ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «ПРОГРАМНО-АПАРАТНИЙ КОМПЛЕКС ДЛЯ МОДЕЛЮВАННЯ ТА ОПТИМІЗАЦІЇ КОМП'ЮТЕРНИХ МЕРЕЖ»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми «Комп'ютерна інженерія»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *В'ячеслав СЛАДЬ*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач(ка) вищої освіти гр. КІД-42
В'ячеслав СЛАДЬ
Ім'я, ПРІЗВИЩЕ

Керівник: Людмила АСЄЄВА
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерна інженерія

Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сладь В'ячеслав Олександрович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Програмно-апаратний комплекс для моделювання та оптимізації комп'ютерних мереж»
керівник кваліфікаційної роботи: Людмила АСЄЄВА, phD (доктор філософії), доцент ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025р. №56
2. Строк подання кваліфікаційної роботи «05» червня 2025р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, стандарти та протоколи TCP/IP, принципи віртуалізації мереж, інструменти контейнеризації (Docker, Docker Compose), засоби тестування та моделювання мережевого трафіку (iperf), системи моніторингу та візуалізації (Prometheus, Grafana), архітектура комп'ютерних мереж, методи оптимізації та аналізу продуктивності.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Теоретичні основи моделювання та оптимізації комп'ютерних мереж
    2. Побудова програмно-апаратного комплексу для моделювання
    3. Аналіз ефективності та перспективи розвитку
5. Перелік ілюстративного матеріалу: *презентація*
    1. Мета, об'єкт, предмет дослідження
    2. Актуальність роботи
    3. Загальна архітектура запропонованої системи
    4. Розгортання віртуального ізольваного середовища
    5. Генерація трафіку за допомогою iperf
    6. Алгоритми функціонування у ізольованій системі
    7. Демонстрація функціонування системи (візуалізація)
    8. Методи покращення існуючої системи
    9. Порівняння запропонованої системи зі альтернативами

10. Висновки
11. Публікації та апробація роботи

6. Дата видачі завдання «24» лютого 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	25.02-19.03.2025	
2	Обробка матеріалу	20.03-02.04.2025	
3	Розробка теоретичної частини	03.04-07.04.2025	
5	Побудова програмно-апаратного комплексу для моделювання	08.04-27.04.2025	
6	Аналіз ефективності та перспективи розвитку	28.04-11.05.2025	
7	Висновки за результатами аналізу	12.05-15.05.2025	
8	Розробка презентації	16.05-26.05.2025	
9	Передзахист	27.05-29.05.2025	
10	Здача в деканат	30.05-05.06.2025	

Здобувач(ка) вищої освіти _____ В’ячеслав СЛАДЬ
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Людмила АСЄЄВА
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 92 стор., 52 рис., 6 табл., 26 джерел.

Мета роботи – спрощення процесу моделювання та оптимізації комп'ютерних мереж за рахунок використання принципів контейнеризації, інструментів збору метрик та візуалізації продуктивності.

Об'єкт дослідження – процес функціонування комп'ютерних мереж у віртуалізованому середовищі.

Предмет дослідження – технології віртуалізації, засоби тестування мережевого трафіку та системи моніторингу для моделювання та оптимізації мереж.

Короткий зміст роботи: У роботі представлено підхід до побудови програмно-апаратного комплексу для моделювання комп'ютерних мереж із використанням інструментів віртуалізації та контейнеризації. Змодельовано віртуальну мережу, яка складається з кількох ізольованих вузлів, на яких відтворюються типові сценарії мережевого трафіку за допомогою iperf. Для збору і аналізу метрик продуктивності інтегровано Prometheus і Grafana, що дозволило отримати візуальні уявлення про параметри затримок, пропускної здатності та втрат пакетів.

Особливу увагу приділено архітектурі тестового середовища, реалізації контейнерної інфраструктури та можливостям масштабування. Проведено порівняльний аналіз інструментів, що використовуються у практиці мережевого тестування, виявлено переваги застосування контейнеризації та автоматизованого моніторингу в освітньому і дослідницькому контексті.

КЛЮЧОВІ СЛОВА: ВІРТУАЛІЗАЦІЯ, МЕРЕЖЕВЕ МОДЕЛЮВАННЯ, DOCKER, КОНТЕЙНЕРИ, IPERF, PROMETHEUS, GRAFANA, ОПТИМІЗАЦІЯ, ЗАТРИМКА, ПРОПУСКНА ЗДАТНІСТЬ, МОНІТОРИНГ, ТРАФІК.

ABSTRACT

Text part of the qualification work for obtaining a bachelor's degree: 92 pages, 52 figures, 6 tables, 26 sources.

The purpose of the work is to simplify the process of modeling and optimizing computer networks by using the principles of containerization, metrics collection tools, and performance visualization.

The object of the study is the process of functioning of computer networks in a virtualized environment.

The subject of the study is virtualization technologies, network traffic testing tools, and monitoring systems for modeling and optimizing networks.

Summary of the work: The work presents an approach to building a software and hardware complex for modeling computer networks using virtualization and containerization tools. A virtual network consisting of several isolated nodes is modeled, on which typical network traffic scenarios are reproduced using iperf. Prometheus and Grafana are integrated to collect and analyze performance metrics, which allows us to obtain visual representations of the parameters of delays, bandwidth, and packet loss.

Particular attention is paid to the architecture of the test environment, the implementation of container infrastructure, and scalability capabilities. A comparative analysis of tools used in network testing practice was conducted, and the advantages of using containerization and automated monitoring in an educational and research context were identified.

KEYWORDS: VIRTUALIZATION, NETWORK MODELING, DOCKER, CONTAINERS, IPERF, PROMETHEUS, GRAFANA, OPTIMIZATION, DELAY, BANDWIDTH, MONITORING, TRAFFIC.

ЗМІСТ

ВСТУП..... 9
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ МОДЕЛЮВАННЯ ТА ОПТИМІЗАЦІЇ КОМП'ЮТЕРНИХ МЕРЕЖ 10
1.1 Основи комп'ютерних мереж 10
1.2 Методи моделювання комп'ютерних мереж..... 18
1.3 Програмно-апаратні засоби віртуалізації мереж 24
1.4 Методи оптимізації роботи комп'ютерних мереж 31
1.5 Засоби моніторингу стану комп'ютерних мереж. 38
РОЗДІЛ 2. ПОБУДОВА ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ ДЛЯ МОДЕЛЮВАННЯ. 47
2.1 Архітектура запропонованої системи 47
2.2 Побудова віртуального мережевого середовища 55
2.3 Алгоритми збору й аналізу метрик 62
2.4 Реалізація побудованої системи 73
РОЗДІЛ 3. АНАЛІЗ ЕФЕКТИВНОСТІ ТА ПЕРСПЕКТИВИ РОЗВИТКУ..... 82
3.1 Методи покращення запропонованої системи..... 82
3.2 Порівняння з альтернативними рішеннями..... 89
3.3 Напрямки подальшого розвитку системи..... 95
ВИСНОВКИ..... 100
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 101
Додаток А. Конфігурація "docker-compose.yml" для розгортання системи моніторингу та тестування мережі 103
Додаток Б. Конфігурація дашборда Grafana для візуалізації метрик мережі 105
Додаток В. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)..... 109

ВСТУП

Сучасні комп’ютерні мережі є основою функціонування більшості інформаційних систем у бізнесі, освіті, промисловості та побуті. З ростом обсягів даних, швидкості передавання інформації та складності топологій виникає необхідність у глибокому розумінні принципів їхньої побудови, класифікації, а також характеристик продуктивності. Мережі різних масштабів — від локальних (LAN) до глобальних (WAN) — мають специфічні технічні особливості та завдання, що вимагає відповідного підходу до їх моделювання та тестування.

Ключовими параметрами ефективності роботи мережі є затримка, пропускна здатність, рівень втрати пакетів та стабільність з’єднання, що визначають якість обслуговування та задоволення вимог користувачів. У контексті оптимізації ці параметри відіграють вирішальну роль для розробників, інженерів та дослідників. Традиційні засоби аналізу та побудови мереж вимагають значних ресурсів, часу та часто не дозволяють гнучко змінювати конфігурації без додаткових витрат, що актуалізує використання засобів віртуалізації.

Контейнерні технології, такі як Docker, забезпечують просте, масштабоване та ізольоване середовище для створення тестових мереж, моделювання різних сценаріїв навантаження та аналізу їх продуктивності. У поєднанні з інструментами моніторингу (Prometheus, Grafana) з’являється можливість не лише створювати складні мережеві структури, але й детально відстежувати стан системи, візуалізувати ключові метрики та виявляти вузькі місця, що значно полегшує процес аналізу та експериментального дослідження.

Усе це створює передумови для побудови програмно-апаратного комплексу, який дозволить ефективно моделювати комп’ютерні мережі, аналізувати їх поведінку в реальному часі та вносити коригування з метою оптимізації. Такий підхід стає особливо цінним у дослідницькому середовищі, навчальному процесі та при попередньому тестуванні рішень перед розгортанням у реальній інфраструктурі.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ МОДЕЛЮВАННЯ ТА ОПТИМІЗАЦІЇ КОМП'ЮТЕРНИХ МЕРЕЖ

1.1 Основи комп'ютерних мереж

Комп'ютерні мережі є фундаментальним елементом цифрової інфраструктури, що забезпечує обмін даними між пристроями, сервісами та користувачами в реальному часі. Завдяки мережевим технологіям забезпечується:

- спільний доступ до інформаційних ресурсів;
- централізоване зберігання даних;
- віддалене управління;
- комунікація між віддаленими об'єктами;
- інтеграція багатьох інформаційних систем.

Без побудови якісної та стабільної мережевої архітектури неможливо уявити ефективну роботу підприємств, організацій і сервісів, які обслуговують велику кількість користувачів, а також забезпечують цілодобову доступність до ресурсів. Функціонування таких мереж вимагає правильного проєктування, що включає аналіз архітектури, моделювання навантажень, підбір відповідних протоколів і топологій з урахуванням майбутньої масштабованості [1-2].

Для забезпечення високої надійності й ефективності мережевих рішень необхідно враховувати безліч чинників – від фізичної організації зв'язків до вибору програмних інструментів оптимізації. Серед них важливу роль відіграють:

- швидкість передачі даних;
- пропускна здатність каналів;
- затримка сигналу;
- кількість вузлів;
- структура взаємозв'язків.

Саме тому при створенні сучасної мережі особливої ваги набуває процес моделювання її структури до реального впровадження, що дозволяє виявити вузькі місця, спрогнозувати поведінку системи при пікових навантаженнях та скоригувати параметри без фінансових втрат. Важливим аспектом є і правильне використання аналітичних та моніторингових інструментів, які допомагають у процесі оцінювання ефективності функціонування мереж [1-2].

Класифікація комп'ютерних мереж. Загальна класифікація комп'ютерних мереж передбачає поділ за масштабом покриття, способами з'єднання та призначенням. Найпоширенішим типом є локальні мережі (LAN), що охоплюють обмежені території на кшталт офісу, будівлі чи кампусу. Вони характеризуються високою швидкістю передачі даних, централізованим управлінням і низькою затримкою, що робить їх ідеальними для внутрішнього використання підприємствами. Наступним рівнем є MAN – міські мережі, які об'єднують кілька LAN у межах одного населеного пункту, зазвичай перебуваючи під управлінням інтернет-провайдерів чи великих установ. WAN – глобальні мережі – охоплюють країни та континенти, забезпечуючи міжконтинентальну передачу даних завдяки використанню маршрутизаторів, протоколів динамічної маршрутизації та волоконно-оптичних каналів. Існують також PAN – персональні мережі, які охоплюють дуже малі відстані, наприклад зв'язок між смартфоном і навушниками, або ноутбуком і мишею через Bluetooth чи USB, що формує тимчасове приватне з'єднання. Додатково зустрічаються вузькоспеціалізовані типи мереж як-от Campus Area Network (CAN) або Storage Area Network (SAN), проте в загальному аналізі класифікації доцільно зосередитися на чотирьох основних типах:

— Local Area Network (LAN);
— Metropolitan Area Network (MAN);
— Wide Area Network (WAN);
— Personal Area Network (PAN).

Масштабне зростання кількості пристроїв, потреба в динамічному масштабуванні інфраструктури, підтримка безперервної доступності сервісів і

зростаючий обсяг переданих даних вимагають нових підходів до розробки мережевих рішень (див. рис. 1.1).



Рисунок 1.1 – Класифікація комп'ютерних мереж за масштабом

Традиційні способи розгортання фізичних мереж часто не здатні оперативно реагувати на зміну вимог, тому все частіше виникає потреба в моделюванні мереж на основі віртуалізації та контейнеризації, що дозволяє ефективно перевіряти архітектурні рішення, оптимізувати параметри продуктивності, тестувати конфігурації й знижувати витрати на інфраструктуру [1].

Мережеві топології та протоколи передачі даних. Окрім класифікації комп'ютерних мереж, важливо розуміти їхню топологічну структуру та принципи організації передачі даних, адже мережі функціонують не лише як набір пристроїв, а як впорядкована система, в якій взаємодія забезпечується через конкретні з'єднання за визначеною логікою. Мережева топологія описує фізичну або логічну конфігурацію з'єднань між вузлами, і саме ця конфігурація визначає спосіб, у який дані циркулюють у межах мережі. Однією з найпростіших є топологія шини, коли всі пристрої підключені до єдиної лінії передачі даних; її реалізація є досить дешевою, але через вразливість до відмови центрального каналу шина втрачає надійність у великих мережах. У випадку топології зірки, яка передбачає з'єднання всіх вузлів із центральним комутатором або маршрутизатором, забезпечується вища стабільність і простота масштабування, проте вона також має слабке місце —

залежність від працездатності центрального вузла, відмова якого виводить із ладу всю систему.

Існує також топологія кільця, при якій кожен вузол підключений до двох інших, утворюючи замкнене кільце, і дані передаються від одного до іншого в одному напрямку, що дозволяє уникати колізій, але будь-який збій у кільці призводить до зупинки комунікації. На відміну від неї, сіткова топологія (mesh) пропонує часткове або повне з’єднання кожного вузла з усіма іншими, що забезпечує високу відмовостійкість та продуктивність, хоча і вимагає значних витрат на обладнання та складність реалізації. Зараз спостерігається поширення споживчих рішень на основі сіткової топології, які дозволяють побудувати локальну безпровідну мережу в будинку або офісі без прокладки додаткових кабелів, коли один маршрутизатор передає сигнал до іншого, формуючи своєрідну мережеву сітку, хоча в реальних умовах досягти стабільної гігабітної швидкості на кожному вузлі складно через вплив перешкод і загальних обмежень безпровідних технологій.

У віртуалізованих середовищах, зокрема в Docker-контейнерах, моделювання мереж часто здійснюється шляхом створення логічних топологій, схожих на зірку або сітку, де кожен контейнер, що виконує роль віртуального вузла, має заздалегідь визначене з’єднання з іншими, утворюючи функціональні структури для обміну даними у межах контейнерної інфраструктури, що дозволяє ефективно імітувати складні мережеві взаємодії у контрольованому середовищі та забезпечувати відповідні експерименти з продуктивністю (див. рис. 1.2) [1-2].

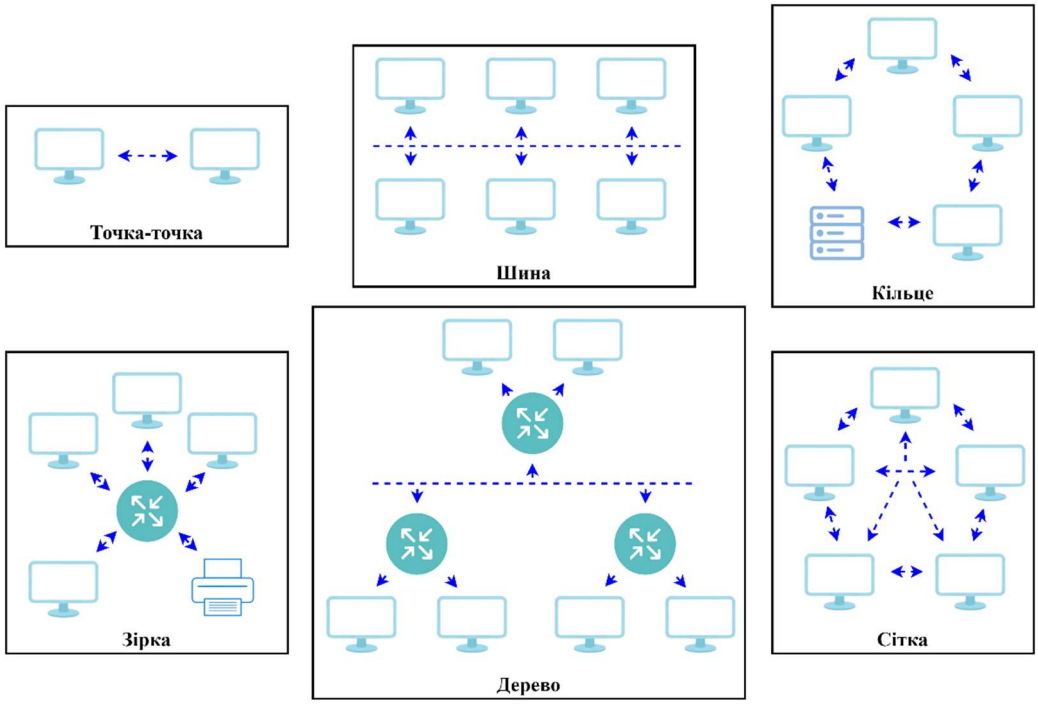


Рисунок 1.2 – Типи топологій локальних мереж

Крім топології, не менш важливим аспектом є протоколи передачі даних, які формалізують правила взаємодії між пристроями та визначають, яким чином інформація передається від одного вузла до іншого. Ці протоколи базуються на моделі OSI або TCP/IP і включають різні рівні взаємодії (див. рис. 1.3).



Рисунок 1.3 – Протоколи моделі OSI

Наприклад, TCP (Transmission Control Protocol) забезпечує надійну доставку даних, контролює порядок слідування пакетів та гарантує їх отримання, тоді як UDP (User Datagram Protocol), будучи менш надійним, не гарантує доставку, але забезпечує високу швидкість, що робить його доцільним для потокових сервісів, таких як відео чи VoIP, де важлива швидкість, а не досконаліть доставки. У випадках веб-доступу, обміну файлами або безпечного підключення використовуються протоколи прикладного рівня — HTTP, HTTPS, FTP, SSH, DNS — які реалізують безпосередню комунікацію користувача з сервісами мережі. У цьому контексті мова йде про провідні або безпровідні канали у рамках локальних комп'ютерних мереж, а не про специфічні безпровідні стандарти, як Wi-Fi або Bluetooth, хоча останні теж можна частково вважати протоколами передачі, проте вони мають окрему специфіку використання [1-2].

Ключові параметри продуктивності мережі. Для повноцінного розуміння ефективності комп'ютерної мережі важливо оцінювати її за кількома ключовими параметрами продуктивності (див. рис. 1.4).

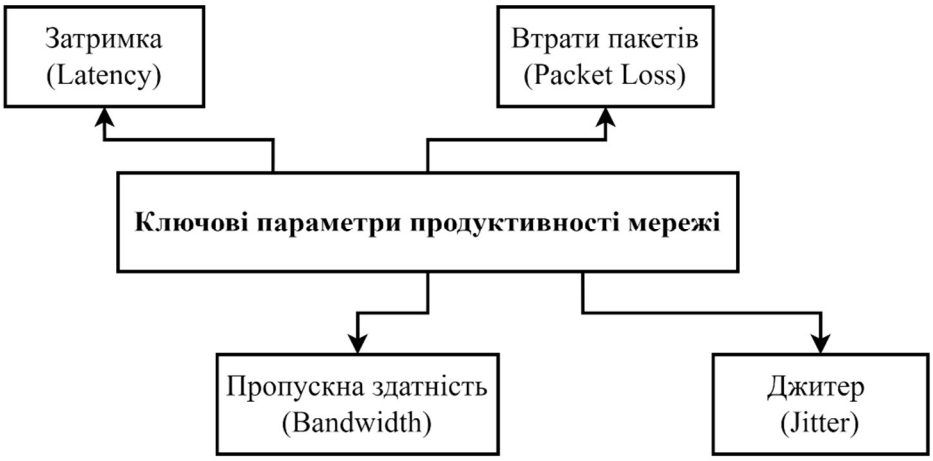


Рисунок 1.4 – Параметри продуктивності мережі

Серед них перше місце займає затримка (latency), тобто час, необхідний для доставки пакету від точки джерела до точки призначення, що включає обробку, маршрутизацію і черги в мережі. У випадках реального часу, таких як онлайн-ігри або відеозв'язок, затримка стає критичною, оскільки навіть незначне її зростання

може негативно впливати на якість взаємодії. Додатковим параметром є пропускна здатність (bandwidth), яка визначає максимальну кількість даних, що може бути передана мережею за певну одиницю часу, наприклад 100 Мбіт/с, і залежить як від конфігурації мережі, так і від характеристик фізичного середовища та навантаження.

Ще одним показником є втрати пакетів (packet loss), коли частина даних не досягає кінцевого вузла через перевантаження буферів або помилки каналу, що може спричинити потребу у повторній передачі, що погіршує загальну продуктивність мережі, призводить до артефактів у відео чи звуку або пошкодження файлів. Менш відомий, але важливий параметр — джитер (jitter), тобто варіація затримки між послідовними пакетами, що є особливо критичним у потокових застосунках, де навіть за стабільної середньої затримки коливання у часі можуть викликати суттєве зниження якості [1-2].

Проблеми масштабування та керування сучасними мережами. Окрім класифікації мереж та визначення принципів їхньої роботи, необхідно також враховувати зростаючі складності, що виникають у процесі масштабування та керування сучасними комп'ютерними системами, адже з часом будь-яка мережа досягає межі своїх можливостей і потребує розширення. Збільшення кількості пристроїв, які підключаються до однієї інфраструктури, є одним із головних факторів, що спричиняють потребу у масштабуванні, зокрема це помітно як у корпоративних комп'ютерних мережах, так і в системах Інтернету речей, де кількість сенсорів, мікроконтролерів та інших пристроїв може досягати тисяч, що створює надмірне навантаження на канали зв'язку та інші ресурси інфраструктури. У таких умовах необхідно забезпечити не лише стабільну роботу, але й підтримку безперебійної адресації, динамічного керування ресурсами та захисту переданої інформації, оскільки без цього мережа починає втрачати функціональність, що проявляється у перебоях зв'язку, збої в маршрутизації та втраті доступу до окремих вузлів.

Зі зростанням числа пристроїв зростає і об'єм передаваного трафіку, що, у свою чергу, вимагає більшої пропускної здатності та вдосконалення механізмів

керування трафіком, зокрема застосування концепцій якості обслуговування, що дозволяє пріоритезувати критичні сервіси. Великі об’єми даних, потокове відео, онлайн-сервіси — усе це накладає додаткові вимоги до фізичного середовища передачі, до потужності обладнання та до здатності мережі масштабуватися динамічно, без зупинки чи перебудови існуючих систем. З огляду на це, виникає необхідність впровадження більш гнучких і стійких рішень, здатних адаптуватися до змін навантаження без втрат у продуктивності чи стабільності [1-2].

Окрім технічних параметрів, варто враховувати і аспект безпеки, адже зі зростанням кількості підключених вузлів підвищується і рівень загрози зі сторони атак, таких як DoS, перехоплення трафіку, сканування вразливостей. Сучасні атаки часто використовують навіть непомітні недоліки в конфігурації для проникнення в систему або порушення її нормальної роботи, тому обов’язковим стає впровадження сегментації, застосування міжмережевих екранів, протоколів шифрування, обмеження доступу за принципами найменших прав, а також моніторинг активності в реальному часі [2].

Ще однією проблемою масштабування є складність ручного адміністрування великої кількості пристроїв та маршрутів. Традиційне керування мережею за допомогою інтерфейсів командного рядка маршрутизаторів або комутаторів є малоефективним при великомасштабних змінах, тому набувають актуальності підходи, пов’язані з автоматизацією — інфраструктура як код (Infrastructure as Code), використання таких засобів як Terraform, Ansible чи NetBox, що дозволяють швидко, повторно та стандартизовано розгортати нові сегменти мережі або змінювати параметри вже існуючих без потреби у ручному втручанні в кожен вузол, що особливо важливо в умовах динамічних змін, які характерні для сучасного IT-середовища, де мережеві ресурси повинні бути доступними миттєво, а конфігурації — відповідати політикам безпеки й продуктивності [2].

Ситуація ускладнюється ще більше при переході до хмарних середовищ, де віртуалізація, розподілені сервіси та оркестрація контейнерів створюють нові виклики до побудови логічних топологій, ізоляції мереж і забезпечення стабільності з’єднань. У таких випадках мережа повинна бути максимально

гнучкою і здатною до швидкого налаштування в залежності від навантаження, типу трафіку або вимог до безпеки. Активне впровадження контейнеризації, інфраструктурних шаблонів, симуляції мережевих структур дозволяє створювати середовища, які в реальному часі відображають реакцію системи на зміну умов, тим самим сприяючи тестуванню, оптимізації і виявленню вузьких місць ще до реального впровадження у виробниче середовище.

Масштабування сучасних комп'ютерних мереж — це не лише питання розширення кількості пристроїв або збільшення трафіку, а комплексний виклик, що охоплює автоматизацію, безпеку, продуктивність, адаптацію до хмарних технологій та здатність мережі змінюватися відповідно до нових вимог без шкоди для основних функцій, що і стає одним з пріоритетних напрямів у сучасному підході до проєктування мережевої інфраструктури.

1.2 Методи моделювання комп'ютерних мереж

Моделювання комп'ютерних мереж є важливим етапом у процесах проєктування, оптимізації та тестування мережевих архітектур без потреби у фізичному розгортанні інфраструктури та реальному обладнанні, дозволяючи імітувати поведінку різних типів мереж, досліджувати нові технології, оцінювати продуктивність, стійкість до збоїв і впроваджувати нові мережеві протоколи у контрольованому, прогнозованому середовищі. У цьому контексті доцільно розглядати основні рівні, підходи та інструменти моделювання, а також зважати на ті практичні обмеження, які притаманні конкретним засобам, особливо коли йдеться про вибір сучасних технологій, зокрема контейнеризованих середовищ, що будуть застосовуватись у цій роботі, адже вони дозволяють сформувати більш гнучке уявлення про сучасні підходи до побудови моделей у комп'ютерних мережах [3-5].

Рівні моделювання – логічне, фізичне, симуляційне. Залежно від мети дослідження, моделювання мереж може проводитись на різних рівнях абстракції, які умовно поділяються на (див. рис. 1.5):

— логічний;

— фізичний;

— симуляційний.



Рисунок 1.5 – Рівні мережевого моделювання

Логічне моделювання орієнтоване на структуру та функціонування мережі без фокусування на фізичних компонентах, охоплюючи визначення логічної топології, схем маршрутизації, віртуальних зв'язків між вузлами, що особливо характерно для архітектур SDN, де правила пересилання трафіку задаються програмно. У той час фізичне моделювання вже враховує реальні характеристики апаратного забезпечення, зокрема типи кабелів, моделі маршрутизаторів, топологію розміщення вузлів та інші елементи фізичної інфраструктури, що робить цей тип придатним для реалізації рішень, близьких до впровадження в реальне середовище. Додатково симуляційне моделювання базується на математичному та програмному описі поведінки мережі в часі, дозволяючи створювати віртуальні вузли, що імітують генерацію трафіку, черги, втрати, затримки та інші параметри, необхідні для тестування роботи мережі в змінних умовах. Ці три підходи можуть комбінуватись для досягнення більш точних і об'єктивних результатів: логічне

задає структуру, симуляційне визначає поведінку, а фізичне – технологічні обмеження [3-4].

Інструменти моделювання – NS-3, GNS3, Cisco Packet Tracer. З метою реалізації описаних підходів до моделювання використовуються спеціалізовані інструменти, які поділяються на:

— симулятори;

— емулятори.

До симуляторів належить, зокрема, NS-3 — потужне академічне рішення для симуляції на рівні пакетів, яке дозволяє моделювати роботу протоколів нижніх рівнів OSI, таких як MAC та PHY, а також реалізовує моделі LTE, Wi-Fi, TCP, UDP та інших мережевих технологій. Завдяки написанню мовами C++ та Python, цей симулятор забезпечує високу гнучкість і точність, дозволяючи реалізовувати моделі з тисячами вузлів, однак характеризується складним налаштуванням і високим порогом входу, що обмежує його використання поза дослідницьким або академічним середовищем [3].

Емулятори, як-от GNS3 (Graphical Network Simulator 3), хоча й носять назву «симулятор», фактично є потужними інструментами для моделювання з використанням реальних образів операційних систем, маршрутизаторів, комутаторів і серверів, дозволяючи створювати лабораторні мережі з реалістичним налаштуванням, інтеграцією з Docker-контейнерами і віртуальними машинами. Такий підхід є актуальним для навчання і підготовки до сертифікацій, як-от Cisco чи Juniper, а графічний інтерфейс забезпечує зручність при розгортанні складних конфігурацій у порівнянні з текстовими сценаріями, які властиві іншим інструментам. Ще один приклад — Cisco Packet Tracer, який є полегшеним симулятором, розробленим з освітньою метою, що дозволяє створювати базові віртуальні мережі з підтримкою основних функцій маршрутизації та комутації. Його зручний інтерфейс і вбудовані навчальні сценарії дозволяють студентам ефективно засвоювати мережеві технології, однак відсутність можливості запуску справжніх операційних систем і обмежений функціонал знижують його

придатність для глибшого дослідження чи використання в умовах, де потрібна висока точність (див. рис. 1.6) [4-5].

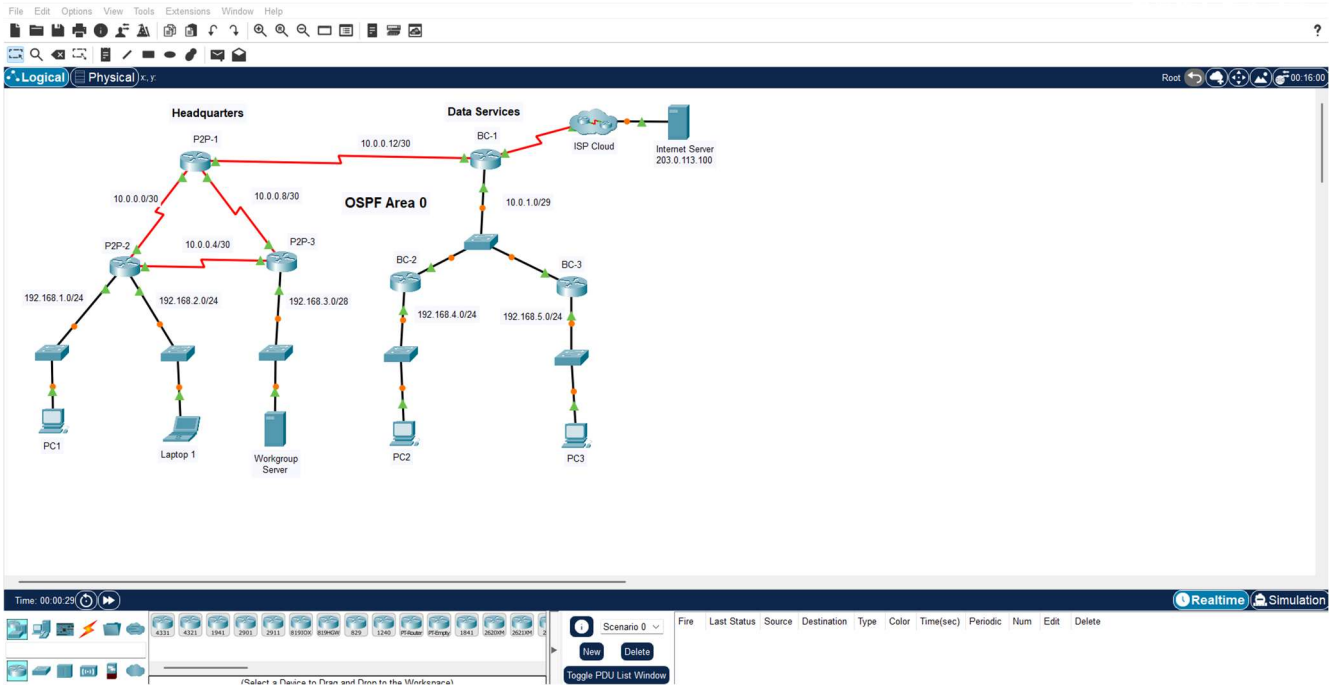


Рисунок 1.6 – Інтерфейс Cisco Packet Tracer

Однак у рамках цієї роботи акцент буде зроблено на сучасніші підходи, що базуються на контейнеризації та реальному середовищі, зокрема використання таких інструментів, як Docker, iperf, Grafana і Prometheus, які хоч і не класифікуються традиційно як симулятори або емулятори, однак дають змогу створювати масштабовані моделі з високим ступенем відповідності до продуктивного середовища. Їх використання в контексті хмарних та контейнеризованих інфраструктур, що стрімко розвиваються, забезпечує гнучкість, швидке розгортання, можливість відтворення складних топологій і аналізу навантаження в реальному часі, що особливо важливо при тестуванні рішень, орієнтованих на сучасні динамічні інфраструктури. Такий підхід дозволяє сформувати програмно-апаратний комплекс для реалістичного моделювання, що відповідає актуальним вимогам до дослідження, тестування та оптимізації комп’ютерних мереж у хмарному середовищі.

Порівняння симуляції з емуляцією. Симуляція та емуляція є двома різними підходами до моделювання комп'ютерних мереж, які хоча й мають схожі цілі, однак реалізуються через різні принципи, що визначає їх застосування у відповідних контекстах. Симуляція зазвичай використовується для дослідження поведінки мереж у гіпотетичних або майбутніх умовах, наприклад, при аналізі нових протоколів або алгоритмів керування трафіком, які ще не мають готової реалізації в обладнанні. У такому разі моделі не відображають фізичні компоненти, а лише імітують функціональну логіку процесів. Натомість емуляція дозволяє запускати реальні програмні компоненти мережевих пристроїв або їх віртуальні образи в умовах максимально наближених до продуктивного середовища, що забезпечує перевірку вже існуючих конфігурацій і рішень перед їх розгортанням. Саме тому часто застосовується комбінований підхід, де спочатку використовуються симуляційні засоби для побудови ідеальної логіки, а після цього емулятори перевіряють її життєздатність у реальному часі з урахуванням технічних обмежень [3-5].

Таблиця 1.1 – Порівняння симуляції з емуляцією

Характеристика	**Симуляція**	**Емуляція**
Реалізм	Спрощена модель	Робота реального ПЗ або ОС
Приклад	NS-3, Packet Tracer	GNS3, EVE-NG, Docker
Мета	Дослідження протоколів, тестування нових алгоритмів	Відтворення умов реальної мережі
Продуктивність	Залежить від складності моделі	Обмежена потужністю хост-машини
Гнучкість	Висока: можна задавати будь-які параметри	Обмежена існуючими прошивками
Точність у часі	Зазвичай поділений час (event-based)	Реальний час

Усе це (див. табл. 1.1) дозволяє краще відслідковувати причинно-наслідкові зв’язки, виявляти потенційні проблеми і коригувати архітектуру ще до того, як вона стане частиною продуктивної мережі.

Обмеження традиційних моделей у реальному середовищі. Разом із тим варто звернути увагу на низку обмежень, притаманних традиційним симуляційним та емуляційним інструментам у практичному застосуванні. Однією з головних проблем є недостатній рівень реалістичності, оскільки більшість симуляторів не враховують такі чинники, як апаратні збої, обмеження буферів, енергоспоживання чи відмовостійкість інтерфейсів, що унеможливлює повноцінне відтворення ситуацій, з якими мережа може стикатися у реальних умовах. Додатково варто враховувати обмеження масштабованості – наприклад, моделювання тисячі вузлів у симуляторах на зразок NS-3 може вимагати значних обчислювальних ресурсів, і в певний момент система починає працювати повільно, що обмежує її застосування у великомасштабних мережевих проєктах. Окремо слід зазначити високий поріг входу, притаманний таким платформам як NS-3 чи Mininet, що потребують глибоких знань у галузі мережевих протоколів, програмування та роботи в Linux-середовищі, чим ускладнюється їх широке впровадження у початкових або навчальних цілях. Крім того, традиційні симулятори та емулятори не завжди мають підтримку найновіших технологій, зокрема нових стандартів мобільного зв’язку на зразок 5G чи інноваційних протоколів маршрутизації, що потребує певного часу на оновлення відповідних моделей або модулів. Навіть такі популярні засоби, як Cisco Packet Tracer, починають лише частково включати новітні рішення, однак не можуть повністю відтворити повноцінне функціонування протоколів, оскільки не підтримують запуск справжніх образів програмного забезпечення. Ще одним суттєвим обмеженням є відсутність тісної інтеграції з процесами CI/CD, що є важливим у сучасному корпоративному середовищі, де автоматизація тестування та розгортання стає ключовим етапом життєвого циклу систем. Через це дедалі частіше зростає популярність контейнеризованих підходів до моделювання, де створення мережевих моделей базується на Docker-контейнерах, а перевірка працездатності здійснюється за допомогою реальних утиліт, як-от "ping" або

"traceroute", з інтеграцією до DevOps-інфраструктури, зокрема через системи моніторингу типу Prometheus і візуалізацію на базі Grafana. Такий підхід дозволяє не лише масштабовано будувати мережі відповідно до потреб, а й оперативно адаптуватися до змін та тестувати варіативність сценаріїв, що виникають у динамічному середовищі сучасних інформаційних систем [3-5].

1.3 Програмно-апаратні засоби віртуалізації мереж

У сучасних комп'ютерних системах, особливо в умовах стрімкого поширення хмарних обчислень, де інфраструктури дедалі частіше будуються з орієнтацією на гнучкість, масштабованість і автоматизацію, віртуалізація мережі набуває особливого значення, оскільки забезпечує можливість абстрагування мережевої взаємодії від фізичного обладнання, що відкриває широкий спектр нових сценаріїв — від створення ізольованих середовищ до масштабування ресурсів, автоматизованого розгортання і тестування у складних багаторівневих мережевих конфігураціях, які постійно змінюються відповідно до нових вимог. З огляду на це виникає потреба в більш глибокому аналізі саме тих програмно-апаратних засобів, що забезпечують таку віртуалізацію і дозволяють не тільки відтворювати функціонування мережі, але й активно оптимізувати її роботу в умовах змінної інфраструктури [1-2, 6-7, 10].

Поняття віртуалізації та контейнеризації. Розглядаючи базові поняття, пов'язані з віртуалізацією та контейнеризацією, слід насамперед зосередитись на тому, що віртуалізація передбачає створення віртуального аналога фізичного ресурсу, такого як сервер, операційна система, мережевий пристрій чи сховище, які працюють як незалежні об'єкти у межах єдиного фізичного середовища. У випадку мережевої інфраструктури це означає створення таких елементів, як віртуальні комутатори, маршрутизатори, сегменти мереж, тунелі, а також інших віртуалізованих мережевих об'єктів, які дають змогу будувати складні, гнучкі

топології без необхідності задіювати фізичне обладнання. Поряд із цим контейнеризація виступає сучаснішою формою віртуалізації, що в низці сценаріїв уже почала витісняти традиційну модель, хоча повністю її не замінює, оскільки обидва підходи залишаються важливими для різних завдань. На відміну від повноцінної віртуальної машини, контейнер не містить власного ядра операційної системи, а використовує ядро хост-системи, маючи при цьому ізольоване середовище з власними бібліотеками, залежностями і конфігураціями, що дозволяє зменшити розмір, пришвидшити запуск і спростити управління [1-2].

Контейнери значно ефективніші з точки зору продуктивності, дозволяють запускати десятки або сотні ізольованих вузлів на одному хості, підтримують вбудовані механізми маршрутизації та фільтрації трафіку, а також забезпечують функціональність одночасної роботи багатьох віртуальних мереж на основі таких технологій, як "bridge", "overlay", "host", "macvlan", що особливо важливо при побудові складних сценаріїв взаємодії у віртуалізованому середовищі. Однією з найбільш значущих переваг у цьому контексті є можливість інтеграції з інструментами автоматизації, зокрема такими як Kubernetes, Ansible, Terraform, що дозволяє досягати високого рівня контролю і повторюваності при розгортанні складних мережевих рішень (див. рис. 1.7) [1-2].

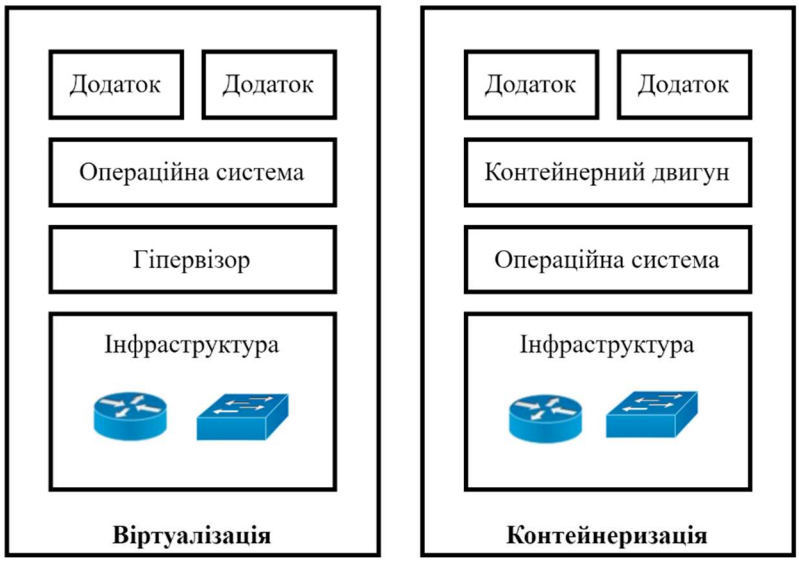


Рисунок 1.7 – Відмінність віртуалізації та контейнеризації

В результаті формується нова парадигма побудови мережевої інфраструктури — Network Function Virtualization (NFV), яка замінює традиційні апаратні засоби, як-от фізичні маршрутизатори чи фаєрволи, на повністю віртуалізовані програмні компоненти, що в поточних умовах дає змогу оперативно адаптувати інфраструктуру до потреб бізнесу, зменшуючи витрати на обслуговування та підвищуючи швидкість впровадження змін [6].

Docker як інструмент створення ізольованих середовищ. У контексті створення ізольованих середовищ важко обійтись без такої широко поширеної платформи, як Docker, що сьогодні є однією з найпопулярніших систем контейнеризації, яка забезпечує повноцінне створення, розгортання та управління контейнерами з високою гнучкістю та ефективністю. Основу Docker становить використання контейнерів, кожен з яких функціонує як легкий, ізольований екземпляр із власною файловою системою, мережею, процесами та середовищем виконання, що дозволяє досягати стабільної роботи незалежно від конфігурації хостової системи [7].

У сфері мережевої інфраструктури Docker підтримує декілька типів вбудованих драйверів, зокрема "Bridge", "Host", "Overlay", "Macvlan", а також дає змогу створювати користувацькі ізольовані мережі з чітко визначеними IP-адресами, підмережами та маршрутизаторами, що значно спрощує побудову складних топологій порівняно з ручним налаштуванням. Вбудована можливість швидкого з’єднання контейнерів у внутрішній мережі через команди на зразок "docker network create" полегшує розгортання сервісно орієнтованих систем [6-7].

Окрім можливостей мережевої взаємодії, Docker також дозволяє керувати публічною і приватною адресацією, інтегрує DNS і "iptables" для підтримки функціональних мережевих конфігурацій, а його архітектура складається з ключових компонентів, таких як Docker Engine, що відповідає за запуск і управління контейнерами, Docker Image як шаблон з усіма конфігураціями та файлами, Docker Container — безпосередньо запущений примірник образу, а також Docker Network — програмно визначена мережа з налаштуванням для кожного контейнера окремо. Завдяки цьому підходу моделювання мереж з використанням

Docker дозволяє створювати віртуальні системи, що імітують повноцінні мережі з серверами, клієнтами, маршрутизаторами, і одночасно тестувати пропускну здатність, затримки, потенційні вразливості, особливо у зв’язці з утилітами типу "ip", "netcat", "tcpdump", Wireshark, що в сукупності відкриває великі можливості для глибокого аналізу й тестування мережевих конфігурацій [6-7].

Контейнери та віртуальні машини в контексті мережевого моделювання. Порівнюючи контейнери та віртуальні машини в контексті моделювання мережі, попри їх спільну здатність до ізоляції процесів і мереж, існують суттєві архітектурні та функціональні відмінності (див. табл. 1.2), які мають критичне значення під час вибору технології для конкретного завдання.

Таблиця 1.2 – Порівняння контейнерів та віртуальних машин

Характеристика	**Контейнери (Docker)**	**Віртуальні машини (VirtualBox, VMware)**
Час запуску	Секунди	Хвилини
Використання ресурсів	Мінімальне	Високе (повноцінна ОС)
Ізоляція	На рівні процесів	Повна ОС і ядро
Мережеві можливості	Гнучкі, вбудовані драйвери	Емуляція апаратного рівня
Масштабованість	Висока	Обмежена ресурсами хоста
Швидкість створення	Миттєва (на основі образів)	Потребує встановлення ОС
Інтеграція з DevOps	Висока (CI/CD, orchestration)	Складна або ручна

Наприклад, у характеристиці часу запуску контейнери на базі Docker забезпечують миттєвий старт, що вимірюється секундами, тоді як запуск віртуальних машин, таких як ті, що працюють на VirtualBox або VMware, триває значно довше — хвилини, через необхідність повного завантаження операційної системи. У плані

споживання ресурсів контейнери мають суттєву перевагу, оскільки працюють на базі ядра хост-системи і використовують лише ті ресурси, що потрібні для виконання окремої програми або сервісу, на відміну від віртуальних машин, які вимагають окремого ядра, драйверів і повного стеку ОС, навіть коли ці компоненти не є критично необхідними [1-2, 7].

Ізоляція в контейнерах реалізується на рівні процесів, що забезпечує швидку та ефективну ізоляцію виконання без дублювання системних компонентів, у той час як у віртуальних машинах вона повна — на рівні ОС, з власним ядром і повною ізоляцією пам’яті, що підвищує безпеку, проте значно збільшує складність і вимоги до ресурсів. У мережевому аспекті Docker забезпечує гнучкі механізми завдяки вбудованим мережевим драйверам і можливостям створення ізольованих віртуальних топологій, тоді як віртуальні машини емулюють мережеві адаптери на апаратному рівні, часто використовуючи спеціалізовані драйвери, такі як Intel PRO/1000 або специфічні VirtualBox-драйвери, що ускладнює реалізацію гнучких сценаріїв і вимагає додаткової конфігурації. Масштабованість у Docker досягається простою і швидкою адаптацією контейнерів до різних конфігурацій середовища без потреби перевстановлення ОС, у той час як віртуальні машини потребують більше часу на розгортання, налаштування і копіювання, навіть якщо використовується система резервного відновлення [1-2, 7].

Особливо важливим чинником при виборі платформи виступає інтеграція з DevOps-процесами, оскільки Docker надає повну підтримку CI/CD, оркестрації, автоматизованого масштабування і управління через інструменти на кшталт Kubernetes, що дозволяє використовувати контейнери як окремі мікросервіси у складі великої автоматизованої системи. Віртуальні машини також можуть бути інтегровані у DevOps-середовище, проте цей процес часто є складнішим і менш гнучким, що обмежує їх використання в сучасних швидкозмінних інфраструктурах. Таким чином, у контексті мережевого моделювання контейнери дають змогу оперативно розгортати тестові сценарії, масштабувати логіку мережі, імітувати атаки, перевіряти поведінку протоколів у динамічних умовах без суттєвого перевантаження ресурсів, тоді як віртуальні машини залишаються

доцільними для глибокої емуляції повноцінного середовища з ОС, повним доступом до драйверів, ядра і низькорівневих процесів, що актуально при моделюванні апаратних компонентів або специфічних умов середовища.

Приклади застосування Docker у мережевих дослідженнях. Розглядаючи Docker у контексті дослідження комп'ютерних мереж, завдяки гнучкості, простоті налаштування та широкому функціоналу, він часто використовується для практичного моделювання мережевих сценаріїв, особливо коли йдеться про перевірку пропускної здатності, тестування безпеки чи загальну оптимізацію інфраструктури. У процесі моделювання мережі для аналізу таких параметрів, як швидкість передавання даних, затримки або втрати пакетів, Docker дозволяє створювати ізольовані середовища з кількома контейнерами, які виконують функції клієнтів і серверів та взаємодіють через віртуальну мережу, сформовану спеціально для експерименту. Наприклад, створення мережі через команду "docker network create testnet", розгортання серверного контейнера "docker run -dit --name server --network testnet ubuntu", встановлення необхідних утиліт та запуск клієнта з аналогічним підключенням дозволяє отримати повноцінну платформу для вимірювання пропускної здатності за допомогою "iperf3", що забезпечує детальне оцінювання характеристик передавання даних у змодельованій мережевій інфраструктурі [6-7].

Наступним прикладом є емуляція атак та тестування систем виявлення та запобігання вторгненням (IDS/IPS), де Docker дозволяє запускати як цільові сервіси (вебсервери, бази даних тощо), так і атакуючі інструменти типу Metasploit чи Nmap, розміщуючи їх в ізольованому середовищі з можливістю підключення відповідних засобів безпеки на кшталт Suricata або Snort. Така конфігурація дає змогу не лише перевірити здатність системи виявляти і блокувати вторгнення, але й забезпечує безпечні умови для дослідження шкідливої активності без загрози для основного середовища.

Ще одним напрямом використання є дослідження ефективності балансування навантаження, що передбачає запуск декількох екземплярів серверів, наприклад, nginx, та генерацію навантаження через такі інструменти як Apache

Benchmark (ab) або WRK. Docker дає змогу моделювати взаємодію з балансувальниками типу HAProxy або Traefik у змінних умовах, дозволяючи фіксувати реакцію системи на зростання навантаження, розподіл запитів, а також перевіряти відмовостійкість заздалегідь визначених компонентів [6-7].

До цього всього додається можливість розгортання моніторингових інструментів на зразок Grafana, Prometheus або Netdata, які в середовищі Docker можуть бути запущені як окремі контейнери, що працюють у реальному часі й забезпечують візуалізацію завантаження ресурсів, аналіз мережевого трафіку, а також спостереження за динамікою змін у всій інфраструктурі. Таке інтегроване середовище дозволяє виявляти вузькі місця, контролювати ефективність налаштувань і загалом приймати обґрунтовані рішення щодо оптимізації архітектури мережі [6-7, 10].

Не менш важливим прикладом застосування Docker є автоматизоване тестування мережевих топологій з використанням Docker Compose, де в YAML-файлах описується конфігурація всієї інфраструктури, що включає вузли, мережі, залежності, і дозволяє запускати складні сценарії одним рядком команди, що значно спрощує проведення повторюваних досліджень, забезпечує узгодженість конфігурацій на різних етапах експерименту та дозволяє проводити масштабовані симуляції з мінімальними витратами часу на розгортання (див. рис. 1.8) [6-7].

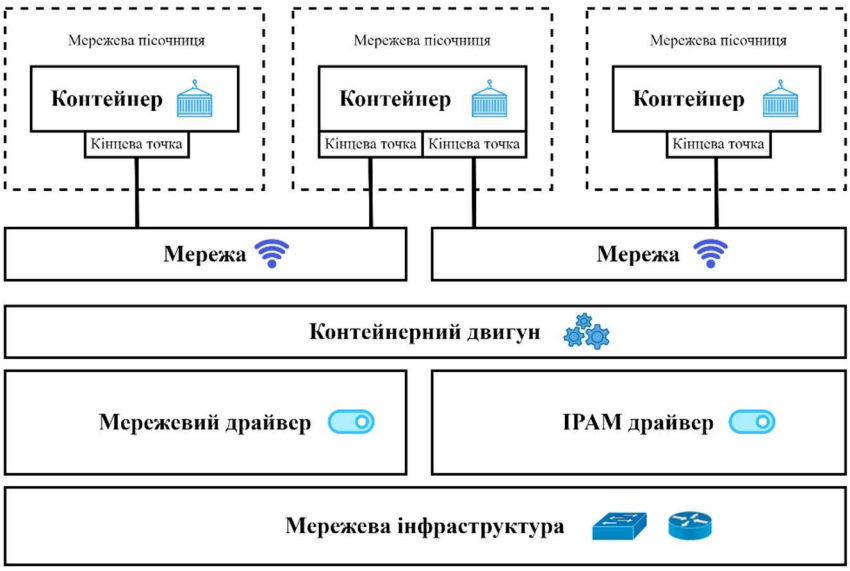


Рисунок 1.8 – Інтеграція Docker у процесі моделювання мереж

У підсумку, застосування технологій контейнеризації на базі Docker у сфері мережевого моделювання відкриває нові можливості для дослідження, тестування та вдосконалення інфраструктури, де контейнерні середовища дозволяють створювати масштабовані, ізольовані, швидко керовані системи з гнучкою інтеграцією до існуючих DevOps-процесів. Порівняно з класичними віртуальними машинами, контейнери мають суттєві переваги в частині швидкості розгортання, споживання ресурсів і можливостей автоматизації, що дає змогу ефективно реалізовувати сценарії від тестування пропускної здатності до перевірки надійності систем безпеки в умовах змінних топологій і навантажень.

1.4 Методи оптимізації роботи комп'ютерних мереж

Оптимізація комп'ютерних мереж є критично важливим аспектом забезпечення їх ефективної, надійної та масштабованої роботи, особливо в умовах зростаючого навантаження, постійного збільшення кількості користувачів і сервісів, а також поширення нових типів додатків і технологій, що потребують стабільної передачі даних і мінімальних затримок. У такому середовищі зростає актуальність впровадження спеціалізованих методів, що стосуються як маршрутизації, так і управління трафіком, балансування ресурсів, виявлення вузьких місць та усунення точок відмови, що у свою чергу дозволяє створити гнучку і стійку архітектуру, здатну адаптуватися до змін в режимі реального часу. Серед найбільш поширених напрямів оптимізації можна виділити впровадження алгоритмів маршрутизації, які забезпечують правильну доставку даних за найефективнішими маршрутами та дозволяють швидко реагувати на зміни в мережевій топології, підтримуючи безперервність сервісів навіть у випадку часткових збоїв або перевантаження окремих вузлів [6-10, 25].

Алгоритми маршрутизації – OSPF, BGP, RIP. У цьому контексті одним із базових рішень є використання протоколів маршрутизації, які реалізують

алгоритми прийняття рішень для вибору шляху доставки даних між джерелом і призначенням. Наприклад, протокол RIP (Routing Information Protocol), що відноситься до класу дистанційно-векторних, ґрунтується на принципі вибору маршруту з найменшою кількістю переходів (hop count), і завдяки простоті реалізації та сумісності з багатьма пристроями часто використовується в простих мережах, хоча його обмеження у масштабованості та максимальна глибина маршруту у 15 хопів, а також повільна конвергенція роблять його менш придатним для складних або динамічних інфраструктур. Натомість більш досконалим є протокол OSPF (Open Shortest Path First), який належить до внутрішньої динамічної маршрутизації, базується на алгоритмі Дейкстри, і використовує топологічну карту мережі для обчислення найкоротших маршрутів на основі метрик ваги. Такий підхід дозволяє досягти високої швидкості конвергенції, ефективного розподілу навантаження і масштабованості, що робить його особливо ефективним у великих корпоративних інфраструктурах та дата-центрах, де важливо забезпечити стабільну маршрутизацію з урахуванням розподілених вузлів і сегментів (див. рис. 1.9) [8].



Рисунок 1.9 – Алгоритм роботи маршрутизації OSPF

Поряд із внутрішніми протоколами значну роль у взаємодії між автономними системами відіграє BGP (Border Gateway Protocol), що реалізує міждоменну маршрутизацію і ґрунтується на прийнятті рішень, виходячи з політик, префіксів та атрибутів маршруту. Його головною перевагою є гнучкість, контроль над маршрутами і висока стабільність, хоча це супроводжується складністю налаштування і повільною реакцією на зміни в мережі, що у деяких випадках може

вимагати додаткового втручання для забезпечення швидкого відновлення маршрутів. У практиці побудови складних інфраструктур часто використовується комбінований підхід, коли OSPF застосовується всередині автономної системи, забезпечуючи швидке і надійне маршрутизування трафіку в межах організації, а BGP використовується на кордоні для взаємодії з провайдерами, іншими автономними системами або зовнішніми партнерами, що дозволяє досягти максимальної гнучкості та адаптивності на різних рівнях мережевої взаємодії [8-9].

Оптимізація мереж за рахунок правильного вибору та налаштування алгоритмів маршрутизації є одним із фундаментальних методів удосконалення її функціонування, що дозволяє досягати високих показників продуктивності, стійкості до збоїв і ефективного використання наявних ресурсів, адаптуючи мережеву інфраструктуру до сучасних вимог цифрових сервісів.

QoS та механізми пріоритезації трафіку. Оптимізація роботи комп'ютерних мереж в умовах високої динаміки навантаження, зростання кількості користувачів та збільшення обсягів передаваних даних забезпечується застосуванням комплексних підходів, що включають як управління трафіком, так і виявлення та усунення вузьких місць. Методи QoS забезпечують управління трафіком із гарантіями доставки важливих даних шляхом реалізації набору технологій, які дозволяють пріоритезувати потоки залежно від типу трафіку, наприклад, голосових дзвінків, даних чи відеоконференцій, що особливо важливо у мережах реального часу, де навіть невеликі затримки можуть впливати на якість обслуговування. Класифікація пакетів здійснюється на основі різних критеріїв за допомогою механізмів, що дозволяють задавати пріоритети відповідно до класу сервісу, а також встановлювати політику черг, завдяки якій трафік із високим пріоритетом обслуговується швидше, що досягається за допомогою таких методів, як фільтрація трафіку, шейпінг і диференційоване обслуговування, що використовує спеціальні мітки у заголовках IP-пакетів для класифікації та подальшої маршрутизації відповідно до встановлених політик (див. рис. 1.10) [6].

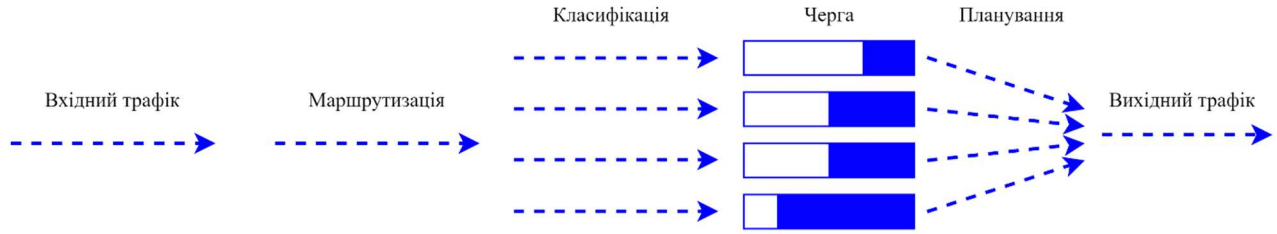


Рисунок 1.10 – Схема QoS-пріоритезації трафіку

У системах, що реалізують QoS, пріоритезація трафіку дозволяє забезпечити якісне обслуговування важливих сервісів, наприклад голосового зв’язку по протоколу SIP, що забезпечує більш стабільні дзвінки навіть під час високого навантаження, завдяки чому відбувається розподіл ресурсів у мережі таким чином, що критично важливі дані отримують пріоритетний доступ, а менш важливий трафік обмежується за допомогою методів Traffic Shaping, що сприяє зменшенню затримок і втрат пакетів. Фільтрація трафіку здійснюється за допомогою налаштування правил на маршрутизаторах і комутаторах, що дозволяє впровадити схеми, де певні типи даних передаються швидше, тоді як інші — обмежуються для збереження належного рівня продуктивності мережі, що дозволяє ефективно реагувати на перевантаження та інші зміни у мережевій інфраструктурі.

Засоби виявлення «вузьких місць» і балансування навантаження. Засоби виявлення вузьких місць та балансування навантаження є невід’ємною частиною оптимізації мереж, оскільки мережеві вузли можуть стати критичними ланками, що обмежують загальну пропускну здатність або викликають затримки у передачі даних, що негативно впливає на стабільність сервісу. Інструменти моніторингу, такі як Prometheus та Grafana, дозволяють збирати метрики щодо завантаження CPU, мережевого трафіку, помилок і затримок, а детальний аналіз трафіку за допомогою утиліт типу "ping", "traceroute" або Wireshark дозволяє визначити місця, де виникають колізії або втрати даних, що є першочерговим завданням при виявленні вузьких місць. Крім того, оптимізація мережевої інфраструктури може включати застосування методів оптимізації таблиць маршрутизації, агрегацію лінків (LACP) для збільшення пропускної здатності, а також впровадження

балансувальників навантаження, які розподіляють трафік між кількома вузлами для забезпечення рівномірного навантаження, що дозволяє уникнути перевантаження окремих компонентів. Балансування на різних рівнях, як-от Layer 4 за TCP/UDP і Layer 7 за протоколом HTTP/S, дозволяє адаптувати розподіл запитів до конкретних умов, що робить систему більш стійкою до пікових навантажень, а використання технологій DNS-based балансування дозволяє враховувати регіональні особливості та розподіляти трафік згідно з навантаженням у різних сегментах мережі (див. рис. 1.11) [7, 10].

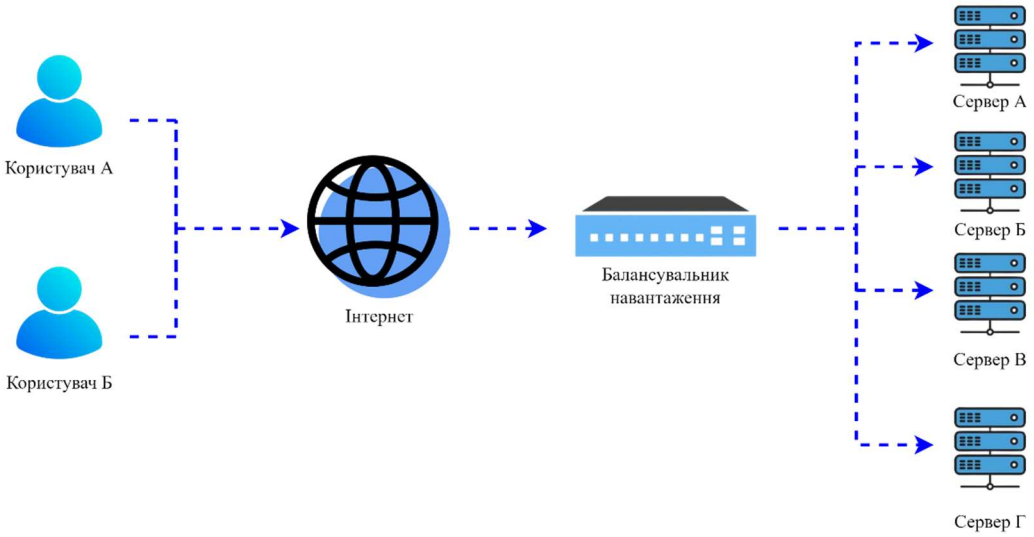


Рисунок 1.11 – Балансування навантаження на маршрутизаторі

Методи оптимізації роботи комп’ютерних мереж підвищують ефективність роботи систем за рахунок управління як параметрами передачі даних, так і шляхами маршрутизації, що дозволяє забезпечити стабільну роботу мережевої інфраструктури навіть у випадку значного зростання навантаження або кількості користувачів. За допомогою впровадження алгоритмів, що керують пріоритетами пакетів, налаштовують черги обробки, здійснюють фільтрацію і шейпінг трафіку, системи здатні адаптуватись до змін умов експлуатації, забезпечувати мінімальні затримки при передачі даних і уникати втрат, що критично для онлайн-сервісів, відеоконференцій або інших застосувань, де важлива негайна реакція мережі. Додатково можливість впровадження балансування навантаження дозволяє

розподілити трафік між серверними вузлами таким чином, щоб стабілізувати пропускну здатність всієї мережі, що особливо актуально у корпоративних середовищах з високою кількістю одночасних запитів.

Підхід до оптимізації на рівні контейнерних мереж. Мережі, побудовані на контейнеризаційних технологіях, таких як Docker чи Kubernetes, стикаються з низкою специфічних викликів, серед яких динамічне зростання кількості вузлів, потреба в масштабуванні сервісів та ускладнення трасування трафіку, і саме ці фактори визначають необхідність застосування окремих підходів до оптимізації на рівні контейнерних мереж. Одним із базових рішень у такому контексті є впровадження віртуальних мереж, які реалізуються через функціональність самого Docker із використанням режимів "bridge", "overlay", "host" або "macvlan", що дозволяє організувати ізольований та контрольований обмін даними між контейнерами. У випадку з Kubernetes взаємодія між подами часто забезпечується за допомогою CNI-плагінів, зокрема таких як Calico, Flannel, Weave або Cilium, що дозволяє досягти більшої масштабованості, прозорого маршрутування і дотримання політик мережевої безпеки [7, 25].

Для подальшої оптимізації застосовується підхід на рівні сервісної сітки, де компоненти на кшталт Istio, Linkerd чи Traefik виконують функції маршрутизації, балансування навантаження, а також забезпечують додаткові шари безпеки та спостереження за мережею. Такий підхід дозволяє абстрагувати мережеву взаємодію між сервісами, централізовано визначати правила маршрутизації, здійснювати шифрування та автентифікацію трафіку, а також збирати метрики для подальшого аналізу. Використання проксі-серверів, таких як Envoy чи HAProxy, дозволяє забезпечити деталізовану маршрутизацію запитів та контроль над мережею на рівні L7, а інтеграція з такими інструментами як eBPF, NetFlow або тим самим Cilium відкриває можливості для глибокого профілювання трафіку, виявлення аномалій і аналізу поведінкових шаблонів у мережевому середовищі [7].

Оптимізація контейнерних мереж дозволяє реалізовувати автоматичне масштабування компонентів, зменшувати затримки завдяки локалізованій маршрутизації всередині вузлів, налаштовувати політики безпеки з високим

ступенем гнучкості за допомогою Network Policy, а також реалізовувати прозору інтеграцію з системами моніторингу та логування, що спрощує контроль за станом системи та пришвидшує виявлення проблем. У практичному прикладі система, побудована на Docker, може використовувати Docker Compose для розгортання кількох взаємопов'язаних сервісів з балансуванням запитів між ними, а Prometheus у такій конфігурації виступає інструментом збору даних і аналітики, що дозволяє в реальному часі спостерігати вплив змін конфігурації на продуктивність і виявляти потенційні вузькі місця (див. рис. 1.12) [7, 25].

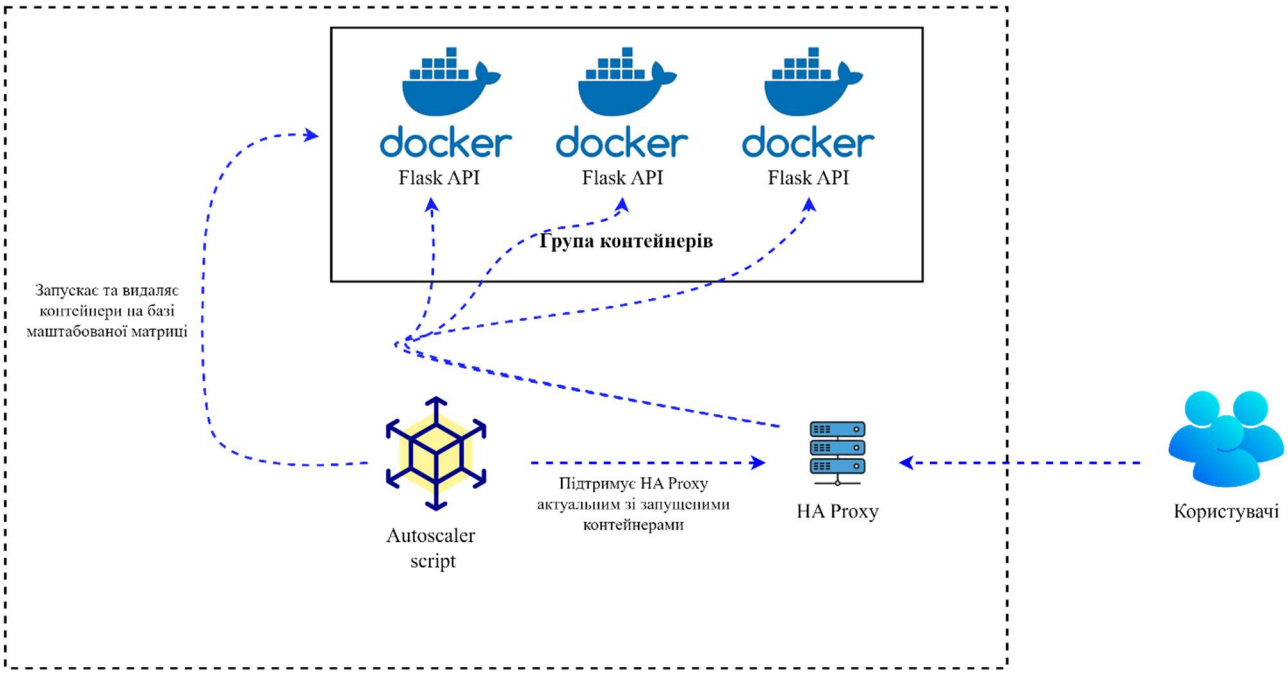


Рисунок 1.12 – Оптимізація мережі через контейнерне масштабування

У підсумку, підходи до оптимізації мережі на рівні контейнеризації стають усе більш актуальними, оскільки класичні протоколи поступово доповнюються більш гнучкими сучасними інструментами, що орієнтовані на автоматизацію, масштабованість і підвищену спостережуваність. У сучасному середовищі, де сервіси постійно змінюють свій стан, масштаби і кількість викликів, ефективність мережевої взаємодії залежить не лише від базової архітектури, а й від здатності системи адаптуватися до змін через інтеграцію з сервісними сітками, застосуванням QoS, балансуванням навантаження та аналітикою в реальному часі.

Усе це формує цілісну та комплексну стратегію підтримки стабільної і ефективної інфраструктури, яка здатна залишатися стійкою до зовнішніх викликів і внутрішніх змін, що особливо важливо в умовах постійного зростання складності розгортання сучасних розподілених систем.

1.5 Засоби моніторингу стану комп'ютерних мереж

У сучасних умовах існування комп'ютерних мереж, коли мережеві технології інтегруються у всі сфери діяльності — від бізнесу до критичної інфраструктури — моніторинг стану комп'ютерних мереж вже давно перестав бути допоміжним інструментом і трансформувався в одну з основних функцій, необхідних для забезпечення стабільної, безпечної та продуктивної роботи інформаційних систем. У ситуаціях, де навіть короткочасне порушення зв'язку чи втрата продуктивності може спричинити фінансові або операційні збитки, своєчасне виявлення аномалій, збір телеметричних даних, аналіз ключових показників продуктивності та їх візуалізація стають необхідним мінімумом для ефективної підтримки інфраструктури, що базується на мережевій взаємодії. Саме тому засоби моніторингу, орієнтовані на аналіз трафіку, стану обладнання, поведінки сервісів та взаємодії між ними, становлять основу для будь-якої системи підтримки як реальної, так і віртуалізованої мережі, включно із системами моделювання, які дозволяють протестувати різні конфігурації до їх реального впровадження, а також дослідити ефективність оптимізаційних заходів у симульованому середовищі [3-7, 10].

Основи збору метрик – SNMP, NetFlow, телеметрія. Моніторинг мережі, як процес, передбачає безперервне отримання показників про стан як самого обладнання, так і мережевого трафіку, затримок, навантаження на канали зв'язку та інші параметри, що визначають загальний стан функціонування інфраструктури. Для того щоб цей процес був ефективним і масштабованим, використовуються

різноманітні протоколи та технології збору метрик, з яких одними з найпоширеніших є:

— SNMP;

— NetFlow;

— IPFIX;

— sFlow;

— телеметрія.

Протокол SNMP (Simple Network Management Protocol), будучи одним із перших загальноприйнятих стандартів, забезпечує механізм обміну інформацією між керуючим елементом мережі — менеджером — і агентами, встановленими на мережевих пристроях, таких як комутатори, маршрутизатори або сервери. Агент SNMP може надавати дані про завантаження інтерфейсів, стан портів, кількість помилок або втрат пакетів і надсилати ці значення SNMP-менеджеру через стандартні запити (Get, GetNext) або події у реальному часі (Trap). Попри простоту впровадження, широку підтримку серед обладнання та мінімальне ресурсне навантаження, SNMP у версіях 1 та 2 має суттєві обмеження у сфері безпеки та не дозволяє реалізувати детальну аналітику, необхідну для глибшого аналізу сучасних мереж (див. рис. 1.13) [10].

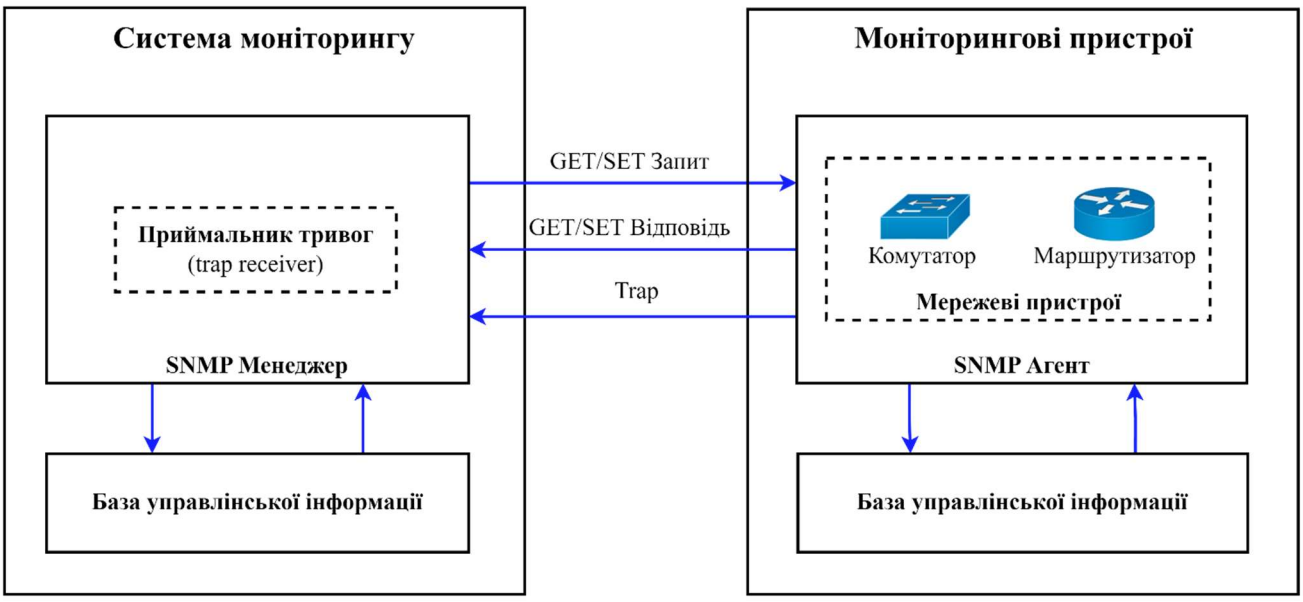


Рисунок 1.13 – Модель збору метрик SNMP

Альтернативою або доповненням до SNMP є протоколи потокового аналізу, серед яких NetFlow (розроблений Cisco) та його аналоги IPFIX і sFlow, що дозволяють збирати інформацію не просто про стан пристроїв, а про характеристики мережевого трафіку. Маршрутизатори та комутатори з підтримкою цих технологій здатні формувати записи про мережеві потоки, зокрема джерело та призначення IP-адрес, номери портів, обсяги переданих даних, тривалість з'єднання, після чого ці дані передаються до систем аналітики. У результаті формується цілісна картина того, який трафік домінує в мережі, які напрямки споживають найбільше ресурсів, що дозволяє виявляти аномалії, зокрема DoS-атаки або внутрішній зловмисний трафік. Серед переваг цього підходу — висока деталізація та аналітична глибина, однак при великій кількості мережевих потоків зростає навантаження на обладнання, що може потребувати попередньої оптимізації конфігурацій або використання потужніших пристроїв [10].

Сучасні підходи до телеметрії, такі як Streaming Telemetry, базуються на ідеї безперервного надсилання структурованих даних про стан мережевого обладнання в реальному часі за допомогою протоколів gRPC або HTTP/2 у форматах JSON, GPB тощо. Така форма телеметрії дозволяє зменшити затримки, забезпечити високу масштабованість та інтеграцію з хмарними платформами аналітики. На відміну від SNMP або NetFlow, які передбачають періодичне опитування або обробку агрегованих даних, стрімінгова телеметрія працює у режимі постійного потоку, що особливо важливо в умовах змінного навантаження або потреби швидкої реакції на інциденти. Водночас такий підхід передбачає наявність новішого мережевого обладнання, а також складніше впровадження, зокрема у частині узгодження форматів даних і налаштування відповідних приймальних сервісів [10].

Незалежно від обраного підходу або протоколу, всі ці інструменти утворюють технічний фундамент для збору інформації, яка далі надходить до систем моніторингу, таких як Prometheus, Zabbix, Grafana, PRTG, Datadog або інші платформи, здатні не лише відображати стан мережі, але й автоматично реагувати на аномалії, генерувати сповіщення, будувати динамічні дашборди та

забезпечувати можливість ретроспективного аналізу подій. Таким чином, ефективний моніторинг забезпечує не лише підтримку працездатності мережі, але й слугує основою для прийняття рішень щодо її оптимізації, масштабування або перебудови у відповідь на зміну навантаження, архітектури або бізнес-вимог.

Огляд інструментів моніторингу – Prometheus, Grafana, Zabbix, Nagios. У сучасних умовах різноманітні інструменти моніторингу комп'ютерних мереж застосовуються залежно від:

— потреб;
— варіантів розгортання;
— вимог до інтеграції з інфраструктурними компонентами;
— рівня деталізації та візуалізації даних.

Частина з них призначена лише для базового спостереження, інші ж дозволяють гнучко налаштовувати системи оповіщення, створювати складні логіки аналізу й інтегруватися з хмарними або контейнеризованими платформами. Серед найбільш уживаних варто розглянути Prometheus, який є інструментом збору часових рядів з відкритим вихідним кодом, архітектура якого базується на pull-моделі, що дозволяє йому періодично витягувати метрики з визначених таргетів. Завдяки мові запитів PromQL, широкому переліку експортерів на кшталт "node_exporter" чи "blackbox_exporter", а також легкій інтеграції з контейнеризованими середовищами, Prometheus вирізняється високою масштабованістю та продуктивністю. Водночас, обмежена підтримка SNMP та потокового аналізу трафіку створює певні труднощі для розширеного мережевого аналізу без застосування додаткових рішень (див. рис. 1.14) [10].

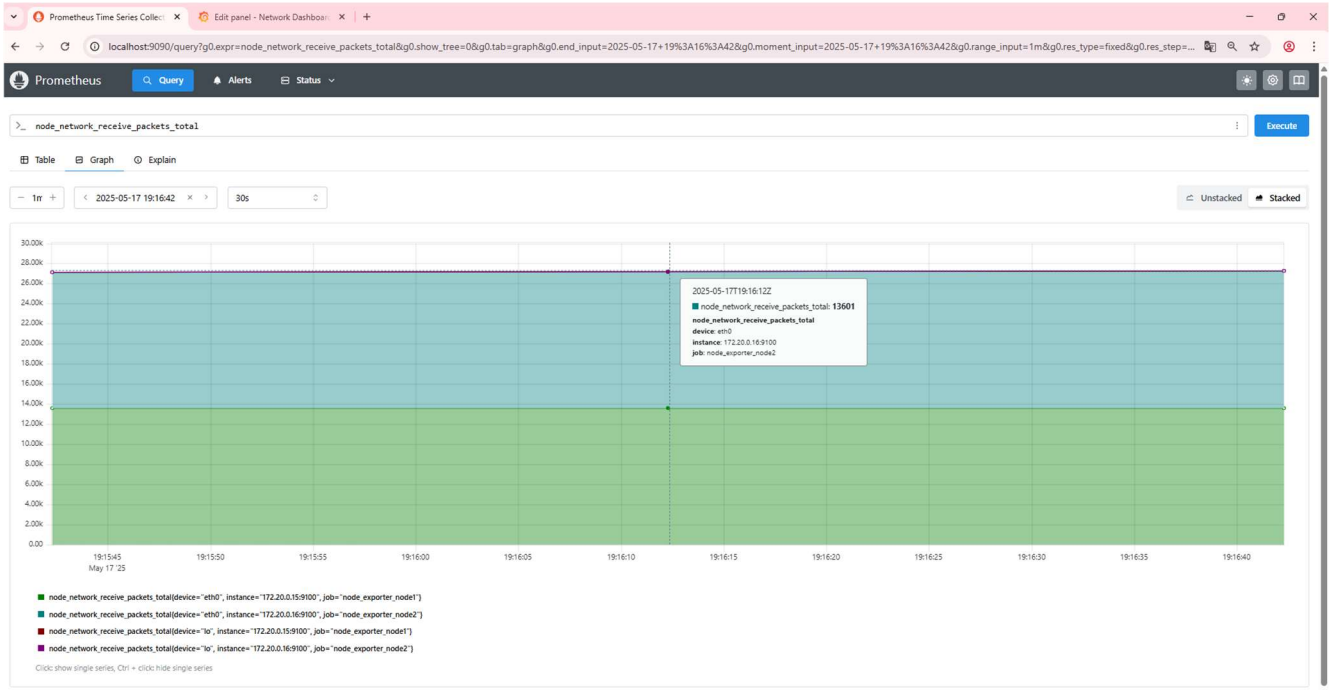


Рисунок 1.14 – Система аналізу метрик Prometheus

Для візуалізації зібраних метрик зазвичай використовується Grafana – платформа для побудови інтерактивних дашбордів, яка дозволяє поєднувати дані з численних джерел, зокрема Prometheus, InfluxDB, Elasticsearch та інші. Вона підтримує налаштування алертів, роботу з шаблонами, інтерактивні графіки, таблиці, карти та віджети, що дозволяє швидко створити зручне візуальне представлення метрик із гнучкою адаптацією під конкретні сценарії. Grafana особливо ефективна при створенні уніфікованих панелей моніторингу для аналітики системної продуктивності, навантаження інтерфейсів або пошуку аномалій у мережевому трафіку [10].

Іншим представником систем моніторингу є Zabbix – рішення з агентською архітектурою, яке забезпечує активний моніторинг, автодетекцію пристроїв, підтримку SNMP, а також розвинуту систему сповіщень і звітності. Платформа здатна працювати як із серверними інфраструктурами, так і з мережею загалом, проте конфігурація системи є складнішою, а взаємодія з динамічними середовищами на кшталт Kubernetes менш зручна порівняно з більш контейнерно-орієнтованими рішеннями. Ще одним класичним засобом моніторингу вважається Nagios, що відзначається широким використанням плагінів для перевірки стану

сервісів, дисків, ресурсів. Його основні переваги полягають у стабільності, гнучкості та підтримці великої спільноти, однак масштабування та робота з потоковими метриками у реальному часі є дещо обмеженими [10].

Таблиця 1.3 – Порівняння інструментів моніторингу

Параметр	**Prometheus**	**Grafana**	**Zabbix**	**Nagios**
Збір метрик	Так	Ні	Так	Частково
Візуалізація	Через Grafana	Так	Так	Через плагіни
Алерти	Так	Так	Так	Так
Підтримка SNMP	Через експортер	Ні	Вбудована	Через плагіни
Орієнтація	Контейнери, хмари	Аналітика	Класика	Сервери

Узагальнене порівняння цих систем (див. табл. 1.3) свідчить про те, що Prometheus забезпечує активний збір метрик, але для візуалізації потребує інтеграції з Grafana, яка надає зручний інтерфейс для виведення графіків, алертів і комбінування джерел даних. Zabbix, у свою чергу, реалізує функції збору й візуалізації всередині однієї системи з розвинутими SNMP-можливостями, а Nagios здатний частково виконувати функції збору через плагіни та є доречним у класичних серверних розгортаннях. Таким чином, у рамках даної роботи, зосередження на зв’язці Prometheus і Grafana є доцільним, оскільки обидва продукти мають відкритий код, не потребують додаткових витрат і дозволяють гнучко реалізувати необхідні функції для моделювання та оптимізації мережевої інфраструктури.

Візуалізація даних про стан мережі. Окрему увагу необхідно приділити візуалізації стану мережі, адже вона є невід’ємною складовою ефективного моніторингу, дозволяючи не тільки спостерігати за поточним навантаженням або доступністю ресурсів, а саме

— виконувати ретроспективний аналіз;

— виявляти тренди та аномалії;

— формувати звітність.

Серед основних типів візуалізації можна виокремити графіки часових рядів, які ілюструють динаміку зміни затримок, пропускної здатності, навантаження, а також таблиці станів, що відображають статус вузлів і сервісів. Додатково застосовуються теплові карти для аналізу активності за напрямами трафіку або портами й навіть географічні карти для розподілених мереж. Усі ці типи зручно реалізуються в Grafana, яка дозволяє створювати дашборди з такими метриками, як використання ЦП, оперативної пам’яті, пропускна здатність інтерфейсів, зміни рівня трафіку, виявлення пікових навантажень або відсутності відповідей, що надає змогу оперативно локалізувати проблемні вузли або сегменти мережі та швидко реагувати на інциденти (див. рис. 1.15).

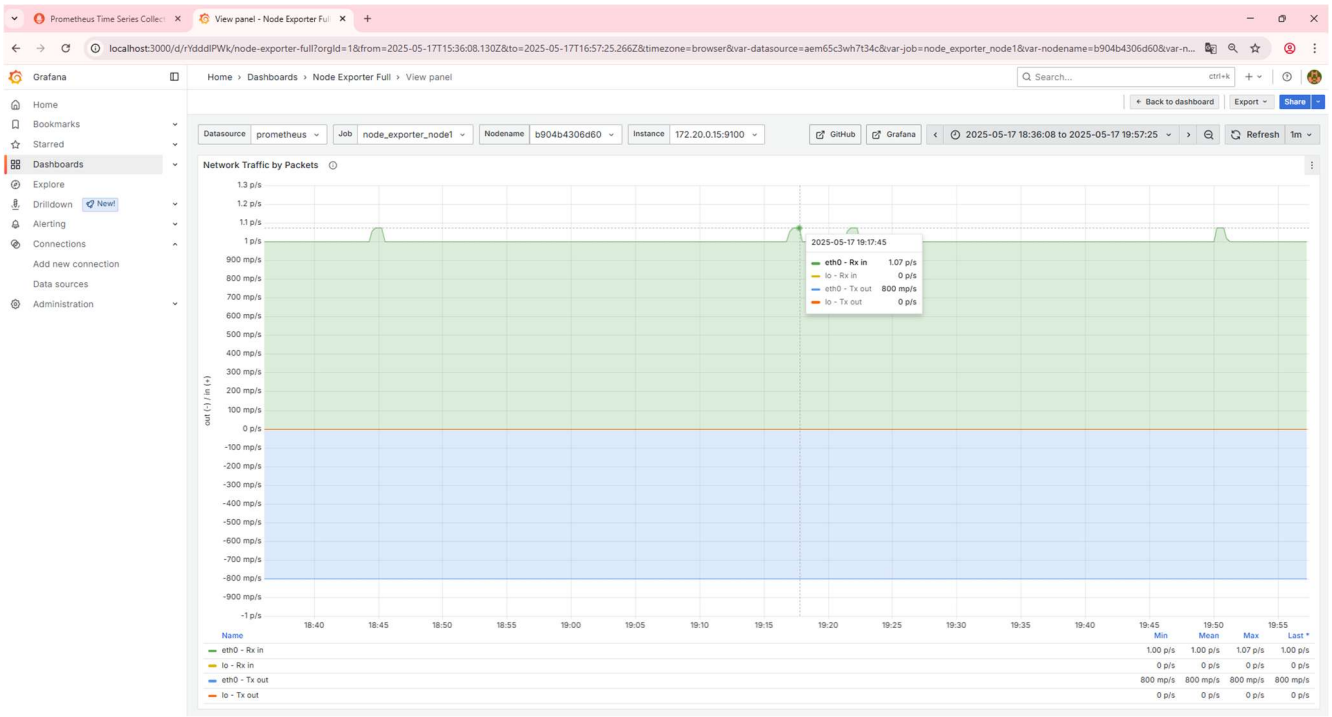


Рисунок 1.15 – Візуалізація зібраних метрик у Grafana

Саме така взаємодія між системами збору та візуалізації метрик у поєднанні з ретельно продуманими інтерфейсами дозволяє забезпечити ефективний контроль за стабільністю, безпекою та продуктивністю мережевих інфраструктур [10].

Переваги інтеграції моніторингу у моделювальні системи. Інтеграція засобів моніторингу у моделювальні середовища, зокрема за допомогою інструментів Docker, "iperf", системи збору метрик Prometheus та платформи для візуалізації

Grafana, дозволяє сформувати цілісне уявлення про поведінку комп'ютерної мережі в експериментальних умовах. Такий підхід відкриває можливість детального аналізу результатів, зокрема щодо впливу змін конфігурацій або навантаження на затримки, пропускну здатність, втрати пакетів чи адаптивність сервісів у змінних мережевих умовах, при цьому забезпечуючи об'єктивну оцінку ефективності прийнятих рішень. Зокрема, можна простежити, як впровадження QoS або зміна топології впливають на якість обслуговування, а також оперативно виявляти помилки або неочевидні конфігураційні конфлікти (див. рис. 1.16) [6-7, 10].

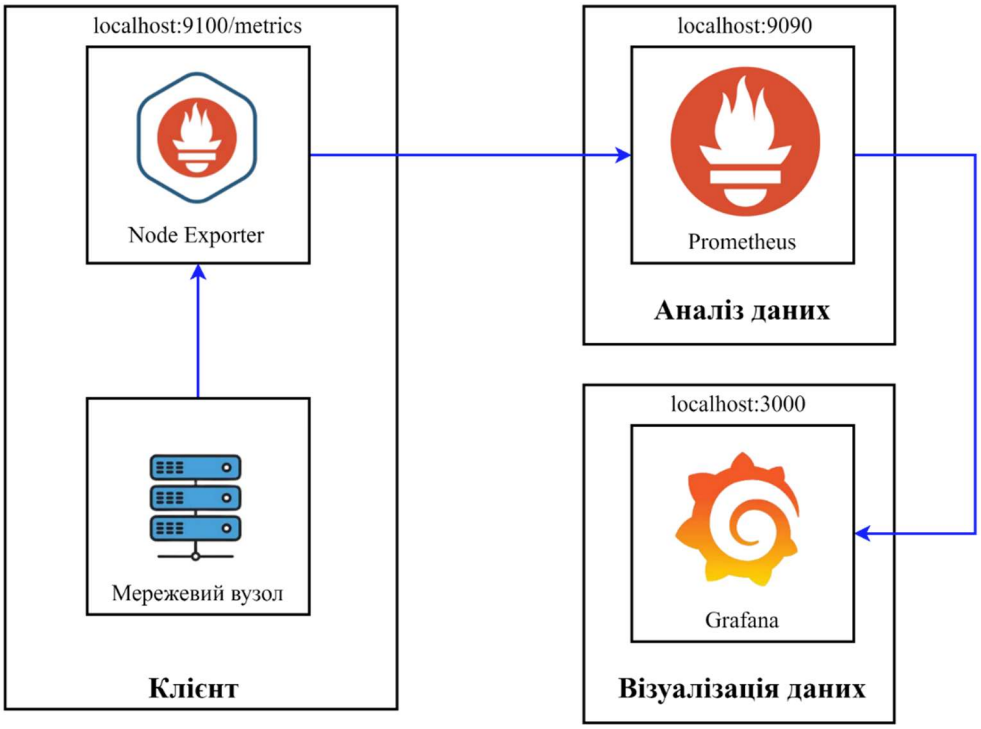


Рисунок 1.16 – Інтеграція моніторингу у моделювальне середовище

Використання автоматизованого збору даних у таких системах дозволяє створювати шаблони експериментів, в межах яких усі метрики, пов'язані з мережею та навантаженням, збиратимуться без потреби ручного втручання, що підвищує точність та оперативність аналізу. Крім того, така автоматизація дає змогу досягнути високої відтворюваності експериментів, що є важливим у дослідженнях: усі вхідні параметри, метрики та події можуть бути зафіксовані і

використані повторно, що робить можливим відтворення ідентичних сценаріїв з метою перевірки або валідації результатів. Підсумкові дані, отримані у процесі моделювання, зокрема метрики часу відгуку, кількості втрат пакетів чи рівня використання ресурсів, можуть бути зручно виведені у вигляді графіків, таблиць або інших структурованих форм за допомогою Grafana, що значно спрощує підготовку звітів, презентацій та подальшу аналітику.

Наприклад, під час моделювання сценарію з обмеженням пропускної здатності в середовищі Docker через засіб "traffic shaping", за участі Prometheus та Grafana можна отримати динамічну картину змін продуктивності: як сервіси реагують на зміни, чи встигають адаптуватися, чи спостерігається втрата якості з точки зору затримок і стабільності з’єднання. Такий підхід надає змогу не просто проводити тестування, а робити його системним та керованим, формуючи середовище максимально наближене до реального функціонування мережевої інфраструктури [6-7, 10].

Узагальнюючи, сучасні комп’ютерні мережі вимагають цілісного, багаторівневого підходу до моніторингу, що поєднує класичні протоколи збору метрик на кшталт SNMP та NetFlow із новітніми засобами телеметрії і візуалізації. Включення таких рішень, як Prometheus, Grafana, Zabbix у структуру моделювального середовища дозволяє не лише відстежувати поточний стан мережі, а й здійснювати глибоку аналітику, ідентифікувати слабкі місця, адаптувати конфігурацію під специфіку навантаження та перевіряти коректність реалізованих змін. Таким чином, інтеграція моніторингу в моделювання відкриває новий рівень у розумінні поведінки мережевої інфраструктури, що дозволяє як вдосконалювати її роботу, так і забезпечити достовірну перевірку результатів досліджень.

РОЗДІЛ 2 ПОБУДОВА ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ ДЛЯ МОДЕЛЮВАННЯ

2.1 Архітектура запропонованої системи

У контексті побудови стійкої системи для моделювання комп'ютерних мереж необхідно реалізувати архітектуру програмно-апаратного комплексу, що забезпечує не лише створення віртуальних вузлів, генерацію трафіку та збір метрик, а й надає змогу подальшого аналізу продуктивності та виявлення нестабільностей або вузьких місць у мережевій інфраструктурі. Загальною метою такої системи є створення гнучкого, масштабованого середовища для симулювання мережевих сценаріїв із різними конфігураціями та параметрами навантаження, де всі компоненти будуть ізольовані, проте взаємодіятимуть через логічну мережу, забезпечуючи [11]:

— повноцінний цикл тестування;

— моніторингу;

— візуалізації.

Опис загальної схеми – вузли, моніторинг, тестування. Загальна архітектура будується на основі контейнеризації в межах ізольованого середовища, що реалізується за допомогою Docker Desktop, встановленого на операційній системі Windows 11 або іншій сумісній платформі. Як базовий блок виступає Docker Engine, через який розгортаються всі необхідні сервіси. Основними елементами системи є два віртуальні вузли Node1 та Node2, реалізовані як окремі контейнери на базі надлегкого дистрибутиву Linux Alpine, що завдяки своїй малій вазі забезпечує швидке розгортання та мінімальне використання ресурсів. Ці вузли імітують комп'ютери або мережеві пристрої в мережі й є основними учасниками трафікогенерації та вимірювання параметрів пропускної здатності.

Далі у структурі знаходяться контейнери з інструментами для навантаження мережі — iperf-клієнт і iperf-сервер, які виконують генерацію контрольованого мережевого трафіку для аналізу таких показників, як затримка, втрати пакетів і загальна продуктивність. З іншого боку реалізується моніторингова частина, де встановлюються агенти Node Exporter на кожному з вузлів, що збирають системні метрики, включаючи:

— використання CPU;

— оперативної пам’яті;

— мережевого інтерфейсу.

Ці дані передаються до централізованого контейнера з Prometheus, що виступає сервером збору метрик і зчитує інформацію з експортерів через HTTP-запити відповідно до конфігурацій [11].

З метою візуалізації даних використовується Grafana, яка встановлена в окремому контейнері та підключається до Prometheus як джерела даних. Побудова дашбордів дозволяє бачити повну картину стану мережі в реальному часі, використовуючи готові шаблони або створюючи кастомні варіанти через JSON-конфігурації. Мережеве середовище всієї архітектури реалізовано на основі Docker Network, яка автоматично конфігурує підмережу та DNS-іменування, завдяки чому контейнери можуть звертатися один до одного за логічними іменами без додаткового налаштування IP-адрес. Уся система виглядає як повністю ізольоване середовище, де кожен компонент працює автономно в межах свого контейнера, але водночас є частиною загальної логічної структури. Трафік між вузлами моделюється за допомогою "iperf", а спостереження за його поведінкою здійснюється через зібрані метрики в Prometheus і візуалізацію у Grafana (див. рис. 2.1) [10-11].

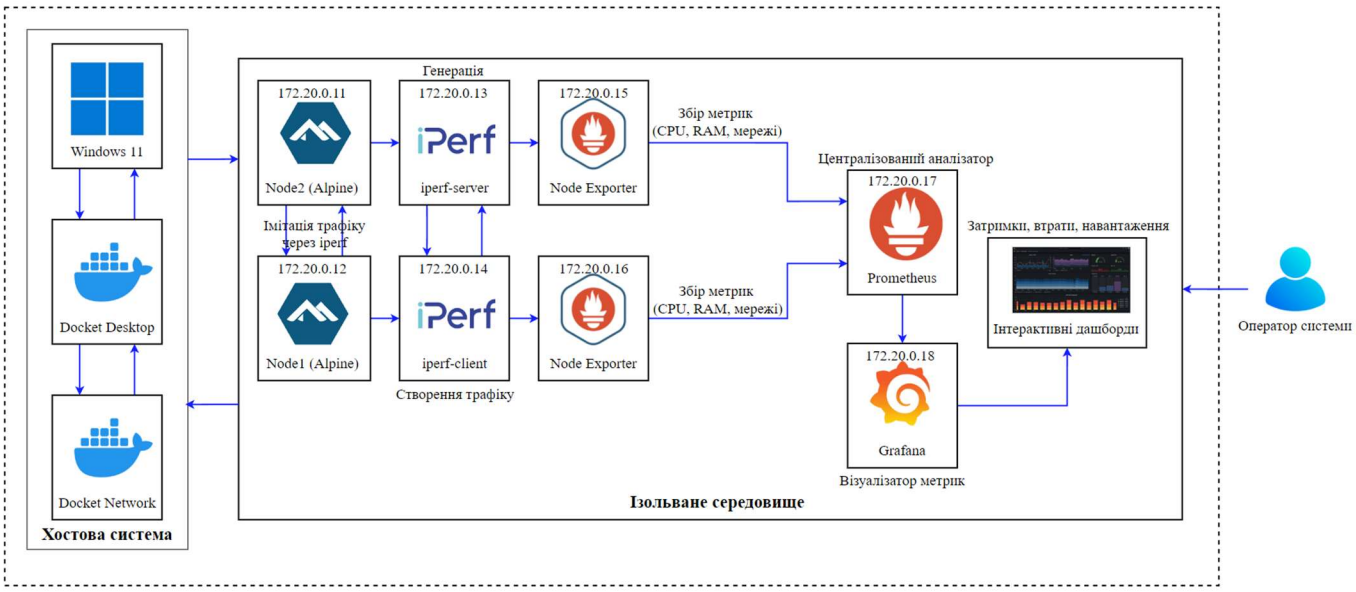


Рисунок 2.1 – Архітектура ізольованої системи моделювання та тестування мереж зі використанням платформи контейнерезації

Додатково передбачена можливість вимикання iperf-клієнта після виконання тестів для зниження навантаження на систему, що робить систему адаптивною до різних сценаріїв тестування.

Мережеве середовище в рамках цієї архітектури дозволяє масштабувати кількість вузлів без значних витрат часу і ресурсів, оскільки додавання нових контейнерів відбувається шляхом простого запуску ще одного екземпляра з потрібним образом. Ізоляція забезпечується контейнерною технологією, при цьому використання одного ядра хост-системи дозволяє досягти кращої продуктивності в порівнянні з класичною віртуалізацією. У випадку збільшення числа вузлів можна також масштабувати Prometheus шляхом конфігурації таргетів та побудови окремих панелей для моніторингу кожного з них, що також підтримується в Grafana за допомогою змінних і фільтрації [11].

У результаті отримується система, яка забезпечує повну емуляцію мережі з реальними навантаженнями та збором метрик, в якій можна наочно бачити, як змінюється продуктивність при внесенні тих чи інших змін у конфігурацію. Структура типу Node1 та Node2 з iperf, Node Exporter, Prometheus та Grafana створює завершений цикл аналізу, тестування та оптимізації мережі, що дозволяє

використовувати цю архітектуру не лише як навчальну або експериментальну платформу, а й як базову інфраструктуру для валідації конфігурацій у прикладних завданнях з оптимізації комп'ютерних мереж у сучасних умовах.

Docker як базова платформа віртуалізації мережі. В основі запропонованої архітектури розміщується технологія контейнеризації Docker, а саме використання Docker Desktop, що дозволяє створювати легкі ізольовані середовища для запуску вузлів, серверів, агентів моніторингу та інших компонентів, необхідних для побудови програмно-апаратного комплексу з метою моделювання та оптимізації комп'ютерних мереж у будь-якому середовищі. Серед основних переваг використання Docker у контексті мережевого моделювання вирізняється повна ізоляція кожного вузла, що функціонує у власному контейнері без конфліктів по портах, бібліотеках або залежностях, що значно спрощує керування, а також забезпечує стабільність усієї системи при масштабуванні [11].

Додатково варто враховувати фактор надзвичайно швидкого запуску контейнерів у порівнянні з віртуальними машинами, оскільки розгортання контейнера відбувається буквально за кілька секунд, що робить процес тестування мереж значно оперативнішим. Також надзвичайно важливою є спрощена конфігурація, яка реалізується за допомогою одного загального файлу "docker-compose.yaml", в якому описуються:

- мережеві параметри;
- маршрути;
- змінні;
- сервіси;
- логіка взаємодії.

Завдяки цьому вдається здійснювати повне розгортання інфраструктури єдиною командою, без складного налаштування вручну, що особливо актуально у разі багаторазового повторення експериментів з однаковими параметрами. Повторне використання конфігурацій у вигляді YAML-файлів забезпечує відтворюваність середовища без втрати часу на кожне нове налаштування.

У структурі реалізованої архітектури передбачено використання таких компонентів Docker, як "docker-compose" для запуску повної інфраструктури, кастомна мережа для з’єднання вузлів, а також томи (volumes) для збереження метрик, зокрема з боку Prometheus (див. рис. 2.2).

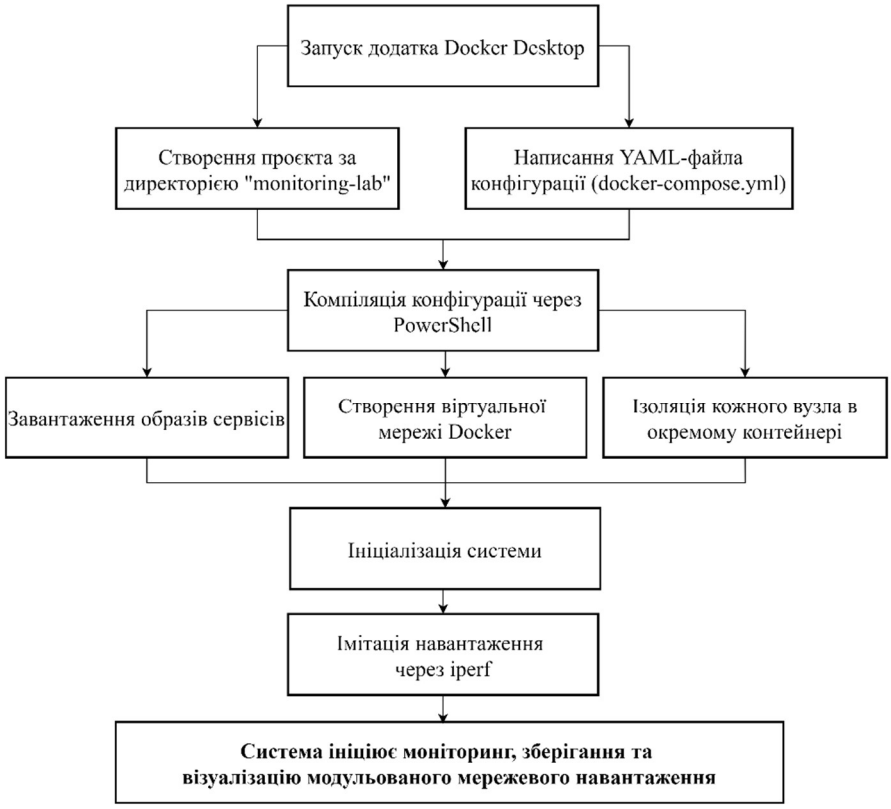


Рисунок 2.2 – Концептуальна схема розгортання та функціонування системи

Усі ці компоненти дозволяють створити динамічну, гнучку систему, яка легко масштабується та переноситься між машинами, зберігаючи стабільність і продуктивність завдяки спільному використанню ядра хост-системи, що робить її ефективнішою за класичну віртуалізацію.

Обґрунтування вибору інструментів – iperf, Prometheus, Grafana. Обґрунтовуючи вибір конкретних інструментів у побудованій системі, визначаються три основні програмні засоби — "iperf", Prometheus і Grafana, що забезпечують повноцінну модель взаємодії в межах симульованої мережі. Перший з них — "iperf3" — використовується для створення контрольованого мережевого

трафіку TCP та UDP, дозволяє вимірювати пропускну здатність, затримки та втрати пакетів. Його перевагами є:

— простота у використанні;

— широка підтримка в практиці мережевого тестування;

— можливість тонкого налаштування параметрів трафіку;

— сумісність із контейнерним середовищем.

Другий компонент — Prometheus — виконує функції збору системних метрик із вузлів за допомогою агентів Node Exporter. Перевагами є:

— власна база даних для збереження метрик;

— розвинена мова запитів PromQL;

— підтримка автоматичного виявлення таргетів;

— повна інтеграція з Docker.

Вибір цього інструмента зумовлений відкритим ліцензуванням, активною спільнотою підтримки, регулярними оновленнями та високою гнучкістю у налаштуваннях.

Третій компонент — Grafana — відповідає за побудову інтерактивних дашбордів та візуалізацію зібраних метрик у вигляді графіків, таблиць, теплових карт та інших віджетів. Вона забезпечує широкі можливості для створення багаторівневих панелей моніторингу з алертами, підтримує шаблони, інтегрується з Prometheus і дозволяє гнучке налаштування зовнішнього вигляду та логіки виводу даних. У межах побудованої архітектури Grafana слугує завершальним етапом аналітичного циклу, перетворюючи зібрані метрики на наочну інформацію для аналізу.

Інтеграція "iperf", Prometheus і Grafana в одну систему дозволяє забезпечити повний цикл симуляції, тестування, збору та аналізу мережевого навантаження. Завдяки взаємодії цих трьох інструментів можна точно моделювати мережеві сценарії, бачити вплив змін у конфігурації на поведінку трафіку та приймати обґрунтовані рішення щодо подальшої оптимізації в реальному середовищі. Такий підхід дозволяє досягати глибокого розуміння мережевих процесів ще до розгортання у продуктивних умовах.

Сценарії використання – перевірка стабільності з'єднання, трафіку. У межах розгортання програмно-апаратного комплексу, що орієнтований на моделювання та моніторинг комп'ютерних мереж, сценарії використання визначаються як базові механізми для перевірки стійкості, продуктивності й реакції мережевих вузлів на зміну умов. Система дозволяє реалізовувати ряд типових сценаріїв, що можуть застосовуватись у навчальних, дослідницьких та лабораторних цілях і дають змогу оцінити ключові аспекти поведінки віртуалізованого середовища при впливі навантаження або змін у конфігурації. Одним із таких сценаріїв є перевірка стабільності з'єднання, основна мета якої полягає у фіксації впливу перевантаження або мережевих збоїв на доступність вузлів та час відгуку. Реалізація цього сценарію передбачає генерацію ICMP-трафіку між двома контейнеризованими вузлами за допомогою iperf-клієнта у режимі постійного пінгу, при якому одночасно ведеться збір метрик про затримку, доступність та стабільність каналу, що дає змогу аналізувати загальний стан з'єднання через інтегровані панелі Grafana (див. рис. 2.3).

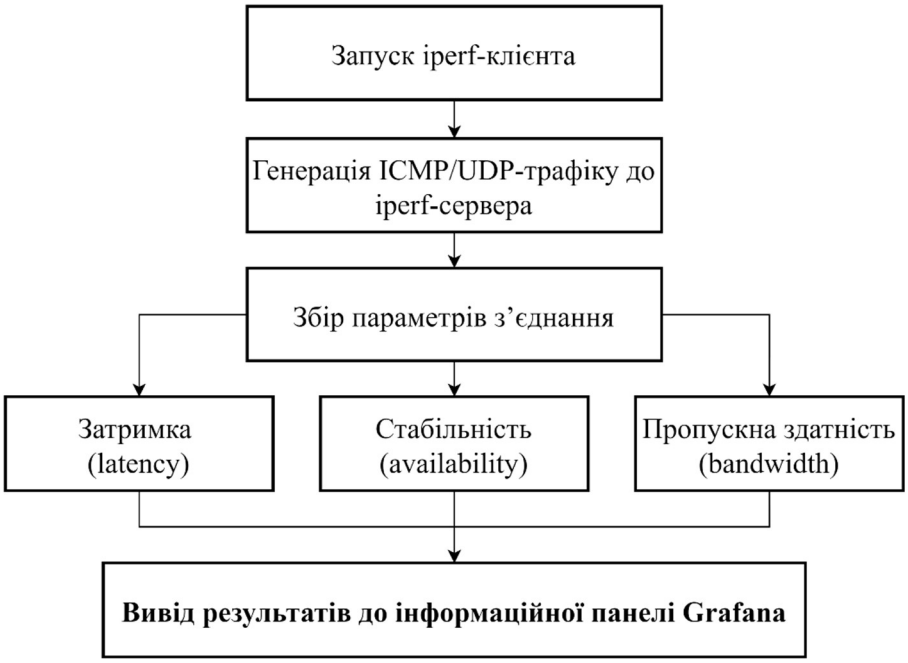


Рисунок 2.3 – Принцип використання "iperf" для імітації трафіку

Наступний сценарій — тестування пропускної здатності — зосереджується на визначенні максимальної швидкості передавання даних між вузлами. У рамках цього сценарію здійснюється запуск "iperf3" між вузлами у режимі TCP або UDP, і за результатами виконання вимірюються значення миттєвої пропускної здатності, затримок, втрат пакетів, які потім виводяться у вигляді графіків, що відображають зміну продуктивності в режимі реального часу, дозволяючи виявити моменти перевантаження або вузькі місця конфігурації. Ще одним варіантом використання виступає симуляція мережевих збоїв, суть якої полягає в імітації втрати з'єднання або навмисному порушенні топології з метою дослідження реакції системи. Такий підхід реалізується шляхом відключення одного з контейнерів або створення штучної затримки в передаваному трафіку, наприклад за допомогою утиліти "tc", і подальшого спостереження за результатами у Prometheus, де фіксується відсутність відповіді від вузла, помилки зчитування метрик, зміни часу доступу до сервісу або повна недоступність окремих системних елементів, що візуалізується через відповідні дашборди.

Сукупність цих сценаріїв демонструє потенціал побудованого комплексу для моделювання різноманітних ситуацій, з якими може зіткнутися реальна інфраструктура, що у свою чергу дозволяє:

— відлагоджувати конфігурації;

— шукати оптимальні шляхи маршрутизації;

— аналізувати стабільність каналів;

— аналізувати ефективність балансування навантаження.

Кожен із етапів виконується без потреби у складному апаратному забезпеченні завдяки використанню сучасних контейнеризованих інструментів і спрощеному середовищу Docker, де за допомогою "iperf" забезпечується генерація навантаження, через Prometheus здійснюється збір і зберігання даних, а Grafana відповідає за гнучке графічне представлення параметрів у реальному часі.

У підсумку побудована система демонструє ефективний підхід до реалізації архітектури мережевого моделювання, в якому використання контейнерної платформи Docker дозволяє спростити процес розгортання, а інтеграція Prometheus

та Grafana забезпечує повний цикл від генерації трафіку до збору та аналітики даних. Така архітектура дозволяє:

— здійснювати експерименти з реалістичною поведінкою мережевих вузлів;
— швидко виявляти вузькі місця;
— реагувати на збої;
— оцінювати продуктивність у різних конфігураціях.

2.2 Побудова віртуального мережевого середовища

Для побудови віртуального мережевого середовища, що забезпечує функціональність базового середовища, в якому можливо виконувати моделювання комп’ютерної мережі з кількома вузлами, а також здійснювати тестування продуктивності та аналізувати поведінку мережі при змінному навантаженні, необхідно забезпечити високу гнучкість, масштабованість та швидке розгортання. Архітектура такої системи базується на використанні контейнеризованого підходу, де віртуальне середовище реалізується у вигляді ізольованих контейнерів, що поєднані у спільну логічну мережу та можуть розгортатися на будь-якій платформі, що підтримує Docker [12].

Опис конфігураційного файлу для моделювання мережі. Основу конфігурації становить файл "docker-compose.yml", який виконує функцію основного конфігураційного документа для опису та ініціалізації середовища (див. табл. 2.1). У межах цього файлу визначається структура контейнеризованих вузлів, тестових агентів, компонентів моніторингу та візуалізації, які об’єднуються у єдину підмережу через Docker Network.

Таблиця 2.1 – Опис "docker-compose.yml" для моделювання мережі

Назва контейнера	**Образ**	**IP-адреса**	**Роль у системі**	**Зв'язки / Залежності**
node1	alpine	172.20.0.11	Перший віртуальний вузол (імітація клієнта)	Взаємодіє з node2, моніториться node_exporter_node1
node2	alpine	172.20.0.12	Другий вузол (імітація сервера)	Взаємодіє з node1, моніториться node_exporter_node2
iperf-server	networkstatic/iperf3	172.20.0.13	Приймає трафік під час тесту продуктивності	Приймає запити від iperf-client
iperf-client	networkstatic/iperf3	172.20.0.14	Генерує трафік для вимірювання швидкості	Залежить від iperf-server (depends_on)
node_exporter_node1	prom/node-exporter	172.20.0.15	Агент збору метрик з node1	Надсилає дані до prometheus
node_exporter_node2	prom/node-exporter	172.20.0.16	Агент збору метрик з node2	Надсилає дані до prometheus
prometheus	prom/prometheus	172.20.0.17	Збір та зберігання метрик з експортерів	Отримує дані з node_exporter_*
grafana	grafana/grafana	172.20.0.18	Інтерфейс візуалізації зібраних метрик	Підключається до prometheus

У секції "services" описано окремі компоненти, що складають логіку системи: віртуальні вузли "node1" і "node2", трафікові агенти "iperf-server" і "iperf-client", два

"node_exporter", сервер збору метрик "prometheus" та система візуалізації "grafana". Кожен із сервісів має власне ім’я контейнера, IP-адресу у межах підмережі "monitoring_net", а також у разі потреби — налаштування початкових команд або залежностей, наприклад, "iperf-client" має затримку перед запуском і виконує підключення до "iperf-server", після чого завершує свою роботу автоматично, не навантажуючи систему.

Параметри мережі визначені у секції "networks", де створюється підмережа з драйвером "bridge" та вказаним діапазоном IP-адрес, що забезпечує коректне адресування і взаємодію між усіма контейнерами. Такий підхід дозволяє централізовано керувати маршрутизацією, зв’язками між сервісами, взаємодією між клієнтськими і серверними компонентами, а також організовувати ізольовану, стабільну інфраструктуру без конфліктів портів чи ресурсів.

На відміну від класичних віртуальних машин, які вимагають інсталяції повноцінних операційних систем, налаштування мережевих адаптерів та створення складної топології вручну, контейнеризована інфраструктура у межах "docker-compose.yml" дає змогу створювати симульовану мережу за кілька секунд. Усі конфігурації зберігаються у вигляді одного YAML-файлу, що забезпечує відтворюваність експериментів, повторюваність сценаріїв у будь-який час і мінімізацію витрат часу на розгортання середовища (див. рис. 2.4) [12].

```yaml
version: "3.8"

services:
  node1:
    image: alpine
    container_name: node1
    command: tail -f /dev/null
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.11

  node2:
    image: alpine
    container_name: node2
    command: tail -f /dev/null
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.12

  iperf-server:
    image: networkstatic/iperf3
    container_name: iperf-server
    command: -s
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.13

  iperf-client:
    image: networkstatic/iperf3
    container_name: iperf-client
    depends_on:
      - iperf-server
    entrypoint: >
      sh -c "sleep 5 && iperf3 -c 172.20.0.13 -t 30"
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.14

  node_exporter_node1:
    image: prom/node-exporter
    container_name: node_exporter_node1
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.15

  node_exporter_node2:
    image: prom/node-exporter
    container_name: node_exporter_node2
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.16

  prometheus:
    image: prom/prometheus
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.17

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    depends_on:
      - prometheus
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.18

networks:
  monitoring_net:
    driver: bridge
    ipam:
      config:
        - subnet: 172.20.0.0/24
```

Рисунок 2.4 – Конфігурація "docker-compose.yml" для розгортання системи

Крім цього, використання легковагих образів контейнерів, таких як alpine, дозволяє значно зменшити споживання ресурсів, а також прискорити запуск системи навіть на обмежених хостах. Завдяки такому підходу віртуальне середовище наближається до реалій практичного застосування, даючи змогу дослідити мережеву поведінку з високою точністю, фіксувати затримки, втрати, продуктивність і зміну доступності вузлів, а також автоматизувати проведення тестів на основі завчасно підготовлених конфігурацій, що особливо актуально в рамках навчальних або дослідницьких завдань.

Віртуальні вузли на базі Alpine (node1, node2). Розглядаючи контейнери, а саме вузли "node1" і "node2", слід зазначити, що це імітація двох мережевих вузлів, які можуть взаємодіяти між собою, виконуючи базові функції в комп’ютерній мережі. Вони побудовані на основі мінімалістичного образу Linux Alpine, який широко використовується в сучасних DevOps-середовищах завдяки своїй малій вазі та швидкому часу запуску. Образ Alpine важить менше ніж 5 мегабайт, при цьому вже включає базові мережеві утиліти, такі як "ping", "netcat", "ip", "route", що

робить його надзвичайно зручним для обмежених середовищ та лабораторних досліджень. Відсутність зайвих компонентів і простота конфігурації дозволяє зосередитися на дослідженні поведінки мережі в умовах, максимально наближених до реальних (див. рис. 2.5).



Рисунок 2.5 – Віртуальні вузли на базі Alpine у вигляді контейнерів

Такі вузли дозволяють емулювати реальні ситуації, включаючи втрату зв’язку, перевантаження, затримки та інші аномалії, які неможливо детально змоделювати в умовах абстрактних симуляторів. Основна функція цих вузлів полягає в безперервному обміні пакетами з використанням протоколу ICMP, що реалізується через постійне надсилання ping-запитів між вузлами "node1" та "node2" для перевірки доступності та стабільності з’єднання. Крім цього, для збору системних метрик, таких як навантаження на процесор, використання пам’яті або мережевий трафік, на кожному з вузлів встановлено агент "node_exporter", який передає зібрані дані до Prometheus.

Вузли також можуть грати різні ролі, включаючи клієнт, сервер або маршрутизатор, що дозволяє розширити сценарії використання та провести більш гнучке тестування. Зручність такої конфігурації полягає в тому, що всі вузли мають унікальні імена в мережі Docker і можуть звертатися один до одного за DNS-іменами контейнерів, наприклад, "node1", "node2", "iperf-server", що значно спрощує налагодження взаємодії без потреби у ручному налаштуванні IP-адрес, що робить процес створення, підтримки і масштабування мережі швидким, ефективним і повторюваним, що особливо важливо для дослідницьких і навчальних середовищ.

Тестування трафіку між вузлами через iperf. Для моделювання навантаження між вузлами використовується інструмент "iperf3", який є

загальновизнаним стандартом у сфері тестування мереж. Його основною функцією є генерація контрольованого трафіку між двома точками з подальшим вимірюванням ключових показників, таких як пропускна здатність, затримки, втрати пакетів та нестабільність трафіку. У системі реалізовано два компоненти цього інструменту:

1) "iperf-server" – приймає запити на стандартному порту 5201.

2) "iperf-client" – ініціює підключення, генерує трафік і виводить результати вимірювань.

Типовий запуск може бути здійснений через команду "docker exec iperf-client iperf3 -c iperf-server", що ініціює з'єднання між вузлами та формує звіт із ключовими показниками продуктивності, який потім може бути виведений у терміналі або інтегрований у систему візуалізації Grafana (див. рис. 2.6).

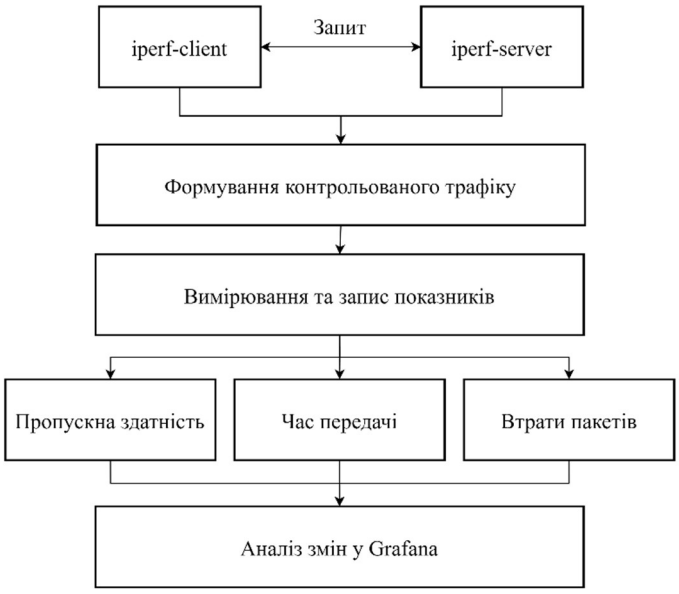


Рисунок 2.6 – Схема взаємодії iperf-server та iperf-client

До налаштувань цього інструменту можуть входити параметри:

— -t 10 (тривалості сеансу в секундах);

— -u (використання UDP замість TCP);

— -b 100M (встановлення ширини каналу).

Усе це дозволяє емулювати різні умови передавання даних. За допомогою таких гнучких налаштувань можна реалізувати численні сценарії навантаження та адаптації мережі до змінних умов, що є невід'ємною частиною тестування та оптимізації віртуального мережевого середовища.

Імітація реального навантаження. У цьому випадку для створення більш реалістичного тестового середовища на етапі реалізації необхідно моделювати певні типи навантаження, які виникають у реальних мережах, і варіанти імітації таких процесів можуть бути різними залежно від поставлених цілей. Наприклад, одним із базових сценаріїв є постійний пінг між вузлами, де головна мета полягає в контролі доступності та часі відповіді, а реалізація такого сценарію відбувається через нескінченний цикл надсилання ping-запитів від одного вузла до іншого, що дозволяє відстежувати стабільність з'єднання протягом усього періоду експерименту. Інший сценарій може передбачати періодичну передачу даних з різною інтенсивністю, де "iperf3" використовується зі змінною швидкістю передачі, що дає можливість симулювати пікові навантаження і дослідити, як система реагує на раптове зростання мережевого трафіку. Такий підхід дозволяє не лише згенерувати значне навантаження, а й вивчити поведінку мережі під час зміни параметрів.

Окремо можна виділити сценарій навмисного перевантаження вузла, що реалізовується шляхом запуску додаткових процесів або скриптів всередині контейнера з метою навмисного підвищення споживання ресурсів CPU, оперативної пам'яті або дискової підсистеми. У поєднанні з агентами моніторингу, які фіксують зміни навантаження в реальному часі, це дозволяє аналізувати поведінку системи у граничних умовах і виявляти вузькі місця, які потребують оптимізації. Такий підхід забезпечує високу гнучкість під час дослідження реальних сценаріїв навантаження, зокрема в умовах обмежених ресурсів або критичної чутливості до затримок.

Відзначаючи переваги такого підходу, слід підкреслити, що він дозволяє моделювати нестабільності та граничні умови, формуючи у результаті реалістичну платформу для тестування адаптації мережевих компонентів до змінних умов

середовища. Усе це формує основу для подальшої оптимізації конфігурації, оскільки результати експериментів дають змогу обґрунтовано підібрати параметри, які будуть застосовані вже на реальних апаратних рішеннях.

Підсумовуючи, побудова віртуального мережевого середовища за допомогою контейнеризації Docker, інструментів "iperf" і дистрибутиву Alpine створює ефективну, гнучку та масштабовану платформу для емулювання мережевої поведінки. Завдяки ізольованості середовищ, підтримці сценаріїв реального трафіку та широким можливостям конфігурації така система стає універсальним інструментом для дослідження продуктивності та взаємозв’язків між елементами мережі в умовах, що змінюються динамічно, що дозволяє не лише виявляти потенційні проблеми, а й проводити практичну оцінку ефективності рішень, підготовлених до впровадження в реальні інфраструктури [12].

2.3 Алгоритми збору й аналізу метрик

Для ефективного моніторингу стану віртуального мережевого середовища необхідно організувати системний підхід до збору, зберігання та візуалізації метрик, що надходять з усіх вузлів, і ці метрики дозволяють аналізувати продуктивність системи, виявляти проблеми, пов’язані з навантаженням, ресурсами чи комунікацією, а також формувати обґрунтовані висновки щодо оптимізації інфраструктури. У такому підході основними елементами реалізації стають системи збору даних, зокрема Prometheus, та візуалізації — Grafana, які в тісній інтеграції з експортерами метрик, наприклад Node Exporter, забезпечують єдине замкнене коло — від отримання інформації з вузлів до її аналізу в зручному графічному вигляді [13].

Встановлення та конфігурація Prometheus. Prometheus є системою моніторингу з відкритим кодом, яка побудована на моделі pull, тобто сервер самостійно опитує вказані цілі, щоб зібрати потрібні дані. Завдяки гнучкій

архітектурі, підтримці масштабування, розвиненій мові запитів PromQL та простоті конфігурації він широко використовується в сучасних інфраструктурах. В рамках реалізації пропонованої системи Prometheus конфігурується за допомогою "docker-compose", де додається відповідний сервіс з посиланням на зовнішній конфігураційний файл, в якому описується інтервал опитування та адреси вузлів-експортерів.

У "docker-compose.yml" необхідно прописати блок для Prometheus, де вказується образ, контейнерне ім’я, порт для доступу до інтерфейсу (9090) і мережеві параметри, а також volume, в якому розміщено конфігураційний файл Prometheus (див. рис. 2.7).

```
prometheus:
  image: prom/prometheus
  container_name: prometheus
  volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml
  ports:
    - "9090:9090"
  networks:
    monitoring_net:
      ipv4_address: 172.20.0.17
```

Рисунок 2.7 – Фрагмент конфігурації "docker-compose.yml" для розгортання Prometheus

Окремо створюється файл "prometheus.yml", де задаються правила збору метрик. У секції "global" встановлюється інтервал опитування, наприклад 5 секунд, а в секції "scrape_configs" вказуються джерела даних — адреси експортерів на вузлах, які повертають системні метрики (див. рис. 2.8).

```
global:
  scrape_interval: 5s

scrape_configs:
  - job_name: 'node_exporter_node1'
    static_configs:
      - targets: ['172.20.0.15:9100']
  - job_name: 'node_exporter_node2'
    static_configs:
      - targets: ['172.20.0.16:9100']
```

Рисунок 2.8 – Конфігурація "prometheus.yml" для збору метрик

Усе це дозволяє Prometheus стабільно взаємодіяти з вузлами навіть після перезапуску контейнерів, оскільки IP-адреси задано вручну в "docker-compose.yml". Для збору метрик використовуються Node Exporter — агенти, що запускаються в окремих контейнерах на кожному вузлі, й автоматично надають інформацію про використання CPU, RAM, дисків, мережеві інтерфейси, кількість процесів та інші показники. Їхні конфігурації включають базовий образ "prom/node-exporter", ім’я контейнера та IP-адресу, що входить до однієї спільної мережі (див. рис. 2.9).

```
node_exporter_node1:
  image: prom/node-exporter
  container_name: node_exporter_node1
  networks:
    monitoring_net:
      ipv4_address: 172.20.0.15

node_exporter_node2:
  image: prom/node-exporter
  container_name: node_exporter_node2
  networks:
    monitoring_net:
      ipv4_address: 172.20.0.16
```

Рисунок 2.9 – Фрагмент конфігурації "docker-compose.yml"
для розгортання агентів Node Exporter

Після запуску системи Prometheus автоматично починає опитувати ці два вузли, зберігає зібрані метрики у власній базі даних, що дозволяє не лише переглядати поточні значення, а й проводити історичний аналіз. На базі цієї інформації в подальшому формується інтерактивна візуалізація за допомогою Grafana, а також забезпечується можливість створення сповіщень при виявленні аномалій. Таким чином, організація збору метрик через Prometheus і Node Exporter створює стійке й масштабоване підґрунтя для безперервного контролю мережевого середовища, яке є основою для якісної оптимізації та прийняття рішень на основі реальних даних.

Засіб Grafana для візуалізації. Коли йдеться про налаштування Grafana для візуалізації, мається на увазі використання системи побудови інтерактивних дашбордів, яка інтегрується з Prometheus через HTTP API, забезпечуючи можливість динамічного аналізу метрик, що надходять із вузлів. Її основною

перевагою є гнучкість у створенні графіків, підтримка великої кількості джерел даних і можливість формування кастомізованих панелей для моніторингу. Розгортання Grafana у контейнеризованому середовищі здійснюється аналогічно до Prometheus, за допомогою конфігурації у "docker-compose.yml", де вказується образ системи, порт доступу (3000) та мережа, яка дозволяє контейнеру взаємодіяти з іншими компонентами середовища, зокрема з самим Prometheus (див. рис. 2.10).

```
grafana:
  image: grafana/grafana
  container_name: grafana
  ports:
    - "3000:3000"
  depends_on:
    - prometheus
  networks:
    monitoring_net:
      ipv4_address: 172.20.0.18
```

Рисунок 2.10 – Фрагмент конфігурації "docker-compose.yml" для розгортання Grafana

Збір метрик CPU, мережі, навантаження через агенти. Крім самого налаштування Grafana, необхідно врахувати ключові аспекти збору метрик із вузлів, що виконуються за допомогою Node Exporter. До базових категорій метрик, які мають значення при аналізі продуктивності, належать показники (див. табл. 2.2):

— процесора;

— оперативної пам’яті;

— мережевої активності.

Таблиця 2.2 – Збір метрик CPU, мережі, навантаження

Категорія	**Метрика / Формула**	**Опис і значення**
CPU	rate(node_cpu_seconds_total{mode!="idle"}[1m])	Час, витрачений на користувацькі процеси; дозволяє виявити перевантаження процесора
Пам’ять	node_memory_MemTotal_bytes node_memory_MemAvailable_bytes	Фактично використана пам’ять у байтах, сигналізує про заповнення оперативної пам’яті
Мережа	rate(node_network_receive_bytes_total[1m])	Вхідний трафік, швидкість отримання даних через мережевий інтерфейс
	rate(node_network_transmit_bytes_total[1m])	Вихідний трафік, швидкість передачі даних через інтерфейс
	rate(node_network_receive_packets_total[1m])	Кількість отриманих пакетів за секунду
	rate(node_network_transmit_packets_total[1m])	Кількість переданих пакетів за секунду

Продовження таблиці 2.2 – Збір метрик CPU, мережі, навантаження

Категорія	Метрика / Формула	Опис і значення
Мережа	node_network_receive_errs_total	Кількість помилок під час отримання даних
	node_network_transmit_errs_total	Кількість помилок під час передачі даних
	node_network_receive_drop_total	Втрачені пакети при отриманні (drop)
	node_network_transmit_drop_total	Втрачені пакети при передачі

Для процесора доцільно фіксувати час роботи користувацьких процесів, що дозволяє виявляти перевантаження та пікову активність. Оцінювання оперативної пам’яті базується на показнику фактичного використання, що дає змогу вчасно виявити перевитрати або нестачу ресурсів. Для аналізу мережевої активності важливо враховувати швидкість передачі вхідного та вихідного трафіку, кількість переданих і отриманих пакетів, а також наявність помилок чи втрат, що може сигналізувати про нестабільну роботу вузла або перевантаження інтерфейсу (див. рис. 2.11) [13].



Рисунок 2.11 – Основні метрики CPU, RAM, мережі у Grafana

Такі базові метрики формують мінімально необхідний набір для оцінки стану вузлів і поведінки мережі в умовах емульованого трафіку, зокрема під час використання інструменту "iperf". Незважаючи на те, що кількість доступних метрик у Node Exporter значно перевищує сотню, у практиці достатньо сконцентрувати увагу саме на ключових — зокрема тих, що пов’язані з процесором, оперативною пам’яттю та мережею, адже саме вони першими вказують на потенційні проблеми в архітектурі мережевої взаємодії. Таким чином, використання Grafana у зв’язці з Prometheus та Node Exporter дозволяє реалізувати повноцінну систему збору, інтерпретації та візуального контролю всіх найважливіших параметрів, забезпечуючи своєчасне реагування на будь-які відхилення у роботі віртуального мережевого середовища.

Приклади типових графіків і висновків на їх основі. Також якщо розглядати приклади типових графіків і описувати, що саме вони можуть означати та як інтерпретуються у контексті моніторингу (див. рис. 2.12).

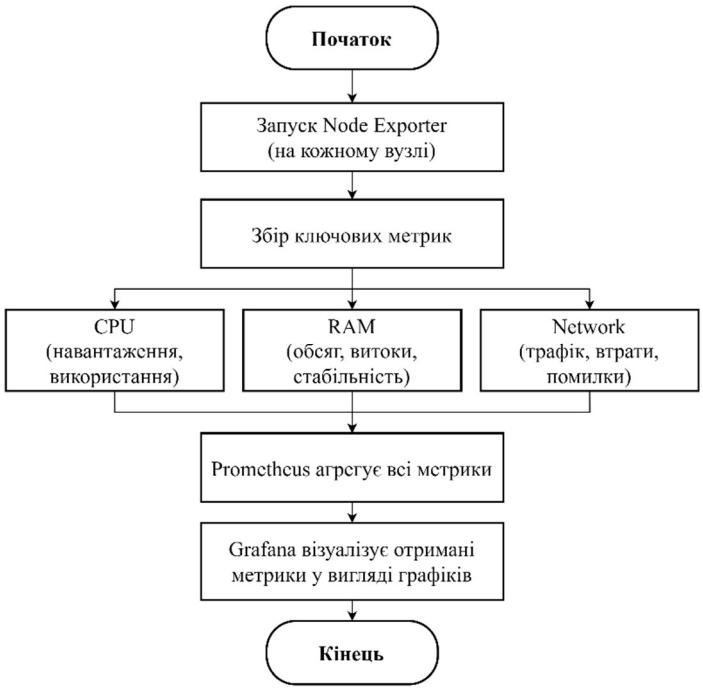


Рисунок 2.12 – Алгоритм збору й аналізу метрик

Одним із перших важливих візуальних елементів є графік навантаження на центральний процесор у часі. У такому випадку різке зростання навантаження на

CPU здатне свідчити про аномальні запити, перевищення планових ресурсів або навіть імітацію розподіленої атаки на відмову в обслуговуванні, що може проявлятись у збільшенні часу відповіді або деградації загальної продуктивності системи (див. рис. 2.13).

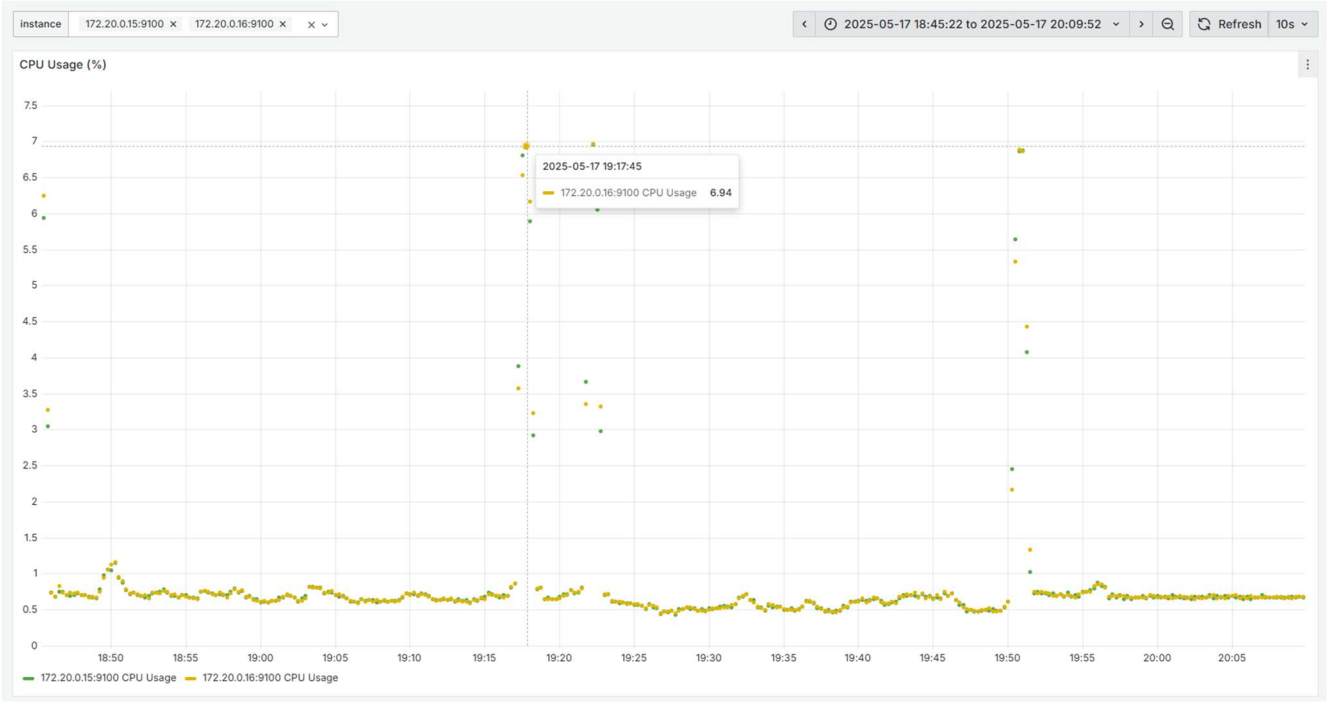


Рисунок 2.13 – Графік навантаження на CPU: імітація навантаження iperf-client

Такий графік дозволяє миттєво виявити періоди пікових навантажень і зрозуміти, коли саме система перестає справлятися з поставленими завданнями

Іншим типом важливого графіку є візуалізація використаної оперативної пам’яті, особливо якщо застосовується кумулятивна подача — накопичення використання у часі. За допомогою таких графіків можливо виявити ситуації з витоком пам’яті у сервісах, які з часом починають споживати усе більше ресурсів без вивільнення (див. рис. 2.14).

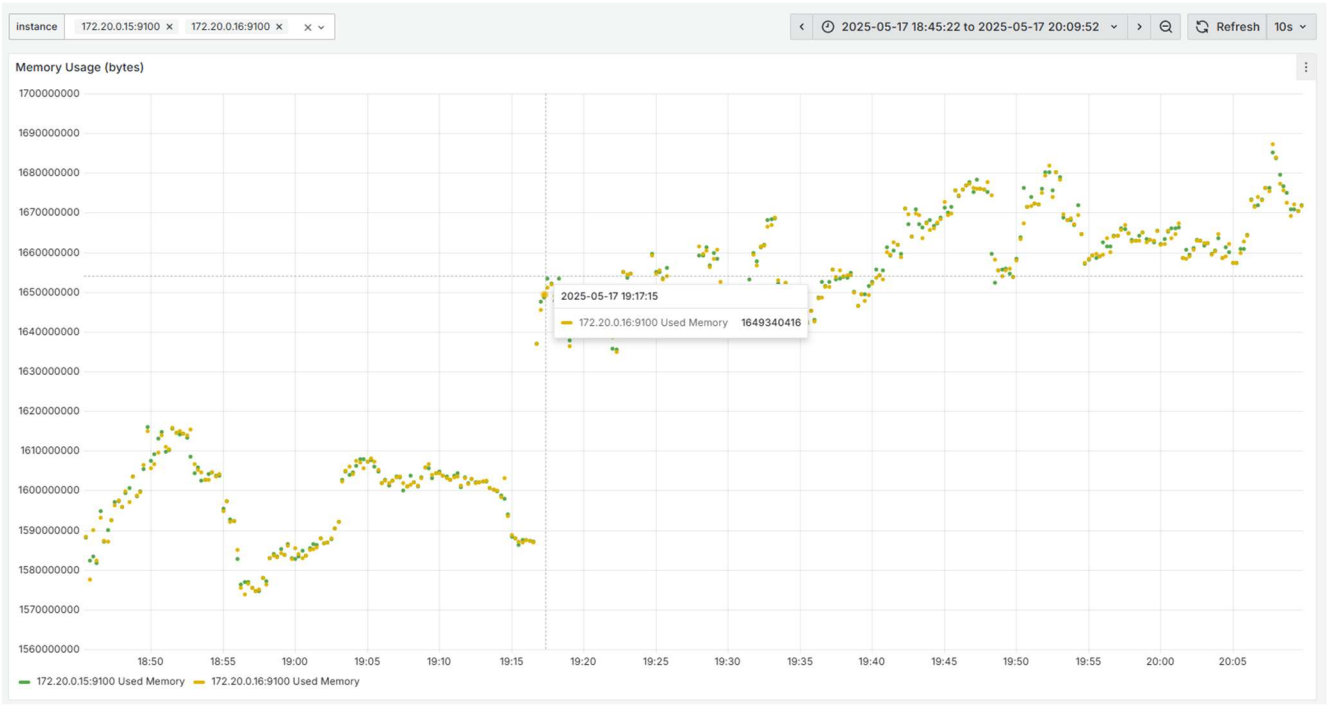


Рисунок 2.14 – Графік використання RAM: імітація навантаження iperf-client

Також такий графік дозволяє виявити нестабільну поведінку системи під час масштабування або неефективну роботу буферів, що призводить до зайвого використання оперативної пам’яті навіть у спокійному режимі [13].

Графіки мережевої активності, а саме передавання і приймання байтів або кількості мережевих пакетів у заданий період, демонструють інтенсивність роботи вузлів у режимі тестування трафіку, наприклад при запуску "iperf". На таких графіках чітко простежується момент запуску трафіку, його тривалість і завершення, що дозволяє побачити динаміку навантаження на мережу в реальному часі (див. рис. 2.15).

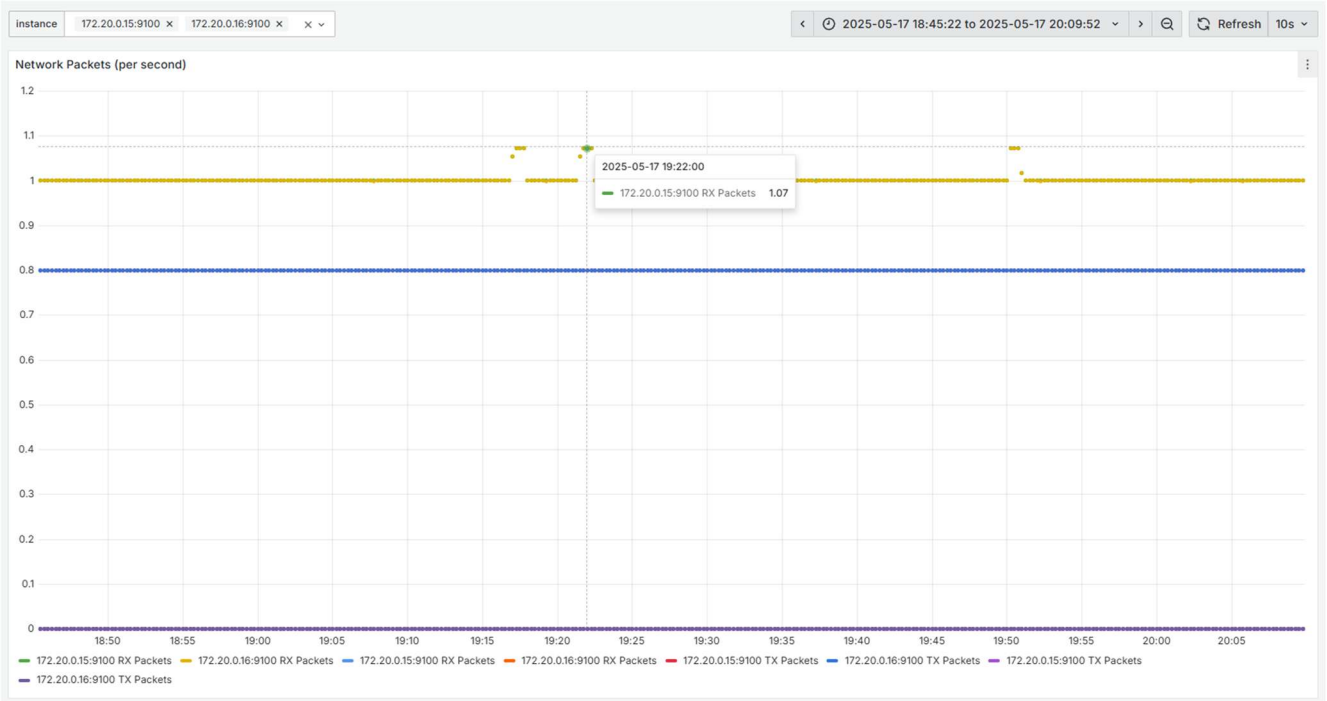


Рисунок 2.15 – Графік мережевої активності: імітація навантаження iperf-client

Зокрема, коли iperf-клієнт автоматично запускається і припиняє роботу, його поведінка чітко фіксується як хвиля активності на мережевих графіках, що є дуже зручним при повторюваних експериментах.

Крім цього, якщо в інфраструктурі реалізована емуляція затримок або втрат пакетів, наприклад через утиліту "tc" усередині контейнера, додаткові графіки дозволяють фіксувати затримки, втрати або нестабільність у доставці даних (див. рис. 2.16).

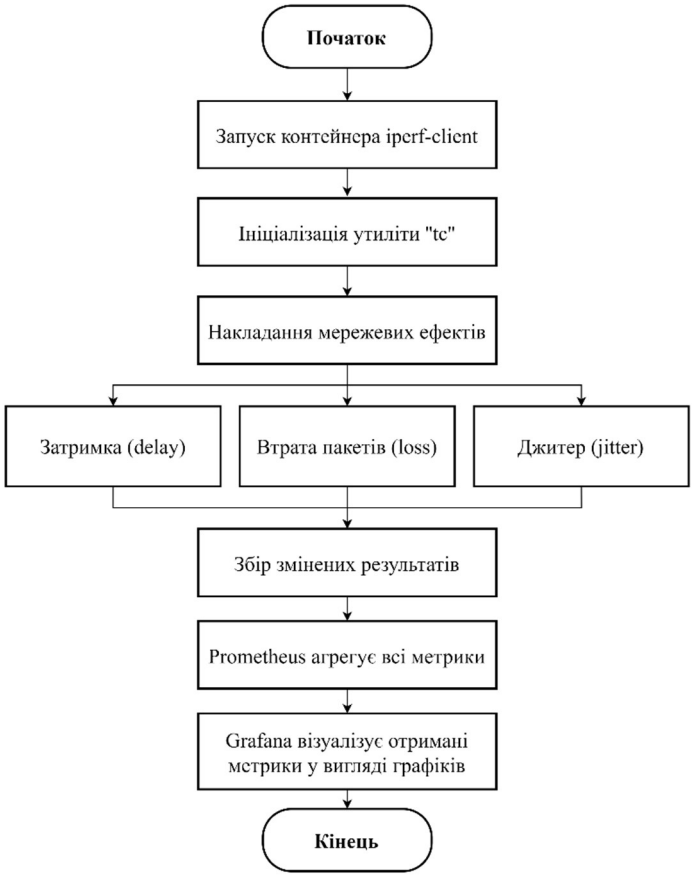


Рисунок 2.16 – Алгоритм емуляції затримок або втрат пакетів

Хоча такі механізми не були активно впроваджені у рамках поточного експерименту, реалізація їх можлива у межах тієї ж інфраструктури. Вони особливо корисні при тестуванні відеоконференцзв'язку, VoIP, онлайн-ігрових систем або інших сценаріїв, чутливих до мережевих коливань.

Обґрунтовуючи загальний підхід до аналізу графіків, можна зазначити, що у зв'язці Prometheus і Grafana формується потужна архітектура моніторингу для віртуального середовища. Її гнучкість дозволяє зчитувати ключові метрики з усіх вузлів — процесора, пам'яті, мережі, дискових ресурсів — у режимі реального часу. Використовуючи шаблонні або створені вручну дашборди, можна:

— миттєво локалізувати проблемні зони;

— виявляти тренди навантаження;

— здійснювати порівняння з попередніми періодами;

— формувати на основі цього звіти або аналітичні висновки.

Отримані графіки не лише візуалізують процеси, що відбуваються у системі, а й формують базу для глибшого аналітичного аналізу, дозволяючи будувати логіку оптимізації на реальних даних. Така візуалізація дає змогу швидко приймати обґрунтовані рішення щодо змін у конфігурації, адаптації системи до нових умов роботи або реагування на потенційні відмови ще до того, як вони призведуть до повного порушення функціональності. Таким чином, навіть базові типи графіків дозволяють сформувати цілісну картину стану середовища, що досліджується.

2.4 Реалізація побудованої системи

Коли мова йде про реалізацію самої системи, необхідно описати повний цикл створення програмно-апаратного комплексу для моделювання комп'ютерних мереж на основі віртуалізованого середовища контейнерів Docker, у якому розглядається не лише підготовка робочого середовища та запуск контейнерних сервісів, а й демонстрація взаємодії мережевих вузлів через інструмент "iperf" та візуалізація результатів у Grafana на основі попередньо зібраних у Prometheus даних, приділяючи при цьому окрему увагу питанням масштабування, додаванню нових вузлів і змінам у конфігурації при потребі [14, 26].

Підготовка середовища – Docker Desktop та Docker Compose. Для розгортання системи використовується контейнерна платформа Docker, яка дозволяє створювати ізольовані середовища та ефективно ними управляти, і в цьому випадку було обрано також елемент оркестрації Docker Compose, що значно спрощує керування кількома сервісами через єдиний конфігураційний файл, а отже й сам процес запуску в один клік. Під час підготовки середовища необхідно спершу встановити Docker Desktop на обрану платформу, що може бути Windows, macOS або Unix-подібна система, після чого слід переконатися в коректності роботи через команду "docker --version", яка повертає актуальну інформацію про встановлену версію. Якщо все працює правильно, наступним кроком стає перевірка наявності

Docker Compose, який, починаючи з версії 2, вже інтегровано у Docker Desktop, і цього цілком достатньо для одночасного запуску декількох контейнерів у межах єдиного сценарію (див. рис. 2.17) [14].

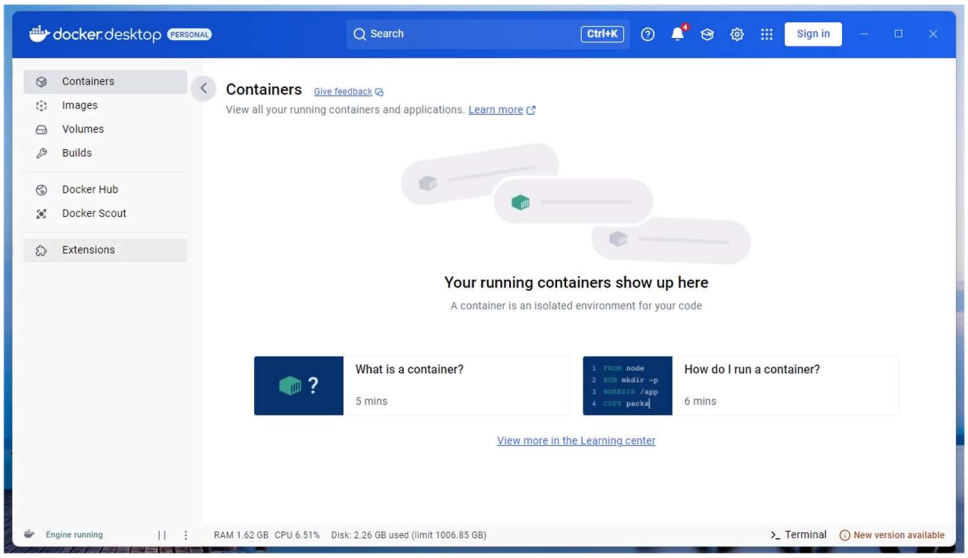


Рисунок 2.17 – Інтерфейс Docker Desktop

Після підготовки інструментів переходять до створення структури проєкту, у якій ключовим є файл конфігурації "docker-compose.yml", що визначає архітектуру усієї системи, включаючи опис усіх сервісів і вузлів, зокрема "node1", "node2", iperf-сервер і клієнт, Node Exporter для кожного вузла, Prometheus і Grafana. Файл повинен бути розміщений у кореневій директорії проєкту, де зберігаються всі конфігураційні елементи, і його можна адаптувати під будь-яке середовище, змінюючи IP-адреси, образи або параметри запуску в залежності від потреб. Ієрархія каталогу проєкту може виглядати приблизно наступним чином (див. рис. 2.18).

```
monitoring-lab/
├── docker-compose.yml
├── prometheus.yml
```

Рисунок 2.18 – Ієрархія каталогу проєкту

У цій структурі "docker-compose.yml" відповідає за загальний опис сервісів, "prometheus.yml" — за налаштування збору метрик, а каталог "grafana" містить шаблони або інструкції для автоматичного створення дашбордів. Усі ці файли розміщуються у зручному вигляді, що дозволяє повторно використовувати готову конфігурацію для моделювання або експериментів на інших платформах.

Щодо апаратних вимог до хост-машини, достатньо мати:

— від 2 до 4 ядер CPU;

— щонайменше 4 ГБ оперативної пам’яті;

— близько 2 ГБ вільного місця на диску.

Усе це дозволить без проблем запустити необхідні сервіси та забезпечити їхню стабільну роботу. У Docker Desktop також доступна можливість вручну визначати, скільки ресурсів виділяється окремим контейнерам, що дозволяє дуже гнучко масштабувати або обмежувати навантаження в залежності від експериментального сценарію.

Завдяки такому підходу реалізація системи перетворюється на просту процедуру розгортання конфігураційного середовища, у якому взаємодіють логічно ізольовані вузли, з підтримкою тестування, збору метрик і візуалізації результатів у реальному часі без необхідності налаштування повноцінних віртуальних машин або складних систем. Усі етапи, від підготовки до запуску, охоплюють весь життєвий цикл тестового стенду, призначеного для гнучкого та багаторазового використання з можливістю його масштабування та модифікації у майбутньому.

Процедура розгортання комплексу. Після завершення підготовки конфігураційного середовища необхідно здійснити запуск усіх елементів системи, який виконується однією командою, що суттєво спрощує процедуру. Для цього потрібно спочатку перейти до відповідної директорії, у якій зберігається створений конфігураційний файл "docker-compose.yml" разом із додатковим файлом конфігурації "prometheus.yml", як правило, це директорія з назвою "monitoring-lab", де розміщено всю структуру проєкту. Важливо переконатися, що Docker Desktop запущено, після чого відкривається термінал, наприклад, PowerShell або будь-який

інший сумісний інтерфейс командного рядка. Перебуваючи у потрібній директорії, запускається команда "docker-compose up -d", яка ініціалізує процес створення контейнерів, що може зайняти певний час залежно від кількості сервісів, зазначених у конфігурації (див. рис. 2.19).

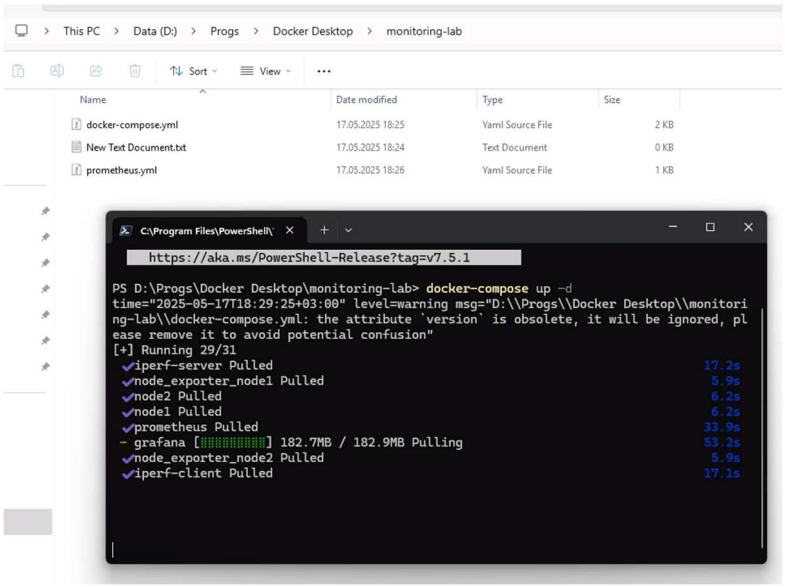


Рисунок 2.19 – Запуск середовища через "docker-compose up"

Після завершення запуску виводиться інформація про створення кожного контейнера, і за допомогою команд "docker ps" або "docker stats" можна перевірити їхній статус і поточне навантаження (див. рис. 2.20).



Рисунок 2.20 – Перевірка працездатності розгорнутої системи

У середовищі Docker Desktop на Windows 11 також можна візуально побачити всі активні контейнери, включаючи інформацію про їхню активність. У випадку з

iperf-клієнтом, який працює лише 30 секунд і автоматично завершує свою роботу, його контейнер може бути неактивним уже через короткий проміжок часу, що є передбаченою поведінкою (див. рис. 2.21).

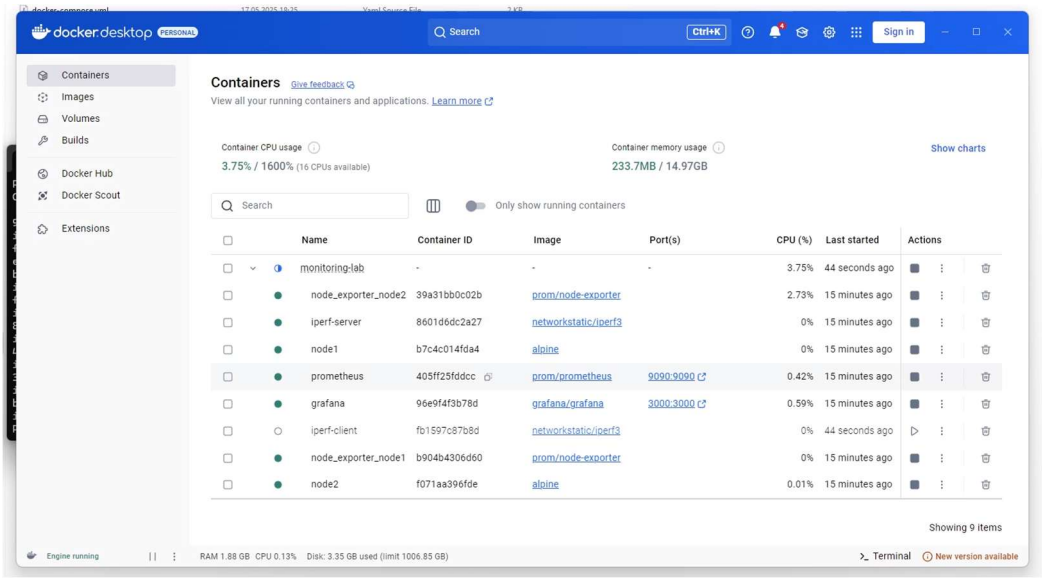


Рисунок 2.21 – Загальна інформація по розгорнутій системі

Після перевірки статусу контейнерів наступним кроком є відкриття веб-інтерфейсів Prometheus і Grafana. Prometheus доступний за адресою "http://localhost:9090", де інтерфейс завантажується без потреби у введенні логіна або пароля. Grafana відкривається за адресою "http://localhost:3000", де стандартні облікові дані для входу — це логін "admin" і пароль "admin" (див. рис. 2.22).

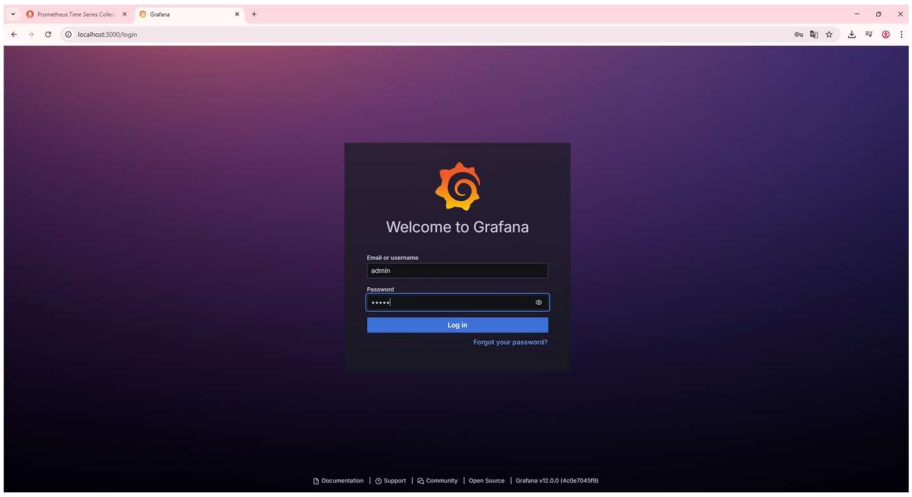


Рисунок 2.22 – Перший запуск засоба моніторингу Grafana

Після входу у Grafana необхідно додати джерело даних, вибравши Prometheus і вказавши його адресу як "http://prometheus:9090", що відповідає імені контейнера згідно з налаштуванням у "docker-compose" (див. рис. 2.23).

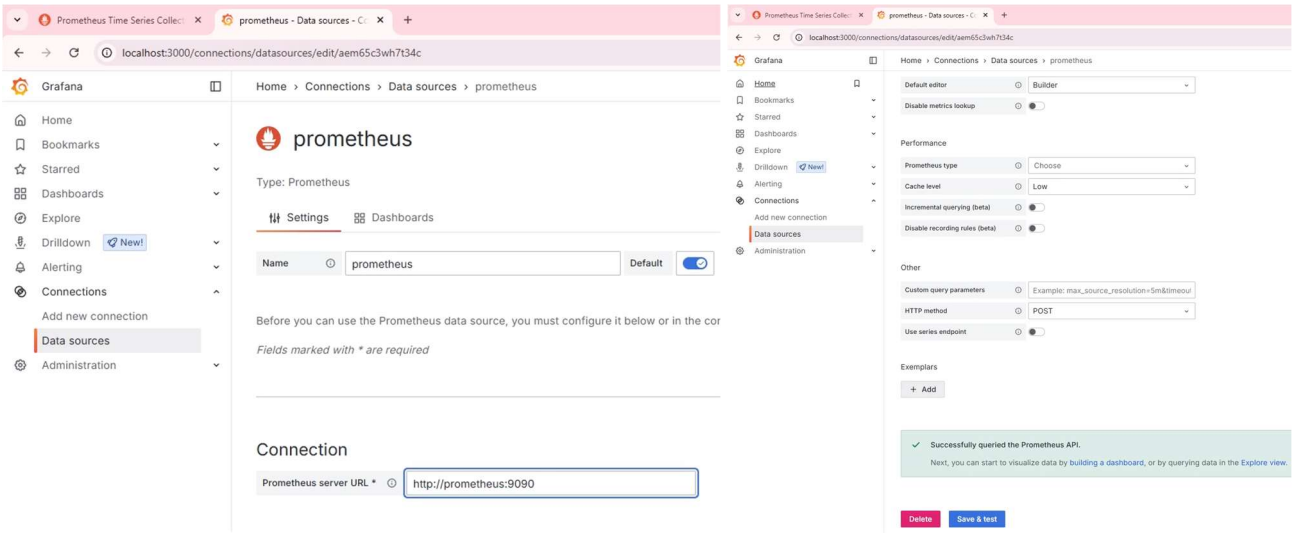


Рисунок 2.23 – Вибір Prometheus як джерела даних

Після під’єднання Prometheus до Grafana можна імпортувати готовий шаблон дашборда, наприклад, з "ID 1860", який відповідає "Node Exporter Full Dashboard" (див. рис. 2.24-2.25).

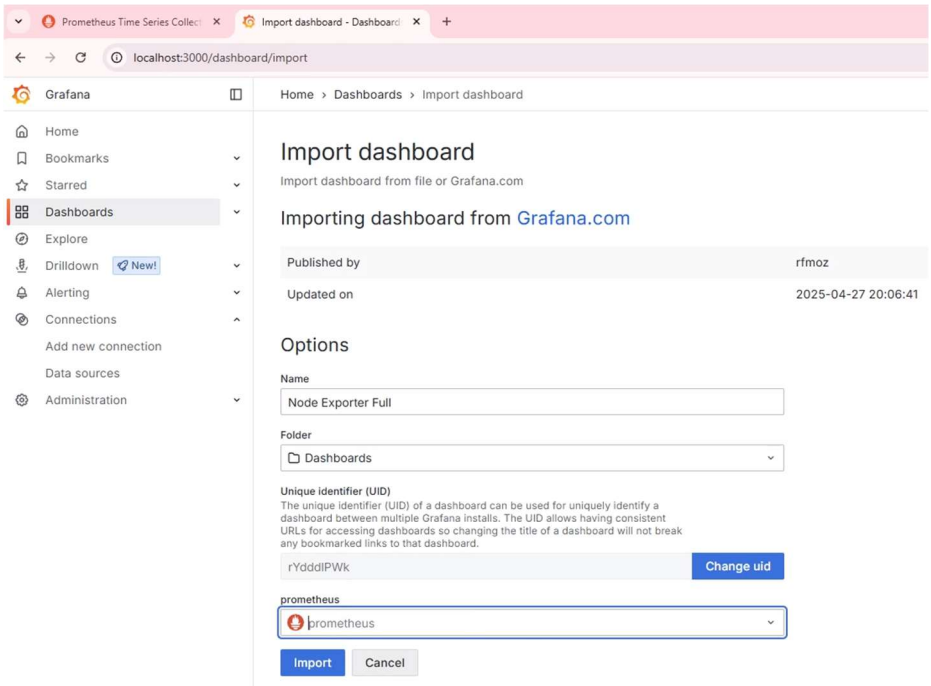


Рисунок 2.24 – Імпорт готового шаблона для агентів Node Exporter

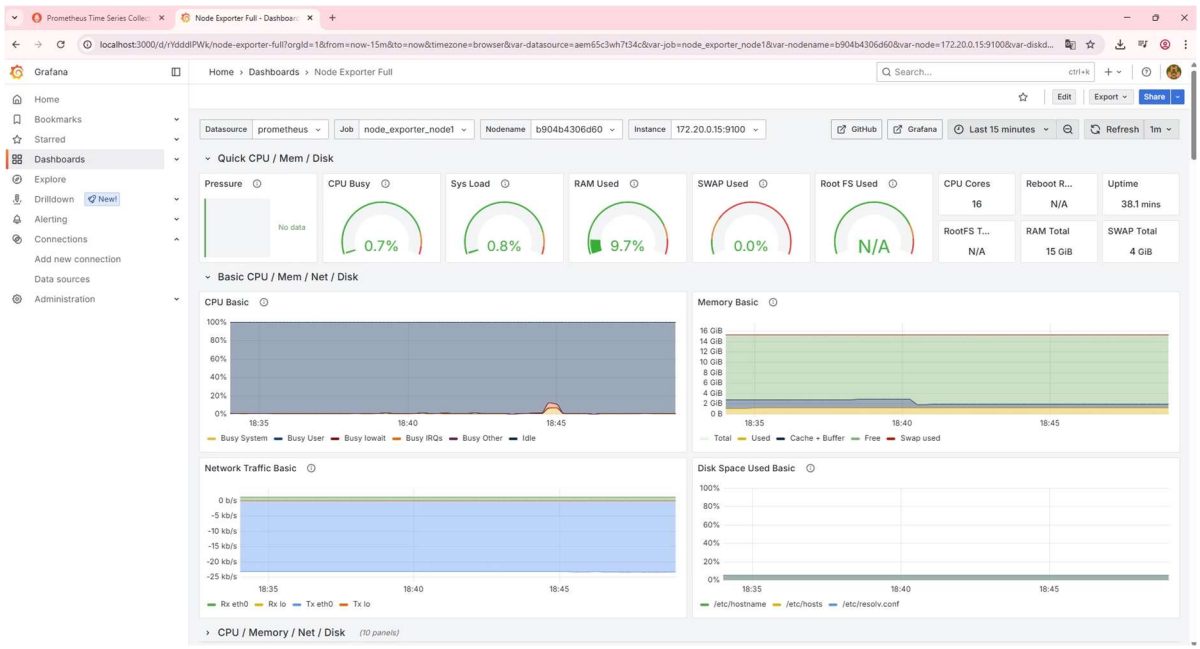


Рисунок 2.25 – Приклад готового дашборда для візуалізації метрик зі агентів Node Exporter в Grafana

Також можна скористатися попередньо створеним власним шаблоном, адаптованим під структуру конкретної системи. Під час використання кастомного шаблону необхідно враховувати, що метрики повинні точно відповідати назві полів і джерел, інакше графіки не відображатимуться коректно (див. рис. 2.26).

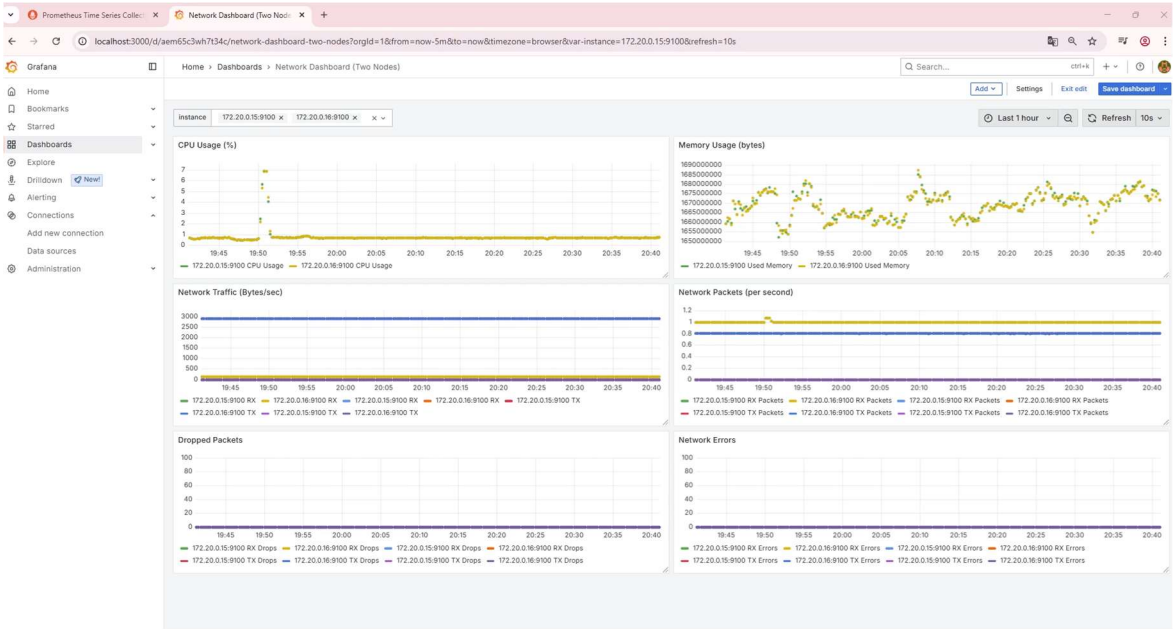


Рисунок 2.26 – Приклад власного налаштованого дашборда для візуалізації метрик зі агентів Node Exporter в Grafana

Якщо якийсь із вузлів буде недоступним, Prometheus це покаже в розділі статусів, а у Grafana просто зникнуть відповідні графіки (див. рис. 2.27).

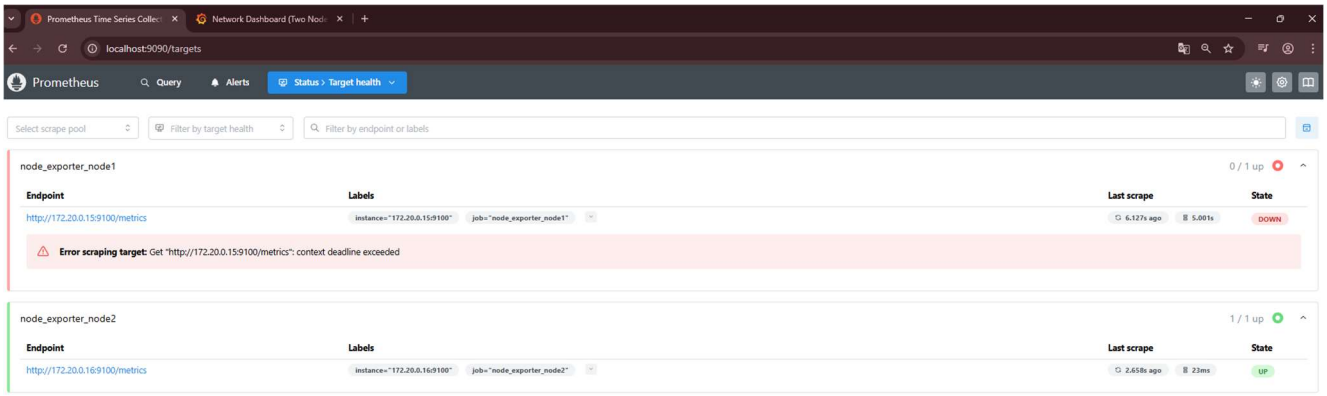


Рисунок 2.27 – Імітація недоступності одного із вузлів у Prometheus

Демонстрація роботи iperf і відображення результатів у Grafana. Після запуску системи і під'єднання всіх компонентів демонструється робота iperf для перевірки продуктивності з'єднання між вузлами. Інструмент "iperf3", що реалізований у двох контейнерах як сервер і клієнт, генерує контрольований трафік, вимірюючи швидкість передачі даних і надаючи телеметричні дані, які зчитуються агентами Node Exporter і передаються до Prometheus для подальшої візуалізації в Grafana. У конфігураційному файлі було задано параметри автоматичного запуску клієнта, що ініціалізує передачу даних упродовж 30 секунд, після чого вимикається, створюючи при цьому короткочасне, але інтенсивне навантаження на мережу. Ці зміни миттєво відображаються у Grafana на графіках, які демонструють активність процесора, обсяг використаної оперативної пам'яті, а також інтенсивність мережевої взаємодії у вигляді обсягу переданих байтів або кількості пакетів (див. рис. 2.28).

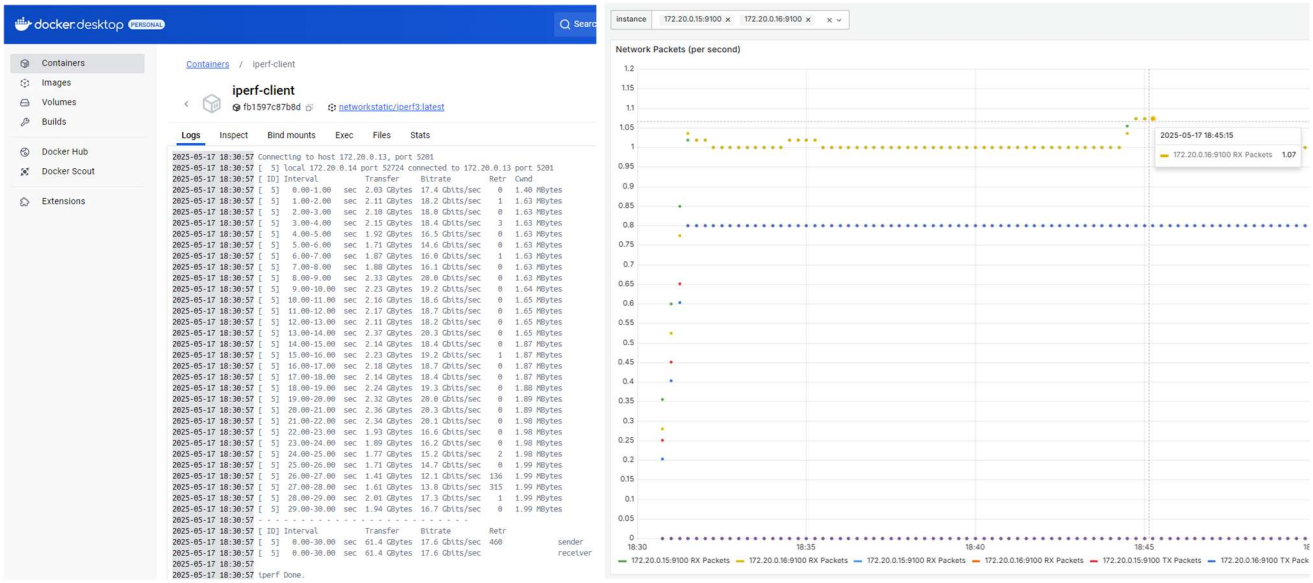


Рисунок 2.28 – Візуалізація роботи iperf у реальному часі

У разі, якщо конфігурація підтримує і додаткові параметри, такі як тип трафіку (TCP або UDP), тривалість передачі або ширина каналу, можна побачити ще точніші метрики, які відображаються на окремих панелях дашборда. Усе це дозволяє оцінити, наскільки стабільною є мережа в умовах навантаження, і які вузли або компоненти можуть бути критичними для роботи всієї інфраструктури. Такий підхід дозволяє протестувати систему в динаміці, візуально контролювати її поведінку і отримати практичне уявлення про продуктивність моделі взаємодії, що надалі може бути використано для аналізу, оптимізації і масштабування системи у нових експериментальних або виробничих умовах [26].

РОЗДІЛ 3 АНАЛІЗ ЕФЕКТИВНОСТІ ТА ПЕРСПЕКТИВИ РОЗВИТКУ

3.1 Методи покращення запропонованої системи

У базовому варіанті запропонована система ефективно реалізує моделювання мережевої взаємодії за допомогою контейнеризації, інструментів моніторингу та тестування навантаження, однак при переході від демонстраційного або навчального стенду до більш реалістичного масштабованого прототипу мережі доцільно розглядати покращення в кількох напрямках, серед яких:

— гнучке керування топологією;

— повноцінна маршрутизація;

— динамічна конфігурація;

— підвищення рівня безпеки.

Усе це в сукупності дозволить розширити функціональність моделі та наблизити її до умов реального середовища [15-16].

Впровадження Open vSwitch для складніших топологій. Розглядаючи одну з проблем, що виникає у межах стандартного мережевого стека Docker, можна відзначити обмеження, пов’язані з тим, що типова мережа за замовчуванням побудована на простій bridge-архітектурі з мінімальним рівнем комутації, що ускладнює створення більш гнучких сценаріїв з розгалуженими сегментами, VLAN, центральними маршрутизаторами або умовною маршрутизацією між зонами. Одним із ефективних рішень для вирішення цієї задачі виступає інтеграція з Open vSwitch, або коротко OVS, який є високопродуктивним програмним комутатором рівня L2/L3, що підтримує широкі мережеві протоколи, такі як VLAN, GRE, VXLAN, OpenFlow, і здатен емулювати складні топології з глибоким контролем за мережею, що значно розширює можливості моделювання.

Серед основних переваг впровадження Open vSwitch варто виокремити:

— повноцінне моделювання L2-комутаторів із підтримкою VLAN;

— trunk-з’єднань, ізоляції трафіку між групами контейнерів;

— ручного або автоматизованого формування логічних містків між контейнерами через OVS bridge;

— інтеграцію з OpenFlow.

Усе це відкриває доступ до програмного управління потоками і створення гнучких правил маршрутизації трафіку. Додатковою перевагою є масштабованість, адже рішення придатне для роботи із сотнями віртуальних портів, що дозволяє безперешкодно будувати складні сценарії тестування з великою кількістю логічних вузлів (див. рис. 3.1).

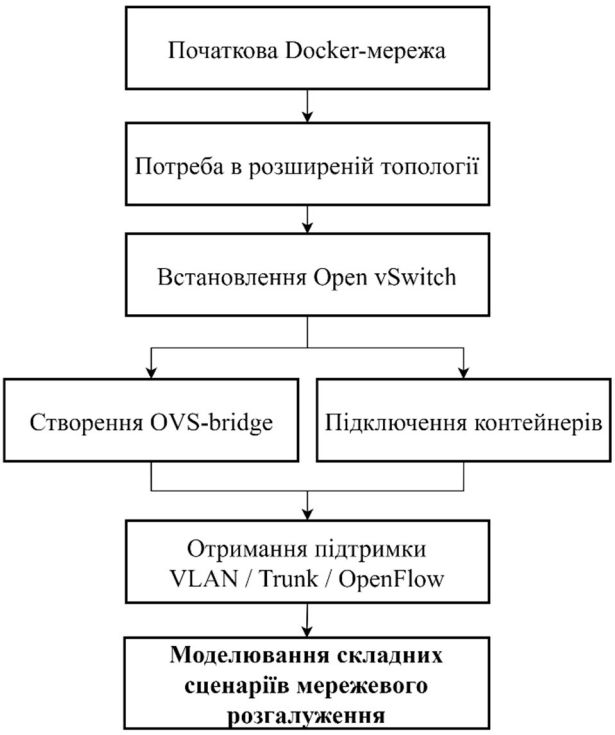


Рисунок 3.1 – Метод застосуванням Open vSwitch

Усе це дозволяє створити міст OVS і прив’язати до нього інтерфейс контейнера. У межах Docker можна використовувати підхід з опцією "--network=none" при запуску контейнера і подальшим ручним підключенням інтерфейсу до OVS, або створювати власні скрипти під’єднання. Також для кожного сегмента мережі можна створити окрему групу комутації, що дає змогу симулювати сценарії з окремими підмережами, точками маршрутизації, фільтрацією трафіку, які не

можуть бути реалізовані засобами стандартного bridge-режиму Docker Compose [15].

Інтеграція Open vSwitch дозволяє не лише відтворити архітектуру з централізованим маршрутизатором, кількома логічними сегментами та транзитними вузлами, але й здійснювати глибоке тестування сценаріїв, пов’язаних з керуванням потоками, оптимізацією трафіку, виявленням конфліктів у маршрутних таблицях або симуляцією балансування навантаження, що робить таку модернізацію важливим етапом при побудові більш реалістичного мережевого середовища.

Додавання FRRouting для маршрутизації між контейнерами. Ще однією з технічних обмежень, що спостерігається у базовій реалізації, є відсутність повноцінної маршрутизації на рівні L3, оскільки реалізація за замовчуванням передбачає лише базовий шлюз або використання bridge-інтерфейсів Docker, що не дозволяє повноцінно моделювати складні багаторівневі маршрутизовані сценарії. Рішенням у цьому випадку є використання стека FRRouting, скорочено FRR, який надає повноцінну підтримку динамічних протоколів маршрутизації, зокрема OSPF версій 2 і 3, BGP, RIP, Static Routes та EIGRP, при цьому він може бути як окремим маршрутизуючим вузлом, так і інтегруватися у будь-який контейнерний елемент. Його можна реалізувати як окремий контейнер із підключенням до кількох мереж, а сама конфігурація виконується через файл "frr.conf", де задаються потрібні протоколи та мережі, наприклад у випадку з OSPF типовий блок конфігурації може містити "router ospf, network 10.0.1.0/24 area 0, network 10.0.2.0/24 area 0", що дозволяє налаштувати взаємодію між підмережами у рамках єдиного домену маршрутизації (див. рис. 3.2).

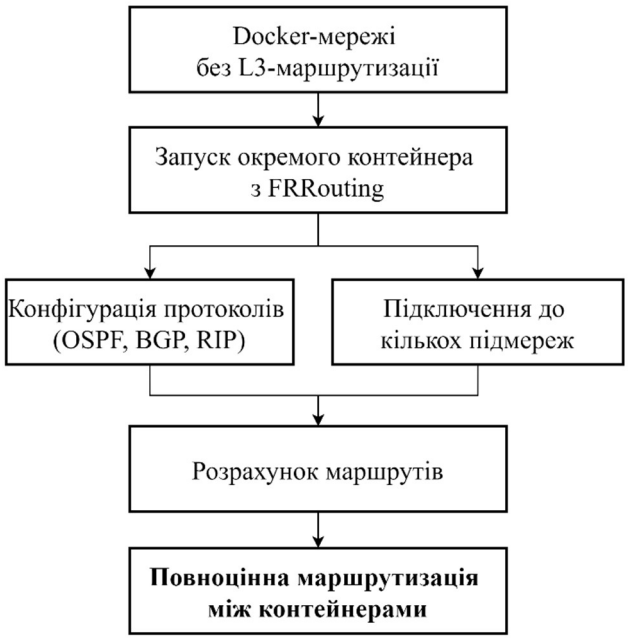


Рисунок 3.2 – Метод впровадження FRRouting у Docker

Основними перевагами такого підходу виступає підтримка динамічної адаптації маршрутів до змін топології, що дозволяє моделювати сценарії з відмовостійкістю, резервуванням каналів зв'язку, мультихопом, а також наближує поведінку системи до реальної мережі з інтернет-подібною структурою. Така гнучкість у маршрутизації дозволяє відтворити як сценарії enterprise-рівня, так і ISP-мережі, з кількома AS, BGP-сесіями або відмовами інтерфейсів, що значно підвищує дослідницьку цінність платформи при збереженні контейнерного підходу [16].

Створення REST API для динамічної конфігурації. Крім розширення мережевої логіки маршрутизації, у контексті побудови динамічного стенду виникає потреба в інтерактивному керуванні топологією, оскільки поточна реалізація побудована на основі статичних конфігурацій YAML, що потребують ручного редагування та перезапуску всіх сервісів у разі навіть незначної зміни у структурі. Для подолання цього обмеження доцільно впровадити REST API інтерфейс, який дозволить взаємодіяти з інфраструктурою у режимі реального часу. Такий інтерфейс забезпечить:

— можливість створення нових вузлів;

— запуску тестів з iperf між будь-якими двома контейнерами;

— зміни маршрутизації в FRRouting;

— динамічного моніторингу статусу кожного вузла через стандартні HTTP-запити.

Для реалізації такого функціоналу можуть використовуватись технології backend-розробки, зокрема Python з Flask або FastAPI, або Node.js, а також Docker SDK для Python, що дозволяє працювати із запущеними контейнерами через API. Додатково необхідні бібліотеки типу "requests", "docker", "subprocess", які реалізовують звернення до системи і контейнерного оточення (див. рис. 3.3).

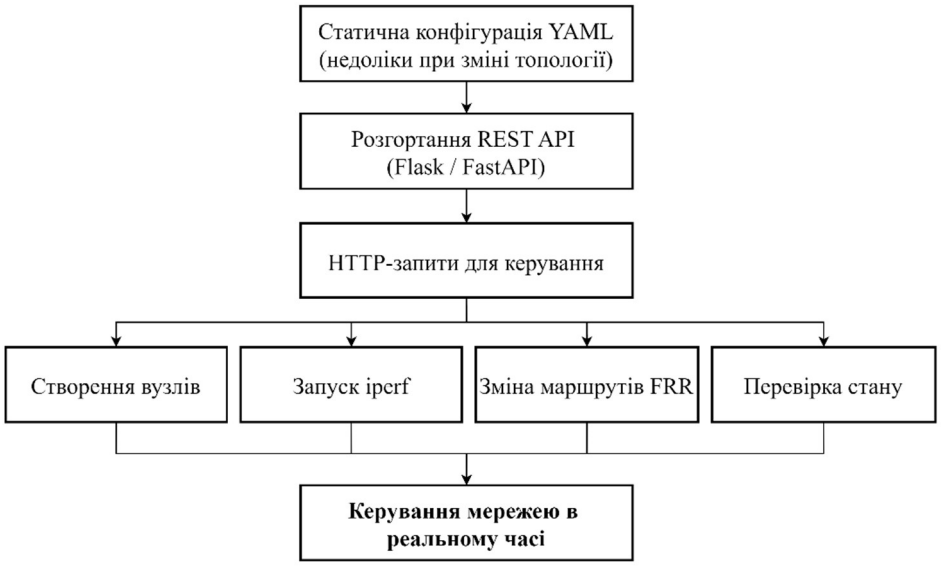


Рисунок 3.3 – Метод динамічного конфігурування REST API

Усе дозволяє одразу ініціювати новий контейнер, приєднаний до потрібної мережі без зміни загального конфігураційного файлу. Такий REST API рівень перетворює статичну структуру в динамічне кероване середовище, придатне для інтеграції з веб-інтерфейсами, CI/CD пайплайнами, автоматизованими тестами, симуляціями з оновленням у реальному часі або для застосування в освітньому процесі, де потрібна швидка зміна конфігурацій без складних ручних операцій, що загалом суттєво підвищує адаптивність системи до нових завдань.

Підвищення безпеки – ізоляція мережевих просторів. Також досить поширеною проблемою при побудові віртуалізованих мереж на основі контейнерних технологій залишається питання безпеки, зокрема недостатня

ізоляція мережевого трафіку у стандартному стеку Docker, оскільки мережеві простори, створені за замовчуванням через режими "bridge" або "overlay", зазвичай не мають чітко визначених обмежень між підмережами, не застосовують політики доступу за принципом "zero trust", що створює ризики неконтрольованого трафіку та ненавмисних або потенційно шкідливих взаємодій між контейнерами в єдиному середовищі.

Одним із дієвих підходів до вирішення цієї проблеми є впровадження логічної ізоляції мережевих просторів, коли середовище поділяється на кілька окремих сегментів за логікою функціонування, наприклад "frontend", "backend", "monitoring", "traffic-gen" або "security-zone", причому кожен логічний домен під’єднується лише до своєї віртуальної мережі, а далі здійснюється жорстке обмеження міжмережевого доступу за допомогою вбудованих інструментів Docker або зовнішніх фаєрволів. У такій реалізації контейнер Prometheus може бути налаштований так, щоб мати змогу опитувати лише ті вузли, на які вказано у конфігурації, тоді як між генератором трафіку "iperf" і системою моніторингу може бути повністю заблокована передача пакетів, що виключає ризики змішування каналів призначених для тестування із тими, які відповідають за системне спостереження.

Додатковим інструментом для підвищення ізоляції є використання технологій типу "macvlan" або "ipvlan", які дозволяють контейнерам отримувати IP-адреси з фізичної або зовнішньої підмережі, при цьому вони сприймаються як окремі фізичні пристрої з повноцінною мережею, що дає змогу будувати сегментовані зони з незалежним маршрутизуванням, де ізоляція виконується не на рівні Docker bridge, а через зовнішні L2-комутатори або маршрутизатори. У поєднанні з застосуванням "iptables" або "docker firewall" можна реалізувати складні правила фільтрації трафіку, що визначають допустимі напрямки передачі даних між логічними групами та окремими вузлами, забезпечуючи рівень безпеки, подібний до класичної корпоративної DMZ [15-16].

Також у перспективі варто впроваджувати більш гнучке управління доступом на основі імен контейнерів або ролей за допомогою сторонніх систем типу Traefik,

Consul або Cilium, які дозволяють створювати політики доступу, що описуються не IP-адресами, а логічними іменами або тегами, наприклад: дозволити "role=monitoring" підключення до "role=metrics", але заборонити доступ до "role=traffic-generator", що особливо важливо при масштабуванні моделі, де ручне керування конфігураціями стає надмірно громіздким. Такий підхід дозволяє автоматизувати контроль трафіку, реалізувати безпечну аутентифікацію і авторизацію між вузлами навіть у межах одного фізичного хоста [15-16].

Ізоляція мережевого трафіку у симульованому середовищі особливо важлива при моделюванні багаторівневих архітектур із проксі-серверами, тестування атак типу DoS або симуляції захищених зон, де важливо, щоб окремі вузли мали доступ лише до певного набору сервісів або мали обмеження на типи запитів, що дозволяє проводити не лише тести на пропускну здатність, а й тестування стабільності при підвищеному навантаженні в ізольованому середовищі або за умов неповної доступності (див. рис. 3.4).



Рисунок 3.4 – Метод ізоляції мережевих просторів через "iptables"

Узагальнюючи, покращення базової архітектури системи через впровадження Open vSwitch, FRRouting, RSTP та ізоляцію мережевих доменів

дозволяє вийти за межі демонстраційного середовища, перетворюючи платформу у повноцінну лабораторію з корпоративного рівня симуляцією, що підтримує:

— гнучке управління топологією;

— безпечну маршрутизацію;

— контроль доступу;

— масштабування.

Завдяки таким удосконаленням система стає придатною як для тестування складних сценаріїв, так і для автоматизації досліджень, експериментів, навчання майбутніх спеціалістів у сфері комп'ютерних мереж або кібербезпеки.

3.2 Порівняння з альтернативними рішеннями

Розроблена система моделювання комп'ютерних мереж загалом базується на використанні контейнерних технологій і інструментів з відкритим вихідним кодом таких як Docker, "iperf", Prometheus, Grafana, і для того щоб повноцінно оцінити ефективність і доцільність обраного підходу, доцільно також порівняти його з іншими популярними середовищами для мережевого моделювання та дослідження, зокрема Cisco Packet Tracer, GNS3 та Mininet, аби висвітлити переваги використання контейнерів, обґрунтувати існуючі обмеження та охарактеризувати продуктивність запропонованого рішення у вигляді цілісного програмно-апаратного комплексу [3-4, 17].

Порівняння основних інструментів моделювання мереж – Cisco Packet Tracer, GNS3 та Mininet. Розглядаючи Cisco Packet Tracer як одне з найбільш розповсюджених середовищ у навчальному процесі, варто зазначити, що це пропрієтарне програмне забезпечення, розроблене Cisco спеціально для симуляції мережевих пристроїв у віртуальному середовищі, і воно активно використовується в рамках Cisco Network Academy. Його ключовими перевагами:

— наявність інтуїтивного графічного інтерфейсу;

— підтримка протоколів STP, OSPF, EIGRP;

— моделювання роботи логіки комутаторів, маршрутизаторів і ПК.

Водночас основні обмеження полягають у закритості коду, неможливості інтеграції з реальними сервісами, такими як SNMP, Prometheus чи iperf, а також відсутності повноцінного Linux-оточення, що робить його непридатним для моделювання реальних інфраструктур із широким спектром інструментів. Крім того, Packet Tracer має низьку масштабованість, погано справляється з великими топологіями і не дозволяє реалізовувати сучасні підходи до телеметрії та моніторингу [3-4, 17].

GNS3 або Graphical Network Simulator 3 є потужною платформою для більш реалістичної емуляції мережевих пристроїв і систем, оскільки підтримує віртуалізацію через QEMU, Dynamips, Docker і може працювати з реальними образами Cisco IOS, MikroTik RouterOS та іншими. Однією з його сильних сторін є наявність графічного інтерфейсу для побудови складних мережевих схем із повноцінною підтримкою віртуальних машин, маршрутизаторів та комутаторів, а також можливість запуску Docker-контейнерів як частини мережевої інфраструктури. До переваг додається:

— широка підтримка спільноти;

— гнучка система розширень;

— підтримка сучасних сценаріїв для підготовки до сертифікацій.

Однак до недоліків можна віднести складність у налаштуванні, оскільки для запуску повноцінної симуляції потрібно розгортання окремого GNS3-сервера. Також платформа вимагає значних апаратних ресурсів, особливо при роботі з повноцінними віртуальними машинами, не має нативної підтримки REST API або систем аналітики на кшталт Prometheus, і не призначена для автоматизованих сценаріїв [4].

Ще одним варіантом для порівняння є Mininet — легкий емулятор програмно визначених мереж (SDN), який дозволяє створювати віртуальні топології з хостами, комутаторами та контролерами на базі OpenFlow.

Головними перевагами є:

— підтримка SDN-протоколів (OpenFlow, Ryu, POX);

— споживання мінімальних ресурсів хост-системи;

— можливість інтеграції з Python API (для автоматизованого створення сценаріїв).

Проте серед основних обмежень слід зазначити обмежену підтримку нестандартних або сторонніх протоколів, слабку розширюваність стандартного стеку мережевих функцій, жорстку прив’язку до "namespace" ядра Linux, а також відсутність повноцінної ізоляції контейнерів та процесів, що обмежує його використання у багатокористувацькому або безпечному середовищі.

Переваги використання контейнерного підходу. Якщо розглядати переваги контейнерного підходу порівняно з іншими варіантами моделювання, то насамперед можна виокремити високу ізоляцію та безпеку, оскільки контейнери функціонують у власному середовищі, що забезпечує відокремленість процесів, дозволяє одночасний запуск різних вузлів без конфліктів та дає можливість ефективно контролювати ресурси. Також однією з ключових характеристик є легка масштабованість, що дає змогу запускати десятки й сотні контейнерів із незначним навантаженням на систему, створюючи великі, розгалужені топології без потреби у потужному апаратному забезпеченні. Крім цього, контейнери мають дуже швидкий запуск, оскільки вони стартують за лічені секунди, що є недосяжним показником для повноцінних віртуальних машин. Реалістичність моделювання також зростає завдяки використанню справжніх Linux-утиліт, таких як "iperf", "netstat", "tcpdump", "tc" та інших, що дозволяє проводити глибоке дослідження продуктивності в умовах, наближених до промислових.

Окремою перевагою є підтримка автоматизації, зокрема інтеграція з CI/CD-процесами, що дозволяє створювати автоматизовані тести продуктивності, збирати метрики, перевіряти стабільність мережевих з’єднань і отримувати звіти у візуалізованому вигляді. Можливість підключення таких систем, як Jenkins, GitLab CI, GitHub Actions, значно розширює варіанти інтеграції для корпоративного або дослідницького використання.

Недоліки моделі на основі Docker. Водночас модель, що базується на Docker, має і низку обмежень. Найперше – відсутність графічного інтерфейсу, що вимагає від користувача постійної взаємодії з командним рядком або з конфігураційними файлами, що може бути незручним або складним для новачків. Потреба у знаннях CLI Linux, зокрема для запуску мережевих утиліт або налаштування маршрутизації між контейнерами, ускладнює початкове входження. Оскільки більшість змін у топології потребують редагування файлу "docker-compose.yml" і перезапуску всієї системи, це не дозволяє динамічно вносити корективи під час роботи, як це реалізовано в системах з підтримкою REST API або візуального конструктора. Також візуалізація обмежується побудовою графіків у Grafana, а візуального представлення самої мережі як структури, як це зроблено в Packet Tracer чи GNS3, немає [3-4, 17].

Щодо продуктивності, контейнерні рішення мають перевагу в ефективному використанні ресурсів. Наприклад, запуск двадцяти контейнерів із тестовими агентами типу "iperf-server" може споживати менше одного гігабайта оперативної пам’яті, тоді як подібна кількість віртуальних машин у GNS3 потребувала б значно більше. Пропускна здатність контейнерних мереж також залишається високою – до 10 Гбіт/с і більше при локальному трафіку, що наближається до нативного рівня, на відміну від обмеженої швидкості в симуляторах. Затримки в контейнерах мінімальні, оскільки вони працюють на ядрі хоста без значних накладних витрат, що дозволяє точно моделювати вплив навантаження, втрати пакетів і затримки. Завдяки Docker Compose нову топологію із десятків вузлів можна розгорнути менш ніж за 5 секунд, тоді як у GNS3 це потребувало б щонайменше 30 секунд або й більше, а в Packet Tracer — ручної побудови (див. табл. 3.1, рис. 3.5) [3-4, 17].

Таблиця 3.1 – Порівняння зі основними інструментами моделювання мереж

Критерій	**Cisco Packet Tracer**	**GNS3**	**Mininet**	**Запропонована система (Docker)**
Тип моделювання	Симуляція	Емуляція	Емуляція	**Емуляція реальних сервісів**
Підтримка реальних утиліт	Присутня	Частково	Частково	**Присутня**
Графічний інтерфейс	Присутній	Присутній	Відсуній	**Відсуній (CLI/API, можливе розширення)**
Масштабованість	Обмежена	Обмежена (через ресурси)	Середня	**Висока (контейнери)**
Складність налаштування	Низька	Висока	Середня	**Середня**
Автоматизація	Відсутня	Обмежено	Присутня (через Python)	**Присутня (через Compose, скрипти, API)**
Використання ресурсів	Низьке	Високе	Низьке	**Помірне**
Підтримка SDN/контролерів	Відсутня	Присутня	Присутня (OpenFlow)	**Присутня (можна додати вручну)**

Продовження таблиці 3.1 – Порівняння зі основними інструментами моделювання мереж

Критерій	**Cisco Packet Tracer**	**GNS3**	**Mininet**	**Запропонована система (Docker)**
Ізоляція між вузлами	Обмежена	Висока (VM)	Легка (namespace)	**Висока (контейнери)**
Ліцензія / відкритість	Пропрієтарна	Open Source	Open Source	**Open Source**
Підтримка моніторингу (Prometheus/Grafana)	Відсутня	Відсутня	Відсутня	**Присутня**
Продуктивність (мережа)	Низька	Середня	Висока	**Висока (майже нативна)**
Час запуску топології	5–10 сек	>30 сек	~5 сек	**2–5 сек**

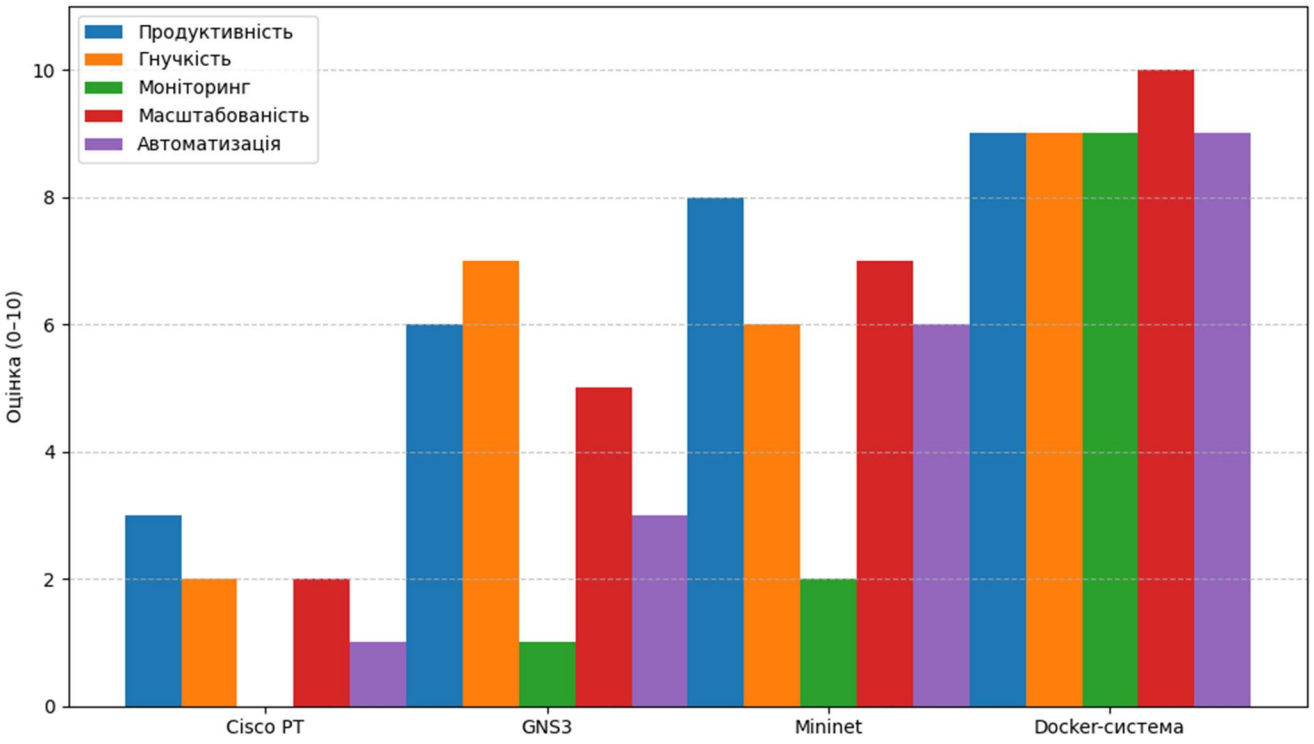


Рисунок 3.5 – Діаграма аналізу технічної ефективності інструментів моделювання мереж

Усе це дає підстави дійти висновку, що система моделювання мереж на основі контейнеризації не тільки не поступається популярним інструментам у гнучкості, функціональності й автоматизації, а в багатьох випадках може їх перевершувати. Особливо вигідною вона є для сценаріїв, де потрібна реалістична емуляція з опорою на справжні інструменти Linux та інтеграцію з сучасними CI/CD-середовищами, а відсутність графічного інтерфейсу може бути компенсована через REST API, веб-панелі керування або побудову власного дашборду в майбутньому, що робить цю платформу придатною як для навчального використання, так і для практичних досліджень та інфраструктурного тестування.

3.3 Напрямки подальшого розвитку системи

У процесі розробки та реалізації програмно-апаратного комплексу для моделювання й оптимізації комп'ютерних мереж було виявлено низку перспективних напрямків, які можуть бути реалізовані в подальшому для підвищення функціональності системи, її інтеграційної сумісності з іншими інструментами та загального рівня придатності до використання в дослідницьких, виробничих та освітніх цілях, що відкриває нові горизонти для розвитку як у локальних середовищах, так і в корпоративному або хмарному застосуванні. Одним із таких перспективних напрямків є забезпечення безпекового моніторингу за рахунок інтеграції із SIEM-системами, які дозволяють проводити централізований збір та аналіз подій, а також оперативне реагування на інциденти у межах віртуального середовища [18-24].

Інтеграція з SIEM-системами для безпекового моніторингу. У сучасних умовах, коли навіть тестові лабораторії стають об'єктами потенційного зловмисного впливу, важливим завданням стає побудова захищеної та контрольованої моделі цифрового середовища, де можна вивчати вплив різних атак, симулювати загрози і тренувати засоби їх виявлення. Саме у цьому контексті

актуальною є реалізація інтеграції з платформами на зразок Elastic Stack у складі Elasticsearch, Logstash, Kibana, а також Beats або ж альтернативними рішеннями типу Graylog, які можуть акумулювати журнали подій із мережевих агентів, системних логів або інструментів контролю трафіку, таких як Suricata чи Zeek. Така інтеграція дає змогу в реальному часі моніторити активність у віртуальних мережах, виявляти відхилення від норми, проводити аналіз вторгнень або спроб сканування портів, а також створювати тренувальні сценарії для навчання системних адміністраторів, інженерів з інформаційної безпеки або студентів профільних спеціальностей. Наприклад, один із практичних сценаріїв може полягати в тому, що контейнер з Suricata підключається до віртуального інтерфейсу всередині Docker-мережі, прослуховує весь трафік, а далі передає його у вигляді подій до Elasticsearch, де на підставі заздалегідь налаштованих шаблонів Kibana формує інтерактивні панелі з деталізованою інформацією щодо типів атак, індикаторів компрометації чи аномальної поведінки (див. рис. 3.6).

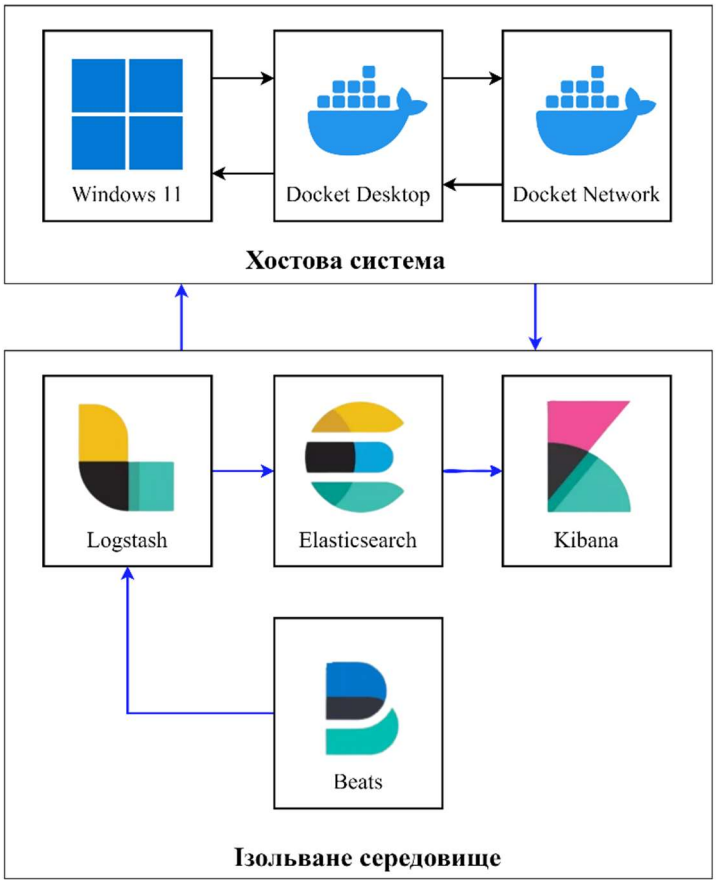


Рисунок 3.6 – Схема інтеграції з SIEM-системами

Застосування такого підходу дає змогу не тільки візуалізувати повну картину подій у системі, а й значно підвищити реалістичність тренувань, проводити аудит дій у мережевому середовищі, відпрацьовувати сценарії реагування на інциденти та, за потреби, адаптувати інструменти до вимог певного підприємства або сфери. Таким чином, інтеграція з SIEM-рішеннями стає одним із ключових напрямів подальшого розвитку комплексу, що перетворює його з класичного симулятора на гнучке й потужне середовище для вивчення кіберзагроз, тестування алгоритмів захисту та забезпечення контролю над усіма аспектами цифрової взаємодії всередині віртуальної інфраструктури.

Автоматизація тестів за допомогою Ansible або Terraform. Автоматизація є необхідним компонентом подальшого вдосконалення будь-якої інфраструктурної системи, особливо коли йдеться про регулярне виконання тестів продуктивності, перевірку стабільності або запуск діагностичних сценаріїв у динамічному середовищі, що унеможливлює ручне втручання при кожному запуску чи зміні конфігурації. У рамках запропонованого комплексу така автоматизація може бути досягнута за допомогою інструментів на кшталт Ansible або Terraform, які дозволяють ефективно керувати інфраструктурою, описуючи її у вигляді коду та сценаріїв. Ansible використовується для автоматизації налаштування контейнерів, встановлення пакетів, запуску скриптів або перевірки стану сервісів, що особливо важливо у випадках багаторазового розгортання однотипних мережевих стендів, де критичною є швидкість конфігурації та уніфікація середовищ. Використання Terraform у свою чергу дозволяє описати всю інфраструктуру як код, автоматично створюючи необхідні вузли, віртуальні мережі, правила безпеки та підключення до зовнішніх систем, що дозволяє легко переносити локальні розгортання в хмару, масштабувати їх і управляти цим централізовано через репозиторії та CI/CD конвеєри (див. рис. 3.7).



Рисунок 3.7 – Автоматизація CI/CD-тестів через Ansible

Наприклад, типовий "playbook" у Ansible може не тільки генерувати "docker-compose.yml", але й запускати сценарії iperf-тестів, збирати метрики з Prometheus та автоматично формувати звіти по доступності, затримках і продуктивності системи, що дає змогу реалізувати повноцінну модель тестування за графіком або після певної події, наприклад коміту в репозиторій [18-21].

Хмарна реалізація – AWS, Azure. Перехід до хмарної моделі, у свою чергу, відкриває додаткові можливості щодо масштабування, мобільності та незалежності від локальних обчислювальних ресурсів. У контексті запропонованої системи це може означати перенесення контейнерної інфраструктури на такі сервіси, як AWS ECS, Azure Container Instances, Google Cloud Run або подібні хмарні платформи, що дозволяють за лічені хвилини розгортати симульовані мережеві середовища з підтримкою усіх необхідних елементів – від вузлів і сервісів до засобів моніторингу та збору метрик. При цьому вхідною точкою для такого автоматизованого створення середовища може бути той же Terraform, який забезпечує опис хмарної інфраструктури у вигляді кодових шаблонів, що включають мережеві групи, політики безпеки, екземпляри віртуальних машин та підключення до централізованих систем моніторингу на кшталт AWS CloudWatch чи Azure Monitor, що дозволяє не лише здійснювати миттєвий запуск або знищення середовища на вимогу, але й забезпечити доступ до системи з будь-якої точки без потреби

встановлення будь-якого програмного забезпечення локально, а також створити сценарій використання, при якому, наприклад, технічний відділ компанії може розгортати стенд за запитом на час експерименту, навчання чи тестування, оплачуючи лише фактичне використання ресурсів (див. рис. 3.8) [22-24].



Рисунок 3.8 – Розгортання середовища у AWS через Terraform

У підсумку, запропонована система моделювання мереж не лише адаптується до потреб локального використання, а й легко інтегрується у сучасну екосистему автоматизованих інфраструктур і хмарних сервісів, що робить її універсальним інструментом для дослідників, адміністраторів, викладачів та інженерів, які працюють у сфері мережевих технологій, автоматизації та безпеки. Її подальший розвиток може охоплювати не лише технічне вдосконалення, а й створення API-доступу, веб-інтерфейсів для спрощеного керування, автоматичної генерації топологій або імпорту з шаблонів, що розширює спектр застосування від навчання до повноцінного промислового використання.

ВИСНОВКИ

У роботі було реалізовано архітектуру програмно-апаратного комплексу для моделювання та оптимізації комп’ютерних мереж із використанням контейнерних технологій. Основу системи склали ізольовані середовища Docker, у яких розгорнуто вузли віртуальної мережі, інструменти генерації трафіку та модулі моніторингу. Побудована схема дозволила гнучко конфігурувати мережеві зв’язки, моделювати трафік і аналізувати взаємодію між компонентами без використання фізичного обладнання.

Для тестування продуктивності мережі застосовувались iperf-клієнт і сервер, що дозволило здійснити вимірювання пропускної здатності й стабільності з’єднання між віртуальними вузлами. Інструменти Prometheus і Grafana забезпечили збір метрик та наочну візуалізацію параметрів системи в режимі реального часу. Зокрема, відображались навантаження на CPU, мережевий трафік, частота запитів та інші характеристики, що дозволяє оперативно оцінювати стан мережі.

Побудована система показала хорошу масштабованість та зручність у використанні для дослідницьких і навчальних цілей. За допомогою "docker-compose" спрощено управління конфігурацією і повторне розгортання експериментального середовища. Архітектура дозволяє легко розширювати функціональність за рахунок додавання нових сервісів, вузлів або протоколів, що робить її гнучкою платформою для тестування різноманітних сценаріїв.

У результаті аналізу існуючих підходів та реалізованої системи були запропоновані напрями подальшої оптимізації, зокрема використання механізмів автоматичної маршрутизації, балансування навантаження та вдосконалення інтерфейсів візуалізації. Порівняння з іншими рішеннями підтвердило доцільність використання відкритих інструментів для побудови ефективного середовища моделювання мереж із мінімальними затратами на інфраструктуру.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Container Virtualization vs VMs: Benefits & Differences. Scale Computing. URL: https://www.scalecomputing.com/resources/container-virtualization-explained (дата звернення: 25.02.2025)
2. Containerization vs. Virtualization: 7 Technical Differences. Trianz. URL: https://www.trianz.com/insights/containerization-vs-virtualization (дата звернення: 28.02.2025)
3. ns-3 | a discrete-event network simulator for internet systems. URL: https://www.nsnam.org/ (дата звернення: 03.03.2025)
4. GNS3 | The software that empowers network professionals. URL: https://www.gns3.com/ (дата звернення: 06.03.2025)
5. What is Cisco Packet Tracer? | Free Training and Download. Cisco Networking Academy. URL: https://www.netacad.com/cisco-packet-tracer (дата звернення: 09.03.2025)
6. Networking - Docker Docs. URL: https://docs.docker.com/engine/network/ (дата звернення: 12.03.2025)
7. Docker Networking Model — Introduction. Medium. URL: https://hwchiu.medium.com/docker-networking-model-introduction-194a2a2c9b68 (дата звернення: 15.03.2025)
8. Open vSwitch. URL: https://www.openvswitch.org/ (дата звернення: 18.03.2025)
9. FRRouting. URL: https://frrouting.org/ (дата звернення: 21.03.2025)
10. Grafana & Prometheus SNMP: beginner's network monitoring guide. Grafana Labs. URL: https://grafana.com/blog/2022/01/19/a-beginners-guide-to-network-monitoring-with-grafana-and-prometheus/ (дата звернення: 24.03.2025)
11. Docker Networking - Basics, Network Types & Examples. Spacelift. URL: https://spacelift.io/blog/docker-networking (дата звернення: 27.03.2025)
12. Free Range Routing (FRR) demo using Docker Compose. GitHub. URL: https://github.com/ksator/frrouting_demo (дата звернення: 30.03.2025)
13. Grafana & Prometheus SNMP: advanced network monitoring guide. Grafana Labs. URL: https://grafana.com/blog/2022/02/01/an-advanced-guide-to-network-monitoring-with-grafana-and-prometheus/ (дата звернення: 02.04.2025)
14. Docker — FRR latest documentation. FRRouting. URL: https://docs.frrouting.org/projects/dev-guide/en/latest/building-docker.html (дата звернення: 05.04.2025)
15. Open vSwitch: Self-service networks. OpenStack Docs. URL: https://docs.openstack.org/neutron/latest/admin/deploy-ovs-selfservice.html (дата звернення: 08.04.2025)
16. Simulating Inter Branch Offices Networking Using FRRouting and Docker Containers. Medium. URL: https://barrihs.medium.com/simulating-inter-branch-offices-

networking-using-frrouting-and-docker-containers-46f6e4bca935 (дата звернення: 11.04.2025)

17. Cisco Packet Tracer: Free Computer Networking Simulation Tool. Cisco Networking Academy. URL: https://www.netacad.com/learning-collections/cisco-packet-tracer (дата звернення: 14.04.2025)

18. Ansible for Network Automation. Ansible Documentation. URL: https://docs.ansible.com/ansible/latest/network/index.html (дата звернення: 17.04.2025)

19. Testing Strategies — Ansible Community Documentation. Ansible Documentation. URL: https://docs.ansible.com/ansible/latest/reference_appendices/test_strategies.html (дата звернення: 20.04.2025)

20. What is Infrastructure as Code with Terraform? - HashiCorp Developer. URL: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code (дата звернення: 23.04.2025)

21. Terraform Infrastructure as Code (IaC) Guide With Examples. Spacelift. URL: https://spacelift.io/blog/terraform-infrastructure-as-code (дата звернення: 26.04.2025)

22. How to Deploy Docker Containers - AWS. Amazon Web Services. URL: https://aws.amazon.com/getting-started/hands-on/deploy-docker-containers/ (дата звернення: 29.04.2025)

23. Quickstart: Deploy a container instance in Azure using the Azure portal. Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/container-instances/container-instances-quickstart-portal (дата звернення: 02.05.2025)

24. Azure Container Instances documentation - Learn Microsoft. URL: https://learn.microsoft.com/en-us/azure/container-instances/ (дата звернення: 05.05.2025)

25. Сладь В.О., Лащевська Н.О. Інтелектуальний підхід до оптимізації комп'ютерних мереж за допомогою програмно-апаратного комплексу моделювання. «Застосування програмного забезпечення в ІКТ» : Всеукр. науково-техн. конф., м.Київ, 24 квітня 2025 р. С.647-649.

26. Сладь В.О., Лащевська Н.О. Застосування програмно-апаратного комплексу для моделювання і оптимізації комп'ютерних мереж у середовищі хмарних технологій. «Сучасні інтелектуальні інформаційні технології в науці та освіті»» : V Всеукр. науково-практ. конф., м.Київ, 15 травня 2025 р. (подано до публікації).

Додаток А

Конфігурація "docker-compose.yml" для розгортання системи моніторингу та тестування мережі

```yaml
version: "3.8"

services:
  node1:
    image: alpine
    container_name: node1
    command: tail -f /dev/null
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.11

  node2:
    image: alpine
    container_name: node2
    command: tail -f /dev/null
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.12

  iperf-server:
    image: networkstatic/iperf3
    container_name: iperf-server
    command: -s
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.13

  iperf-client:
    image: networkstatic/iperf3
    container_name: iperf-client
    depends_on:
      - iperf-server
    entrypoint: >
      sh -c "sleep 5 && iperf3 -c 172.20.0.13 -t 30"
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.14

  node_exporter_node1:
    image: prom/node-exporter
    container_name: node_exporter_node1
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.15

  node_exporter_node2:
    image: prom/node-exporter
    container_name: node_exporter_node2
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.16

  prometheus:
    image: prom/prometheus
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
```

```
      - "9090:9090"
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.17

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    depends_on:
      - prometheus
    networks:
      monitoring_net:
        ipv4_address: 172.20.0.18

networks:
  monitoring_net:
    driver: bridge
    ipam:
      config:
        - subnet: 172.20.0.0/24
```

Додаток Б

Конфігурація дашборда Grafana для візуалізації метрик мережі

```
{
  "annotations": {
    "list": []
  },
  "editable": true,
  "fiscalYearStartMonth": 0,
  "graphTooltip": 0,
  "id": 8,
  "links": [],
  "panels": [
    {
      "type": "graph",
      "title": "Network Traffic (Bytes/sec)",
      "id": 1,
      "datasource": {
        "type": "prometheus",
        "uid": "aem65c3wh7t34c"
      },
      "targets": [
        {
          "expr":
"rate(node_network_receive_bytes_total{instance=~"$instance"}[1m])",
          "legendFormat": "{{instance}} RX",
          "refId": "A"
        },
        {
          "expr":
"rate(node_network_transmit_bytes_total{instance=~"$instance"}[1m])",
          "legendFormat": "{{instance}} TX",
          "refId": "B"
        }
      ],
      "gridPos": {
        "h": 9,
        "w": 24,
        "x": 0,
        "y": 0
      }
    },
    {
      "type": "graph",
      "title": "Network Packets (per second)",
      "id": 2,
      "datasource": {
        "type": "prometheus",
        "uid": "aem65c3wh7t34c"
      },
      "targets": [
        {
          "expr":
"rate(node_network_receive_packets_total{instance=~"$instance"}[1m])",
          "legendFormat": "{{instance}} RX Packets",
          "refId": "C"
        },
        {
          "expr":
"rate(node_network_transmit_packets_total{instance=~"$instance"}[1m])",
          "legendFormat": "{{instance}} TX Packets",
          "refId": "D"
```

Продовження додатка Б

```
        }
      ],
      "gridPos": {
        "h": 9,
        "w": 24,
        "x": 0,
        "y": 9
      }
    },
    {
      "type": "graph",
      "title": "Network Errors",
      "id": 3,
      "datasource": {
        "type": "prometheus",
        "uid": "aem65c3wh7t34c"
      },
      "targets": [
        {
          "expr":
"rate(node_network_receive_errs_total{instance=~"$instance"}[1m])",
          "legendFormat": "{{instance}} RX Errors",
          "refId": "E"
        },
        {
          "expr":
"rate(node_network_transmit_errs_total{instance=~"$instance"}[1m])",
          "legendFormat": "{{instance}} TX Errors",
          "refId": "F"
        }
      ],
      "gridPos": {
        "h": 8,
        "w": 24,
        "x": 0,
        "y": 18
      }
    },
    {
      "type": "graph",
      "title": "Dropped Packets",
      "id": 4,
      "datasource": {
        "type": "prometheus",
        "uid": "aem65c3wh7t34c"
      },
      "targets": [
        {
          "expr":
"rate(node_network_receive_drop_total{instance=~"$instance"}[1m])",
          "legendFormat": "{{instance}} RX Drops",
          "refId": "G"
        },
        {
          "expr":
"rate(node_network_transmit_drop_total{instance=~"$instance"}[1m])",
          "legendFormat": "{{instance}} TX Drops",
          "refId": "H"
        }
      ],
      "gridPos": {
```

Продовження додатка Б


```
        "h": 8,
        "w": 24,
        "x": 0,
        "y": 26
      }
    },
    {
      "type": "graph",
      "title": "CPU Usage (%)",
      "id": 5,
      "datasource": {
        "type": "prometheus",
        "uid": "aem65c3wh7t34c"
      },
      "targets": [
        {
          "expr": "100 - (avg by (instance)
(rate(node_cpu_seconds_total{mode="idle",instance=~"$instance"}[1m])) * 100)",
          "legendFormat": "{{instance}} CPU Usage",
          "refId": "I"
        }
      ],
      "gridPos": {
        "h": 8,
        "w": 24,
        "x": 0,
        "y": 34
      }
    },
    {
      "type": "graph",
      "title": "Memory Usage (bytes)",
      "id": 6,
      "datasource": {
        "type": "prometheus",
        "uid": "aem65c3wh7t34c"
      },
      "targets": [
        {
          "expr": "node_memory_MemTotal_bytes{instance=~"$instance"} -
node_memory_MemAvailable_bytes{instance=~"$instance"}",
          "legendFormat": "{{instance}} Used Memory",
          "refId": "J"
        }
      ],
      "gridPos": {
        "h": 8,
        "w": 24,
        "x": 0,
        "y": 42
      }
    }
  ],
  "preload": false,
  "refresh": "10s",
  "schemaVersion": 41,
  "tags": [],
  "templating": {
    "list": [
      {
        "name": "instance",
```

Продовження додатка Б


```
        "type": "query",
        "datasource": {
          "type": "prometheus",
          "uid": "aem65c3wh7t34c"
        },
        "query": "label_values(node_network_receive_bytes_total, instance)",
        "multi": true,
        "includeAll": false,
        "refresh": 1,
        "sort": 1
      }
    ]
  },
  "time": {
    "from": "now-1h",
    "to": "now"
  },
  "timepicker": {
    "refresh_intervals": [
      "5s",
      "10s",
      "30s",
      "1m",
      "5m",
      "15m",
      "30m",
      "1h"
    ],
    "time_options": [
      "5m",
      "15m",
      "1h",
      "6h",
      "12h",
      "24h",
      "2d",
      "7d",
      "30d"
    ]
  },
  "timezone": "",
  "title": "Network Dashboard (Two Nodes)",
  "uid": "aem65c3wh7t34c",
  "version": 6
}
```

Додаток В

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Комп'ютерної інженерії



КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

НА ТЕМУ: «ПРОГРАМНО-АПАРАТНИЙ КОМПЛЕКС ДЛЯ МОДЕЛЮВАННЯ ТА ОПТИМІЗАЦІЇ КОМП'ЮТЕРНИХ МЕРЕЖ»

Виконав студент групи КІД-42
Сладь В'ячеслав Олександрович
Керівник дипломної роботи:
Асєєва Людмила Анатоліївна,
phD (д-р філософії), доц.

Київ – 2025

Мета, об’єкт, предмет дослідження

Мета роботи – спрощення процесу моделювання та оптимізації комп’ютерних мереж за рахунок використання принципів контейнеризації, інструментів збору метрик та візуалізації продуктивності

Об’єкт дослідження – процес функціонування комп’ютерних мереж у віртуалізованому середовищі

Предмет дослідження – технології віртуалізації, засоби тестування мережевого трафіку та системи моніторингу для моделювання та оптимізації мереж

Актуальність роботи

У роботі представлено підхід до побудови програмно-апаратного комплексу для моделювання комп'ютерних мереж із використанням інструментів віртуалізації та контейнеризації. Змодельовано віртуальну мережу, яка складається з кількох ізольованих вузлів, на яких відтворюються типові сценарії мережевого трафіку за допомогою iperf. Для збору і аналізу метрик продуктивності інтегровано Prometheus і Grafana, що дозволило отримати візуальні уявлення про параметри затримок, пропускної здатності та втрат пакетів.

Особливу увагу приділено архітектурі тестового середовища, реалізації контейнерної інфраструктури та можливостям масштабування. Проведено порівняльний аналіз інструментів, що використовуються у практиці мережевого тестування, виявлено переваги застосування контейнеризації та автоматизованого моніторингу в освітньому і дослідницькому контексті.

Застосування контейнерезації у моделюванні та оптимізації мереж

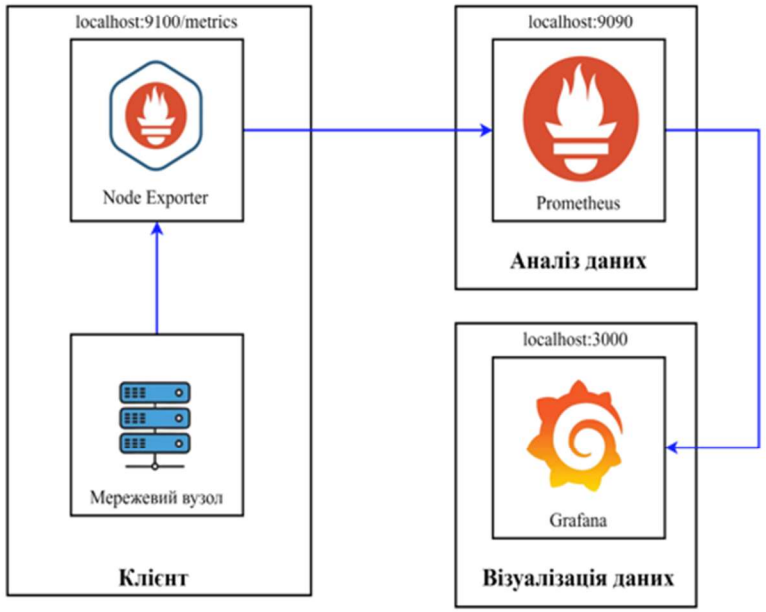


Рис. 1.16 – Інтеграція моніторингу у моделювальне середовище

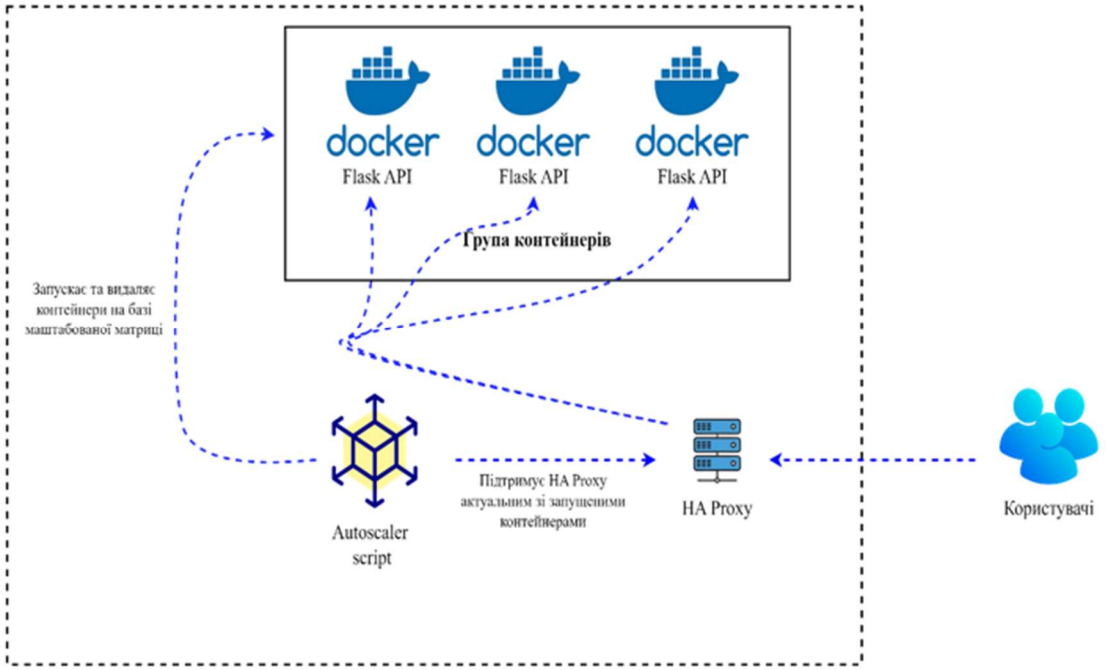


Рис. 1.12 – Оптимізація мережі через контейнерне масштабування

Загальна архітектура запропонованої системи

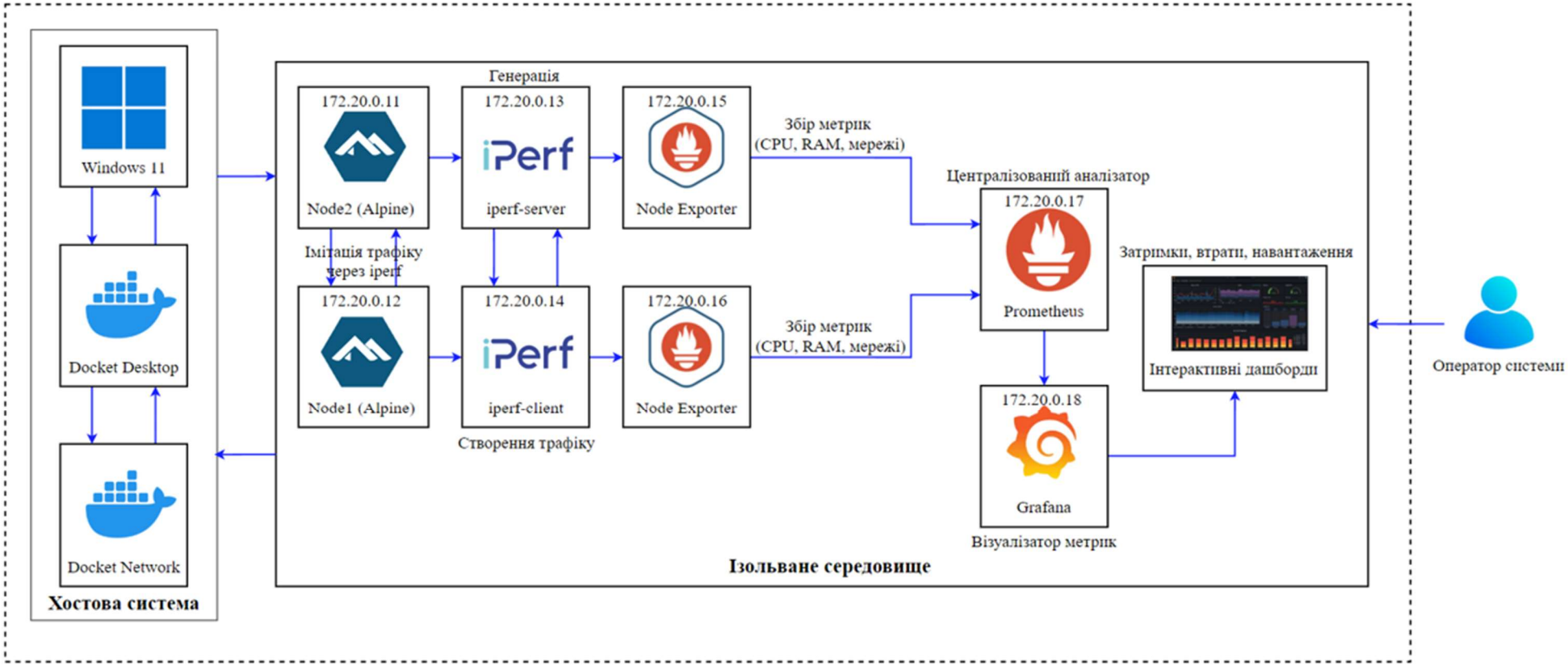


Рис. 2.1 – Архітектура ізольованої системи моделювання та тестування мереж зі використанням платформи контейнерезації

Розгортання віртуального ізольваного середовища

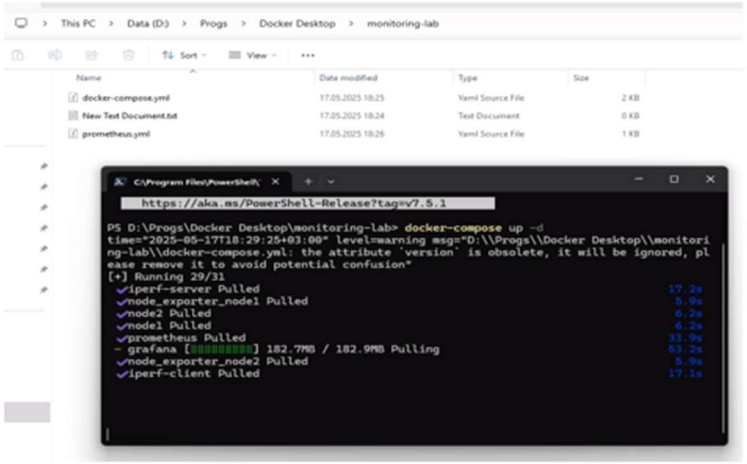


Рис. 2.19 – Запуск середовища через "docker-compose up"

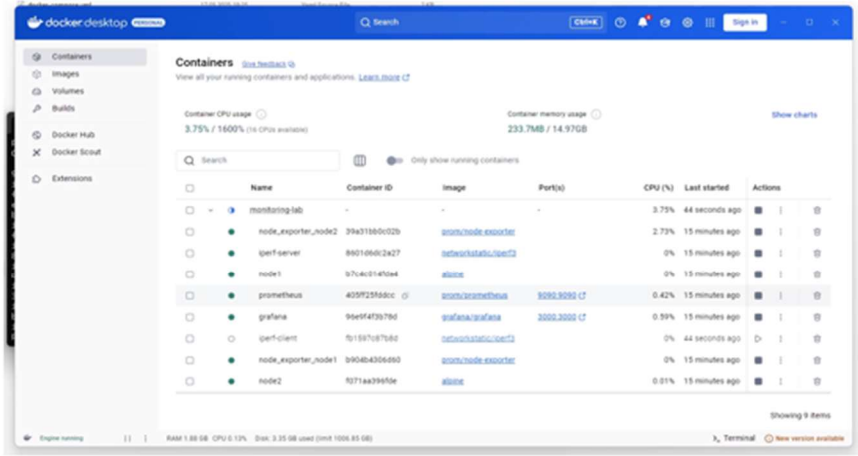


Рис. 2.21 – Загальна інформація по розгорнутій системі

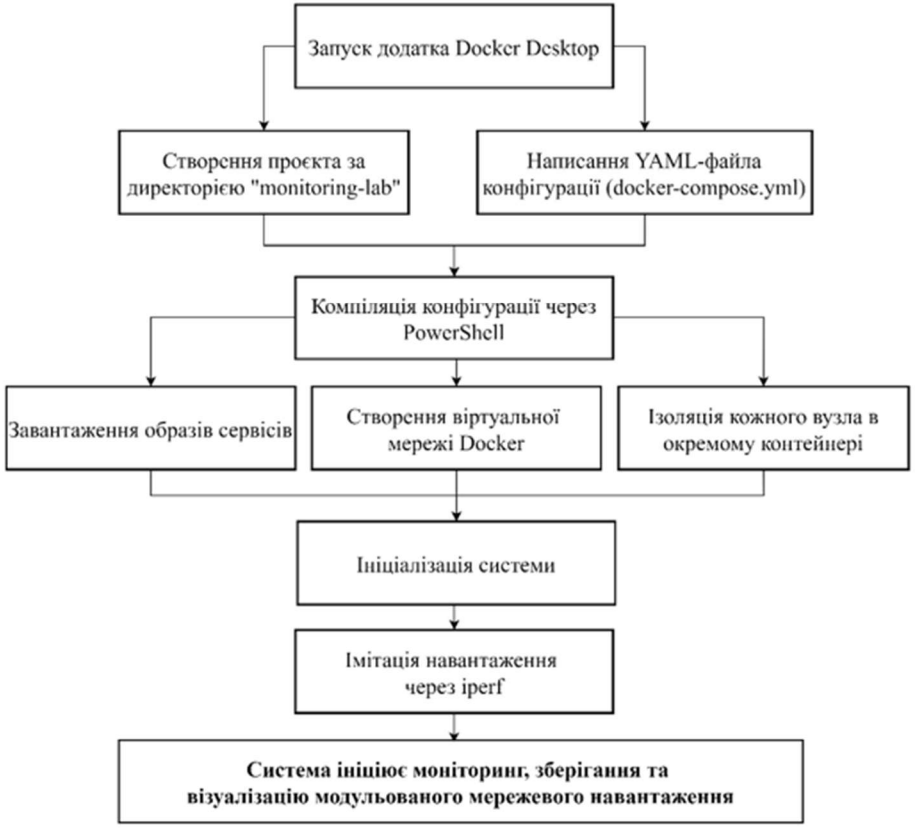


Рис. 2.2 – Концептуальна схема розгортання та функціонування системи

Генерація трафіку за допомогою iperf

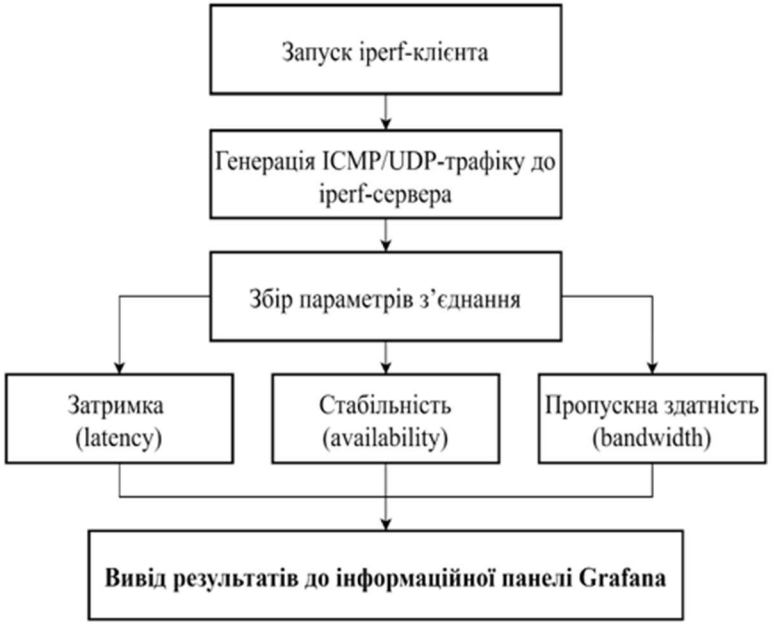


Рис. 2.3 – Принцип використання "iperf" для імітації трафіку

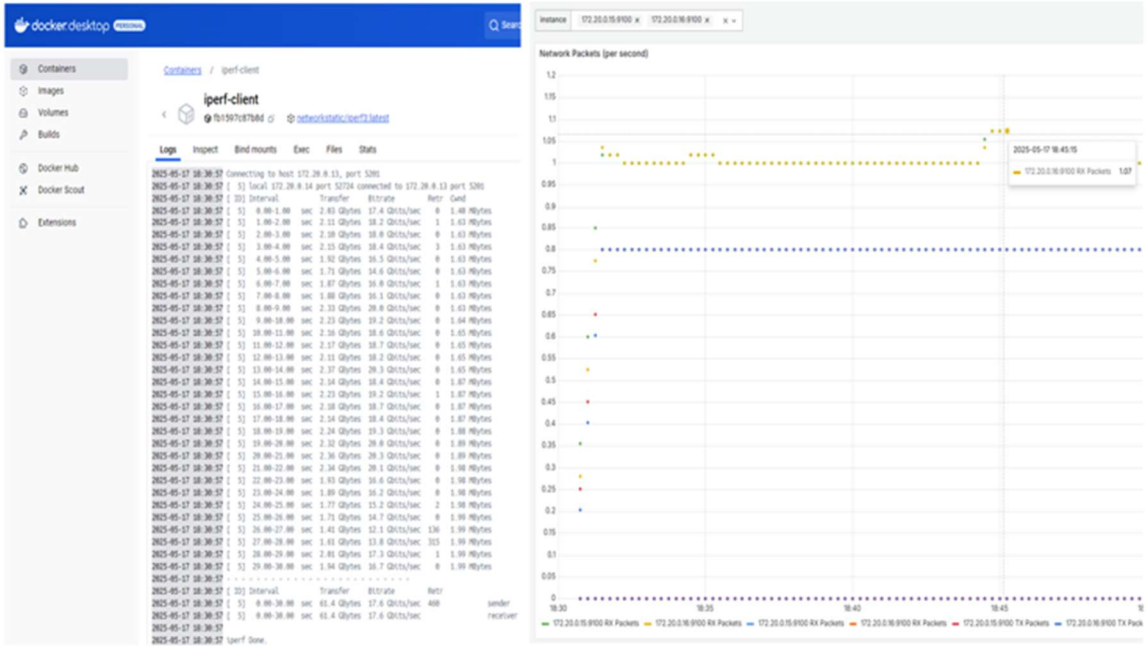


Рис. 2.28 – Візуалізація роботи iperf у реальному часі

Алгоритми функціонування у ізольованій системі

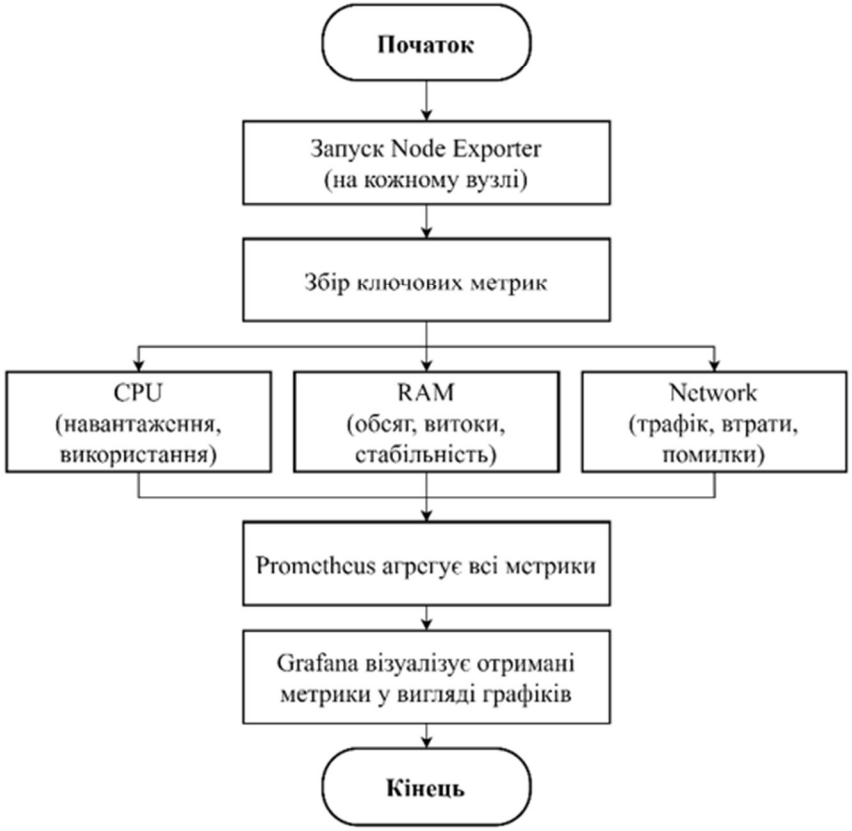


Рис. 2.12 – Алгоритм збору й аналізу метрик

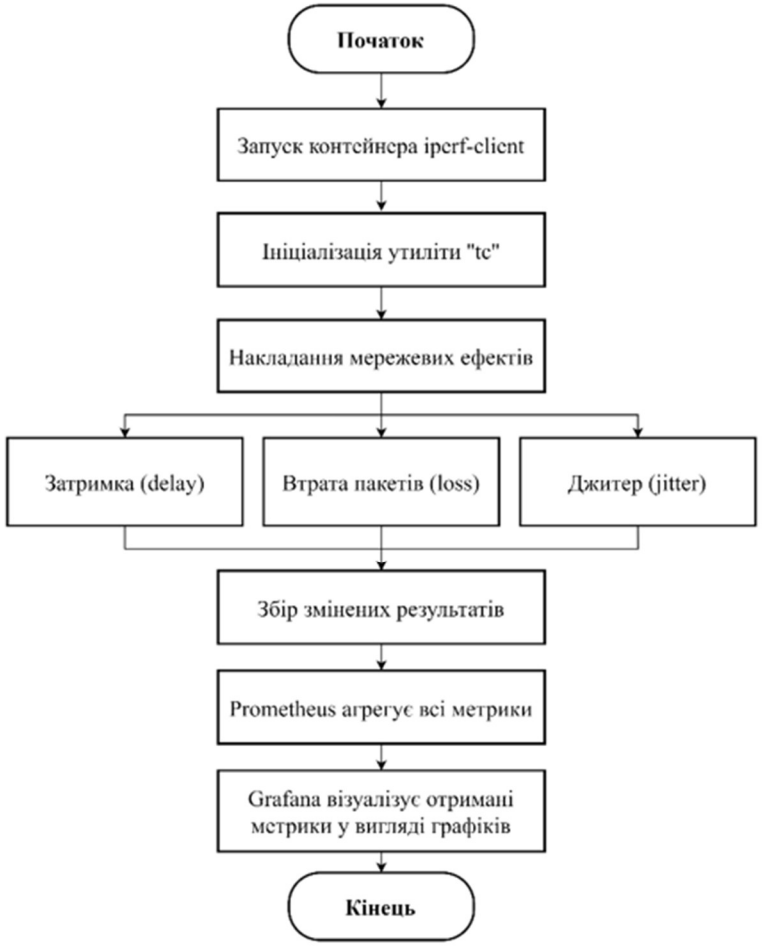


Рис. 2.16 – Алгоритм емуляції затримок або втрат пакетів

Демонстрація функціонування системи (візуалізація)

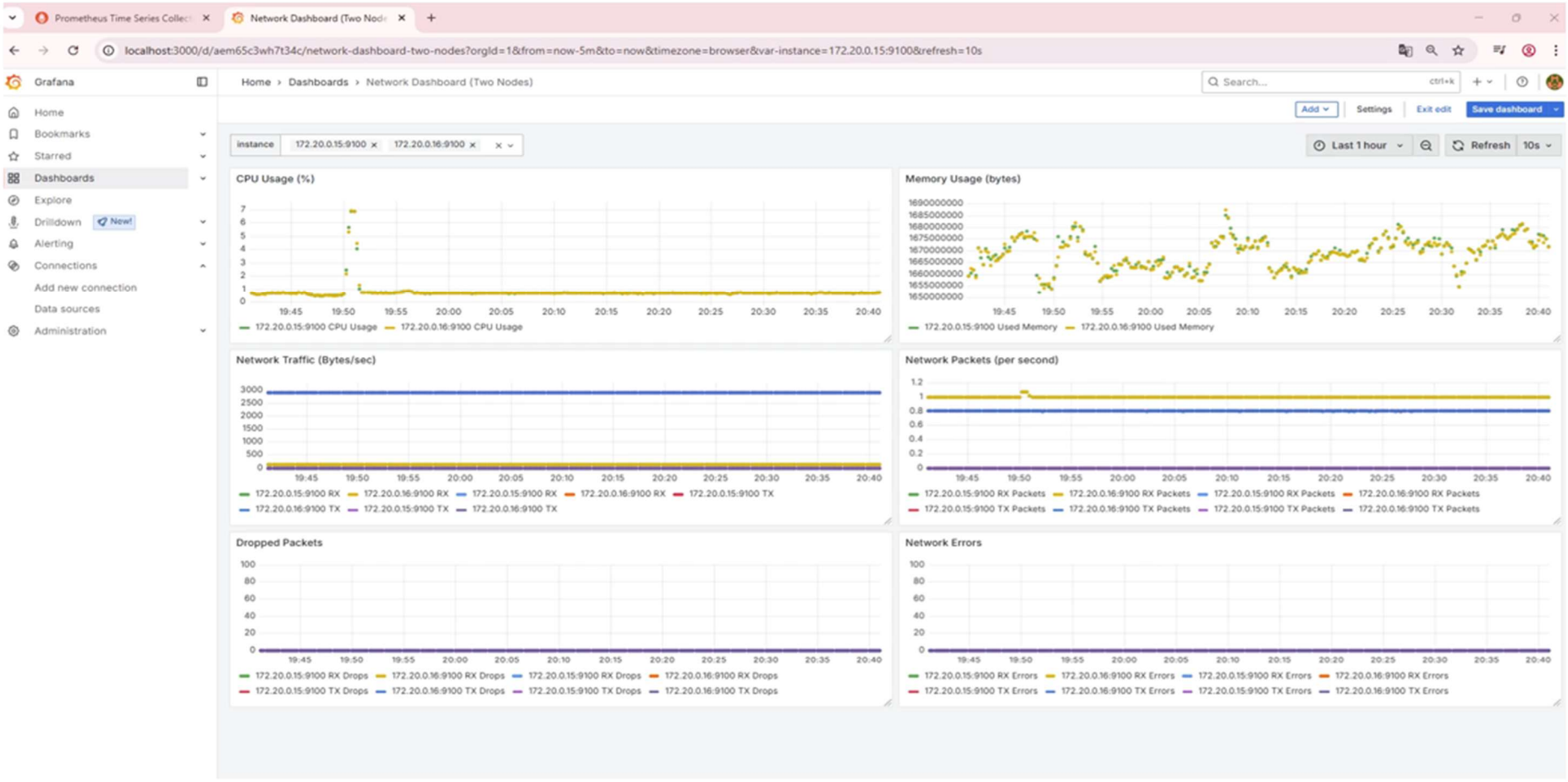


Рис. 2.26 – Приклад власного налаштованого дашборда для візуалізації метрик зі агентів Node Exporter в Grafana 9

Методи покращення існуючої системи

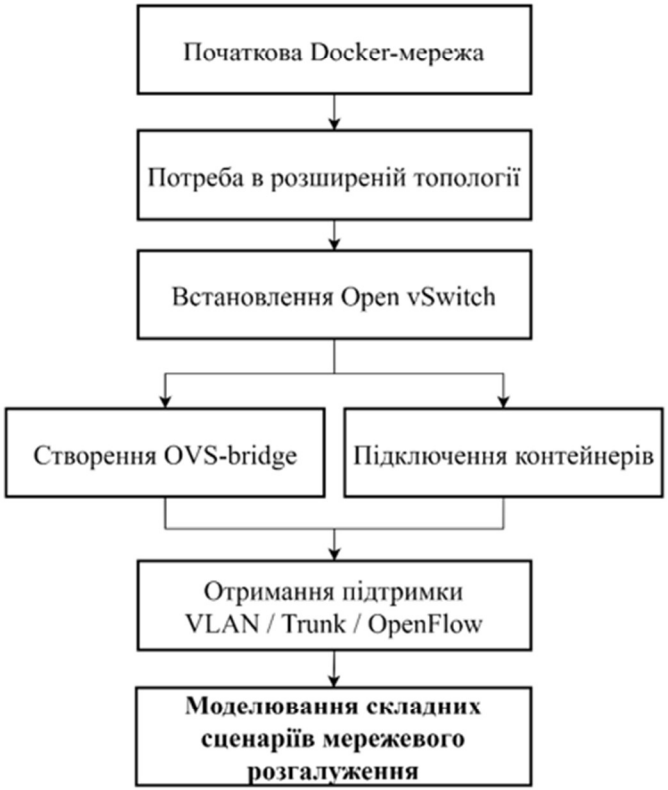


Рис. 3.1 – Метод застосуванням Open vSwitch



Рис. 3.4 – Метод ізоляції мережевих просторів через "iptables"

Порівняння запропонованої системи зі альтернативами

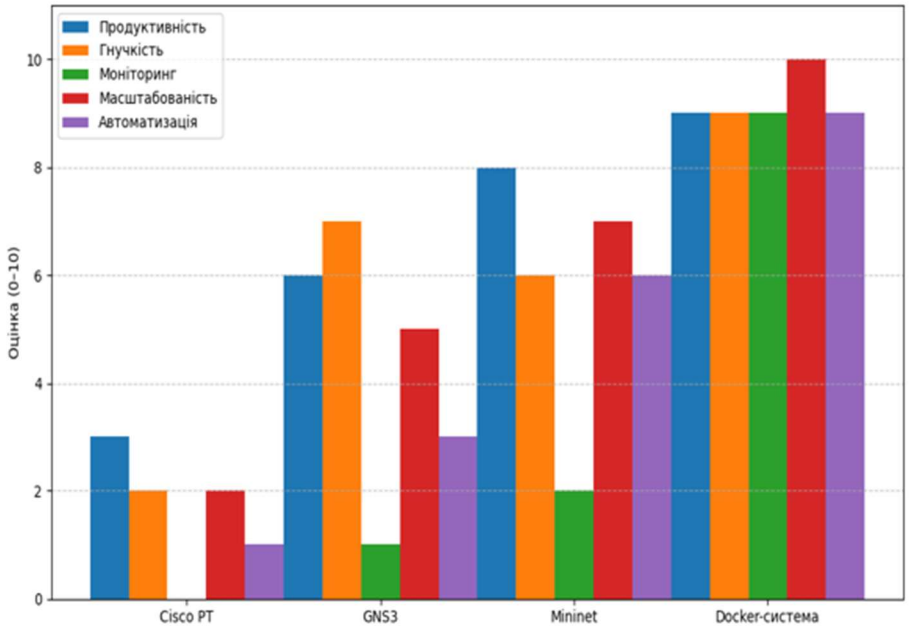


Рис. 3.5 – Діаграма аналізу технічної ефективності інструментів моделювання мереж

Табл. 3.1 – Порівняння зі основними інструментами моделювання мереж

Критерій	**Cisco Packet Tracer**	**GNS3**	**Mininet**	**Запропонована система (Docker)**
Тип моделювання	Симуляція	Емуляція	Емуляція	**Емуляція реальних сервісів**
Підтримка реальних утиліт	Присутня	Частково	Частково	**Присутня**
Графічний інтерфейс	Присутній	Присутній	Відсуній	**Відсуній (CLI/API, можливе розширення)**
Масштабованість	Обмежена	Обмежена (через ресурси)	Середня	**Висока (контейнери)**
Складність налаштування	Низька	Висока	Середня	**Середня**
Автоматизація	Відсутня	Обмежено	Присутня (через Python)	**Присутня (через Compose, скрипти, API)**
Використання ресурсів	Низьке	Високе	Низьке	**Помірне**
Підтримка SDN/контролерів	Відсутня	Присутня	Присутня (OpenFlow)	**Присутня (можна додати вручну)**
Ізоляція між вузлами	Обмежена	Висока (VM)	Легка (namespace)	**Висока (контейнери)**
Ліцензія / відкритість	Пропрієтарна	Open Source	Open Source	**Open Source**
Підтримка моніторингу (Prometheus/Grafana)	Відсутня	Відсутня	Відсутня	**Присутня**
Продуктивність (мережа)	Низька	Середня	Висока	**Висока (майже нативна)**
Час запуску топології	5–10 сек	>30 сек	~5 сек	**2–5 сек**

Висновки

У роботі реалізовано архітектуру програмно-апаратного комплексу для моделювання та оптимізації комп'ютерних мереж на основі контейнерних технологій. Ізольовані середовища Docker дозволили розгорнути віртуальні вузли, інструменти генерації трафіку та модулі моніторингу, що забезпечило повноцінне моделювання мережевої взаємодії без залучення фізичного обладнання. Тестування продуктивності проводилось за допомогою iperf, а метрики продуктивності збирались і візуалізувались у реальному часі за допомогою Prometheus та Grafana, що дало змогу оперативно оцінювати стан системи.

Побудована система продемонструвала високу масштабованість, гнучкість та придатність для навчальних і дослідницьких цілей. Завдяки використанню docker-compose було значно спрощено процес налаштування та повторного розгортання середовища. Система підтримує розширення функціоналу шляхом додавання нових сервісів і протоколів, а також дозволяє реалізувати подальшу оптимізацію, зокрема впровадження автоматичної маршрутизації, балансування навантаження та удосконалення засобів візуалізації. Порівняння з іншими інструментами підтвердило доцільність використання відкритих рішень для створення ефективного середовища моделювання з мінімальними витратами.

Публікації та апробація роботи

1. Сладь В.О., Лащевська Н.О. Інтелектуальний підхід до оптимізації комп'ютерних мереж за допомогою програмно-апаратного комплексу моделювання. «Застосування програмного забезпечення в ІКТ» : Всеукр. науково-техн. конф., м.Київ, 24 квітня 2025 р. С.647-649.
2. Сладь В.О., Лащевська Н.О. Застосування програмно-апаратного комплексу для моделювання і оптимізації комп'ютерних мереж у середовищі хмарних технологій. «Сучасні інтелектуальні інформаційні технології в науці та освіті»» : V Всеукр. науково-практ. конф., м.Київ, 15 травня 2025 р. (подано до публікації).