4.7 SystemController

SystemController надає сервісні можливості платформі та забезпечує діагностичні механізми для адміністратора. Він дозволяє перевіряти стан сервісу, переглядати середовище виконання та отримувати журнал подій (див. Рисунок 3.9).

Основні запити:

- GET /api/system/health — стан системи та підсистем;
- GET /api/system/environment — інформація про середовище виконання;
- POST /api/system/restart — запит перезапуску сервісу;
- GET /api/system/logs — отримання системних логів.

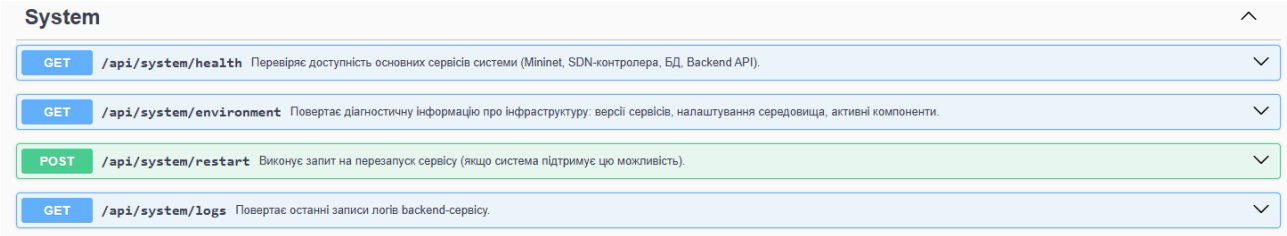

Рисунок 3.9 — Структура контролера System

3.5 Механізм інтеграції алгоритмів балансування

Механізм інтеграції алгоритмів балансування реалізовано як окремий підсистемний компонент серверної частини, що підтримує динамічне завантаження, ініціалізацію та виконання зовнішніх алгоритмічних модулів. Такий підхід забезпечує незалежність алгоритмів від основної логіки сервісу, а також дозволяє розширювати платформу новими методами балансування без модифікації робочого коду ядра.

Алгоритми реалізуються у вигляді окремих динамічних бібліотек, що містять єдиний публічний клас, який відповідає стандартизованому інтерфейсу. Під час завантаження DLL сервер виконує статичний аналіз метаданих: перевіряє наявність потрібного класу, методів, відповідність сигнатур та можливість створення екземпляра без параметрів. Це запобігає включенню у систему модулів, які не відповідають загальній специфікації.

Після завантаження DLL-файлу сервер реєструє модуль у власному реєстрі алгоритмів. Реєстраційний запис містить назву алгоритму, опис, версію, ім’я класу та шлях до файлу. Усі ці дані використовуються для формування каталогу доступних алгоритмів. Під час перемикання активного алгоритму сервер створює окремий екземпляр класу, який залишається основним: він може зберігати внутрішній контекст між обчислювальними циклами, якщо це необхідно для реалізації певної стратегії балансування.

У процесі виконання експерименту алгоритм працює у циклі, синхронізованому з механізмом збору телеметрії. На кожній ітерації сервер формує структуру мережевого стану, що включає статистику портів, активні записи таблиць потоків, завантаженість каналів, інформацію про хости та їхні IP/MAC-відповідності. Стан мережі формується у вигляді складного об’єкта, що містить ієрархію вузлів, зв’язків та метрик, після чого передається алгоритму для опрацювання (див. Таблиця 3.1).

Таблиця 3.1 — Структура вхідних даних алгоритму (NetworkState)

Параметр	**Опис**	**Тип даних**
Switches	Перелік доступних комутаторів OpenFlow	List<SwitchInfo>
Links	Топологічні зв’язки між вузлами	List<LinkInfo>
Hosts	Інформація про хости та їхні IP/MAC	List<HostInfo>
FlowStats	Статистика активних потоків	List<FlowInfo>
PortStats	Статистика портів	List<PortLoadInfo>
LinkLoad	Завантаження каналів	List<LoadInfo>
ActiveTraffic	Активні джерела трафіку	List<TrafficSession>
Timestamp	Час отримання телеметрії	DateTime
CycleIndex	Індекс ітерації алгоритму	int

Після отримання NetworkState алгоритм обробляє дані у межах одного обчислювального циклу й повертає список дій (FlowAction), які визначають, які зміни необхідно зберегти до таблиць потоків OpenFlow-комутаторів. Формат вихідних дій стандартизовано, що забезпечує сумісність із модулем взаємодії з Ryu Controller (див. Таблиця 3.2).

Таблиця 3.2 — Структура вихідних параметрів алгоритму

Параметр	**Опис**	**Тип даних**	**Приклад**
SwitchId	Ідентифікатор комутатора	long	1
Match	Поля OpenFlow для зіставлення	словник	{"in_port":1,"eth_type":2048}
Action	Інструкція обробки	рядок	output:3
Priority	Пріоритет правила	int	200
IdleTimeout	Таймаут неактивності	int	30
HardTimeout	Граничний час життя	int	60
Description	Коментар для логів	string	RR route to h3

Застосування інструкцій відбувається через інтеграційний модуль взаємодії з Ryu Controller, який трансформує дії алгоритму у REST-запити до контролера, що відповідають формату OpenFlow FlowMod. Зміни вносяться безпосередньо у таблиці потоків комутаторів, що дозволяє алгоритму впливати на маршрутизацію у реальному часі. Усі операції виконуються асинхронно, що забезпечує реагування алгоритму на зміну станів лінків без затримок.

Однією з важливих особливостей механізму є ізоляція модулів алгоритмів. Сервер забезпечує незалежність їхнього виконання від внутрішнього середовища застосунку, що дозволяє містити як прості евристики (Round-Robin, Least Connection), так і складніші моделі, включно з адаптивними алгоритмами, використанням статистичного прогнозування чи імпортованими ML-модулями. Забезпечується також можливість використання внутрішнього стану алгоритму між циклами, наприклад, для реалізації історично залежних або контекстних моделей балансування.

У підсумку механізм інтеграції алгоритмів балансування забезпечує відокремлений, керований та розширюваний контур виконання аналітичних модулів, що дозволяє використовувати у системі будь-які типи алгоритмів, від класичних методів до моделей із машинним навчанням, без зміни архітектури та логіки базових компонентів системи.

3.6 Збір та обробка результатів експериментів

Збір даних відбувається у декілька етапів і охоплює як низькорівневі параметри комутаторів, так і агреговані показники продуктивності мережі та окремих потоків трафіку. Усі дані інтегруються у єдину структуру, що дозволяє виконувати подальший аналіз, візуалізацію та формування звітів.

Перший рівень збору інформації здійснюється на стороні SDN-контролера Ryu, який забезпечує доступ до статистики OpenFlow-комутаторів. Контролер періодично генерує FlowStats і PortStats, що містять кількість отриманих і переданих пакетів, обсяг переданих даних, інформацію про активні таблиці потоків та стан портів комутаторів. Ці дані надходять до серверної частини у стандартизованому вигляді через механізм REST-запитів, де вони фіксуються у контексті конкретного експерименту. Отримання статистики відбувається у фонових режимах з певним інтервалом, що забезпечує високу точність відображення динаміки навантаження.

Другий рівень збору результатів пов’язаний із генераторами трафіку, які працюють у середовищі Mininet. Система ініціює запуск iperf-процесів або ICMP-вимірювань, на основі яких отримуються показники пропускної здатності, затримок, стабільності каналів та втрат пакетів. Результати виконання цих процесів збираються у вигляді логів або структурованих відповідей, що містять значення пікової та середньої пропускної здатності, мінімальних, середніх і максимальних значень затримки та інших параметрів якості передавання даних. Кожен такий результат прив’язується до певного етапу експерименту або певної пари вузлів.

Наступний рівень обробки стосується агрегування зібраних даних та їх перетворення у формат аналітичних метрик. Сервер формує узагальнені показники, такі як середнє навантаження на канали, асиметрія потоків, ефективність розподілу, стабільність маршрутів, відхилення значень затримок та інші характеристики, що дозволяють оцінити якість роботи алгоритму.

Обробка результатів включає й перевірку коректності зібраних даних. Серверна частина виконує фільтрацію неконсистентних та синхронізацію тимчасових значень між різними джерелами даних, а також усунення дублювання записів. Застосування цих процедур гарантує точність аналітичних висновків та коректне формування графіків і таблиць.

Усі результати експериментів зберігаються у базі даних разом із метаданими, що описують структуру топології, активний алгоритм, параметри трафіку та часові характеристики експериментального сценарію. Завдяки цьому кожен експеримент може бути відтворений або повторно проаналізований у будь-який момент. Система також підтримує формування історичних даних, що дозволяє виконувати порівняння ефективності різних алгоритмів за однакових умов.

Користувацький інтерфейс взаємодіє із сервером через спеціалізований API для отримання як поточних, так і історичних метрик. Дані можуть відображатися у вигляді графіків реального часу, таблиць, діаграм або текстових звітів залежно від сценарію використання. Це дає можливість користувачу оцінювати стан мережі, фіксувати аномалії та проводити структурний аналіз результатів.

Таким чином, підсистема збору та обробки результатів формує центральну аналітичну складову системи, яка забезпечує повний цикл роботи з даними — від отримання низькорівневої інформації з комутаторів і генераторів трафіку до обробки, структурування, збереження та візуалізації метрик, необхідних для всебічного аналізу ефективності алгоритмів балансування навантаження.

3.7 Модель взаємодії компонентів

Модель взаємодії компонентів відображає логіку обміну даними між основними функціональними модулями системи та визначає порядок проходження інформаційних потоків під час виконання експериментів. Усі компоненти працюють у межах чітко визначеної послідовності, що забезпечує узгодженість процесів моделювання, управління мережею, виконання алгоритмів балансування та збору результатів.

На верхньому рівні взаємодії знаходиться користувацький інтерфейс, який забезпечує ініціацію всіх операцій: створення топологій, запуск експериментів, керування трафіком та перегляд аналітичних даних. Інтерфейс не взаємодіє безпосередньо з емульованим середовищем чи контролером, а працює виключно через REST API, що гарантує стабільність та безпеку доступу до внутрішніх ресурсів системи.

Центральне місце у моделі взаємодії посідає серверна частина, яка виконує роль координатора. Вона отримує команди від клієнта, обробляє їх відповідно до внутрішньої логіки та ініціює взаємодію з іншими компонентами. Сервер містить модулі для керування топологіями, трафіком, алгоритмами балансування та збором метрик, кожен з яких відповідає за певний аспект роботи системи. Завдяки цьому вся комунікація між різними частинами платформи відбувається через уніфікований серверний рівень.

Після отримання команди щодо створення або зміни топології сервер формує відповідні сценарії або параметри та надсилає їх до Mininet. Емулятор створює віртуальних хостів і комутатори, встановлює зв’язки між ними та під’єднує всю мережу до SDN-контролера через протокол OpenFlow. Під час виконання експерименту Mininet генерує трафік між вузлами та забезпечує умови, необхідні для оцінювання роботи алгоритмів балансування.

SDN-контролер Ryu виступає посередником між сервером і емульованою мережею. З одного боку, він збирає статистику портів, потоків та подій PacketIn від комутаторів, а з іншого — застосовує FlowMod-інструкції, що надходять із серверної частини. Саме через контролер відбувається переналаштування поведінки мережі: усі зміни маршрутизації, створені алгоритмами балансування, проходять через Ryu та відображаються у таблицях потоків комутаторів.

Модуль алгоритмів балансування отримує від серверної частини структурований опис поточного стану мережі, що включає статистику потоків, параметри портів, робоче навантаження та опис активної топології. Алгоритм обробляє дані і повертає набір інструкцій щодо перенаправлення трафіку. Після цього сервер трансформує отримані рішення у формати FlowMod і надсилає їх до Ryu для застосування у мережі.

Паралельно з роботою алгоритму модуль збору телеметрії продовжує отримувати статистику з контролера та результати вимірювань від генераторів трафіку. Ці дані повертаються на сервер, де агрегуються, синхронізуються за часовими мітками та зберігаються у базі даних. Інтерфейс користувача може запитувати як поточні дані у режимі майже реального часу, так і історичні метрики будь-якого попереднього експерименту.

Загалом модель взаємодії формує замкнений цикл роботи, у якому кожен компонент відіграє свою визначену роль: інтерфейс формує користувацькі команди, сервер їх обробляє, Mininet виконує моделювання, контролер керує мережею, алгоритм балансування приймає рішення, а модуль телеметрії забезпечує доступом до даних для аналізу. Такий підхід створює чітку ієрархію обміну інформацією та гарантує відтворюваність результатів експериментів, незалежно від складності топології чи типу використовуваного алгоритму балансування навантаження.

3.8 Масштабованість та гнучкість системи

Архітектура системи спроектована таким чином, щоб забезпечити можливість масштабування та адаптації під різні типи досліджень, різні обсяги трафіку та різноманітні алгоритмічні підходи до балансування навантаження. Завдяки модульній структурі та чіткому розмежуванню функціоналу між компонентами система здатна працювати як у локальному середовищі розробника, так і в масштабованих лабораторних інфраструктурах.

Основним фактором масштабованості є відокремлення логіки моделювання мережі від логіки управління та аналізу. Емулятор Mininet може бути розміщений у будь-якому Linux-середовищі, включно з віртуальними машинами та контейнерами. Система дозволяє змінювати кількість хостів, комутаторів, рівнів вкладеності топології й параметри лінків без будь-яких змін у серверному коді. Це дає змогу проводити експерименти як з простими топологіями типу “лінія” чи “зірка”, так і з більш складними структурами Fat-Tree або Clos, що характерні для дата-центрових мереж.

Масштабованість також забезпечується використанням Ryu Controller як централізованого керуючого елементу. Контролер підтримує одночасне обслуговування кількох OpenFlow-комутаторів, що дозволяє моделювати велику кількість потоків і складні мережеві сценарії. Завдяки тому, що серверна частина взаємодіє з Ryu через REST API, масштабування контролера може бути реалізоване як шляхом винесення його в окремий вузол, так і шляхом запуску кількох інстансів для ізольованих експериментів.

Гнучкість системи забезпечена роботою з алгоритмами балансування через механізм динамічного завантаження модулів. Алгоритми можуть бути реалізовані різними методами, включаючи евристичні стратегії, статистичні моделі чи компоненти машинного навчання. Завдяки тому, що сервер очікує від алгоритмів лише набір стандартизованих рішень у вигляді FlowMod-операцій, зміна або

розширення алгоритмів не впливає на інші частини системи. Це дозволяє досліджувати широкий спектр підходів, включно з комбінованими або адаптивними моделями.

Гнучкість архітектури проявляється і в серверній частині, яка побудована на принципах модульності. Кожен контролер API виконує одну чітко визначену групу функцій і може бути розширений або модифікований без впливу на інші компоненти. У такому підході легко додаються нові функції, такі як підтримка нових типів метрик, додаткові інструменти генерації трафіку або розширення можливостей телеметрії. Внутрішня архітектура сервера підтримує можливість інтеграції зовнішніх сервісів, наприклад систем логування або платформ машинного навчання.

Таким чином, система має багаторівневу масштабованість — від гнучкої зміни топологій до можливості горизонтального масштабування компонентів у розподілених середовищах. Модульна архітектура, стандартизовані інтерфейси та використання відкритих SDN-протоколів забезпечують адаптивність і дають змогу розширювати функціональність платформи відповідно до майбутніх дослідницьких або прикладних потреб.

3.9 Висновки до розділу №3

У цьому розділі було представлено архітектурну модель системи тестування ефективності алгоритмів балансування навантаження в SDN-мережах, а також описано принципи взаємодії між її ключовими компонентами. Розроблена архітектура демонструє чітке розмежування функціональних ролей між рівнями моделювання, керування, обробки даних та аналітики, що забезпечує високу структурованість і передбачуваність роботи системи.

Було визначено, що основою мережевої частини є емулятор Mininet та SDN-контролер Ryu, які формують віртуальне середовище для виконання експериментів. Серверна частина системи реалізує механізми оркестрації

топологій, управління експериментами, взаємодії з контролером, виконання алгоритмів балансування та збору телеметрії. Завдяки модульному REST API забезпечується стандартизована, контрольована та розширювана взаємодія між усіма компонентами.

Описаний механізм інтеграції алгоритмів балансування дозволяє динамічно підключати нові модулі та досліджувати різні підходи без втручання у базову логіку системи. Наявність підсистеми збору та обробки експериментальних даних забезпечує можливість отримання детальних метрик і проведення порівняльного аналізу результатів.

Аналіз масштабованості та гнучкості показав, що архітектура системи дозволяє адаптувати її до різних експериментальних сценаріїв, розширювати функціональність, збільшувати складність топологій та використовувати зовнішні обчислювальні модулі. Завдяки цьому система може застосовуватися не лише для академічних досліджень, а й для практичного моделювання роботи SDN-мереж у корпоративних або тестових середовищах.

Таким чином, у межах цього розділу сформовано цілісне бачення архітектурної структури та механізмів взаємодії компонентів системи, що є фундаментальною основою для її подальшої реалізації та аналізу результатів у наступних розділах роботи.

РОЗДІЛ 4. ВИПРОБУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Розгортання та конфігурація тестового середовища

Для роботи системи тестування необхідно підготувати ізольоване середовище, яке включає емулятор мережі Mininet, SDN-контролер Ryu, серверну частину на .NET 8, клієнтський веб-інтерфейс Angular та базу даних для збереження телеметрії. Усі компоненти запускаються окремо, але працюють узгоджено, утворюючи повноцінну платформу для виконання експериментів. Нижче наведено послідовну інструкцію щодо розгортання середовища.

Першим кроком є підготовка Linux-оточення через Windows Subsystem for Linux (WSL2), оскільки Mininet і Ryu функціонують лише в Linux. Спочатку виконується активація необхідних компонентів Windows. Після перезавантаження системи встановлюється дистрибутив Ubuntu (версії 20.04) з Microsoft Store.

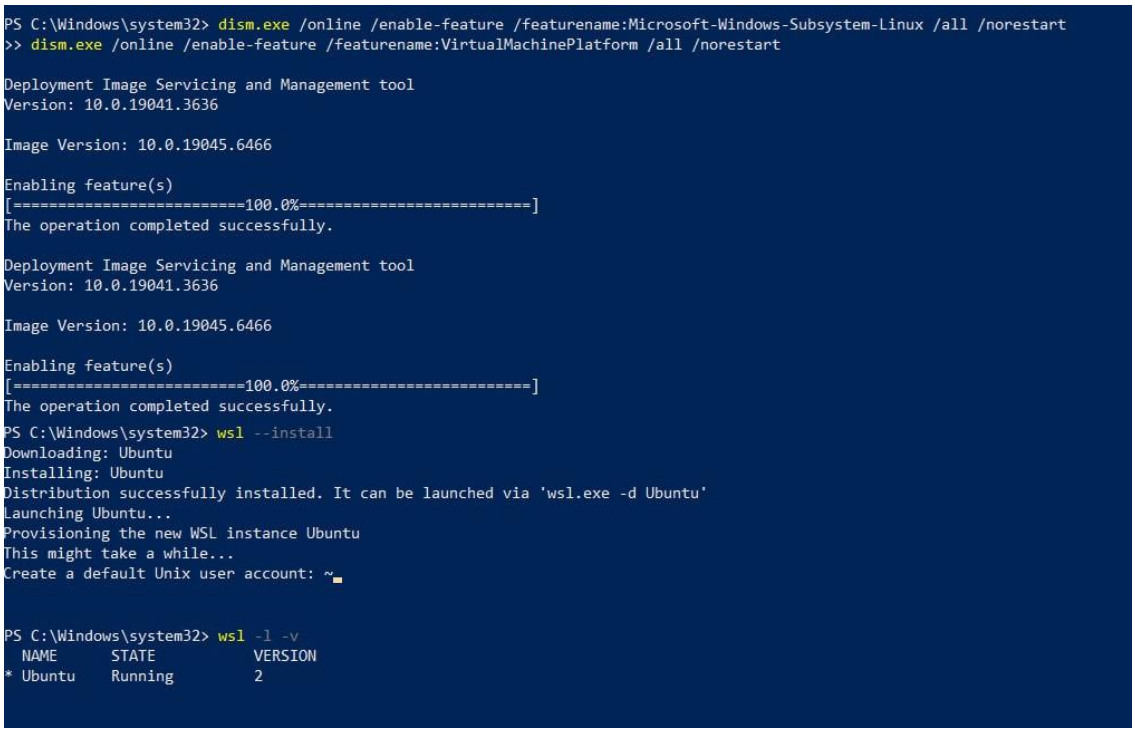

Рисунок 4.1 — Встановлення дистрибутиву Linux

Усередині Linux-середовища виконується базове оновлення командами sudo apt update & sudo apt upgrade -y.

Після підготовки системи встановлюється Mininet разом з Open vSwitch. Для цього необхідно завантажити репозиторій та запустити інсталяційний скрипт:

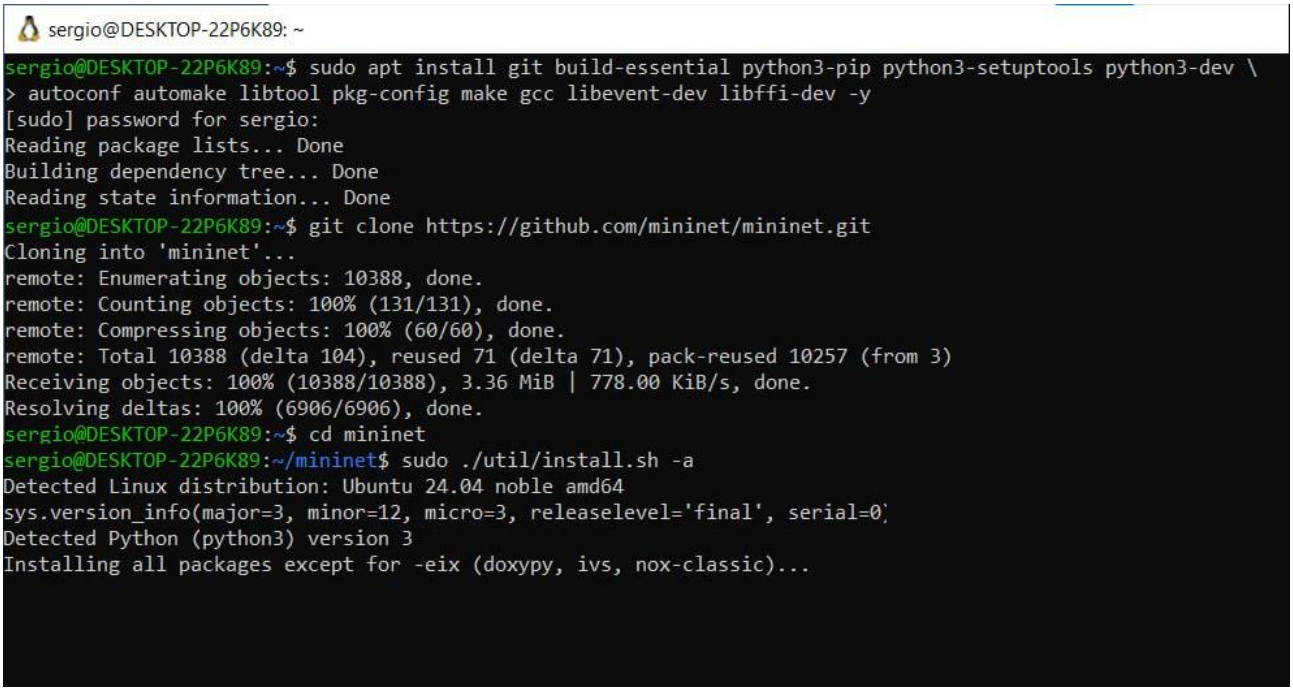

Рисунок 4.2 — Встановлення Mininet

Наступним етапом встановлюється Python-віртуальне середовище та Ryu контролер.

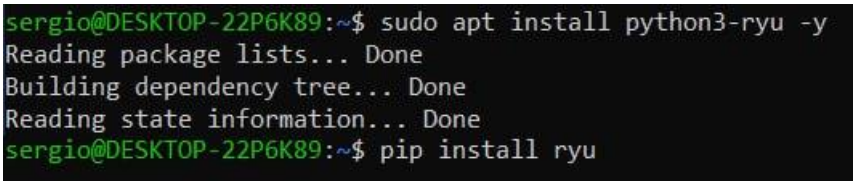

Рисунок 4.3 — Встановлення RyuController

Після того як Mininet і контролер налаштовані, запускається модель мережі та підключення до контролера. У новій вкладці WSL виконуються:

sudo mn --controller=remote,ip=127.0.0.1 --topo tree,depth=2

Якщо підключення успішне, у вікні Ryu з'являється повідомлення hello OFP....

Після розгортання Linux-частини налаштовується серверна складова. На Windows встановлюється .NET SDK 8, після чого завантажується і збирається Backend-сервіс:

git clone https://github.com/SergioKostyukov/SdnTestPlatformApi

cd SdnTestPlatformApi

dotnet restore

У конфігураційному файлі appsettings.json варто перевірити та, за потреби, оновити параметри підключення бази даних.

Після цього сервер запускається командою:

dotnet run

Паралельно налаштовується клієнтський інтерфейс Angular. Після встановлення Node.js та Angular CLI виконується:

cd SdnTestPlatformUI

npm install

ng serve --open

UI відкриється у браузері на http://localhost:4200.

Для збереження телеметрії використовується PostgreSQL, який встановлюється на Windows. Після запуску сервера БД створюється окрема база:

```
psql -U postgres

CREATE DATABASE sdn_tests;
```

Актуальна схема застосовується через виклик команди міграції з backend сервісу:

```
dotnet ef database update
```

Після завершення встановлення всіх компонентів система запускається у такій послідовності: спочатку активується Ryu у середовищі WSL, далі — Mininet з обраною топологією, після цього запускається Backend-сервіс та Angular-інтерфейс. Після відкриття сторінки UI користувач може перейти до конфігурації експерименту. На цьому етапі тестове середовище готове до повного циклу роботи системи.

4.2 Методика проведення експериментів

Процес проведення експериментів у створеній системі базується на чіткій послідовності дій, які користувач виконує для формування мережевого середовища, визначення параметрів навантаження та запуску механізму збору телеметрії.

Після запуску всіх компонентів системи користувач отримує доступ до інтерфейсу керування експериментами або до REST-ендпоінтів, які забезпечують ті самі можливості програмно. Першим кроком є вибір або формування мережевої топології. Користувач задає структуру мережі, зазначаючи кількість комутаторів, хостів та параметри каналів зв’язку. Система передає ці параметри серверній частині, яка формує відповідну конфігурацію для Mininet.

Після визначення топології задаються параметри навантаження. Користувач обирає сценарій трафіку та його характеристики, зокрема тип потоку (TCP/UDP), інтенсивність, інтервали зміни навантаження та тривалість експерименту. Ці значення використовуються для запуску генераторів трафіку у Mininet.

Після завершення налаштування система переходить до етапу запуску експерименту. Серверна частина створює або оновлює топологію, ініціалізує генерацію трафіку та активує модуль збору телеметрії. Під час експерименту сервер з певним інтервалом отримує статистику від SDN-контролера: дані про стан портів, активні потоки, пропускну здатність каналів та затримку між вузлами. Ці значення обробляються та фіксуються у внутрішній моделі стану мережі.

Процес збору даних триває до завершення експерименту або до моменту його примусового зупинення користувачем. Після цього система завершує роботу генераторів трафіку, скидає активні Flow-таблиці у SDN-контролері та формує підсумковий набір даних. Результати експерименту зберігаються у базі даних та можуть бути проаналізовані у подальших підрозділах, присвячених топологіям, навантаженню та поведінці алгоритмів.

Таким чином, методика проведення експериментів передбачає послідовність етапів — створення топології, конфігурація навантаження, запуск, збір телеметрії та завершення. Такий підхід забезпечує формалізованість дослідження та дозволяє отримати порівнювані результати.

4.3 Сценарії навантаження та моделі генерації трафіку

Для проведення експериментів у системі було підготовлено набір типових сценаріїв навантаження, що дозволяють моделювати різні режими роботи мережі та оцінювати поведінку алгоритмів балансування в однакових умовах. Сценарії відрізняються характером зміни інтенсивності, типом трафіку та динамікою появи потоків. Кожен із них може бути застосований у реальному режимі роботи (через утиліти Mininet), що забезпечує повноцінне формування мережевих пакетів і

точність вимірювань. Використання заздалегідь визначених сценаріїв гарантує відтворюваність експериментів та дозволяє порівнювати алгоритми при ідентичних вхідних умовах.

Першим типом навантаження є стабільний рівномірний трафік. Він моделює ситуацію, коли всі потоки підтримують фіксовану інтенсивність протягом усього експерименту. Такий режим використовується для базової перевірки роботи мережевої топології, коректності маршрутизації та оцінки стабільності системи без додаткових динамічних факторів. Рівномірний трафік є також важливою базовою точкою для калібрування та порівняння результатів між різними топологіями.

Другим сценарієм є пікове або імпульсне навантаження. Воно характеризується короткочасними різкими збільшеннями інтенсивності трафіку. Цей тип навантаження використовується для перевірки реакції системи на раптові зміни мережевих умов та оцінки стійкості модулів маршрутизації до високих навантажень. Пікові стрибки дозволяють дослідити поведінку алгоритмів у ситуаціях, де необхідна швидка переорієнтація потоків і мінімізація втрат пакетів.

Третій тип навантаження — стохастичний трафік зі змінною інтенсивністю. У цьому сценарії інтенсивність потоків змінюється випадковим чином у межах заданого діапазону. Він дає змогу моделювати мережі з непередбачуваною поведінкою користувачів та нерівномірним використанням каналів зв’язку. Стохастичний трафік особливо корисний у великих топологіях або в конфігураціях з потенційними «вузькими місцями». Такий режим дозволяє оцінити здатність алгоритмів пристосовуватися до швидких змін трафіку та уникати локальних перевантажень.

Окремо виділяється змішаний сценарій, який поєднує кілька типів потоків: стабільний фоновий трафік, імпульсні пікові значення, випадкові коливання та короткі UDP-запити. Такий режим дозволяє створити умови, близькі до реальних багатосервісних мереж, і є найбільш показовим для комплексного аналізу роботи

системи. Завдяки різнорідності трафіку цей сценарій дає змогу оцінити не лише продуктивність алгоритмів, але й їхню здатність підтримувати збалансований розподіл ресурсів у складних та динамічних середовищах.

Запропоновані сценарії створюють достатнє різноманіття мережевих умов і дозволяють перевіряти поведінку алгоритмів балансування під час репродуктивних, пікових та непередбачуваних режимів роботи. Крім того, використання таких сценаріїв забезпечує можливість систематичного порівняння, оскільки кожне тестування виконується при повністю відтворюваних параметрах навантаження.

4.4 Користувацький інтерфейс та візуалізація результатів

Користувацький інтерфейс системи створено таким чином, щоб забезпечити повний цикл взаємодії з платформою — від налаштування експерименту до перегляду його результатів. Окремою особливістю системи є демонстраційний режим, який дозволяє відтворювати типові сценарії навантаження та візуалізацію телеметрії на основі попередньо зібраних експериментальних даних, не запускаючи щоразу повний цикл формування трафіку. У цьому режимі UI використовує зафіксовані набори метрик, структура та динаміка яких відповідають реальній телеметрії, отриманій від Mininet і SDN-контролера.

4.4.1 Початковий екран та активація демонстраційного режиму

Після запуску Angular-додатка користувач потрапляє на головну сторінку керування експериментами. Система автоматично перевіряє доступність Backend-сервісу та відображає статус підключення. З головного екрану користувач може обрати один із двох варіантів: запуск нового експерименту в реальному режимі або перехід у демонстраційний режим, де інтерфейс використовує попередньо збережені результати раніше виконаних тестів для відтворення поведінки системи.

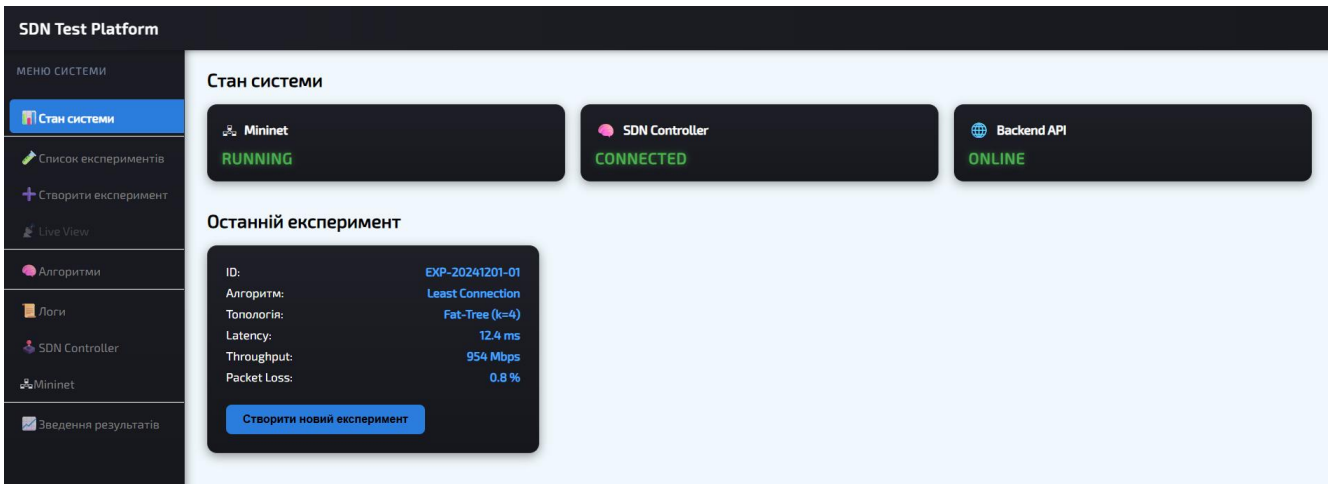

Рисунок 4.4 — Сторінки загального стану системи

4.4.2 Конфігурація експерименту

Після переходу до налаштування експерименту користувач обирає топологію, тип навантаження та алгоритм балансування. У реальному режимі ці параметри формують конфігурацію, яка надсилається Backend-сервісу та використовується для запуску Mininet і SDN-контролера згідно з заданим сценарієм.

У демонстраційному режимі UI відображає схему вибраної топології на основі наперед визначених шаблонів, що дає змогу побачити її структуру без обов’язкового запуску нового експерименту. Для кожного параметра система формує внутрішню конфігурацію, яка в реальному режимі використовується Backend-сервісом, а у демонстраційному — слугує ключем для вибору відповідного набору збережених телеметричних даних.

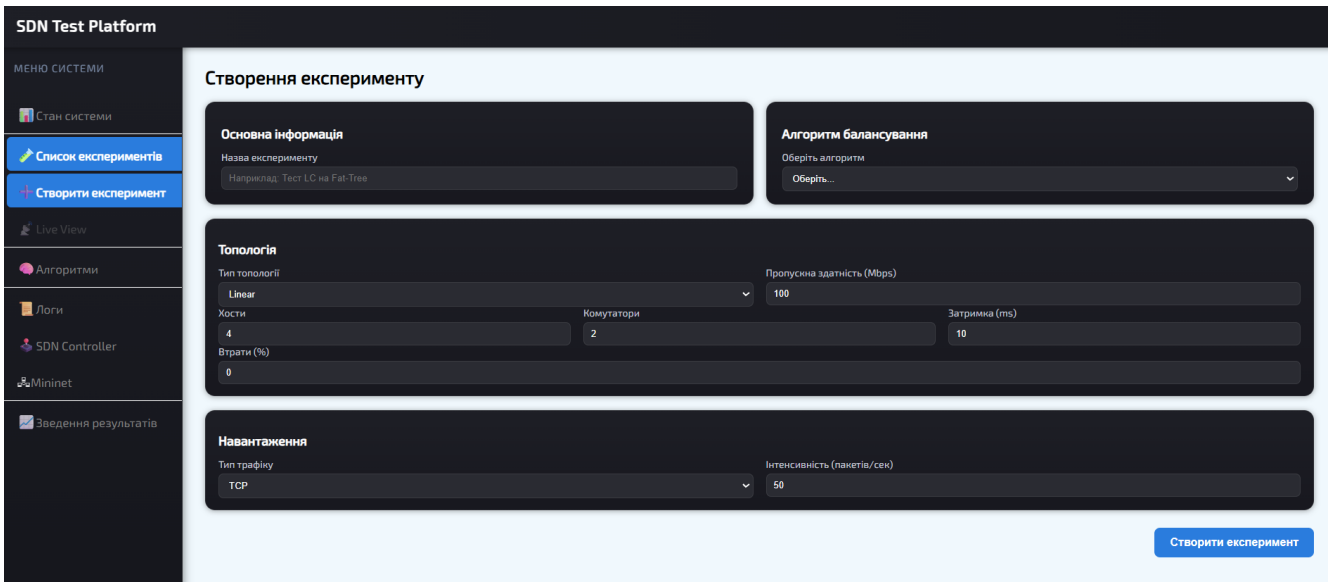

Рисунок 4.5 — Сторінка конфігурації експерименту

4.4.3 Запуск експерименту та перехід до моніторингу

Після підтвердження налаштувань користувач запускає експеримент.
У реальному режимі Backend створює топологію у Mininet, встановлює з’єднання з SDN-контролером та ініціалізує генерацію трафіку згідно з обраним сценарієм.

У демонстраційному режимі UI підключається до локального або серверного джерела збережених метрик і починає послідовно відтворювати їх із заданим інтервалом (300–500 мс), імітуючи потік телеметрії в реальному часі. Після запуску інтерфейс автоматично переходить на сторінку моніторингу.

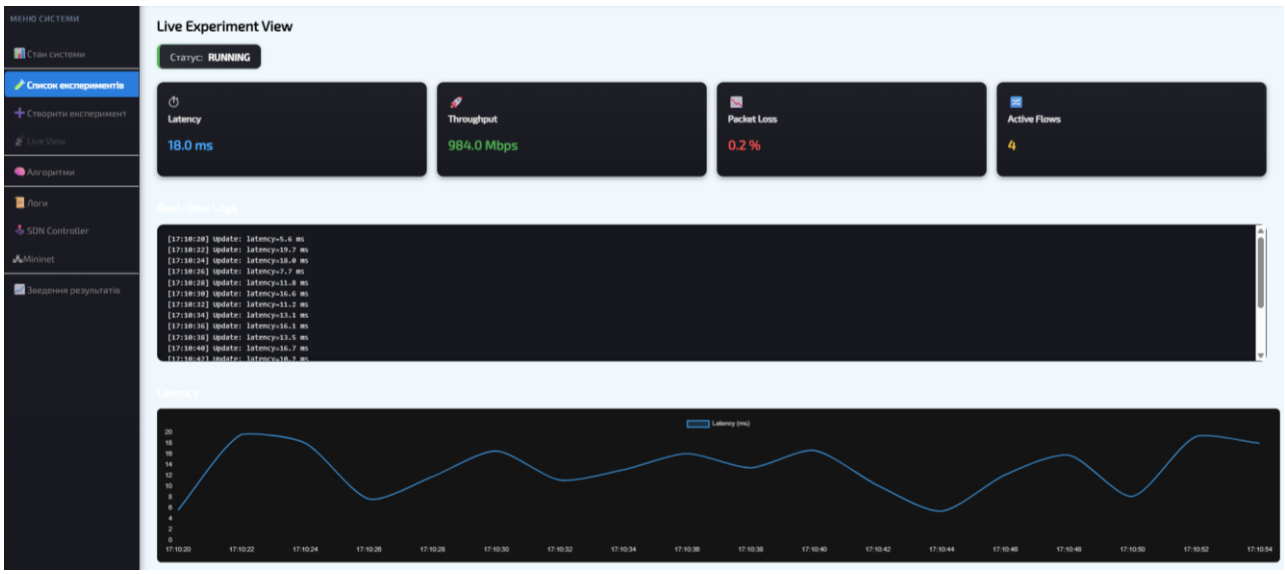

Рисунок 4.6 — Сторінка моніторингу експерименту

4.4.4 Візуалізація результатів у реальному часі

Під час виконання експерименту UI оновлює панелі моніторингу відповідно до отриманих метрик. Усі графіки побудовані таким чином, щоб відображати поведінку мережі та алгоритмів у динаміці.

У режимі онлайн-візуалізації інтерфейс показує:

- пропускну здатність каналів (throughput);
- середню та максимальну затримку;
- кількість активних потоків;
- статистику портів комутаторів;
- загальний стан мережевого навантаження.

У демонстраційному режимі ці значення відтворюються на основі попередньо збережених наборів даних, отриманих під час реальних запусків експериментів. Це дозволяє демонструвати повну картину роботи системи та алгоритмів балансування без виконання нового тесту, зберігаючи при цьому структуру й формат реальної телеметрії.

4.4.5 Завершення експерименту та підсумкова статистика

Після завершення експерименту користувач натискає кнопку «Завершити», після чого UI припиняє збір і оновлення даних. На екран виводиться таблиця ключових агрегованих показників:

- середня та максимальна затримка,
- максимальний throughput,
- загальний обсяг переданих даних,
- кількість втрат пакетів,
- тривалість експерименту.

Завдяки цьому UI фактично виконує роль універсального інструменту моніторингу та візуалізації: у реальному режимі він відображає живу телеметрію від Mininet та SDN-контролера, а в демонстраційному — дозволяє відтворювати та аналізувати вже зібрані результати експериментів у зручному інтерактивному форматі.

4.5 Результати роботи та аналіз поведінки системи

Під час проведення експериментів система продемонструвала стабільну роботу всіх модулів — Mininet, SDN-контролера, Backend-оркестратора та інтерфейсу візуалізації. Для кожної топології було зібрано повний перелік телеметрії, що дозволило оцінити реакцію мережі на зміни навантаження та вплив вибраного алгоритму балансування.

У тестах із рівномірним навантаженням алгоритм Round Robin забезпечив рівномірний розподіл потоків між доступними маршрутами, що відобразилося у стабільній пропускній здатності та симетричному завантаженні портів комутаторів. Затримка залишалася майже сталою, а коливання throughput не перевищували кількох відсотків, що свідчить про коректну реакцію системи на статичні сценарії.

Алгоритм Least Connection показав ефективнішу реакцію на динамічні зміни інтенсивності. У тестах з піковими навантаженнями він швидше вирівнював потоки між менш завантаженими лінками, що призводило до зменшення черг на окремих портах. Це проявилося у нижчій середній затримці під час пікових фаз і зменшенні епізодів перевантаження окремих комутаторів.

У складніших топологіях із кількома альтернативними маршрутами адаптивний алгоритм демонстрував найбільш гнучку поведінку. Він змінював маршрути в залежності від локального стану лінків, що відобразилося у коливаннях у Flow-таблицях та перерозподілі навантаження протягом експерименту. Телеметрія портів підтвердила, що алгоритм коригував напрями передавання потоків при зміні затримки та зростанні завантаження окремих сегментів мережі.

Для всіх алгоритмів система стабільно збирала метрики пропускної здатності, затримки, статистики по портах, а також кількість створених і завершених потоків. Часові графіки показали, що платформа коректно фіксує реакцію мережі на зміну характеру трафіку, а механізми візуалізації дозволяють у реальному часі оцінювати роботу як контролера, так і окремого алгоритму балансування.

Таким чином, система підтвердила можливість проведення порівняльного аналізу методів балансування, забезпечивши відтворюваність умов та точність вимірювань.

4.6 Обмеження розробленої системи та напрями покращення

Проведені експерименти продемонстрували готовність системи до виконання повноцінних досліджень алгоритмів балансування навантаження в SDN-мережах. Разом із тим, поточна реалізація має низку обмежень, характерних для першої інтеграційної версії платформи, у якій основна увага приділялася архітектурі, модульності та цілісності інфраструктури.

Одним із ключових обмежень є масштабованість мережевих топологій. Хоча система підтримує декілька типових конфігурацій — лінійні, деревоподібні, fat-tree та топології з вузьким місцем — їхня складність на цьому етапі обмежена кількома десятками вузлів. Для моделювання дата-центрових сценаріїв із сотнями або тисячами хостів необхідні додаткові оптимізації механізмів створення топологій у Mininet, а також удосконалення логіки обробки телеметрії та виконання алгоритмів у режимі реального часу.

У контексті алгоритмів балансування система наразі включає Round Robin, Least Connection та адаптивний модуль, що працює на основі поточних метрик мережі. Хоча цього достатньо для базового порівняльного аналізу підходів, у майбутніх версіях планується розширення бібліотеки алгоритмів за рахунок моделей машинного навчання та прогнозування. Для їх інтеграції необхідно передбачити інтерфейс для підключення зовнішніх моделей, а також модуль навчання, здатний працювати з історичними даними та прогнозувати навантаження з урахуванням часових залежностей.

Іншим аспектом є збирання та зберігання телеметрії. Поточна реалізація базується на PostgreSQL і демонструє задовільну продуктивність, однак зі зростанням частоти запитів та обсягу даних може виникнути необхідність використання спеціалізованих time-series баз даних (InfluxDB, TimescaleDB) або проміжного буферизуючого шару. Це дозволить зменшити затримки при записі великих потоків телеметрії та забезпечити плавне оновлення метрик для алгоритмів, що залежать від високої частоти опитування SDN-контролера.

Користувацький інтерфейс системи вже надає засоби для моніторингу ключових параметрів — пропускної здатності, затримки, статистики портів та кількості активних потоків. Проте у майбутньому доцільно розширити можливості аналітики: додати панелі для порівняння роботи декількох алгоритмів, теплові карти завантаження, графи топології з динамічним виділенням перевантажених лінків, а також можливість візуалізувати зміни Flow-таблиць у часі.

Ще одним напрямом удосконалення є автоматизація досліджень. Поточна система орієнтована на ручний запуск експериментів, тоді як наукові дослідження часто вимагають виконання серійних тестів із різними параметрами у пакетному режимі. Додавання планувальника експериментів дозволить запускати великі набори бенчмарків автоматично, забезпечуючи повторюваність результатів і спрощуючи проведення багатофакторного аналізу.

Усі наведені обмеження є типовими для першої повністю інтегрованої версії системи та не знижують її практичної цінності. Архітектура платформи спроєктована з урахуванням розширюваності: стандартизовані API, модульність Backend-логіки та уніфікований формат телеметрії дозволяють додавати нові компоненти без модифікації ядра.

У майбутніх версіях планується:

- масштабування підтримуваних топологій до рівня дата-центрових сценаріїв;
- розширення бібліотеки алгоритмів, зокрема AI/ML-моделями;
- оптимізація механізмів збору й обробки телеметрії;
- впровадження time-series баз даних;
- підтримка пакетного запуску експериментів;
- удосконалення UI для глибшої порівняльної аналітики.

Таким чином, розроблена система вже сьогодні є повноцінною платформою для дослідження алгоритмів балансування навантаження в SDN-мережах, а її архітектурна відкритість забезпечує значний потенціал для подальшого розвитку та реалізації складніших експериментальних сценаріїв.

4.7 Висновки до розділу №4

У цьому розділі було продемонстровано роботу створеної системи тестування алгоритмів балансування навантаження в SDN-мережах та проаналізовано її поведінку в умовах різних топологій і сценаріїв трафіку.

Експериментальні запуски підтвердили, що платформа коректно відтворює усі етапи повного циклу роботи SDN-системи — від конфігурації мережевого середовища, створення віртуальної топології та генерації навантаження до збору телеметрії, обробки результатів і їх візуалізації.

Система продемонструвала стабільну роботу при тестуванні різних типів топологій, підтримку кількох профілів навантаження та можливість аналізу поведінки алгоритмів балансування в режимі реального часу. Інтерфейс користувача забезпечив зручний доступ до основних метрик — пропускної здатності, затримки, статистики портів та використання каналів — що дозволило оперативно оцінювати стан мережі та вплив алгоритмів на її продуктивність.

Порівняльний аналіз алгоритмів Round Robin, Least Connection та адаптивного методу показав суттєву різницю в їхній ефективності залежно від обраної топології та характеру навантаження. Виявлені особливості дозволили визначити сильні та слабкі сторони кожного підходу, а також сформувати напрямки для подальших досліджень і оптимізацій.

У межах розділу також було окреслено існуючі обмеження системи та можливості її розвитку — розширення масштабності топологій, інтеграція алгоритмів на основі машинного навчання, вдосконалення механізмів збору телеметрії та покращення засобів аналітики в інтерфейсі користувача.

Загалом система вже на цьому етапі виступає як повноцінна платформа для дослідження механізмів балансування навантаження в SDN-мережах, забезпечуючи відтворюваність експериментів, модульність компонентів та готовність до масштабування і подальшого розширення функціональності.

ВИСНОВКИ

У магістерській роботі проведено всебічне дослідження проблеми оцінювання ефективності алгоритмів балансування навантаження у програмно-визначуваних мережах і створено комплексну систему, яка забезпечує повний цикл моделювання, запуску та аналізу мережевих експериментів у контрольованому середовищі. На основі детального огляду сучасних SDN-технологій, протоколу OpenFlow, принципів централізованого управління трафіком, а також інструментів мережевої емуляції, сформовано науково обґрунтовані вимоги до тестової платформи, що дозволили вибудувати архітектуру, здатну підтримувати відтворювані, керовані та масштабовані експериментальні сценарії.

Розроблена система поєднує Mininet як середовище моделювання мережевої топології, контролер Ryu як механізм централізованого управління потоками, а також серверну частину на платформі .NET, яка оркеструє всі процеси експерименту — від створення топології та запуску трафіку до виконання алгоритмів балансування й обробки телеметрії. Структура системи забезпечує чітке розмежування функціональних ролей і підтримує модульний підхід, що робить можливим динамічне підключення нових алгоритмів, оновлення конфігурацій та розширення функціональності без втручання в основні компоненти платформи.

У ході роботи підтверджено, що SDN-підхід істотно полегшує процес реалізації та дослідження механізмів балансування завдяки централізованому керуванню, глобальному баченню стану мережі та можливості отримання телеметрії в режимі реального часу. Розроблена платформа продемонструвала здатність коректно відтворювати мережеві сценарії різної складності, підтримувати різні типи навантаження, а також забезпечувати об'єктивні метрики, необхідні для аналізу алгоритмів. Це дозволяє використовувати систему як інструмент порівняльного дослідження класичних, адаптивних та інтелектуальних методів

балансування, включно з тими, що базуються на машинному навчанні або прогнозних моделях.

Отримані результати свідчать про відповідність розробленої системи поставленим завданням і її здатність задовольняти потреби як дослідницьких лабораторій, так і практичних інженерних підрозділів, які займаються оптимізацією мережевої інфраструктури. Створена платформа може використовуватися для навчальних цілей, для тестування реальних алгоритмів у прототипних топологіях, а також як основа для побудови більш складних рішень у сфері автоматизації мережевого управління.

Перспективи подальшого розвитку системи охоплюють розширення масштабності мережевих топологій, впровадження спеціалізованих time-series сховищ телеметрії, інтеграцію з інтелектуальними контролерами наступного покоління, а також створення модулів автоматизованого запуску серійних експериментів. Завдяки відкритій архітектурі система є придатною до інтеграції нових алгоритмів, включно з моделями на основі штучного інтелекту, що відкриває широкі можливості для майбутніх досліджень і практичного застосування в галузі управління мережевими ресурсами.

Таким чином, розроблена система тестування ефективності алгоритмів балансування навантаження в SDN-мережах є значущим інструментом для сучасної мережевої інженерії, забезпечуючи надійну експериментальну базу, розширюваний функціонал і високий потенціал для подальшого наукового та прикладного розвитку.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Open Networking Foundation. *Software-Defined Networking Architecture* : електрон. ресурс. 2023. URL: https://opennetworking.org (дата звернення: 15.01.2025).
2. Kreutz D., Ramos F. M. V., Verissimo P., Rothenberg C. E., Azodolmolky S., Uhlig S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*. 2015. Vol. 103, No. 1. P. 14–76.
3. Hu F., Hao Q., Bao K. A survey on software-defined networking and OpenFlow: From concept to implementation. *IEEE Communications Surveys & Tutorials*. 2014. Vol. 16, No. 4. P. 2181–2206.
4. Mininet Team. *Mininet Documentation* : електрон. ресурс. 2023. URL: http://mininet.org (дата звернення: 14.01.2025).
5. Ryu SDN Framework. *Ryu Documentation* : електрон. ресурс. URL: https://ryu-sdn.org (дата звернення: 16.01.2025).
6. Bari M. F., Roy A. R., Chowdhury N. M. K. Dynamic load balancing in software-defined networks: A survey. *IEEE Communications Surveys & Tutorials*. 2020. Vol. 22, No. 1. P. 472–503.
7. Shchur V. Optimized adaptive load balancing method in software-defined networks. *Науковий вісник Львівського національного університету імені Івана Франка*. 2023. № 4. С. 62–66.
8. Smirnov A., Lee J. Load-aware traffic engineering for SDN data centers. In: *Proceedings of IEEE ICC 2021*. Montreal, 2021. P. 1–6.
9. Azhar S., Shahria I., Rahman M. *Software Defined Networking: Principles and Applications*. London : Springer, 2021. 242 p.
10. Sezer S., Scott-Hayward S., Chouhan P. *Software-Defined Networking and Security*. Hoboken : Wiley, 2018. 336 p.
11. Орлова М. М., Багінський Є. С. Оптимізація балансування навантаження в стільникових мережах LTE/TE-A. *Прикладна математика та комп'ютинг*

ПМК–2018 : тези доповідей, 21–23 березня 2018 р. Київ : КПІ ім. І. Сікорського, 2018. С. 87–91.

12. Мартинова О.П., Костюков С.В. Система тестування ефективності алгоритмів балансування навантаження в SDN мережах. Прикладна математика та комп'ютинг ПМК–2025 : тези доповідей, 19–21 листопада 2025 р. Київ : КПІ ім. І. Сікорського, 2025. С. 179–183.
13. Мартинова О.П., Костюков С.В., Кучмій О.О. Контейнеризація та її вплив на процес розробки програмного забезпечення. Збірник тез доповідей XV Міжнародної науково-практичної конференції «Комп'ютерні системи та мережні технології» (CSNT-2024), 25–26 квітня 2024 р., м. Київ: НАУ, 2024. – С. 91-92. https://csnt.nau.edu.ua/files/2024/sbirnyk2024.pdf.
14. Мартинова О.П., Костюков С.В. Застосування штучного інтелекту для балансування навантаження серверів мереж SDN. «Комп'ютерні інтелектуальні системи та мережі»: Матеріали XVIII Всеукраїнської науково практичної WEB конференції аспірантів, студентів та молодих вчених (25-27 березня 2025 р.). – Кривий Ріг: Криворізький національний університет, 2025.– С. 52-55. https://sites.google.com/view/kicm#h.p_2GJDU4EZFcVZ