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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «МЕРЕЖЕВА ІНФРАСТРУКТУРА З ПРОГРАМНО-ВИЗНАЧЕНИМ УПРАВЛІННЯМ МАРШРУТИЗАЦІЄЮ»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерної інженерії
(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

_____ *Ілля ЛЕВАДА*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КІД-42
Ілля ЛЕВАДА
Ім'я, ПРІЗВИЩЕ
Керівник: Віра БАБЕНКО
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут Інформаційних технологій

Кафедра 123 Комп'ютерної інженерії
Ступінь вищої освіти Бакалавр
Спеціальність 123 Комп'ютерної інженерії
Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедрою Наталія ЛАЩЕВСЬКА
Ім'я, ПРІЗВИЩЕ
«_____» _____ 2026р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Левада Ілля Анатолійович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Мережева інфраструктура з програмно-визначеним управлінням маршрутизацією
керівник кваліфікаційної роботи Віра БАБЕНКО,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» 02 2026р. № 51
2. Строк подання кваліфікаційної роботи «08» 05 2026р.
3. Вихідні дані до кваліфікаційної роботи: Науково-технічна література, методи системного аналізу, методи математичного моделювання для розрахунку алгоритмів маршрутизації, математичний апарат, емпіричні методи
Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Дослідження сучасних тенденцій та перспектив еволюції програмно-конфігурованих мереж
2. Інструментальні засоби та програмне середовище для проектування й симуляції sdn-мереж
3. Практичні підходи до проектування та розгортання топологій програмно-конфігурованих мереж у середовищі mininet
4. Перелік ілюстративного матеріалу: *презентація*
5. Дата видачі завдання «09» 02 2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	25.02-24.03.2026	Виконано
2	Обробка матеріалу	23.03-05.04.2026	Виконано
3	Розробка теоретичної частини	06.04-10.04.2026	Виконано
4	Розробка практичної частини	11.04-20.04.2026	Виконано
5	Висновки за результатами аналізу	21.04-06.05.2026	Виконано
6	Оформлення пояснювальної записки	07.05-08.05.2026	Виконано
7	Розробка презентації	09.05-11.05.2026	Виконано
8	Передзахист	27.05-29.05.2026	Виконано
9	Здача в деканат	02.06-05.06.2026	Виконано

Здобувач вищої освіти _____ Ілля ЛЕВАДА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Віра БАБЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 64 стор., 24 рис., 1 табл., 25 джерел.

Мета дослідження – Дослідження та розробка мережевої інфраструктури з використанням концепції програмно-визначеного управління маршрутизацією (SDN) для підвищення гнучкості та ефективності передачі даних.

Об'єкт дослідження – Мережева інфраструктура на базі технології програмно-конфігурованих мереж.

Предмет дослідження – Методи та алгоритми програмно-визначеного управління маршрутизацією.

Короткий зміст роботи: Сучасні комп'ютерні мережі стикаються з проблемою масштабованості та складності управління через свою жорстку, децентралізовану архітектуру. Традиційний підхід, де кожен маршрутизатор самостійно приймає рішення на основі протоколів динамічної маршрутизації, ускладнює швидке реагування на зміни топології та впровадження нових сервісів. Тому виникає потреба в інноваційних підходах до управління трафіком, які дозволяють централізовано контролювати всю мережеву інфраструктуру. Таку можливість надає концепція програмно-конфігурованих мереж (SDN).

Вона забезпечує фізичне та логічне відокремлення площини управління від площини передачі даних. Завдяки цьому маршрутизатори перетворюються на прості пристрої пересилання пакетів, а вся інтелектуальна логіка маршрутизації зосереджується у централізованому програмному контролері. Для того, щоб правильно реалізувати цей підхід, варто розглянути принципи взаємодії контролера з мережевим обладнанням за допомогою стандартизованих протоколів, таких як OpenFlow.

На основі проведеного дослідження було з'ясовано актуальність використання SDN в сучасних корпоративних мережах та ЦОД, особливості побудови таких інфраструктур, алгоритми балансування навантаження та проведено практичне моделювання маршрутизації у віртуальному середовищі Mininet.

КЛЮЧОВІ СЛОВА: ПРОГРАМНО-КОНФІГУРОВАНІ МЕРЕЖІ, SDN, OPENFLOW, МАРШРУТИЗАЦІЯ, КОНТРОЛЕР, ПЛОЩИНА УПРАВЛІННЯ, МЕРЕЖЕВА ІНФРАСТРУКТУРА, MININET.

ABSTRACT

Textual part of the qualification work for the bachelor's degree: 64 pages, 24 figures, 1 tables, 25 sources.

The aim of the work is the research and development of a network infrastructure using the concept of software-defined routing management (SDN) to increase the flexibility and efficiency of data transmission.

Object of research – Network infrastructure based on Software-Defined Networking (SDN) technology.

Subject of research – Methods and algorithms of software-defined routing management.

Summary of the work: Modern computer networks face problems of scalability and management complexity due to their rigid, decentralized architecture. The traditional approach, where each router independently makes decisions based on dynamic routing protocols, makes it difficult to quickly respond to topology changes and deploy new services. Therefore, there is a need for innovative traffic management approaches that allow centralized control of the entire network infrastructure. This capability is provided by the concept of Software-Defined Networking (SDN).

It provides physical and logical separation of the control plane from the data plane. As a result, routers become simple packet-forwarding devices, and all intelligent routing logic is concentrated in a centralized software controller. In order to properly implement this approach, it is worth considering the principles of controller interaction with network equipment using standardized protocols such as OpenFlow.

Based on the study, the relevance of using SDN in modern corporate networks and data centers, the features of building such infrastructures, and load balancing algorithms were clarified, and practical modeling of routing in the Mininet virtual environment was conducted.

KEYWORDS: SOFTWARE-DEFINED NETWORKING, SDN, OPENFLOW, ROUTING, CONTROLLER, CONTROL PLANE, NETWORK INFRASTRUCTURE.

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

ПОДАННЯ
ГОЛОВІ ДЕРЖАВНОЇ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня бакалавра

Направляється здобувач Левада І. А. до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 123 Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)
на тему: «Мережева інфраструктура з програмно-визначеним управлінням маршрутизацією».
Кваліфікаційна робота і рецензія додаються.

Директор ННІ _____ Катерина НЕСТЕРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач Левада І. А. показав відмінну теоретичну підготовку, уміння володіти та розуміння новими комп'ютерними технологіями, користуватися навчальною і науково-технічною літературою. Працюючи над завданнями, які були поставлені керівником, проявив ініціативність, сумлінність та хист до інженерної роботи.
Все це дозволяє оцінити кваліфікаційну роботу студента Левади Іллі Анатолійовича на оцінку «відмінно», та присвоїти йому кваліфікацію «бакалавр з комп'ютерної інженерії».

Керівник кваліфікаційної роботи _____ Віра БАБЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

«_» _____ 2026 року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач *Левада І. А.* допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедрою Комп'ютерної інженерії _____ Наталія ЛАЩЕВСЬКА
(назва) *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну бакалаврську роботу

здобувача вищої освіти Левада Ілля Анатолійович
(прізвище, ім'я, по батькові)
на тему: «Мережева інфраструктура з програмно-визначеним управлінням маршрутизацією»

Актуальність:

Бакалаврська кваліфікаційна робота присвячена одному з найперспективніших напрямків розвитку сучасних телекомунікаційних та IT-систем — побудові мережевої інфраструктури на базі концепції програмно-конфігурованих мереж (SDN). В умовах, коли традиційні статичні підходи до маршрутизації вичерпують свій потенціал через неможливість гнучко адаптуватися до хмарних середовищ та стрімко зростаючих обсягів трафіку, тема дослідження, спрямована на програмно-визначене управління потоками даних, є надзвичайно актуальною.

Позитивні сторони:
1 Автор вдало аргументував переваги фізичного відокремлення площини управління від площини передачі даних, що дозволяє уникнути проблем традиційної децентралізованої маршрутизації.
2 Автор продемонстрував високу компетентність у практичній частині, успішно розгорнувши та протестувавши різні мережеві топології (однорівневі, лінійні, деревоподібні) у середовищі моделювання Mininet.
3 Автор не лише описав теорію, але й емпірично довів високу швидкість конвергенції SDN-мережі при моделюванні розриву каналів зв'язку.

Недоліки:
1 У роботі недостатньо уваги приділено питанню безпеки самого централізованого контролера SDN.
2 Бажано було б розширити дослідження, розглянувши питання масштабованості. Для великих корпоративних мереж використання одного контролера може стати «вузьким місцем», тому короткий огляд мультиконтролерних архітектур додав би роботі більшої комплексності.

Висновок: *кваліфікаційна бакалаврська робота заслуговує оцінку " відмінно", а здобувач Левада І. А. заслуговує присвоєння кваліфікації: бакалавра з комп'ютерної інженерії.*

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

ЗМІСТ

ВСТУП 11
1 ДОСЛІДЖЕННЯ СУЧАСНИХ ТЕНДЕНЦІЙ ТА ПЕРСПЕКТИВ ЕВОЛЮЦІЇ ПРОГРАМНО-КОНФІГУРОВАНИХ МЕРЕЖ 13
1.1 Історичні передумови та еволюція становлення технології SDN 13
1.2 Аналіз структурної побудови та архітектурних особливостей технології SDN 16
1.3 Практичні сценарії впровадження та сфери застосування програмно-конфігурованих мереж 21
2 ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ТА ПРОГРАМНЕ СЕРЕДОВИЩЕ ДЛЯ ПРОЕКТУВАННЯ Й СИМУЛЯЦІЇ SDN-МЕРЕЖ 26
2.1 Аналіз середовищ віртуалізації та спеціалізованих віртуальних машин для моделювання мережевої інфраструктури 26
2.2 Аналіз функціональних можливостей та архітектури мережевого емулятора Mininet 29
2.3 Аналіз функціональних можливостей та конфігурування графічного середовища MiniEdit та додатків емуляції мережі. 33
3 ПРАКТИЧНІ ПІДХОДИ ДО ПРОЕКТУВАННЯ ТА РОЗГОРТАННЯ ТОПОЛОГІЙ ПРОГРАМНО-КОНФІГУРОВАНИХ МЕРЕЖ У СЕРЕДОВИЩІ MININET 43
3.1 Аналіз та практичне моделювання архітектурних мережевих топологій за допомогою інструментарію Mininet 43
3.2 Практична реалізація та конструювання мережевих топологій у віртуальному середовищі Mininet 53
3.3 Аналіз продуктивності та результати тестування змодельованих мережевих топологій 56
ВИСНОВКИ 59

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....61
Додаток А 64

ВСТУП

В наш час комп’ютерні мережі є фундаментом для функціонування будь-яких інформаційних систем, від невеликих корпоративних інфраструктур до глобальних центрів обробки даних та хмарних сервісів. Із стрімким зростанням обсягів трафіку та кількості підключених пристроїв, традиційні підходи до побудови мереж починають виявляти свої обмеження. У класичній архітектурі логіка управління маршрутизацією (control plane) та безпосереднє пересилання даних (data plane) жорстко об'єднані всередині кожного окремого маршрутизатора чи комутатора. Це означає, що мережа є глибоко децентралізованою: кожен пристрій повинен самостійно збирати інформацію про топологію, обчислювати маршрути та обмінюватися оновленнями з сусідами за допомогою складних протоколів, таких як OSPF чи BGP. Такий підхід робить мережу вразливою до довгої конвергенції при збоях, а процес внесення глобальних змін до політик маршрутизації перетворюється на складне ручне налаштування кожного вузла окремо.

Для того, щоб інфраструктура стала гнучкою, масштабованою та автоматизованою, виникла необхідність у кардинальній зміні парадигми управління. Саме таку революційну можливість надає технологія програмно-конфігурованих мереж (Software-Defined Networking, SDN). Концепція SDN пропонує відокремити інтелектуальну площину управління від апаратного забезпечення пересилання даних. Завдяки цьому маршрутизатори делегують завдання з прийняття рішень спеціалізованому програмному контролеру, який має глобальне бачення всієї топології. Це усуває проблему вузьких місць в управлінні, дозволяє програмувати поведінку мережі в режимі реального часу, ефективно балансувати навантаження та миттєво перебудовувати маршрути у випадку виходу з ладу окремих ліній зв'язку. Дослідження та впровадження мережевої інфраструктури з програмно-визначеним управлінням

маршрутизацією є ключовим кроком для створення надійних, сучасних та високопродуктивних IT-систем.

1 ДОСЛІДЖЕННЯ СУЧАСНИХ ТЕНДЕНЦІЙ ТА ПЕРСПЕКТИВ ЕВОЛЮЦІЇ ПРОГРАМНО-КОНФІГУРОВАНИХ МЕРЕЖ

1.1 Історичні передумови та еволюція становлення технології SDN

Хоча парадигма програмно-конфігурованих мереж (Software-Defined Networking, SDN) часто сприймається як новітнє технологічне досягнення останніх років, насправді вона має глибоке історичне коріння та еволюціонувала протягом тривалого часу. Розвиток цієї технології можна розглядати як логічну кульмінацію багаторічних досліджень, головною метою яких було перетворення статичних мереж на гнучкі, повністю програмовані інфраструктури. Сучасні класичні комп'ютерні мережі характеризуються надзвичайно високим рівнем архітектурної та управлінської складності. Вони об'єднують величезний масив різноманітного спеціалізованого обладнання, серед якого маршрутизатори, комутатори, засоби трансляції мережевих адрес, а також складні системи глибокого аналізу та розпізнавання трафіку[1]. Традиційний підхід до управління вимагає від мережевих адміністраторів ручного та децентралізованого налаштування кожного окремого пристрою. При цьому фахівці змушені суворо дотримуватися глобальних правил маршрутизації для збереження загальної цілісності мережевої топології. Ситуація значно ускладнюється тим, що кожен виробник апаратного забезпечення використовує власні інтерфейси конфігурації та пропрієтарні протоколи[2-4]. Навіть застосування методів пакетного конфігурування лише частково вирішує проблему, залишаючи процес масштабування, модернізації та загального розвитку мережі вкрай обтяжливим. Така ригідність традиційних архітектур не лише гальмує впровадження інноваційних рішень, але й генерує невиправдано високі капітальні та операційні витрати для компаній. Візуальне зображення архітектури такої мережі наведено на рис. 1.1.

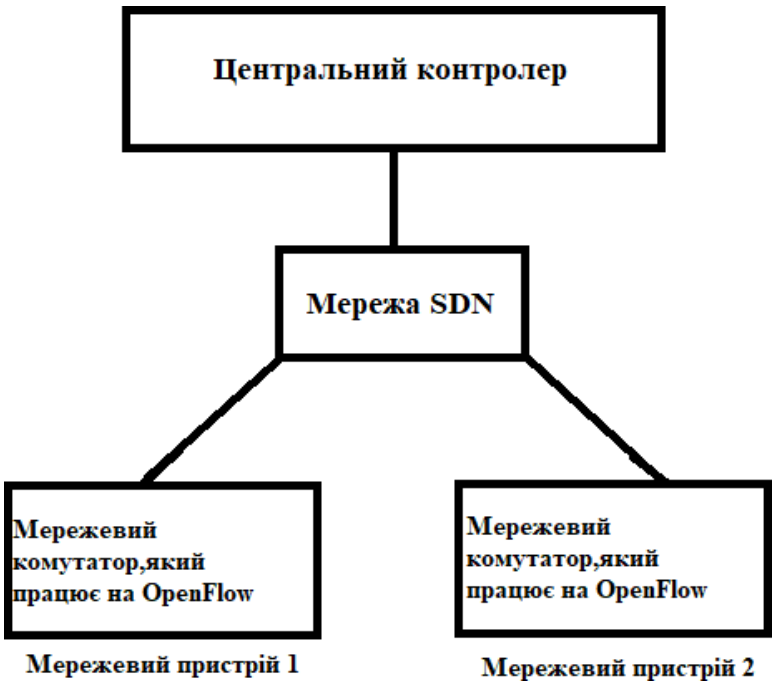


Рисунок 1.1 — Структурна схема та концептуальна модель організації програмно-конфігурованої мережі (SDN)

Впровадження програмно-конфігурованих мереж докорінно трансформує фундаментальні принципи проектування та адміністрування мережевих інфраструктур. Головною інновацією SDN є фізичне та логічне відокремлення площини управління, яка приймає глобальні рішення щодо прокладання маршрутів, від площини передачі даних, що безпосередньо займається пересиланням трафіку на основі отриманих зверху інструкцій. Крім того, ця архітектура передбачає централізацію логіки: замість розподілених обчислень на кожному пристрої, керування зосереджується у спеціалізованому програмному забезпеченні на сервері, який одночасно контролює безліч пристроїв площини даних[5]. Ця взаємодія реалізується через стандартизовані відкриті інтерфейси прикладного програмування (API). Найбільш відомим та поширеним прикладом такого інтерфейсу є протокол OpenFlow. Для успішного розгортання SDN-інфраструктури критично важливо, щоб усі комутаційні вузли на апаратному рівні підтримували цей стандарт. У такому середовищі мережеві елементи оперують однією або декількома таблицями, кожна з яких містить специфічні правила

маршрутизації. Ці правила чітко детермінують алгоритм дій щодо пересилання пакетів для кожного окремого потоку трафіку. Завдяки такій універсалізації, кожен комутатор з підтримкою OpenFlow здатний гнучко виконувати ролі звичайного комутатора, маршрутизатора або транслятора мережевих адрес залежно від потреб системи[6].

Незважаючи на стрімке зростання популярності концепції SDN у сучасному IT-середовищі, самі ідеї, закладені в її основу, розробляються вже понад двадцять років. Деякі дослідники простежують витоки цього підходу ще з часів становлення перших телефонних мереж із комутацією каналів. Саме там вперше було успішно реалізовано принцип відокремлення систем управління мережею від безпосередніх каналів передачі голосового трафіку, що дозволило суттєво спростити адміністрування та пришвидшити розгортання нових послуг. У контексті комп'ютерних мереж із комутацією пакетів поява концепції «програмних комутаторів» також стала важливим кроком, оскільки за своєю логікою функціонування та архітектурною реалізацією вона має багато спільних рис із сучасними SDN[7].

Історично рух у бік програмованих мереж активізувався з появою дослідницької концепції «активних мереж», яка стала першим масштабним баченням парадигми SDN, проте через низку технічних обмежень так і не набула масового поширення. Згодом, керуючись виключно прагматичними цілями оптимізації процесів, розробники здійснили нові спроби розділити площини управління та передачі даних, щоб отримати більш ефективні інструменти для спрямування мережевого трафіку[8].

Вирішальним проривом у цій галузі стала розробка протоколу OpenFlow та створення спеціалізованих мережевих операційних систем, які змогли забезпечити ідеальний баланс між сміливим теоретичним баченням та суворими вимогами практичної експлуатації. Під час адаптації принципів SDN для потреб віртуалізації мережевих ресурсів, саме цей компроміс між широкою стратегічною перспективою та наявністю дієвої стратегії для масового впровадження виявився найбільш цінним[9].

Сьогодні технологія програмно-конфігурованих мереж продовжує активно вдосконалюватися, і ми спостерігаємо тенденцію до її глибокої комерціалізації. Хоча маркетингові матеріали часто позиціонують SDN як універсальну панацею від будь-яких проблем з інфраструктурою, з інженерної точки зору варто розуміти об'єктивну реальність. SDN — це насамперед потужний архітектурний інструмент, створений для значного полегшення рутинних процесів мережевого адміністрування. Цей підхід створює сприятливе середовище для розробки інноваційних програмних додатків і надає принципово нові, більш ефективні способи вирішення давно відомих інфраструктурних завдань[10].

1.2 Аналіз структурної побудови та архітектурних особливостей технології SDN

Фундаментальна концепція програмно-конфігурованих мереж (SDN) базується на парадигмі чіткого фізичного та логічного відокремлення площини управління від площини передачі даних. Цей архітектурний підхід передбачає делегування інтелектуальних функцій керування, які традиційно виконувалися безпосередньо на маршрутизаторах і комутаторах, спеціалізованому програмному забезпеченню. Таке забезпечення функціонує на виділених серверах, що називаються контролерами[11]. Залежно від масштабу та складності інфраструктури, роль контролера може виконувати як один потужний сервер, так і цілий кластер серверів із відповідним програмним забезпеченням. Водночас апаратні мережеві пристрої, позбавлені функцій прийняття рішень щодо маршрутизації, зосереджуються виключно на своєму базовому завданні — високошвидкісному пересиланні пакетів. Така абстракція дозволяє повністю відокремити логіку управління від фізичного обладнання, роблячи мережу програмованою. Базова транспортна інфраструктура стає

незалежною від високорівневих мережевих служб і додатків. Завдяки цьому адмініструванні всією мережею централізується на виділених обчислювальних ресурсах контролера SDN, що дає змогу розглядати всю складну фізичну топологію як один єдиний логічний комутатор або маршрутизатор для потреб прикладного програмного забезпечення.

Екосистему програмно-конфігурованих мереж прийнято розділяти на три взаємопов'язані ієрархічні рівні, кожен з яких виконує специфічний набір функцій та складається з окремих компонентів. Візуальне представлення цієї багаторівневої моделі наведено на рис. 1.2.

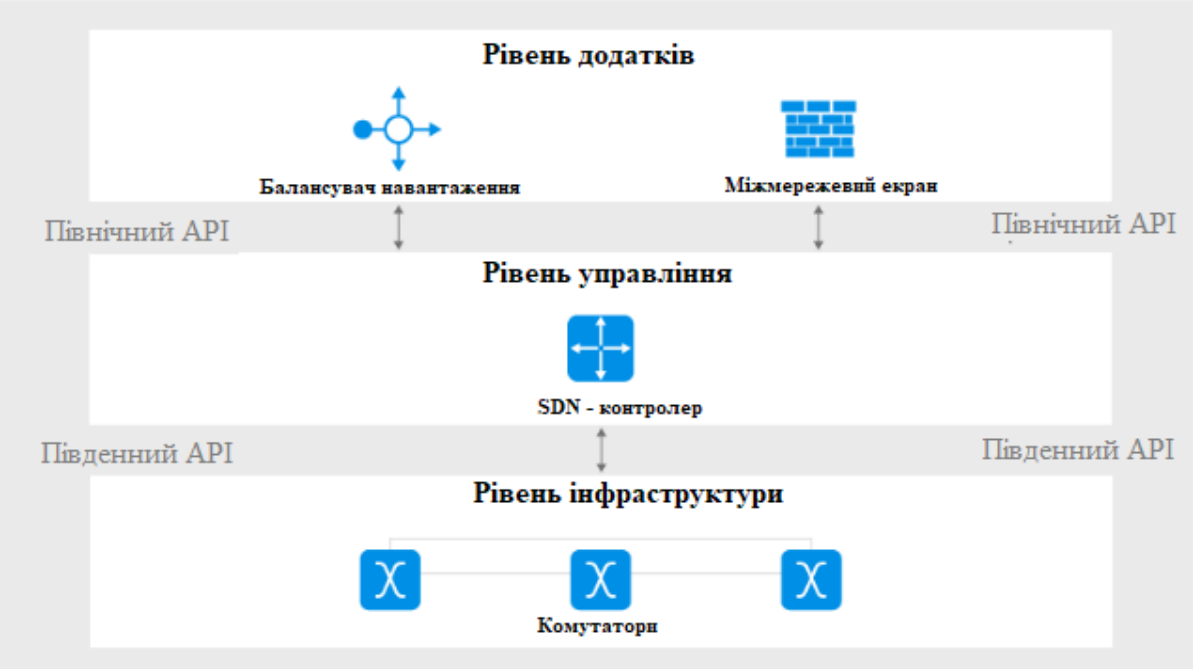


Рисунок 1.2 — Ієрархічна багаторівнева структура програмно-конфігурованої мережі

Верхній рівень архітектури — рівень додатків (прикладний рівень) — відповідає за реалізацію гнучкого управління мережевою поведінкою за допомогою різноманітних бізнес-додатків. Його головне завдання полягає у формуванні високорівневих вимог до мережі для виконання конкретних прикладних завдань[12-15]. Контролер взаємодіє з цими додатками, отримуючи від них запити через відкриті інтерфейси прикладного

програмування (API), і транслює їх у зрозумілі для мережевого обладнання команди.

Центральною ланкою системи є рівень управління (контрольна площина), який забезпечує побудову маршрутів та управління доступом для всього мережевого трафіку. На цьому рівні функціонує мережева операційна система (SDN-контролер), яка здійснює централізоване конфігурування, формує правила обробки пакетів та розподіляє їх між комутаторами. Контролер виступає сполучною ланкою: він збирає телеметрію та дані про стан мережі від апаратних пристроїв і передає цю інформацію на прикладний рівень, а у зворотному напрямку — спускає команди від додатків до обладнання. Завдяки централізації інтелекту на цьому рівні відпадає необхідність у ручному налаштуванні кожного окремого маршрутизатора. Крім того, рівень управління забезпечує підтримку базових мережевих протоколів адресації (IPv4, IPv6, ARP) та централізовано керує логікою протоколів динамічної маршрутизації, таких як RIP, OSPF, EIGRP та BGP[17].

Найнижчий рівень — інфраструктурний (рівень даних) — концептуально відповідає фізичному та канальному рівням традиційної еталонної моделі OSI. Він об'єднує всі фізичні та віртуальні мережеві пристрої (комутатори з підтримкою SDN, гібридні пристрої), а також канали зв'язку. Головна і єдина функція цього рівня полягає у безпосередньому апаратному пересиланні пакетів від вхідного порту до вихідного виключно на основі тих інструкцій та таблиць потоків, які були отримані від рівня управління[16].

Незважаючи на те, що всі програмно-конфігуровані мережі об'єднує ідея централізованого програмного управління потоками даних, на практиці існують різні моделі її реалізації. Відкрита модель SDN базується на використанні стандартизованих протоколів, таких як OpenFlow, завдяки яким адміністратори напряму керують площиною даних віртуальних та фізичних комутаторів. Альтернативним підходом є SDN на базі API, де взаємодія та управління пересиланням даних на мережевих вузлах реалізується через специфічні програмні інтерфейси. Окремо виділяють оверлейну (накладену)

модель SDN, у якій віртуальні мережі розгортаються поверх існуючої класичної апаратної інфраструктури шляхом створення динамічних тунелів між локальними та віддаленими дата-центрами. Цей метод дозволяє гнучко розподіляти пропускну здатність та логічно ізолювати канали без необхідності фізичної перебудови базової мережі. Для поступової модернізації інфраструктури часто використовується гібридна модель SDN, яка гармонійно поєднує інноваційне програмне управління з традиційними мережевими протоколами в єдиному середовищі. У такому сценарії частина трафіку обробляється за класичними алгоритмами маршрутизації, тоді як інша частина керується SDN-контролером, що дозволяє компаніям здійснювати плавну міграцію[18-20].

Ключова різниця між традиційними мережами та архітектурою SDN полягає у підході до розподілу інтелекту. Традиційна мережа є глибоко розподіленою системою, де кожен маршрутизатор є автономним вузлом, який самостійно збирає інформацію про стан сусідніх каналів зв'язку та використовує складні алгоритми для обчислення найкоротшого шляху до цілі. У разі зміни топології або виходу з ладу обладнання, маршрутизатори починають інтенсивний обмін оновленнями та перераховують таблиці маршрутизації. Це забезпечує високу надійність та автоматичне відновлення завдяки децентралізації. Натомість мережа SDN покладається на контролер як на єдиний центр прийняття рішень, який має глобальне бачення всієї топології. Проте така централізація означає, що швидкість конвергенції та загальна працездатність мережі критично залежать від доступності контролера, що створює потенційну єдину точку відмови. Детально архітектурні відмінності між цими двома підходами проілюстровано на рис. 1.3.

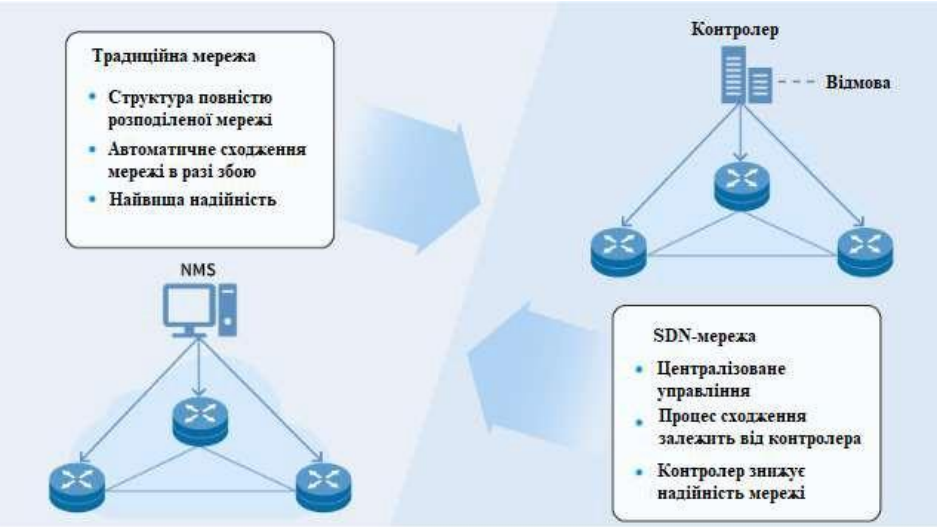


Рисунок 1.3 — Порівняльний аналіз архітектур традиційної та програмно-конфігурованої мережі

Перехід до програмно-конфігурованих мереж забезпечує низку вагомих переваг порівняно з класичними мережевими парадигмами. Насамперед, це безпрецедентний рівень гнучкості та швидкості керування інфраструктурою. Замість ручного налаштування пропрієтарного обладнання від різних виробників, інженери отримують можливість використовувати готові програмні контролери, часто на базі відкритого коду, для централізованого управління трафіком. Це значно розширює вибір апаратного забезпечення та знімає проблему прив'язки до одного вендора[21-23].

Крім того, архітектура SDN формує інфраструктуру, що динамічно налаштовується. Адміністратори можуть у режимі реального часу перерозподіляти віртуальні ресурси, оптимізувати потоки даних та встановлювати пріоритети для тих бізнес-додатків, які потребують найвищого рівня доступності та гарантованої пропускної здатності.

Ще однією суттєвою перевагою є значне підвищення рівня інформаційної безпеки. Централізоване управління надає адміністраторам повну видимість усіх процесів у мережі, що дозволяє набагато глибше аналізувати трафік та швидше виявляти потенційні загрози. У сучасних реаліях з величезною кількістю підключених пристроїв, SDN дозволяє архітекторам створювати ізольовані сегменти мережі з різними політиками

доступу та миттєво блокувати скомпрометовані вузли, надійно захищаючи систему від поширення хакерських атак[24].

1.3 Практичні сценарії впровадження та сфери застосування програмно-конфігурованих мереж

Технологія програмно-конфігурованих мереж демонструє свою універсальність та високу ефективність у надзвичайно широкому спектрі сучасних інформаційних галузей. Зокрема, архітектура SDN успішно інтегрується в інфраструктури, що забезпечують роботу додатків для спільної роботи, оптимізують складний мережевий обмін даними, підтримують розгортання послуг мобільного зв'язку нового покоління, а також лежать в основі функціонування масштабних центрів обробки даних (ЦОД). Аналізуючи ключові сценарії практичного застосування цих мереж, можна виділити кілька магістральних напрямків, які докорінно змінюють підходи до управління корпоративним та провайдерським трафіком.

Одним із найважливіших векторів впровадження є організація доменів доступу та агрегація комунікаційних мереж загального користування спільно з магістральними мережами обробки інформації. У цьому контексті домен мережі доступу обов'язково охоплює транспортну транзитну мережу (mobile backhaul), яка виконує критично важливу функцію надійного високошвидкісного з'єднання базових станцій стільникового зв'язку з ядром мережі оператора. Завдяки впровадженню принципів глибокої віртуалізації, що є фундаментальними для SDN, з'являється можливість логічно та безпечно ізолювати різноманітні групи мережевих послуг одна від одної. Це дозволяє створювати єдину надійну фізичну інфраструктуру, ресурси якої можуть незалежно використовуватися одразу кількома різними телекомунікаційними операторами без ризику взаємного впливу чи зниження якості обслуговування.

З іншого боку, розгортання SDN-рішень у середовищі центрів обробки даних відкриває безпрецедентні можливості для оптимізації їхньої роботи. Програмне керування дозволяє суттєво вдосконалити алгоритми балансування навантаження та забезпечити повноцінний, еластичний розподіл наявних обчислювальних ресурсів у режимі реального часу. У результаті коефіцієнт корисного використання як пропускної здатності мережевих каналів, так і потужностей серверного обладнання значно зростає. Крім того, централізована архітектура SDN максимально спрощує процеси «живої» міграції віртуальних машин між фізичними серверами, оскільки всі необхідні зміни в конфігурації комутаторів для збереження зв'язності відбуваються повністю автоматично. Більше того, розподілені принципи управління стимулюють розробку архітектур для ЦОД, побудованих на базі стандартизованих, комерційно доступних комутаторів (bare-metal або white-box рішень). Це дає змогу компаніям відмовитися від закупівлі вузькоспеціалізованого та надзвичайно дорогого пропрієтарного апаратного забезпечення, радикально знижуючи капітальні витрати на розбудову інфраструктури[25].

На сьогоднішній день саме центри обробки даних становлять одну з найбільших та стратегічно найважливіших сфер інтересу для глобальних постачальників послуг, оскільки вони є серцевиною нових масштабних корпоративних мереж. У цьому глобальному вимірі провайдери територіально розподілених мереж (WAN), які об'єднують віддалені дата-центри, відіграють вирішальну роль у забезпеченні їхньої рентабельної та безперебійної роботи. Традиційно оператори зв'язку змушені спиратися на консервативні, статичні механізми ручної ініціалізації для виділення та маршрутизації ресурсів у глобальних мережах. Такий підхід не лише гальмує загальну швидкість взаємодії між центрами обробки даних, але й непропорційно збільшує операційні витрати на підтримку системи. Відповідно, виникла гостра технологічна потреба перетворити глобальну мережу зв'язку між дата-центрами на такий самий гнучкий пул віртуальних ресурсів, який можна

розподіляти за запитом з високою швидкістю та адаптивністю. Саме архітектура SDN виступає ідеальним інструментом для реалізації цих завдань, забезпечуючи операторам необхідну динамічність в управлінні WAN-інфраструктурою.

Яскравим прикладом практичної реалізації подібних концепцій на ринку є Huawei Agile Controller, який виступає центральним інтелектуальним ядром у розгортанні програмно-конфігурованих мереж. Цей комплексний контролер оптимізований для застосування у чотирьох ключових бізнес-сценаріях: управління мережами центрів обробки даних, організація складних кампусних мереж, адміністрування глобальних інфраструктур (SD-WAN) та оптимізація маршрутизації в мережах Інтернету. Головна місія цього програмного комплексу полягає у забезпеченні повністю автоматизованої трансляції високорівневих бізнес-вимог від прикладних додатків безпосередньо у фізичну конфігурацію транспортної мережі. Він надає адміністраторам абсолютний візуалізований контроль над усіма доступними ресурсами та гарантує глибоку прозорість метрик для підрозділів технічної підтримки. Оскільки система базується на відкритих технологічних стандартах, вона гарантує максимальну сумісність із мережевим обладнанням від сторонніх виробників, дозволяючи інженерам створювати гетерогенні комплексні рішення. Детальна структурна схема, яка ілюструє концептуальну різницю та логіку архітектурного переходу від традиційної моделі побудови мережі центру обробки даних до сучасної програмно-конфігурованої інфраструктури, наведена на рис. 1.4.

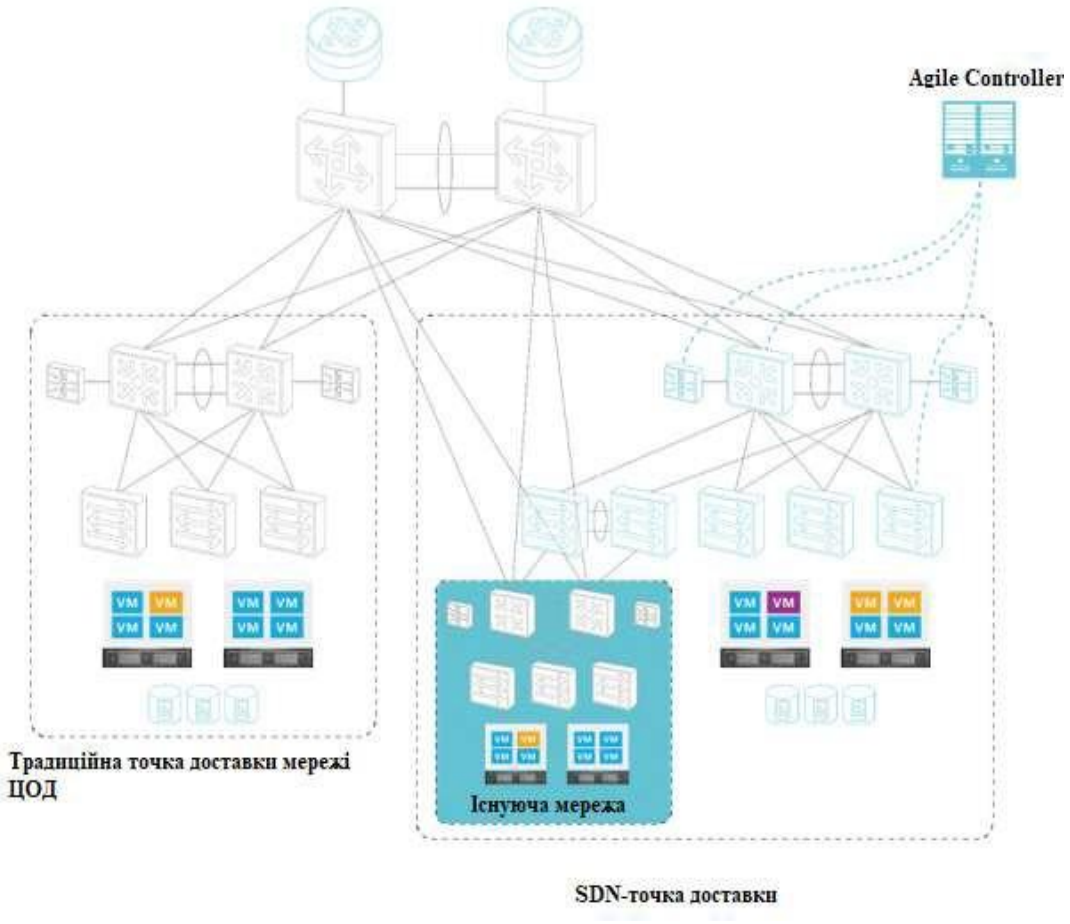


Рисунок 1.4 — Схема еволюційного переходу та архітектурної трансформації традиційної мережі центру обробки даних на базу технології SDN

Окрім самої парадигми SDN, останнім часом визначальну роль в архітектурі мереж провідних телекомунікаційних операторів та великих постачальників хмарних послуг почала відігравати споріднена концепція віртуалізації мережевих функцій (Network Functions Virtualization, NFV). Синергетичне поєднання технологій SDN та NFV дозволяє ефективно долати найскладніші інфраструктурні виклики. Зокрема, вони забезпечують повне абстрагування внутрішньої логіки обробки трафіку від фізичного обладнання та надійно ізолюють конфіденційні бази даних абонентів. Цей інноваційний підхід вирішує проблему масштабування двома основними шляхами: через механізми динамічної оркестровки обчислювальних ресурсів та за допомогою систем інтелектуального управління життєвим циклом мережевих сервісів. Для успішної реалізації процесів динамічної оркестровки на базі NFV

критично необхідним є розгортання єдиного, уніфікованого стеку віртуалізації, який стандартизовано обслуговуватиме всі мережеві додатки. Як правило, таке високопродуктивне середовище формується на базі захищеної приватної хмарної інфраструктури самого оператора, що гарантує еталонний рівень відмовостійкості та інформаційної безпеки.

2 ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ТА ПРОГРАМНЕ СЕРЕДОВИЩЕ ДЛЯ ПРОЕКТУВАННЯ Й СИМУЛЯЦІЇ SDN-МЕРЕЖ

2.1 Аналіз середовищ віртуалізації та спеціалізованих віртуальних машин для моделювання мережевої інфраструктури

Для повноцінної емуляції інфраструктури програмно-конфігурованих мереж на робочих станціях, які володіють обмеженими апаратними та програмними ресурсами, критично важливо залучати сучасні системи віртуалізації та спеціалізовані емулятори з відкритим вихідним кодом. Щоб розгорнути комплексне дослідницьке середовище та детально змоделювати взаємодію SDN-хостів, централізованих контролерів і комутаторів за допомогою протоколу OpenFlow, оптимально використовувати програмний комплекс на базі Oracle VM VirtualBox. Безпосередньо у цьому віртуальному середовищі розгортається мережевий емулятор Mininet, який, у свою чергу, інтегрує низку потужних додаткових інструментів: графічний редактор топологій MiniEdit, термінальний емулятор Xterm для точкового управління вузлами та популярний аналізатор пакетів Wireshark для глибокого дослідження мережевого трафіку.

Oracle VM VirtualBox являє собою потужну кросплатформну систему віртуалізації, що надає користувачам можливість створювати повністю ізольовані віртуальні машини та запускати в них різноманітні гостьові операційні системи безпосередньо поверх основної ОС комп'ютера. Цей програмний продукт відрізняється надзвичайно широкою сумісністю, забезпечуючи стабільну роботу з такими системами, як різноманітні дистрибутиви Linux, FreeBSD, ReactOS, різні версії Solaris та OpenSolaris, Mac OS і навіть застарілі середовища на кшталт DOS. Графічний інтерфейс

користувача та загальний вигляд головного меню цієї програми під час запуску продемонстровано на рис. 2.1.

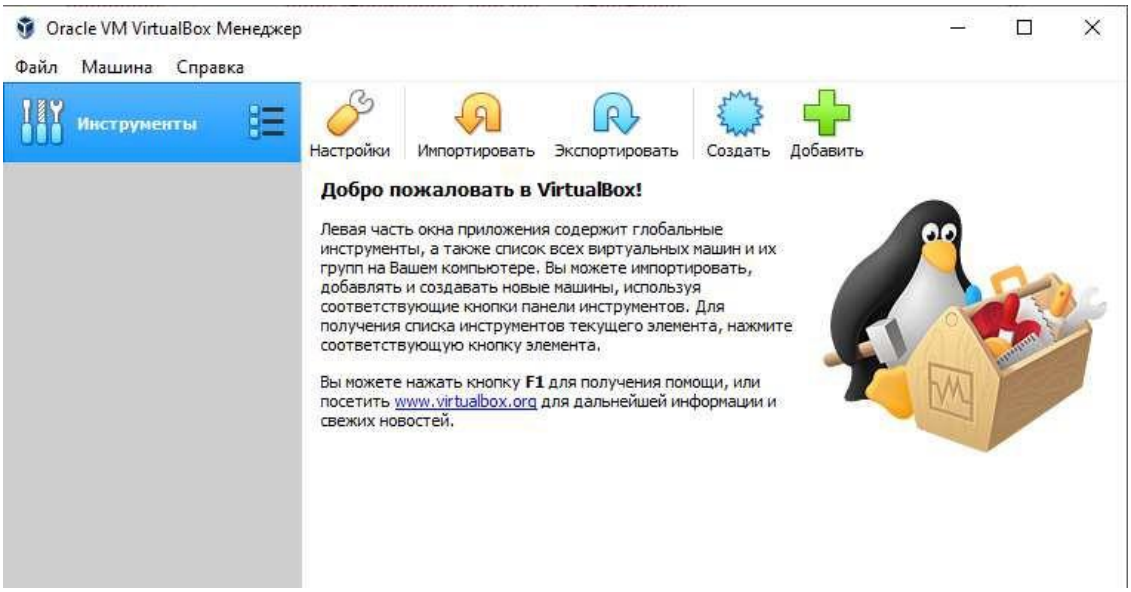


Рисунок 2.1 — Головне вікно управління та базовий інтерфейс користувача гіпервізора Oracle VM VirtualBox

Програмний продукт Oracle VM VirtualBox володіє значним спектром експлуатаційних переваг, що робить його одним із лідерів у своєму сегменті. Насамперед варто відзначити його абсолютно вільне поширення та безкоштовний доступ, що ідеально підходить для освітніх цілей, надаючи інженерам та студентам широкі перспективи для безпечного вивчення принципів роботи з різними операційними системами. Середовище є надзвичайно практичним та гнучким: воно пропонує просунутий мережевий стек із підтримкою різноманітних режимів підключення. Доступні конфігурації включають трансляцію мережевих адрес (NAT), створення внутрішньої ізольованої мережі, а також режим мережевого моста для прямого доступу віртуальної машини до фізичної локальної мережі.

З апаратної та системної точок зору, програма забезпечує надійну підтримку віртуальних USB-контролерів стандартів 1.1 та 2.0, а також пропонує базову сумісність із графічними бібліотеками DirectX та OpenGL. Для досвідчених користувачів вагомою перевагою є наявність розширеного

інтерфейсу командного рядка, який дозволяє гнучко автоматизувати процеси розгортання та налаштування віртуальних машин без використання графічного інтерфейсу. Зручність обміну даними між основною (хостовою) та гостьовою операційними системами легко реалізується через вбудований механізм спільних папок (Shared Folders). Додатково розробники передбачили можливість віддаленого підключення до віртуального середовища через інтегрований RDP-сервер, а наявність портативної версії дозволяє запускати програму навіть із зовнішніх носіїв. Крім того, архітектура гіпервізора без проблем дозволяє одночасно запускати та паралельно використовувати відразу кілька віртуальних машин, легко адаптуючи мову інтерфейсу під потреби користувача.

Незважаючи на беззаперечні плюси, такі як легкість інсталяції та багатогранність налаштувань, ця платформа має й певні специфічні особливості. Наприклад, деякі дослідники відзначають дещо неінтуїтивну та заплутану логіку управління віртуальними дисковими накопичувачами, а також певні обмеження у вбудованих механізмах фіксації стану екрана (створення скріншотів). Тим не менш, концепція віртуалізації загалом є неперевершеним інструментом для безпечного дослідження комп'ютерних систем. Історично для роботи з кількома ОС на одному фізичному пристрої (наприклад, для паралельного використання Windows та Linux) користувачі були змушені застосовувати метод подвійного завантаження системи. Це вимагало складного та ризикованого перерозбиття жорсткого диска на фізичні розділи, а також постійного перезавантаження комп'ютера для перемикання між середовищами, що суттєво уповільнювало робочий процес та створювало незручності.

Сьогодні ж платформи віртуалізації кардинально змінюють підхід до набуття практичного досвіду, особливо у сфері глибокого конфігурування комп'ютерних мереж. VirtualBox надає гнучку можливість розгортати цілі інфраструктури з кількох різних операційних систем, які функціонують одночасно у вигляді ізольованих контейнерів. Важливо, що будь-які зміни, збої

чи експерименти всередині цих віртуальних машин жодним чином не загрожують цілісності та стабільності основної операційної системи хоста. Це формує ідеальний, безпечний полігон для глибокого вивчення новітніх комп'ютерних технологій, відпрацювання складних сценаріїв маршрутизації та постійного вдосконалення професійних навичок мережевого адміністрування.

2.2 Аналіз функціональних можливостей та архітектури мережевого емулятора Mininet

Розглянемо детальніше програмний комплекс Mininet, який являє собою потужний та гнучкий емулятор комп'ютерних мереж. Цей інструмент дозволяє створювати повноцінні віртуальні інфраструктури, до складу яких входять базові системні вузли, такі як кінцеві хости, мережеві комутатори та спеціалізовані контролери з підтримкою протоколу OpenFlow. Використовуючи інтуїтивно зрозумілий синтаксис у вбудованому інтерпретаторі команд, користувач отримує змогу швидко розгортати масштабовані мережі, що складаються з довільної кількості хостів і комутаторів, об'єднаних у найрізноманітніші топології. Головна перевага полягає в тому, що вся ця складна архітектура функціонує в межах однієї єдиної віртуальної машини. На кожному зі створених віртуальних хостів дослідник може вільно змінювати параметри мережевої конфігурації, застосовувати стандартні діагностичні утиліти операційної системи та навіть отримувати повноцінний доступ до командного рядка (терміналу). Водночас на рівні комутаторів адміністратор має можливість налаштовувати специфічні правила обробки пакетів та гнучко керувати маршрутизацією трафіку. У підсумку, середовище Mininet створює ідеальні умови для глибокого вивчення внутрішньої будови та принципів функціонування сучасних комп'ютерних

мереж, повністю усуваючи необхідність у придбанні чи оренді дороговартісного фізичного мережевого обладнання.

Завдяки своїй універсальності, цей емулятор активно використовується для широкого спектра завдань, серед яких проведення наукових досліджень, розробка нових мережевих протоколів, академічне навчання, створення експериментальних прототипів, а також комплексне тестування та налагодження систем. Фактично, він надає інженеру або студенту унікальну можливість мати повноцінну, багатофункціональну мережеву лабораторію безпосередньо на своєму портативному комп'ютері чи робочій станції. Зовнішній вигляд інтерфейсу командного рядка після запуску цього середовища продемонстровано на рис. 2.2.

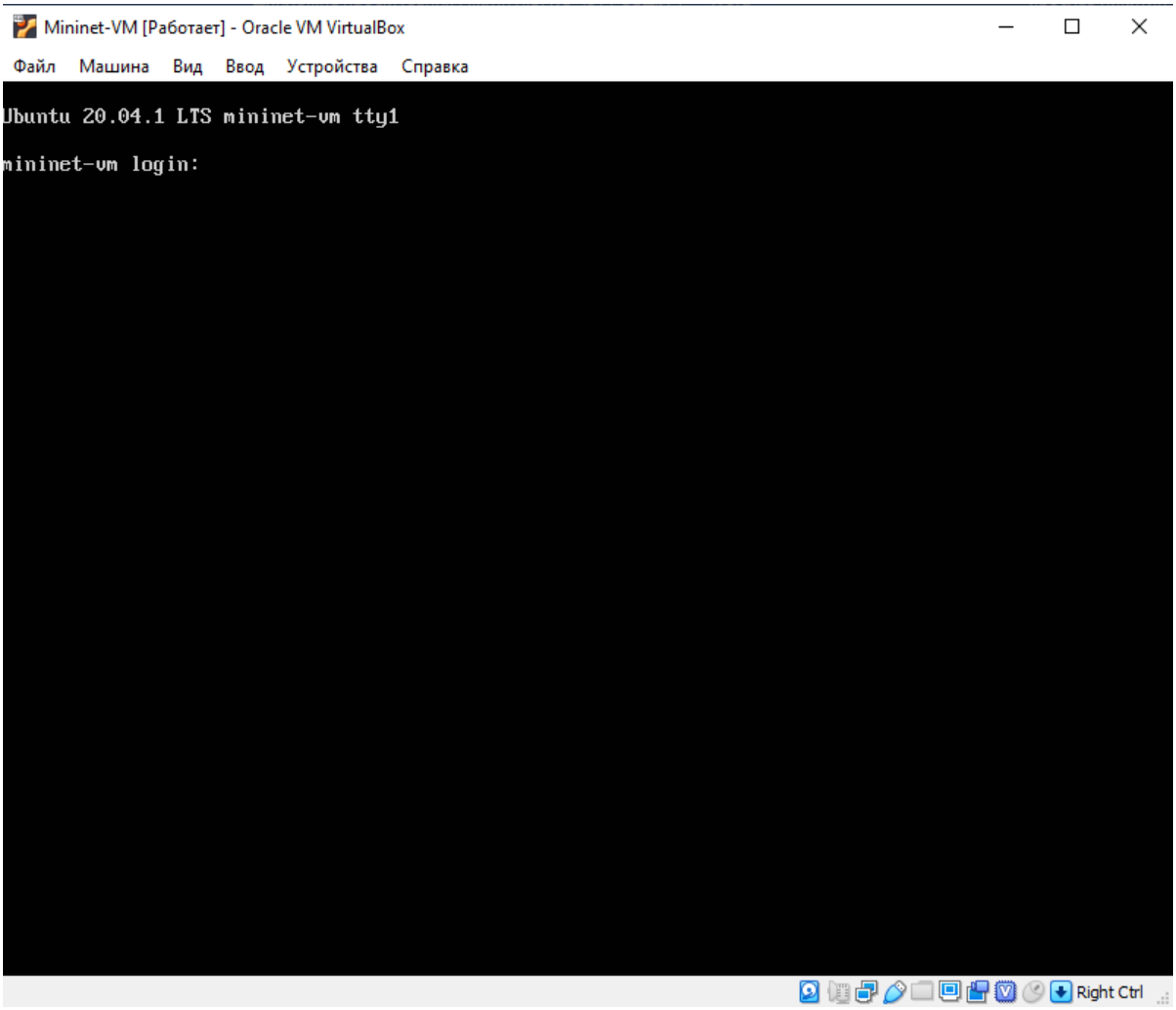


Рисунок 2.2 — Робочий інтерфейс та стартове вікно командного рядка емулятора Mininet

Аналізуючи функціональні можливості Mininet, варто відзначити, що він формує максимально просте, зручне та економічно вигідне середовище для розробки та тестування програмного забезпечення, орієнтованого на архітектуру OpenFlow. Емулятор спроектований таким чином, щоб дозволяти цілим командам розробників паралельно працювати над спільними або ізольованими мережевими топологіями, не заважаючи один одному. Надзвичайно важливою функцією є підтримка відтворюваного системного регресійного тестування. Це означає, що створені сценарії тестування можна легко запакувати, зберегти та багаторазово запускати з абсолютно ідентичними умовами, що є критичним для перевірки стабільності коду. Крім того, інструмент пропонує виняткову можливість моделювати та випробовувати надзвичайно складні, багаторівневі мережеві топології без необхідності прокладання реальних фізичних ліній зв'язку.

Для зручності взаємодії та глибокого налагодження системи у розпорядженні користувача є потужний інтерфейс командного рядка (CLI) з повною підтримкою команд OpenFlow. З коробки система вже містить набір класичних, базових мережевих топологій для швидкого старту, але водночас надає інструментарій для проектування вузькоспеціалізованих, авторських схем підключення. Для максимальної автоматизації процесів та створення нестандартних експериментів розробники можуть використовувати вбудований та чудово документований прикладний програмний інтерфейс (API) на базі мови Python.

Загалом емулятор забезпечує інтуїтивно зрозумілий та надійний механізм перевірки коректності функціонування мережевої логіки, оцінки загальної продуктивності системи та проведення комплексних експериментів із маршрутизацією. Програмне забезпечення Mininet поширюється з відкритим вихідним кодом і тісно інтегрується зі стандартними мережевими програмами операційних систем сімейства Unix/Linux, використовуючи їхні реальні ядра та нативні мережеві стеки. Такий підхід має колосальне значення для розробників, оскільки він гарантує безшовну портативність коду. Це

означає, що програмний код, написаний та ретельно протестований у віртуальному середовищі Mininet для контролерів OpenFlow, модифікованих комутаторів чи кінцевих хостів, може бути перенесений у реальне виробниче середовище з мінімальними, суто косметичними змінами. Таким чином, проекти, які успішно пройшли етап моделювання, здатні максимально плавно мігрувати на реальні апаратні комутатори, забезпечуючи заявлену швидкість передачі пакетів на фізичному рівні.

На відміну від традиційних систем повної віртуалізації, де на кожен віртуальний вузол виділяється окрема громіздка операційна система з величезними витратами ресурсів, Mininet використовує більш витончений підхід, заснований на абстракції процесів та ізоляції мережевих просторів імен. Завдяки цій архітектурній особливості емулятор демонструє низку критичних переваг. По-перше, час завантаження розгалуженої мережі скорочується до лічених секунд. По-друге, система здатна без проблем масштабуватися, підтримуючи одночасну роботу сотень хостів і комутаторів на звичайному ПК, при цьому забезпечуючи високу пропускну здатність віртуальних каналів. Інсталяція також максимально спрощена: достатньо розгорнути образ на базі VMware або VirtualBox з попередньо встановленими інструментами OpenFlow.

Якщо ж порівнювати Mininet із фізичними апаратними тестовими стендами, то його переваги є ще більш очевидними. Це насамперед суттєво нижча вартість розгортання, абсолютна загальнодоступність, а також блискавична швидкість застосування нових налаштувань та перезавантаження топології. Серед безлічі інших емуляторів Mininet беззаперечно виділяється тим, що він запускає абсолютно справжній, немодифікований програмний код, який включає код ядра ОС, утиліти конфігурації, а також логіку контролера OpenFlow. На додаток до цього, інструмент володіє вбудованими простими механізмами для прямого підключення згенерованої віртуальної топології до реальних зовнішніх мереж.

2.3 Аналіз функціональних можливостей та конфігурування графічного середовища MiniEdit та додатків емуляції мережі.

У складі потужного мережевого експлуатаційного комплексу Mininet передбачено спеціалізоване графічне середовище користувача, яке має назву MiniEdit. Цей інструмент надає дослідникам та інженерам унікальну можливість відійти від виключно консольного управління та перейти до візуального, інтуїтивно зрозумілого процесу проектування різноманітних топологій програмно-конфігурованих мереж (SDN). Використовуючи MiniEdit, спеціаліст може на початковому етапі сформувати наочну графічну модель майбутньої інфраструктури, детально відобразивши загальну архітектурну структуру, розташування активних мережевих вузлів та специфіку ліній зв'язку між ними. Важливою особливістю є те, що після повного візуального конструювання мережі та тонкого налаштування всіх необхідних параметрів елементів, отриманий проект можна легко зберегти на фізичний носій у вигляді спеціалізованого файлу з розширенням .mn для подальшого використання. Крім того, графічний інтерфейс дозволяє не лише малювати схеми, але й безпосередньо ініціювати запуск створеної мережі прямо з робочого вікна програми. Це означає, що після глибокого аналізу продуктивності, перевірки пропускної здатності та тестування інших параметрів активної мережі, користувач має змогу в будь-який момент зупинити симуляцію, повернутися до режиму візуального редагування і внести необхідні корективи у віртуальний макет. Однією з найбільш вагомих та корисних переваг середовища MiniEdit є його здатність автоматично генерувати програмний код: розробник може експортувати створений графічний макет безпосередньо у готовий виконуваний файл мовою Python, який вже міститиме всю відповідну логіку та конфігурацію мережевих компонентів. Зовнішній вигляд робочого простору цієї програми під час першого запуску наведено на рис. 2.3.

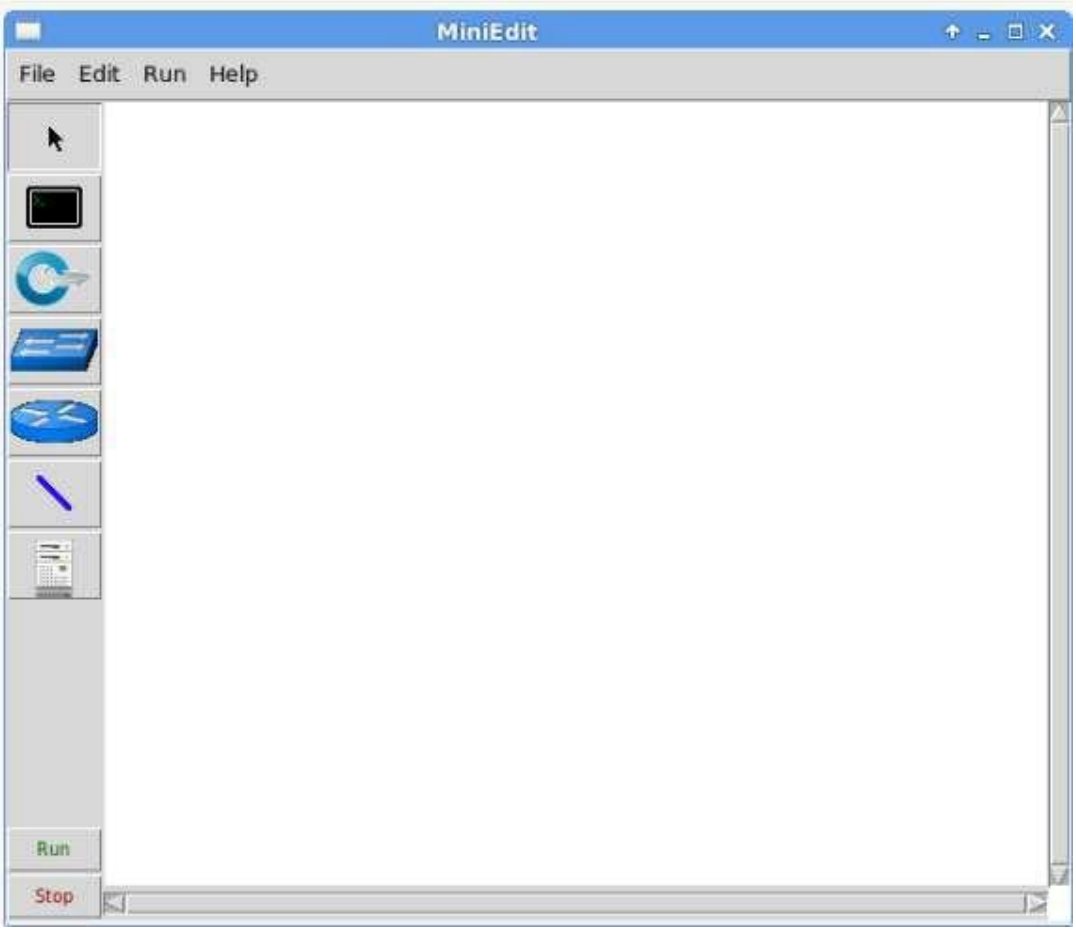


Рисунок 2.3 — Стартове робоче вікно та базовий інтерфейс графічного редактора MiniEdit

За замовчуванням, якщо користувач не вказує інші параметри, система MiniEdit автоматично генерує та підключає базовий еталонний контролер OpenFlow, вбудований у Mininet. Цей стандартний контролер здебільшого емулює класичну поведінку мережевого комутатора з функцією самонавчання, що дозволяє мережі почати функціонувати одразу після запуску без написання складної власної логіки маршрутизації. Таким чином, симбіоз технології Mininet та її графічної надбудови пропонує надзвичайно зручне, комплексне рішення для швидкого створення прототипів та глибокого математичного моделювання інформаційних потоків. Цей підхід слугує високоефективною та економічно виправданою альтернативою побудові дорогих фізичних тестових стендів із реального телекомунікаційного обладнання. Процес створення віртуальної мережевої інфраструктури безпосередньо в середовищі Mininet

відкриває шляхи для миттєвого тестування найрізноманітніших системних конфігурацій, перевірки гіпотез та виявлення вузьких місць в архітектурі. Використання MiniEdit суттєво полегшує візуальне сприйняття складних мережевих процесів та надає дослідникам максимально зручний, багатий функціонал для проведення фундаментальних наукових чи інженерних мережевих досліджень. Візуалізація комплексної архітектури програмно-конфігурованої мережі, розгорнутої за допомогою цього інструменту, представлена на рис. 2.4.

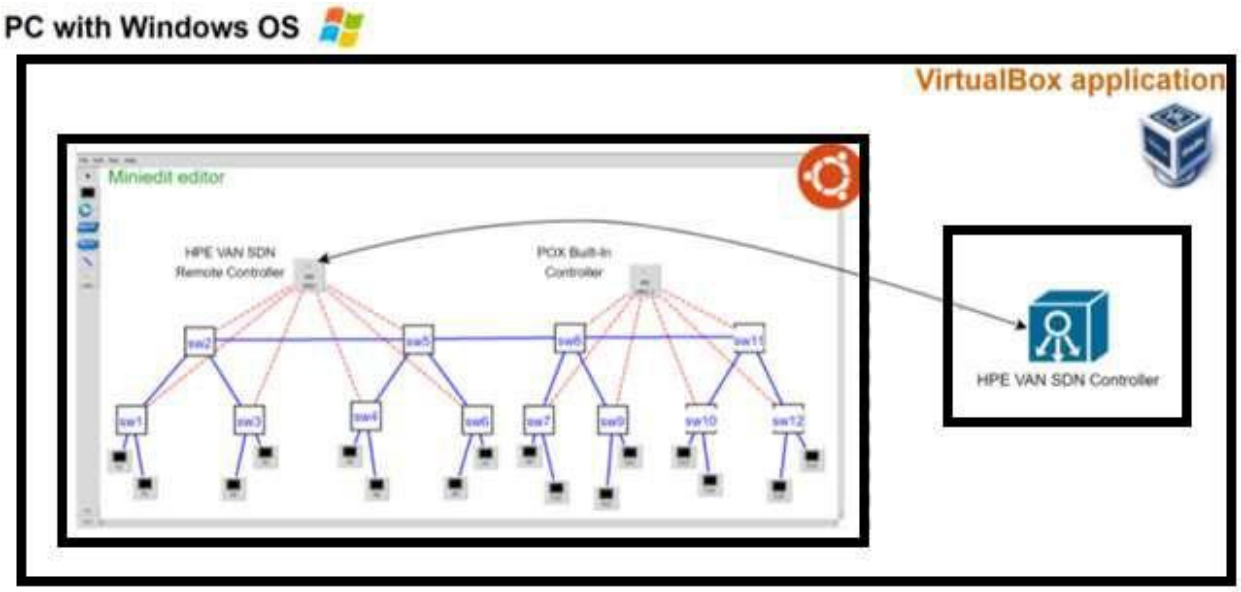


Рисунок 2.4 — Концептуальна структурна схема SDN-мережі, змодельованої у середовищі Mininet

Що стосується практичного аспекту роботи з програмою, то для ініціалізації та подальшого налаштування графічного редактора MiniEdit необхідно виконати певну послідовність системних дій у терміналі операційної системи. Оскільки емулятор глибоко взаємодіє з мережевим стеком ядра, процес запуску вимагає обов'язкової наявності найвищих привілеїв адміністратора (прав доступу рівня root). Відповідний алгоритм виклику програми через передачу параметрів інтерпретатору в середовищі Mininet концептуально відображено на рис. 2.5.



Рисунок 2.5 — Демонстрація процесу ініціалізації графічного середовища MiniEdit через командний рядок

Під час роботи в консолі інколи можуть виникати специфічні системні помилки, пов'язані зі шляхами до файлів або змінними середовища. У таких випадках розробники передбачили альтернативний, більш надійний алгоритм ручного запуску через послідовну навігацію файловою системою. Цей метод полягає у тому, щоб за допомогою базових команд управління директоріями спочатку перейти до системної папки, де безпосередньо зберігається виконуваний файл скрипту, а вже потім ініціювати його виконання. Логіку цього альтернативного, покрокового методу виклику програми проілюстровано на рис. 2.6.

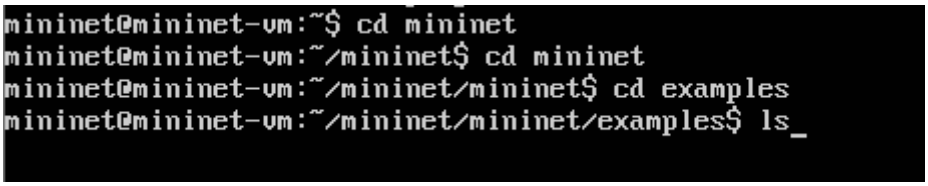


Рисунок 2.6 — Альтернативний алгоритм поетапного запуску програми через навігацію каталогами файлової системи

Зазвичай необхідний інструментарій знаходиться у спеціальному каталозі з прикладами, який постачається разом із базовим пакетом емулятора. Перейшовши до цієї директорії, користувач може переглянути повний перелік доступних демонстраційних скриптів та додатків, що наочно продемонстровано на рис. 2.7.

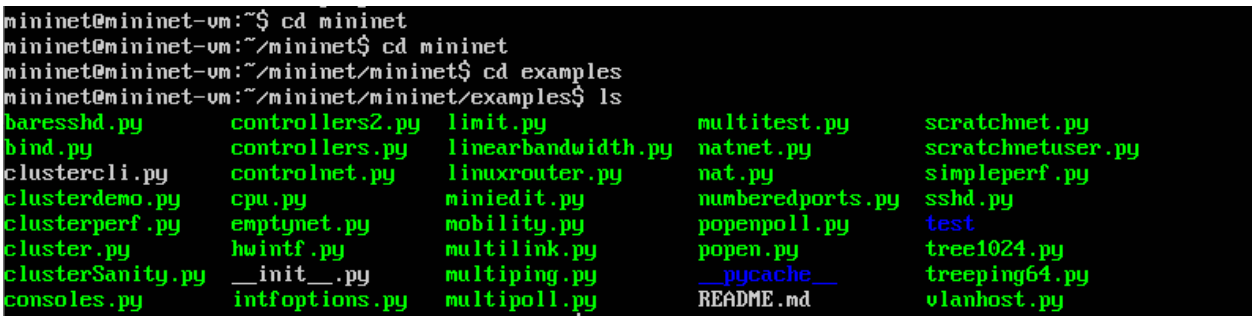


Рисунок 2.7 — Вміст системного каталогу зі стандартними прикладами та скриптами Mininet

Після успішного переходу до правильної папки та візуального підтвердження наявності потрібного файлу скрипту, користувачеві залишається лише виконати фінальну команду його запуску через мовний інтерпретатор, знову ж таки, не забуваючи про права суперкористувача. Цей завершальний крок процесу активації графічного інтерфейсу відображено на рис. 2.8.



Рисунок 2.8 — Фінальний етап виконання стартового скрипту для виклику вікна MiniEdit

Після того як стартовий скрипт успішно відпрацює і на екрані з'явиться головне робоче поле MiniEdit, першочерговим завданням дослідника є перехід до меню базових налаштувань програми. Активація цього пункту меню викликає спеціальне діалогове вікно параметрів, вигляд якого наведено на рис. 2.9.

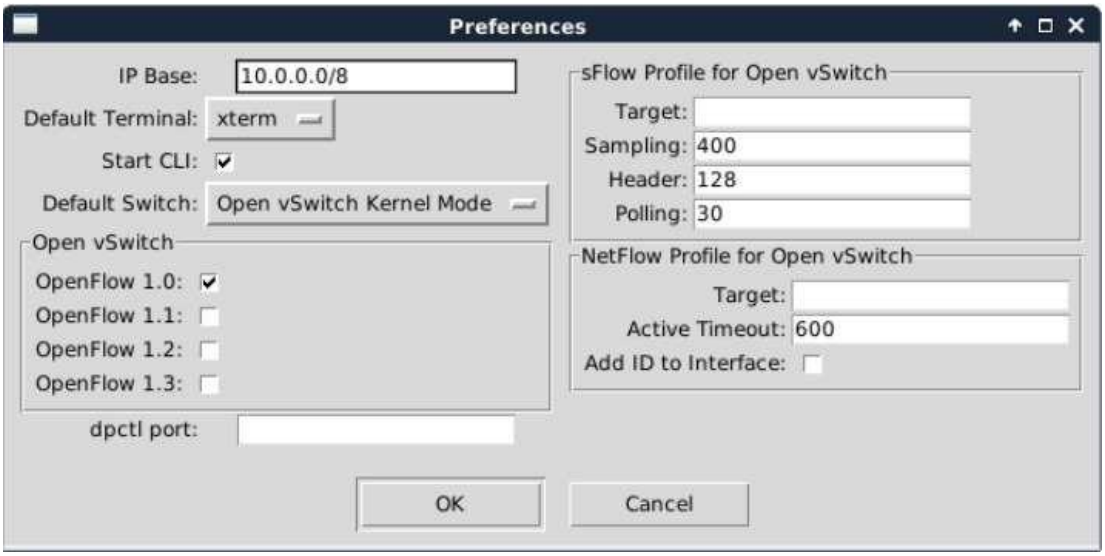


Рисунок 2.9 — Інтерфейс діалогового вікна глобальних налаштувань мережевого середовища

У цьому вікні конфігурації критично важливо активувати спеціальний параметр, який відповідає за паралельний запуск інтерфейсу командного рядка. Встановлення цієї позначки гарантує, що під час запуску графічної симуляції користувач додатково отримає повноцінний доступ до терміналу емулятора для тонкого ручного управління вузлами. Також у цьому ж розділі необхідно чітко визначити та обрати версію протоколу OpenFlow, яка буде використовуватися всіма комутаторами в поточній топології, щоб уникнути конфліктів сумісності з контролером. Усі ці ретельно підібрані налаштування відіграють ключову роль, оскільки саме вони є фундаментом для коректної автоматичної генерації програмних сценаріїв мережі, що в підсумку дозволить віртуальним хостам безперешкодно обмінюватися пакетами даних один з одним. Важливо відзначити, що вказані параметри зберігаються індивідуально для кожного конкретного проекту. Це надає інженеру гнучку можливість встановлювати абсолютно різні версії протоколів та параметри управління для кожної нової розроблюваної мережевої топології.

Базовим інструментом для роботи з командним рядком у середовищах на базі Unix, що використовують графічну підсистему X Window System, є класичний емулятор терміналу Xterm. Ця програма надає користувачам

можливість розгортати безліч незалежних термінальних вікон на одному фізичному моніторі. Кожен такий віртуальний термінал функціонує як повністю автономне середовище, забезпечуючи ізольовані потоки введення та виведення даних для запущених у ньому процесів, якими найчастіше виступають різноманітні командні оболонки Unix (наприклад, bash). Завдяки своїй надійності, архітектура Xterm стала своєрідним стандартом, на основі якого було розроблено багато інших сучасних термінальних клієнтів для X-серверів. Цікавою особливістю оригінального інтерфейсу є те, що за замовчуванням у ньому відсутній звичний видимий рядок меню. Для виклику одного з трьох доступних контекстних меню адміністратору необхідно застосовувати комбінацію утримання клавіші Control разом із натисканням лівої, середньої або правої кнопки миші. Проте розробники передбачили гнучкість: за необхідності підтримку традиційного видимого рядка меню можна жорстко активувати ще на етапі компіляції вихідного коду програми.

Іншим надзвичайно важливим інструментом для мережевого адміністрування є PuTTY — вільно поширюваний програмний комплекс, головне призначення якого полягає у забезпеченні безпечного віддаленого підключення до серверних вузлів. Програма використовує набір стандартизованих мережевих протоколів для ініціалізації та підтримки віддалених сеансів зв'язку, виступаючи надійним клієнтським додатком. З технічної точки зору, PuTTY виконує роль високоякісного емулятора терміналу. Вона віртуально імітує поведінку фізичного пристрою введення-виведення, дозволяючи адміністратору встановлювати захищене з'єднання з віддаленим сервером. Процес роботи виглядає максимально прозоро: команди, введені локально у вікні програми, миттєво передаються та виконуються на стороні сервера, а всі результати обробки та відповіді транслюються назад і відображаються на екрані користувача, дозволяючи здійснювати комплексне віддалене управління. Зовнішній вигляд вікна конфігурації підключень цієї програми наведено на рис. 2.10.

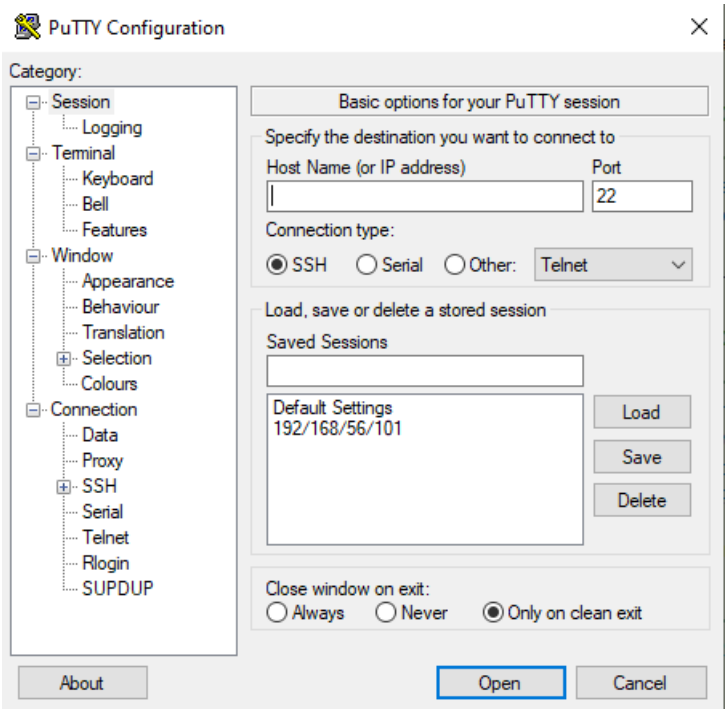


Рисунок 2.10 — Інтерфейс головного вікна конфігурації параметрів підключення у програмі PuTTY

Історично PuTTY розроблялася насамперед для операційних систем сімейства Windows, щоб надати їхнім користувачам зручний інструмент (який також включає логіку емуляції xterm) для взаємодії з Unix-серверами. Програма пропонує багатий арсенал функцій, що значно спрощують рутинну роботу інженера. Серед ключових можливостей варто виділити гнучке налаштування візуального оформлення, включаючи вибір шрифтів, кольорової палітри та роздільної здатності консолі. Для цілей аудиту передбачена функція детального логування сеансів у текстові файли. З точки зору безпеки, програма підтримує збереження криптографічних ключів авторизації для швидкої та надійної автентифікації без вводу паролів. Крім того, комплекс здатен стабільно працювати через проксі-сервери різних типів і містить вбудовані утиліти для безпечної передачі файлів.

Для глибокого інспектування мережевих пакетів незамінним інструментом є Wireshark — потужний аналізатор (сніфер) із відкритим вихідним кодом. Ця програма створена для детального перехоплення та аналізу даних, що циркулюють в мережах Ethernet та інших середовищах. На

відміну від класичних консольних утиліт (таких як tcpdump), Wireshark пропонує надзвичайно зручний графічний інтерфейс. Він володіє потужними вбудованими алгоритмами для складного сортування та багатовимірної фільтрації пакетів. Перевівши мережевий адаптер комп'ютера у «змішаний» режим (promiscuous mode), дослідник отримує можливість перехоплювати абсолютно весь трафік, що проходить через сегмент мережі в реальному часі, отримуючи панорамний огляд інформаційного обміну.

Визначною особливістю Wireshark є його здатність розпізнавати, декодувати та інтерпретувати внутрішню структуру величезної кількості різноманітних мережевих протоколів. Програма автоматично розбирає кожен перехоплений пакет, наочно відображаючи значення конкретних полів на різних рівнях ієрархії. Технічно для захоплення даних на льоту аналізатор покладається на системну бібліотеку pcap, що дещо обмежує пряме перехоплення лише тими типами мереж, які нею підтримуються. Однак цей нюанс нівелюється високою сумісністю Wireshark із зовнішніми форматами: користувач може легко імпортувати для аналізу дампи трафіку, зібрані іншим програмним забезпеченням. Головне робоче вікно цього аналізатора представлено на рис. 2.11.

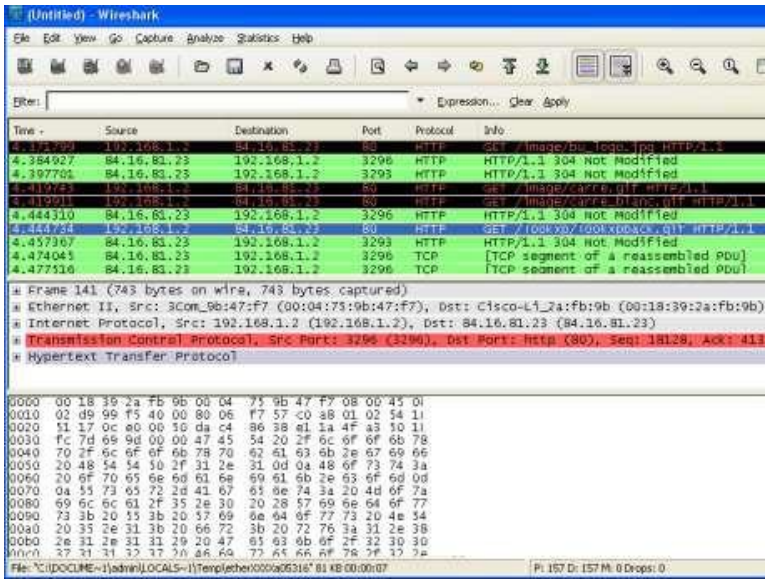


Рисунок 2.11 — Загальний вигляд графічного інтерфейсу та робочого простору аналізатора мережевого трафіку Wireshark

Програмний комплекс Wireshark є абсолютно вільним і поширюється на умовах ліцензії GNU GPL. Для побудови свого мультиплатформного графічного інтерфейсу програма використовує кросплатформну бібліотеку компонентів GTK+ (або Qt). Завдяки цьому аналізатор доступний та коректно функціонує на широкому спектрі операційних систем, включаючи більшість дистрибутивів на базі UNIX (GNU/Linux, Solaris, FreeBSD, NetBSD, OpenBSD), а також у середовищах Mac OS X та Microsoft Windows.

3 ПРАКТИЧНІ ПІДХОДИ ДО ПРОЕКТУВАННЯ ТА РОЗГОРТАННЯ ТОПОЛОГІЙ ПРОГРАМНО-КОНФІГУРОВАНИХ МЕРЕЖ У СЕРЕДОВИЩІ MININET

3.1 Аналіз та практичне моделювання архітектурних мережевих топологій за допомогою інструментарію Mininet

Гнучкий інструментарій віртуального середовища Mininet надає інженерам та дослідникам широкі можливості для точного математичного та логічного моделювання найрізноманітніших сценаріїв побудови комп'ютерних мереж. Це дозволяє проводити глибокі стрес-тести, перевіряти коректність роботи нових мережевих протоколів та відлагоджувати складні алгоритми маршрутизації в безпечних, контрольованих умовах. Кожна з доступних для генерації базових топологій характеризується власними унікальними архітектурними особливостями, перевагами та обмеженнями. Відповідно, вибір конкретної структурної схеми завжди диктується специфічними цілями та завданнями поточного наукового дослідження або інженерного експерименту. Наприклад, одні структури краще підходять для перевірки пропускної здатності, тоді як інші — для аналізу відмовостійкості чи затримок. Глибоке розуміння фундаментальних властивостей цих топологій є критично важливим для максимально ефективного використання всього потенціалу емулятора Mininet під час проведення будь-яких мережевих досліджень.

Найбільш базовим та архітектурно простим варіантом організації мережі є однорівнева (зіркоподібна) топологія. Її концепція полягає в тому, що у системі функціонує лише один центральний комутатор, до якого безпосередньо, радіально підключається певна кількість кінцевих пристроїв (хостів). У такій конфігурації всі хости формують прямі, однорангові логічні з'єднання між собою через цей єдиний центральний вузол комутації. Завдяки

своїй максимальній простоті та передбачуваності, такий тип організації мережі вважається ідеальним стартовим майданчиком для проведення базових ознайомчих експериментів, написання перших скриптів, а також для академічного навчання основам мережевої взаємодії. Для розгортання такої базової топології в консолі емулятора використовується спеціальна команда ініціалізації середовища з параметрами за замовчуванням, процес введення якої зафіксовано на рис. 3.1.

```
mininet@mininet-vm:~$ cd mininet
mininet@mininet-vm:~/mininet$ cd examples
mininet@mininet-vm:~/mininet/examples$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Рисунок 3.1 — Демонстрація синтаксису консольної команди для автоматичної генерації простої однорівневої топології

Після того як базова мережа успішно створена та запущена з правами суперкористувача, дослідникові необхідно переконатися в її працездатності. Для цього в арсеналі інтерфейсу командного рядка Mininet передбачено кілька зручних вбудованих утиліт для діагностики. Одним із найперших кроків є перевірка наскрізної зв'язності між усіма вузлами мережі. Це досягається

шляхом виклику спеціальної команди утиліти автоматичного пінгування, яка послідовно відправляє ICMP-запити від кожного хоста до всіх інших хостів у поточній топології, формуючи матрицю доступності. Результат виконання цієї глобальної перевірки зв'язку наведено на рис. 3.2.

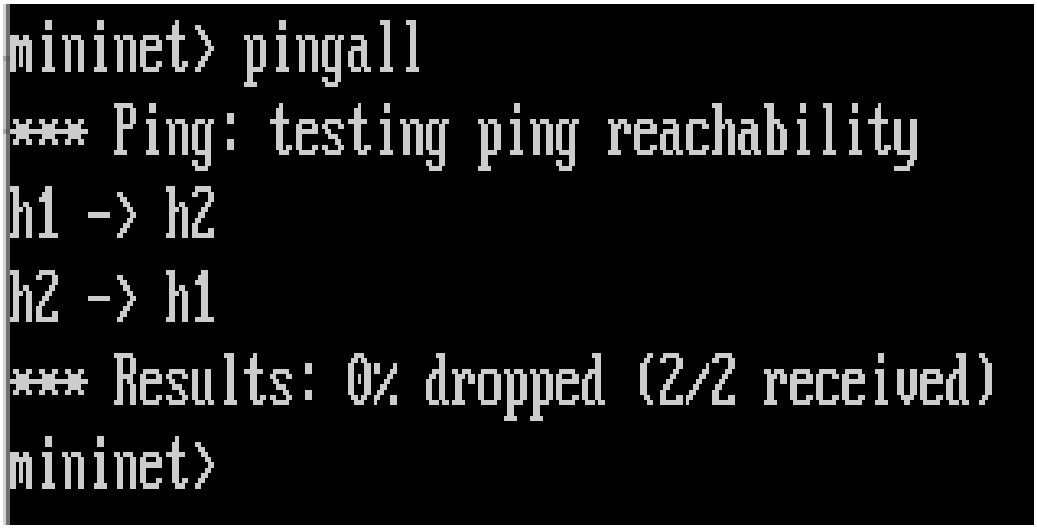


Рисунок 3.2 — Результати виконання вбудованої команди глобальної перевірки зв'язності хостів

Окрім простої перевірки наявності зв'язку, часто виникає потреба оцінити реальну пропускну здатність створених віртуальних каналів. Для цього використовується команда виклику відомого мережевого інструменту генерації трафіку. При її активації емулятор автоматично призначає один хост клієнтом, а інший — сервером, після чого починає інтенсивну передачу TCP або UDP пакетів між ними, фіксуючи максимальну швидкість передачі даних. Процес ініціалізації та результати такого навантажувального тестування проілюстровано на рис. 3.3.



Рисунок 3.3 — Виклик консольної утиліти для вимірювання максимальної пропускної здатності віртуальних каналів зв'язку

Для більш просунутих користувачів, які віддають перевагу управлінню мережею через об'єктно-орієнтовані скрипти (наприклад, використовуючи Python API), існують програмні аналоги згаданих консольних команд. Так, для ініціації глобальної перевірки зв'язності через програмний інтерфейс об'єкта мережі використовується спеціальний метод класу. Виклик цього методу в коді автоматизує процес пінгування всіх вузлів, аналогічно до консольної команди. Логіку застосування цього програмного методу показано на рис. 3.4.

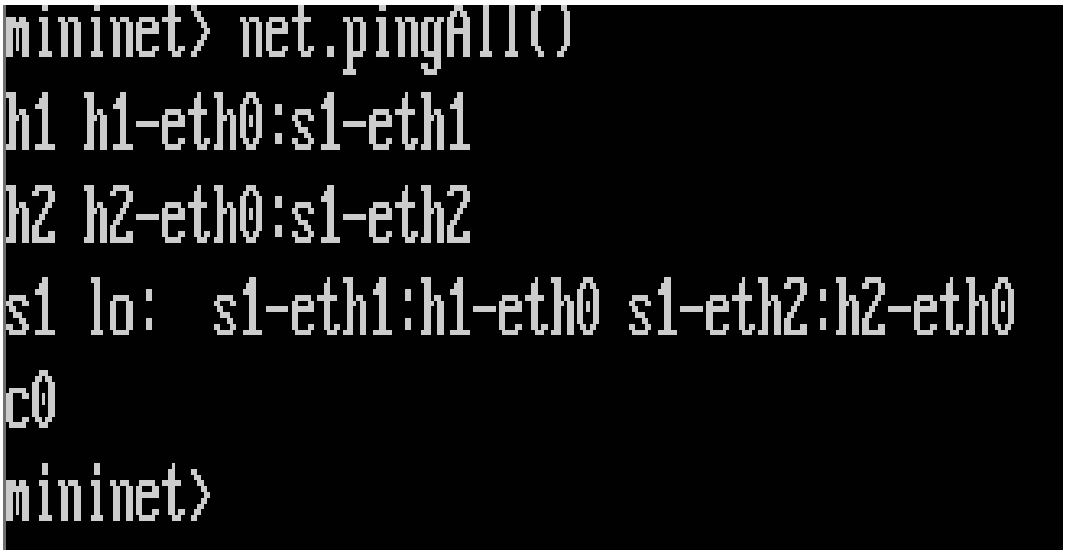


Рисунок 3.4 — Використання програмного методу об'єкта мережі для виконання глобальної перевірки доступності вузлів

Відповідно, для проведення тестів продуктивності з середовища виконання скриптів також використовується спеціалізований метод об'єкта

мережі, який звертається до утиліти вимірювання пропускної здатності. Цей метод дозволяє програмно ініціювати тест між заданими вузлами та отримати результати у вигляді даних, придатних для подальшого аналізу алгоритмом. Приклад виклику цієї програмної функції наведено на рис. 3.5.

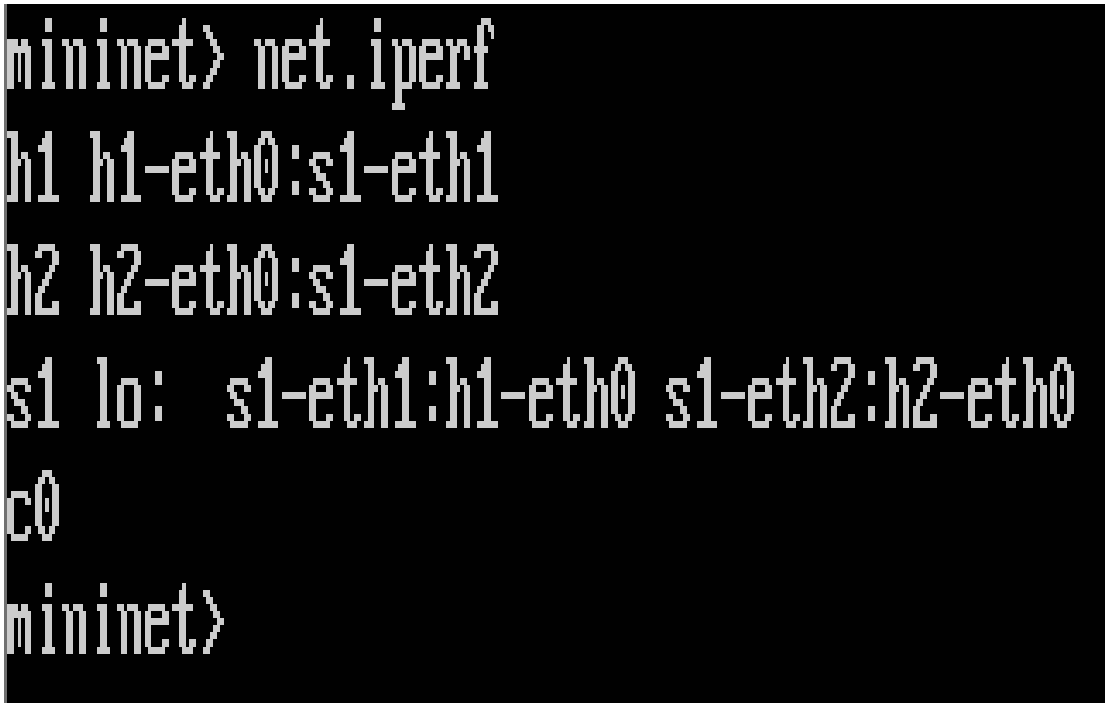


Рисунок 3.5 — Програмна ініціалізація інструменту вимірювання швидкості передачі даних через API емулятора

Використання цих діагностичних інструментів — як консольних, так і програмних — є фундаментальним кроком для підтвердження коректності створеної топології перед початком складніших експериментів. Звісно, залежно від специфіки досліджень, цей базовий інструментарій завжди можна розширити шляхом інтеграції додаткових спеціалізованих утиліт для глибокого аналізу трафіку чи моніторингу затримок.

Більш складною та наближеною до реальних корпоративних мереж є деревоподібна ієрархічна топологія. У цьому сценарії комутаційні пристрої організовані у вигляді суворої багаторівневої структури (дерева). На вершині ієрархії знаходиться головний (кореневий) комутатор, який з'єднується з комутаторами нижчого рівня (агрегації), а ті, у свою чергу, з комутаторами

доступу, до яких вже безпосередньо підключаються кінцеві хости. Така архітектурна модель ідеально підходить для проектування розгалужених, масштабованих мереж із широкими можливостями маршрутизації та резервування каналів. Гнучкість цієї топології полягає в тому, що вона дозволяє точно моделювати поведінку трафіку на різних рівнях абстракції.

Для генерації мережі такого типу в емуляторі використовується консольна команда із зазначенням відповідного ключа конфігурації топології. Якщо запустити цю команду без додаткових аргументів, система автоматично побудує стандартне дерево, що складається з чотирьох рівнів вкладеності комутаторів, де до кожного комутатора нижнього рівня буде підключено по одному хосту. Демонстрація синтаксису цієї команди представлена на рис. 3.6.

```
mininet@mininet-vm:~$ sudo mn --topo tree
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Рисунок 3.6 — Команда ініціалізації ієрархічної деревоподібної мережевої структури

Проте емулятор надає інженерам повну свободу у налаштуванні параметрів цього «дерева». Використовуючи додаткові прапорці в командному рядку, можна чітко специфікувати глибину мережі (кількість рівнів комутаторів) та коефіцієнт розгалуження (кількість дочірніх вузлів, підключених до кожного батьківського). Наприклад, можна передати системі параметри для створення дерева глибиною у три рівні, де кожен комутатор матиме рівно по два підключення вниз. Це дозволяє створювати надзвичайно точні математичні моделі конкретних корпоративних мереж.

Лінійна топологія являє собою послідовний ланцюг, у якому мережеві комутатори з'єднані один за одним у єдину лінію. Відповідно, до кожного комутатора в цьому ланцюжку підключається певна кількість хостів. У такій архітектурі кожен транзитний вузол має лише два прямих з'єднання: зі своїм попереднім та наступним сусідом по магістралі. Цей специфічний вид топології має велику цінність для науковців, які досліджують базові механізми пересилання пакетів, оскільки тут повністю відсутні замкнені контури (петлі маршрутизації).

Для того щоб розгорнути таку мережу, використовується консольна команда з параметром лінійної структури, до якої обов'язково додаються числові аргументи. Ці аргументи визначають загальну кількість хостів та відповідну їм кількість комутаторів, які сформують ланцюг. Наочний приклад виклику команди для створення простої лінії з двох сегментів наведено на рис. 3.7.

```
mininet@mininet-vm:~$ sudo mn --topo=linear,2,2
*** Creating network
*** Adding controller
*** Adding hosts:
h1s1 h1s2 h2s1 h2s2
*** Adding switches:
s1 s2
*** Adding links:
(h1s1, s1) (h1s2, s2) (h2s1, s1) (h2s2, s2) (s2, s1)
*** Configuring hosts
h1s1 h1s2 h2s1 h2s2
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
```

Рисунок 3.7 — Застосування командного рядка для побудови лінійної (ланцюгової) мережевої топології

Головна особливість лінійної структури полягає в маршрутизації: пакет даних, відправлений від першого вузла до останнього, змушений послідовно пройти через абсолютно всі проміжні комутатори. Така лінійна послідовність робить процес проходження трафіку максимально прозорим та легким для аналізу. Саме тому лінійні топології часто використовують для вимірювання кумулятивних мережевих затримок (latency) на кожному хопі, а також для вивчення того, як пропускна здатність деградує при збільшенні довжини магістралі. Таким чином, лінійна мережа в Mininet є чудовим лабораторним інструментом для тестування фундаментальних принципів роботи транзитного трафіку.

Специфічною варіацією класичної лінійної архітектури є так звана "зворотна" (reversed) топологія. Її принципова відмінність полягає у зміненому алгоритмі призначення ідентифікаторів (MAC/IP адрес та номерів портів) під час з'єднання вузлів. Якщо у стандартній лінійній схемі з'єднання створюються у прямому порядку (перший комутатор до другого, другий до третього), то у

зворотній топології логіка ініціалізації портів та лінків відбувається у зворотному напрямку. Ця, на перший погляд, незначна деталь має критичне значення для певного класу завдань.

Зокрема, зворотна топологія є надзвичайно корисною при глибинному тестуванні специфічних протоколів маршрутизації, перевірці правильності роботи складних алгоритмів балансування навантаження, а також під час аудиту політик безпеки. У таких сценаріях сам порядок, у якому комутатори реєструються в контролері та обмінюються службовою інформацією, може суттєво впливати на кінцевий результат роботи алгоритму. Команда для генерації такої нестандартної конфігурації вказується за допомогою відповідного ключа в консолі, що проілюстровано на рис. 3.8.

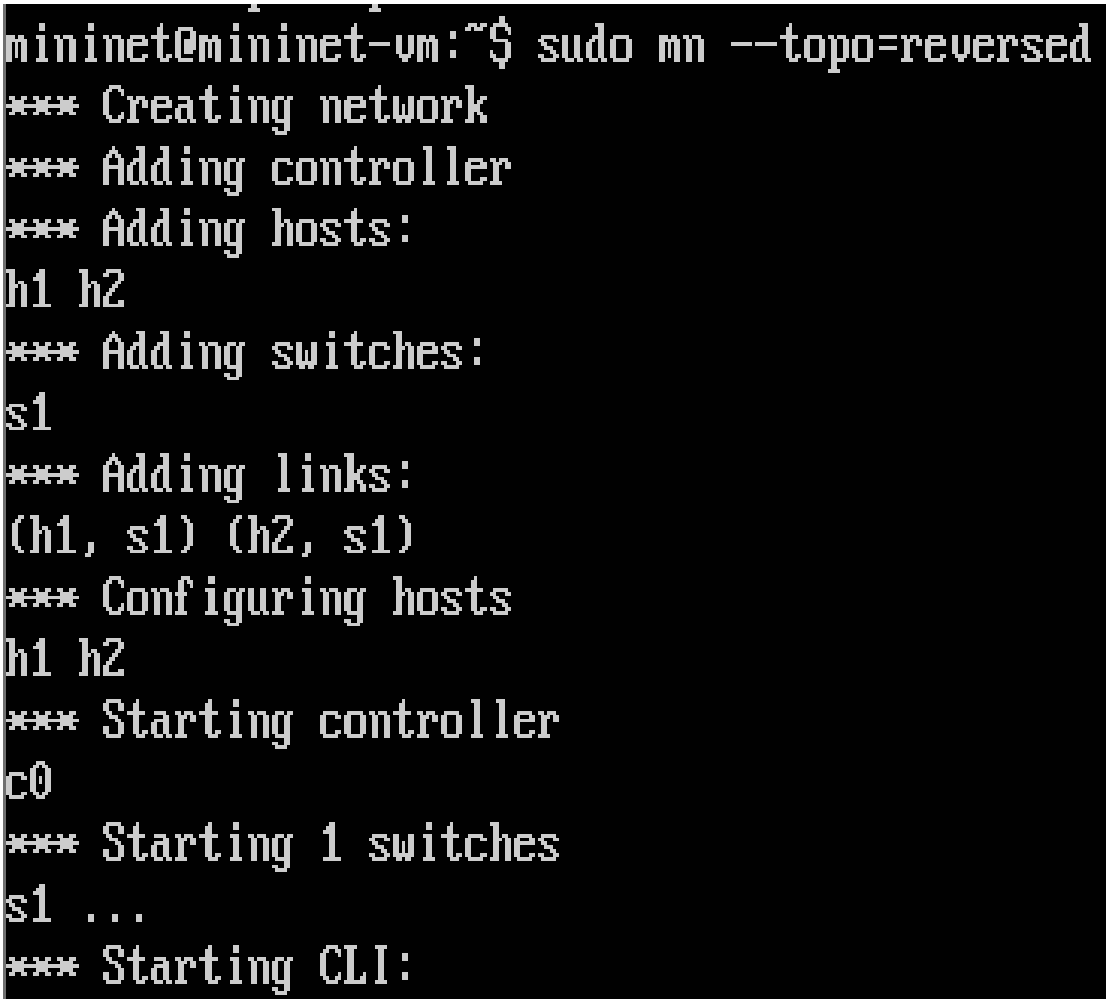


Рисунок 3.8 — Консольна команда для генерації лінійної топології зі зворотним порядком ініціалізації з'єднань

Загалом, це лише базовий перелік стандартних топологій, які можна розгорнути «з коробки». Використовуючи їх як конструктор, інженер здатний імітувати безліч різноманітних умов, аналізуючи, як архітектура мережі безпосередньо впливає на її продуктивність та стабільність роботи додатків.

Окрім створення мереж, дослідникам часто потрібно отримувати детальну інформацію про структуру вже розгорнутої топології під час виконання скриптів. Для цього в арсеналі Mininet (при використанні Python API) є набір спеціальних програмних методів та об'єктів.

Основою взаємодії є звернення до головного об'єкта, який представляє всю поточну запущену мережу в системі. Через цей об'єкт можна отримати доступ до спеціального під-об'єкта топології, який зберігає повну граф-модель створеної мережі. Маючи доступ до цього графа, програміст може використовувати спеціальні функції-гетери для вилучення конкретних даних. Наприклад, існують методи, які повертають повні списки всіх існуючих вузлів (хостів та комутаторів разом) у системі. Якщо потрібна більша деталізація, можна викликати окремі функції, які відфільтрують та повернуть масиви виключно зі списком кінцевих хостів або виключно зі списком активних комутаторів. Крім того, критично важливою є можливість отримати повний масив усіх логічних з'єднань (лінків) між вузлами, що дозволяє програмно аналізувати побудовані маршрути. Використання цих функцій API є незамінним при написанні складних автоматизованих тестів та розробці власних контролерів, яким потрібна актуальна інформація про стан мережевої інфраструктури.

3.2 Практична реалізація та конструювання мережевих топологій у віртуальному середовищі Mininet

Для того щоб програмним шляхом спроектувати та розгорнути базову мережеву топологію, яка складається з одного центрального комутатора та кількох підключених до нього кінцевих вузлів (хостів), необхідно скористатися можливостями спеціалізованої бібліотеки Mininet мовою Python. Замість використання стандартних консольних шаблонів, розробнику потрібно визначити власний об'єктно-орієнтований клас топології, який стане програмним фундаментом для генерації мережі.

Логіка роботи такого програмного скрипту будується на чіткій та послідовній архітектурі. Спочатку в коді оголошується спеціальний клас, який є нащадком базового системного класу Topo з бібліотеки емулятора. Усередині цього класу реалізується метод ініціалізації, який програмно додає в середовище один комутатор та динамічно (зазвичай за допомогою циклу) генерує задану кількість хостів, одразу створюючи логічні канали зв'язку між кожним кінцевим вузлом та цим єдиним комутатором. Далі у скрипті визначається головна виконувана функція. Її завдання полягає в тому, щоб створити екземпляр описаної раніше топології, ініціалізувати на її основі об'єкт мережі Mininet та безпосередньо запустити симуляцію. Після успішного старту ця функція автоматично виводить на екран терміналу службову інформацію про стан усіх створених підключень, а потім ініціює базовий тест зв'язності (пінгування) між усіма вузлами для підтвердження працездатності маршрутизації. Наприкінці скрипту розміщується стандартна перевірка, яка визначає, чи файл запущено як самостійну програму. Якщо так, скрипт встановлює рівень деталізації системного журналу на вивід інформаційних повідомлень та викликає головну функцію тестування. Загалом, цей програмний алгоритм є класичним прикладом автоматизованого створення та базової діагностики простої зіркоподібної мережі.

Процес практичного запуску створеного програмного сценарію вимагає виконання кількох кроків в операційній системі. Досліднику необхідно відкрити системний термінал, перейти до каталогу, де збережено файл зі скриптом, та запустити його за допомогою інтерпретатора Python з відповідними правами доступу. Під час виконання цього коду користувач не побачить графічних вікон; натомість у консолі почне з'являтися детальний текстовий вивід. Цей системний журнал відображатиме процес підняття віртуальних мережевих інтерфейсів, інформуватиме про успішне встановлення з'єднань між хостами та комутатором, а в кінці виведе статистичну таблицю результатів автоматичного тесту зв'язності, демонструючи відсоток успішно доставлених пакетів між усіма парами пристроїв.

Для більш глибокого розуміння можливостей емулятора доцільно розглянути процес створення складнішої, повністю кастомної конфігурації мережі. У цьому сценарії розробляється модель, яка включатиме два комутатори, з'єднані між собою транзитним каналом, та чотири хости, розподілені між ними. Програмна реалізація такої архітектури передбачає створення нового класу топології, де ключова відмінність полягає в перевизначенні методу побудови графа мережі. У тілі цього методу розробник явно декларує створення двох окремих комутаторів та чотирьох незалежних хостів. Після генерації самих об'єктів алгоритм послідовно викликає функцію додавання лінків: спочатку він підключає перші два хости до першого комутатора, потім наступні два хости — до другого. Кульмінацією цього методу є програмне створення магістрального з'єднання між двома комутаторами, що об'єднує два сегменти в єдину інфраструктуру.

Наступним кроком є написання функції запуску цієї кастомної топології. У ній створюється об'єкт розробленого класу та запускається ядро Mininet. Після старту мережі автоматично виконується глобальне пінгування всіх вузлів, результати якого дозволяють переконатися, що трафік успішно проходить навіть через магістральний канал між комутаторами. Після

завершення всіх діагностичних тестів скрипт зобов'язаний коректно зупинити симуляцію та вивільнити системні ресурси за допомогою відповідної команди зупинки. Зберігши цей алгоритм і запустивши його через термінал, користувач отримає консольний звіт, який підтвердить, що всі чотири хости успішно зв'язуються зі своїми локальними комутаторами, а комутатори обмінюються даними між собою, забезпечуючи стовідсоткову наскрізну доступність усієї віртуальної мережі.

У результаті виконання вищеописаного кастомного сценарію формується повноцінна працездатна мережа. Архітектурно це виглядає як два сегменти доступу, об'єднані єдиним мостом: комутатори з'єднані між собою, утворюючи ядро, а кінцеві хости радіально підключені до відповідних комутаторів. Візуальне подання та загальний вигляд цієї створеної кастомної топології наведено на рис. 3.9.

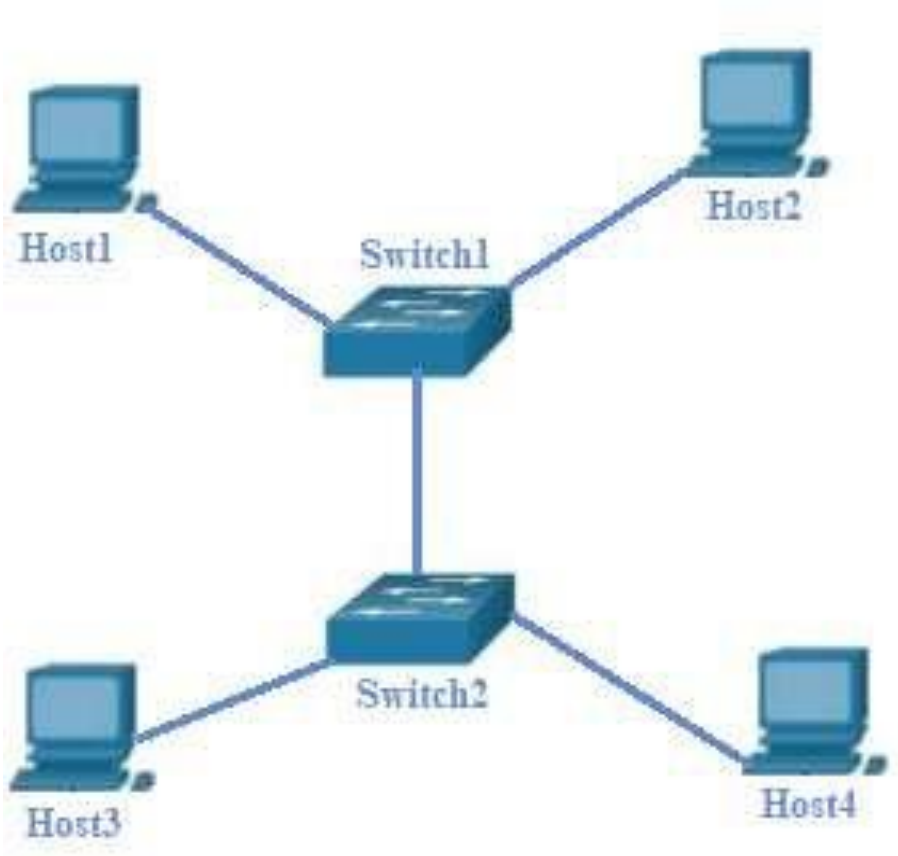


Рисунок 3.9 — Візуальне подання загального вигляду та структурної організації створеної кастомної мережевої топології

3.3 Аналіз продуктивності та результати тестування змодельованих мережевих топологій

Для повноцінної оцінки ефективності розгорнутих програмно-конфігурованих мереж у віртуальному середовищі Mininet було проведено серію навантажувальних тестувань. Головною метою цього етапу стало практичне вимірювання базових метрик якості обслуговування (QoS), таких як максимальна пропускна здатність каналів зв'язку, середній час кругової затримки пакетів (RTT) та відсоток втрачених даних під час пікових навантажень. Як основний інструментарій для генерації трафіку використовувалася вбудована мережева утиліта iperf, тоді як показники затримок фіксувалися за допомогою стандартних ICMP-запитів (ping). Дослідження проводилося для трьох базових конфігурацій: простої однорівневої (зіркоподібної) топології, лінійної магістралі з чотирма вузлами та складної деревоподібної структури з глибиною у три рівні.

Отримані емпіричні дані наочно демонструють, як саме ускладнення архітектури впливає на загальну швидкість обміну інформацією між кінцевими хостами. В однорівневій топології, де всі пристрої підключені до єдиного комутатора, очікувано спостерігаються найкращі показники продуктивності через повну відсутність додаткових транзитних вузлів. Натомість лінійна та деревоподібна структури показують закономірне зростання затримок та певне зниження пропускної здатності. Це безпосередньо зумовлено витратами машинного часу на обробку пакетів та прийняття рішень щодо маршрутизації кожним наступним комутатором у ланцюгу. Узагальнені результати проведених вимірювань структуровано та зведено у порівняльну таблицю 3.1.

Таблиця 3.1. Узагальнені результати проведених вимірювань.

Тип мережевої топології	Середня пропускна здатність (Мбіт/с)	Середній час затримки (RTT, мс)	Відсоток втрати пакетів (%)
Однорівнева (Single, 1 комутатор)	945.2	0.042	0.0
Лінійна (Linear, 4 комутатори)	880.5	0.115	0.0
Деревоподібна (Tree, глибина 3)	815.3	0.284	0.1

Окрім базових метрик пропускної здатності та затримок на рівні площини передачі даних, критично важливим аспектом дослідження програмно-конфігурованих мереж є оцінка навантаження на централізовану площину управління. В архітектурі SDN, коли комутатор отримує новий пакет, для якого ще відсутнє відповідне правило маршрутизації у локальній таблиці потоків, він змушений генерувати спеціальний службовий запит до контролера. Цей процес ініціалізації нових інформаційних потоків створює додаткове навантаження на канал управління та неминуче збільшує початкову затримку передачі найпершого пакета в сесії. Під час експериментального моделювання в середовищі Mininet спеціально фіксувався час реакції еталонного контролера на одночасне масове надходження запитів від різних хостів. Спостереження підтвердили, що за умов різкого лавиноподібного зростання кількості нових підключень, час обробки службових запитів центральним вузлом починає зростати. Це емпірично доводить, що для масштабних промислових впроваджень з інтенсивним трафіком базових контролерів може бути недостатньо, і виникає об'єктивна необхідність

інтеграції більш продуктивних багатопотокових рішень, таких як Ryu, POX або ONOS.

Додатковим етапом практичного дослідження стало моделювання сценаріїв апаратної відмови для перевірки здатності створеної SDN-інфраструктури до швидкого самовідновлення. У рамках складної деревоподібної топології було програмно імітовано раптовий розрив одного з магістральних каналів зв'язку між транзитними комутаторами. Завдяки тому, що контролер володіє глобальним, централізованим баченням усієї мережевої топології, він здатен практично миттєво ідентифікувати недоступність вузла та оперативно перерахувати альтернативні обхідні маршрути. Після обчислення нового шляху контролер автоматично розсилає оновлені інструкції та модифікує таблиці потоків на відповідних комутаторах. Зафіксований час конвергенції мережі — тобто період від моменту імітації падіння лінку до повного відновлення стабільної передачі корисних даних резервним маршрутом — виявився надзвичайно малим. Цей результат наочно підтверджує одну з головних архітектурних переваг SDN порівняно з традиційними децентралізованими протоколами динамічної маршрутизації, яким зазвичай потрібен значно довший час для обміну топологічними оновленнями між усіма автономними маршрутизаторами.

ВИСНОВКИ

Будь-яка сучасна інформаційна система вимагає надійної та гнучкої комп'ютерної мережі для забезпечення безперебійної передачі даних. У традиційних мережах маршрутизація здійснюється децентралізовано, що створює труднощі при масштабуванні, балансуванні навантаження та управлінні політиками безпеки. Оскільки кожен роутер самостійно розраховує маршрути, це може призводити до неоптимального використання каналів зв'язку та тривалого часу відновлення мережі при апаратних збоях. Вирішення цих проблем потребує переходу до централізованої моделі управління, яку надає архітектура програмно-конфігурованих мереж (SDN).

Аналізуючи вище сказане та дослідивши принципи побудови SDN-інфраструктури, протоколи взаємодії та методи централізованого управління трафіком, можна дійти висновку, що впровадження програмно-визначеної маршрутизації докорінно змінює підхід до адміністрування мереж. Використання SDN робить мережу максимально адаптивною, дозволяючи програмно керувати інформаційними потоками на основі глобального бачення всієї топології.

Досліджено поняття площини управління та площини передачі даних, а також обґрунтовано необхідність їх відокремлення. Наведено порівняльний аналіз традиційної децентралізованої маршрутизації та централізованого підходу SDN. Визначено переваги використання контролерів для динамічного обчислення найкоротших шляхів та балансування трафіку.

Приведено приклади практичного моделювання мережевих топологій у віртуальному середовищі Mininet. Описано процес генерації правил маршрутизації та продемонстровано реакцію програмно-конфігурованої мережі на обрив ліній зв'язку. Визначено етапи розгортання такої інфраструктури, починаючи від вибору контролера і закінчуючи тестуванням пропускної здатності та затримок.

Інвестиції в розробку інфраструктур на базі SDN, подальший їх розвиток та інтеграція з хмарними технологіями є актуальним і необхідним кроком. Розвиток програмно-визначеної маршрутизації дозволить значно скоротити експлуатаційні витрати, підвищити надійність мереж та забезпечити високу якість обслуговування (QoS) для критично важливих бізнес-додатків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Lantz B., Heller B., McKeown N. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks // ACM SIGCOMM Computer Communication Review. 2010. Vol. 40, № 1. P. 74–83.

2 Kreutz D., Ramos F. M., Verissimo P. E. Software-Defined Networking: A Comprehensive Survey // Proceedings of the IEEE. 2015. Vol. 103, № 1. P. 14–76.

3 Valencia B., Santacruz S., Becerra L. Y. Mininet: a versatile tool for emulation and prototyping of Software Defined Networking // Entre Ciencia e Ingenieria. 2015. Vol. 9, № 17. P. 62–70.

4 McKeown N., Anderson T., Balakrishnan H. OpenFlow: enabling innovation in campus networks // ACM SIGCOMM Computer Communication Review. 2008. Vol. 38, № 2. P. 69–74.

5 Kaur K., Singh J., Ghumman N. S. Mininet as Software Defined Networking Testing Platform // International Journal of Computer Applications. 2014. Vol. 104, № 11. P. 27–32.

6 Askar S., Keti F. Performance Evaluation of Different SDN Controllers // International Journal of Science and Business. 2021. Vol. 5, № 3. P. 83–91.

7 Nunes B. A., Mendonca M., Nguyen X. N. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks // IEEE Communications Surveys & Tutorials. 2014. Vol. 16, № 3. P. 1617–1634.

8 Sheikh M. N., Hwang I., Raza M. S. A Qualitative and Comparative Performance Assessment of Logically Centralized SDN Controllers via Mininet Emulator // MDPI Computers. 2024. Vol. 13, № 4. P. 85–102.

9 Fawzi S., Din N. M. A Mininet emulation study for SDN fat tree data center sleep mode routing algorithms // Edelweiss Applied Science and Technology. 2024. Vol. 8, № 6. P. 8956–8967.

10 Hu T., Guo Z., Yi P. Multi-controller Based Software-Defined Networking: A Survey // IEEE Access. 2018. Vol. 6, № 1. P. 15980–15996.

11 Baniya P., Nand P., Bhushan B. Performance Analysis of Existing SDN Load Balancing Controller Using Mininet // Future Generation Computer Systems. 2024. Vol. 12, № 1. P. 161–174.

12 Karthika M., Kumar A. Simulation of SDN in mininet and detection of DDoS attack using machine learning // Bulletin of Electrical Engineering and Informatics. 2023. Vol. 12, № 4. P. 2314–2322.

13 Shalimov A., Zuikov D., Zimarina D. Advanced Study of SDN/OpenFlow Controllers // Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia. 2013. Vol. 1, № 1. P. 1–6.

14 Karakus M., Durresi A. Quality of Service (QoS) in Software Defined Networking (SDN): A survey // Journal of Network and Computer Applications. 2017. Vol. 80, № 1. P. 200–218.

15 Jarschel M., Oechsner S., Schlosser D. Modeling and performance evaluation of an OpenFlow architecture // Proceedings of the 23rd International Teletraffic Congress. 2011. Vol. 1, № 1. P. 1–7.

16 Al-Fares M., Loukissas A., Vahdat A. A Scalable, Commodity Data Center Network Architecture // ACM SIGCOMM Computer Communication Review. 2008. Vol. 38, № 4. P. 63–74.

17 Gupta M., Singh A. K. Performance Analysis of Mininet Emulator for SDN based Networks // International Journal of Advanced Research in Computer Science. 2017. Vol. 8, № 5. P. 1234–1239.

18 McKeown N., Anderson T., Balakrishnan H. OpenFlow: enabling innovation in campus networks // ACM SIGCOMM Computer Communication Review. 2008. Vol. 38, № 2. P. 69–74.

19 Salman O., Elhajj I. H., Chehab A. SDN controllers: A comparative study // 18th Mediterranean Electrotechnical Conference. 2016. Vol. 1, № 1. P. 1–6.

20 Khondoker R., Zaalouk A., Marx R. Feature-based comparison and selection of Software Defined Networking (SDN) controllers // World Congress on Computer Applications and Information Systems. 2014. Vol. 1, № 1. P. 1–7.

21 Braun W., Menth M. Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices // Future Internet. 2014. Vol. 6, № 2. P. 302–336.

22 Akyildiz I. F., Lee A., Wang P. A roadmap for traffic engineering in SDN-OpenFlow networks // Computer Networks. 2014. Vol. 71, № 1. P. 1–30.

23 Sezer S., Scott-Hayward S., Chouhan P. K. Are we ready for SDN? Implementation challenges for software-defined networks // IEEE Communications Magazine. 2013. Vol. 51, № 7. P. 36–43.

24 Gude N., Koponen T., Pettit J. NOX: towards an operating system for networks // ACM SIGCOMM Computer Communication Review. 2008. Vol. 38, № 3. P. 105–110.

25 Zhao Y., Iannone L., Riggio R. On the performance of SDN controllers: A reality check // IEEE Conference on Network Function Virtualization and Software Defined Networks. 2015. Vol. 1, № 1. P. 79–85.

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)