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

Навчально-науковий інститут телекомунікаційних систем

Кафедра телекомунікацій

«На правах рукопису»
УДК _____

«До захисту допущено»
Завідувач кафедри
_____ Сергій КРАВЧУК
«___»_____2025 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Інженерія та програмування інфокомунікацій»
зі спеціальності 172 «Електронні комунікації та радіотехніка»

на тему: «Побудова SDN мережі на основі контролера ONOS з елементами ТЕ»

Виконав:
студент ІІ курсу, групи ЦК-41мп
Дрегало Микола Сергійович _____

Керівник:
науковий керівник НН ІТС, професор кафедри ТК НН ІТС,
д.т.н., професор, академік НАН України
Ільченко М.Ю. _____

Консультант з розділу 5
Професор кафедри ТК НН ІТС д.т.н. професор
Романов О.І. _____

Рецензент:
Доцент кафедри ЕКІР НН ІТС, к.т.н., доцент
Бердников О.М. _____

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

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут телекомунікаційних систем
Кафедра телекомунікацій

Рівень вищої освіти – другий (магістерський)

Спеціальність – 172 «Телекомунікації та радіотехніка»

Освітньо-професійна програма «Інженерія та програмування інфокомунікацій»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій КРАВЧУК
«___»_____2024 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Дрегало Миколі Сергійовичу

1. Тема дисертації «Побудова SDN мережі на основі контролера ONOS з елементами ТЕ», науковий керівник дисертації Ільченко Михайло Юхимович, д.т.н., професор, затверджені наказом по університету від «03» листопада 2025 р. № 4772-с.

2. Термін подання студентом дисертації: 15.12.2025 р.

3. Об’єкт дослідження: процеси маршрутизації та управління трафіком у програмно-конфігурованих мережах (SDN).

4. Предмет дослідження: методи, алгоритми та інструменти реалізації інженерії трафіку (Traffic Engineering) у середовищі SDN-контролера ONOS для оптимізації параметрів мережі.

5. Перелік завдань, які потрібно розробити:

1) Розглянути концепцію SDN, специфіку контролера ONOS та архітектуру основних компонентів мережі на його базі.
2) Проаналізувати архітектуру модулів ONOS, принципи кластеризації, розподілену систему управління станом (Atomix) та сервіси управління ресурсами.
3) Дослідити протоколи взаємодії в системі: внутрішньокластерну координацію (Raft, Gossip) та протоколи управління пристроями (OpenFlow, OVSDB,

gNMI).
4) Провести аналіз методів Traffic Engineering.
5) Побудувати та налаштувати віртуальний макет сегменту SDN-мережі на базі кластера ONOS та середовища Mininet.
6) Провести тестування розробленої мережі.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

1) Тема, актуальність, мета та задачі магістерської дисертації;
2) Приклад топології SDN-мережі з централізованим управлінням та багаторівнева архітектура контролера ONOS.
3) Схема віртуального макету для моделювання елементів ONOS мережі та процес обробки запитів REST API.
4) Схема кастомної топології мережі та візуалізація маршрутизації потоків по незалежних шляхах.
5) Графіки та скріншоти результатів тестування.

7. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
5 Розділ	Романов О.І., професор, д.т.н., професор		

8. Дата видачі завдання “02” вересня 2024 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Огляд літератури щодо архітектури SDN та специфіки контролера ONOS. Аналіз концепції Software Defined Networking.	02.09.2024 – 18.10.2024	Виконано
2	Дослідження архітектури ONOS: мікросервіси, кластеризація (Atomix, Raft) та сервіси управління ресурсами.	21.10.2024 – 27.12.2024	Виконано
3	Аналіз протоколів взаємодії (OpenFlow, gNMI, REST API) та методів Traffic Engineering (алгоритми Дейкстри, Єна).	13.01.2025 – 28.02.2025	Виконано
4	Проектування та побудова віртуального макету мережі (ONOS + Mininet). Розробка скриптів топології.	03.03.2025 – 18.04.2025	Виконано
5	Налаштування кластера контролерів, імплементація сценаріїв маршрутизації та проведення експериментів з iperf.	21.04.2025 – 20.06.2025	Виконано

* Якщо визначені консультанти. Консультантом не може бути зазначено наукового керівника магістерської дисертації.

6	Аналіз отриманих результатів, оптимізація мережі за допомогою інтентів. Розробка стартап-проекту.	01.09.2025 – 24.10.2025	Виконано
7	Оформлення пояснювальної записки, підготовка графічного матеріалу та доповіді.	27.10.2025 – 12.12.2025	Виконано

Студент Микола Дрегало

Науковий керівник дисертації Михайло ІЛЬЧЕНКО

РЕФЕРАТ

Пояснювальна записка до магістерської дисертації містить 123 сторінки, 58 рисунків, 8 таблиць, 13 формул, 1 додаток, список використаних джерел (13 джерел).

Актуальність теми. Сучасний етап розвитку телекомунікаційних мереж характеризується стрімким зростанням обсягів трафіку, що вимагає нових підходів до управління ресурсами. Традиційні мережі стають складними для масштабування, тому перехід до програмно-конфігурованих мереж (SDN) та використання контролерів класу Carrier Grade, таких як ONOS, є критично важливим для забезпечення якості обслуговування (QoS) та відмовостійкості.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконується згідно з планом науково-дослідних робіт кафедри телекомунікацій Навчально-наукового інституту телекомунікаційних систем КПІ ім. Ігоря Сікорського.

Мета й завдання дослідження. Метою роботи є підвищення ефективності та надійності функціонування SDN-мереж шляхом розробки та імплементації методів інженерії трафіку (Traffic Engineering) з використанням інструментарію контролера ONOS.

Для цього вирішувалися завдання: аналіз архітектури ONOS та протоколів взаємодії (OpenFlow, gNMI, Raft); дослідження алгоритмів маршрутизації (Дейкстри, Єна); побудова віртуального макету в Onos tutorial; експериментальна перевірка ефективності використання інтентів для уникнення перевантажень; розробка стартап-проекту.

Об'єкт дослідження. Об’єктом дослідження є процеси маршрутизації та управління потоками даних у програмно-конфігурованих мережах.

Предмет дослідження. Предметом дослідження є методи та алгоритми Traffic Engineering, реалізовані в середовищі контролера ONOS.

Методи дослідження. У роботі використано комплекс методів: системний аналіз (для дослідження архітектури ONOS), теорія графів (для аналізу алгоритмів маршрутизації) та методи імітаційного моделювання (для проведення

експериментів з віртуальною топологією мережі).

Наукова новизна одержаних результатів. Новизна полягає у вдосконаленні підходів до управління трафіком в SDN-мережах. Експериментально доведено переваги використання Intent Framework у поєднанні з алгоритмами пошуку k-найкоротших шляхів над стандартними реактивними методами, що дозволяє ефективно уникати колізій та втрат пакетів на спільних каналах.

Практичне значення одержаних результатів. Результатом роботи є розроблений та налаштований віртуальний макет SDN-мережі на базі кластера ONOS та середовища Mininet, а також комплект Python-скриптів для автоматизації розгортання топологій.

Отриманий технологічний результат трансформовано в бізнес-модель стартап-проекту, що підтверджує економічну доцільність впровадження розроблених рішень.

Апробація результатів дисертації. На момент завершення роботи над дисертацією її результати ще не були апробовані на наукових конференціях або в публікаціях.

Публікації. За темою магістерської дисертації на момент її захисту публікації відсутні.

Ключові слова: SDN, ONOS, Traffic Engineering, QoS, маршрутизація, Mininet, OpenFlow, Intent Framework, gNMI.

ABSTRACT

The explanatory note for the master's thesis contains 123 pages, 58 figures, 8 tables, 13 formulas, 1 appendix, and a list of references (13 sources).

Relevance of the topic. The current stage of telecommunications network development is characterized by rapid traffic volume growth, which requires new approaches to resource management. Traditional networks are becoming difficult to scale, making the transition to Software-Defined Networks (SDN) and the use of Carrier Grade controllers such as ONOS critically important for ensuring Quality of Service (QoS) and fault tolerance.

Connection with scientific programs, plans, and themes. The work is carried out in accordance with the research plan of the Department of Telecommunications at the Educational and Scientific Institute of Telecommunication Systems of Igor Sikorsky Kyiv Polytechnic Institute.

Research goal and objectives. The goal of this work is to improve the efficiency and reliability of SDN network operation through the development and implementation of Traffic Engineering methods using ONOS controller tools.

To achieve this, the following tasks were addressed: analysis of ONOS architecture and interaction protocols (OpenFlow, gNMI, Raft); investigation of routing algorithms (Dijkstra, Yen); building a virtual prototype in ONOS tutorial; experimental verification of the effectiveness of using intents to avoid congestion; development of a startup project.

Object of research. The object of research is the routing processes and data flow management in software-defined networks.

Subject of research. The subject of research is Traffic Engineering methods and algorithms implemented in the ONOS controller environment.

Research methods. The work employs a set of methods: systems analysis (for studying ONOS architecture), graph theory (for analyzing routing algorithms), and simulation modeling methods (for conducting experiments with virtual network topology).

Scientific novelty of the obtained results. The novelty lies in the improvement of traffic management approaches in SDN networks. The advantages of using the Intent

Framework combined with k-shortest path search algorithms over standard reactive methods have been experimentally proven, enabling effective avoidance of collisions and packet losses on shared channels.

Practical significance of the obtained results. The result of the work is a developed and configured virtual prototype of an SDN network based on an ONOS cluster and Mininet environment, as well as a set of Python scripts for automating topology deployment.

The obtained technological result has been transformed into a business model for a startup project, confirming the economic feasibility of implementing the developed solutions.

Approbation of dissertation results. At the time of completing work on the dissertation, its results have not yet been presented at scientific conferences or in publications.

Publications. At the time of thesis defense, there are no publications on the topic of the master's dissertation.

Keywords: SDN, ONOS, Traffic Engineering, QoS, routing, Mininet, OpenFlow, Intent Framework, gNMI.

ПЕРЕЛІК СКОРОЧЕНЬ

API	Application Programming Interface (інтерфейс прикладного програмування)
ARP	Address Resolution Protocol (протокол розв'язання адрес)
CLI	Command Line Interface (інтерфейс командного рядка)
DHCP	Dynamic Host Configuration Protocol (протокол динамічної конфігурації вузла)
ECMP	qual-Cost Multi-Path (багатошляхова маршрутизація рівної вартості)
gNMI	gRPC Network Management Interface (інтерфейс управління мережею на базі gRPC)
gRPC	gRPC Remote Procedure Calls (віддалений виклик процедур gRPC)
GUI	Graphical User Interface (графічний інтерфейс користувача)
HTTP	HyperText Transfer Protocol (протокол передавання гіпертексту)
IP	Internet Protocol (міжмережевий протокол)
JSON	JavaScript Object Notation (нотація об'єктів JavaScript)
MAC	Media Access Control (управління доступом до середовища)
ONOS	Open Network Operating System (відкрита мережева операційна система)
OSGi	Open Services Gateway Initiative (ініціатива шлюзів відкритих сервісів)
OVSDB	Open vSwitch Database Management Protocol

	(протокол управління базою даних Open vSwitch)
QoS	Quality of Service (якість обслуговування) **RAM** – Random Access Memory (оперативна пам'ять)
REST	Representational State Transfer (передача стану представлення)
SDN	Software Defined Networking (програмно-конфігуровані мережі)
SNMP	Simple Network Management Protocol (простий протокол управління мережею)
TCP	Transmission Control Protocol (протокол управління передаванням)
TE	Traffic Engineering (інженерія трафіку)
UDP	User Datagram Protocol (протокол користувацьких датаграм)
VLAN	Virtual Local Area Network (віртуальна локальна мережа)

ЗМІСТ

ВСТУП 13

РОЗДІЛ 1 16

Основні компоненти SDN-мережі на базі ONOS 16

1.1 Концепція Software Defined Networking (SDN) 16

1.2 Специфіка ONOS як SDN-контролера 17

1.3 Архітектура та основні компоненти SDN-мережі на базі ONOS 17

Висновки 21

Розділ 2 22

Архітектура та призначення основних модулів ONOS 22

2.1 Загальна концепція та багатошарова архітектура 22

2.2 Розподілена система управління станом та кластеризація 27

2.3 Сервіси управління мережевими ресурсами 30

2.4 Високорівневі сервіси та інтерфейси управління 35

Висновки 40

Розділ 3 41

Аналіз протоколів взаємодії в ONOS 41

3.1 Загальні принципи протокольної взаємодії 41

3.2 Протоколи внутрішньокластерної координації 41

3.3 Southbound протоколи управління мережевими пристроями 51

3.4 Northbound інтерфейс REST API 57

Висновки 60

Розділ 4 62

Методи Traffic Engineering 62

4.1 Концепція Traffic Engineering в SDN-мережах 62

4.2 Алгоритми обчислення шляхів 62
4.3 Механізми управління трафіком 68
4.4 Реактивне переадресування в ONOS 71
Висновки 75
Розділ 5 76
Побудова сегменту SDN-мережі на базі контролеру ONOS 76
5.1 Логічна схема та основні складові віртуального макету ONOS мережі 76
5.2 Порядок налаштування та розгортання ONOS Controller 78
5.3 Тестування SDN мережі на основі ONOS кластера 83
5.4 Налаштування кастомної топології та керування маршрутизацією трафіку. 89
Висновки 110
Розділ 6 112
РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ 112
6.1 Технологічний аудит ідеї проекту 113
Висновки 117
ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ 118
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 120
ДОДАТКИ 122
Додаток А 122

ВСТУП

Актуальність теми. Сучасний етап розвитку телекомунікаційних мереж характеризується стрімким зростанням обсягів трафіку та підвищенням вимог до якості обслуговування (QoS). Традиційні підходи до побудови мереж, де управління жорстко пов'язане з апаратним забезпеченням, стають перешкодою для масштабування та гнучкого налаштування сервісів.

Відповіддю на ці виклики стала концепція програмно-конфігурованих мереж (Software Defined Networking, SDN), яка через відокремлення площини управління (Control Plane) від площини передачі даних (Data Plane) відкрила нові можливості для автоматизації та інтелектуального управління трафіком. Серед існуючих рішень особливу увагу привертає контролер ONOS (Open Network Operating System). Завдяки своїй мікросервісній архітектурі, підтримці кластеризації та високої доступності, він позиціонується як платформа класу Carrier Grade, здатна задовольнити потреби великих сервіс-провайдерів.

У цьому контексті розробка та дослідження методів Traffic Engineering (TE) на базі ONOS є критично важливим завданням, вирішення якого дозволить оптимізувати використання мережевих ресурсів, забезпечити відмовостійкість та гарантувати необхідні параметри якості для критичних додатків.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконується згідно з планом науково-дослідних робіт кафедри телекомунікацій Навчально-наукового інституту телекомунікаційних систем КПІ ім. Ігоря Сікорського.

Метою роботи є підвищення ефективності та надійності функціонування SDN-мереж шляхом розробки та імплементації методів інженерії трафіку (Traffic Engineering) з використанням інструментарію контролера ONOS.

Для досягнення поставленої мети в роботі вирішуються такі **завдання**:

1. Провести аналіз архітектурних особливостей та компонентів SDN-мереж на базі ONOS, визначити переваги його кластерної структури.
2. Дослідити протоколи взаємодії (OpenFlow, OVSDB, gNMI) та механізми синхронізації стану в розподілених системах (Raft, Gossip) для забезпечення

узгодженості даних.

3. Проаналізувати алгоритми обчислення шляхів (модифікований алгоритм Дейкстри, алгоритм Єна) та механізми управління трафіком (Intent Framework).
4. Побудувати віртуальний макет сегмента SDN-мережі в середовищі Mininet та експериментально дослідити ефективність проактивних та реактивних методів маршрутизації.
5. Розробити стартап-проект для комерціалізації створеного рішення з інтелектуального управління трафіком.

Об'єктом дослідження є процеси маршрутизації та управління потоками даних у програмно-конфігурованих мережах.

Предметом дослідження є методи та алгоритми Traffic Engineering, реалізовані в середовищі контролера ONOS.

Методи дослідження. У роботі використано методи системного аналізу для дослідження архітектури ONOS, теорії графів для аналізу алгоритмів маршрутизації, а також методи імітаційного моделювання для проведення експериментів з віртуальною топологією мережі.

Наукова новизна одержаних результатів полягає у подальшому розвитку методів управління трафіком в SDN-мережах. Зокрема, експериментально доведено переваги використання Intent Framework у поєднанні з алгоритмами пошуку k-найкоротших шляхів (алгоритм Єна) над стандартними реактивними методами маршрутизації. Встановлено, що запропонований підхід дозволяє уникнути колізій та втрат пакетів при агрегації потоків на спільних каналах шляхом їх примусового розділення по незалежних фізичних маршрутах.

Практичне значення одержаних результатів. Розроблено та налаштовано повнофункціональний віртуальний макет SDN-мережі на базі кластера контролерів ONOS та середовища Mininet. Створено комплект скриптів на мові Python для автоматизації розгортання кастомних топологій, які можуть бути використані як база для проектування відмовостійких корпоративних мереж та мереж провайдерів. Розроблена концепція стартап-проекту підтверджує економічну доцільність

впровадження запропонованих рішень.

Апробація результатів дисертації. Результати, отримані в ході даного дослідження, на момент завершення роботи над дисертацією ще не були апробовані на наукових конференціях або в публікаціях.

Публікації. За темою магістерської дисертації публікації відсутні.

РОЗДІЛ 1

ОСНОВНІ КОМПОНЕНТИ SDN-МЕРЕЖІ НА БАЗІ ONOS

1.1 Концепція Software Defined Networking (SDN)

Розуміння фундаментальних принципів Software Defined Networking є необхідною основою для дослідження платформи ONOS. Цей підрозділ розглядає сучасний підхід SDN до архітектури мереж, який суттєво змінює традиційні принципи побудови інфраструктури через розділення площини управління та площини даних. Ключові принципи SDN - логічна централізація управління, глобальний погляд на топологію, програмованість та відкритість інтерфейсів - створюють основу для ефективних, керованих та адаптивних мережевих систем, особливо в умовах сучасних вимог до динамічності та автоматизації.

Software Defined Networking (SDN) являє собою сучасний підхід до архітектури мереж, який суттєво змінює традиційні принципи побудови та управління мережевою інфраструктурою. На відміну від класичних мереж, де кожен пристрій автономно приймає рішення щодо пересилання трафіку на основі локальної інформації, SDN розділяє мережу на дві основні площини: площину управління (Control Plane) та площину даних (Data Plane. Цей поділ забезпечує централізоване управління всією мережевою інфраструктурою через програмний контролер.

Фундаментальна ідея SDN полягає в абстракції мережевих ресурсів та створенні програмованого інтерфейсу для управління мереже. Це дозволяє розробникам та адміністраторам динамічно налаштовувати поведінку мережі через програмне забезпечення, а не через ручне конфігурування кожного окремого пристрою. SDN забезпечує гнучкість, масштабованість та можливість швидкого впровадження нових мережевих сервісів та політик.

Ключовими принципами SDN є: логічна централізація управління мережею, глобальний погляд на мережеву топологію, програмованість мережевої поведінки та відкритість інтерфейсів. Ці принципи дозволяють створювати більш ефективні, керовані та адаптивні мережеві інфраструктури, особливо в умовах сучасних вимог до динамічності та автоматизації [8].

1.2 Специфіка ONOS як SDN-контролера

ONOS як SDN-контролер має унікальні характеристики, які відрізняють його від інших рішень на ринку. Цей підрозділ розглядає ключові особливості платформи, зокрема кластерну архітектуру для забезпечення високої доступності, модульну побудову на основі мікросервісів та орієнтацію на потреби сервіс-провайдерів. Ці характеристики роблять ONOS оптимальним вибором для критично важливих промислових мереж, де надійність та масштабованість є пріоритетними вимогами, а відмова централізованого контролера може призвести до серйозних наслідків.

Open Network Operating System (ONOS) представляє собою операційну систему для мереж з відкритим вихідним кодом, розроблену спеціально для потреб сервіс-провайдерів та великих корпоративних мереж. На відміну від інших SDN-контролерів, ONOS орієнтований на забезпечення високої продуктивності, надійності та масштабованості, що робить його ефективним вибором для критично важливих мережевих інфраструктур [5].

ONOS відзначається своєю кластерною архітектурою, яка забезпечує високу доступність та горизонтальне масштабування. Система може функціонувати в режимі розподіленого кластера, де кілька екземплярів контролера працюють синхронно, забезпечуючи відмовостійкість та рівномірний розподіл навантаження. Це особливо важливо для промислових мереж, де відмова централізованого контролера може призвести до серйозних наслідків.

Модульна архітектура ONOS дозволяє гнучко налаштовувати функціональність системи відповідно до специфічних потреб мережі. Контролер побудований на основі мікросервісної архітектури, де кожен модуль відповідає за конкретну функцію та може бути незалежно розроблений, протестований та розгорнутий. Це значно спрощує процес розробки нових функцій та забезпечує високу модульність системи.

1.3 Архітектура та основні компоненти SDN-мережі на базі ONOS

Повноцінна SDN-мережа на базі ONOS складається з багаторівневої структури взаємопов'язаних компонентів, кожен з яких виконує специфічні функції

в загальній архітектурі. Чотири функціональні рівні — управління, додатків, комунікаційний та інфраструктурний — забезпечують логічну організацію системи, тоді як конкретні компоненти (ONOS контролер, площина даних, протоколи взаємодії, API та модулі Traffic Engineering) реалізують практичну функціональність. Розуміння призначення та взаємодії цих елементів є критичним для ефективного проєктування та експлуатації SDN-інфраструктури [9].

Багаторівнева структура SDN-мережі. SDN-мережа на базі ONOS складається з чотирьох функціональних рівнів, що забезпечують повноцінне функціонування програмно-визначеної інфраструктури:

1. Рівень управління включає ONOS контролер (одиночний або кластерний), систему управління конфігурацією та інтерфейси адміністрування (CLI, GUI, REST API).
2. Рівень додатків містить мережеві додатки (Network Applications), модулі Traffic Engineering, системи моніторингу та аналітики і інтент-фреймворк. Комунікаційний рівень представлений Southbound API (OpenFlow, NETCONF, P4Runtime), Northbound API (REST, GraphQL) та механізмами міжкластерної комунікації.
3. Рівень інфраструктури складається з OpenFlow-сумісних комутаторів, програмованих маршрутизаторів та мережевих функцій віртуалізації (VNF).

Взаємодію цих рівнів та приклад загальної топології SDN-мережі зображено на рисунку 1.1.

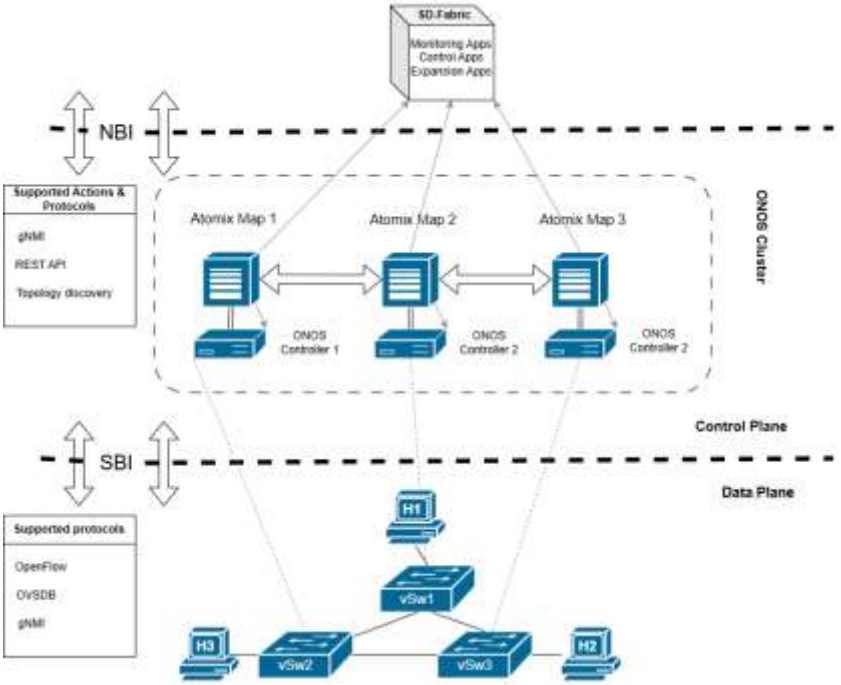

Рис. 1.1. Приклад топології SDN-мережі з централізованим управлінням.

ONOS контролер становить центральний елемент SDN-архітектури та виконує роль операційної системи для мережі. Цей компонент відповідає за централізоване прийняття рішень щодо маршрутизації, політик безпеки, управління ресурсами та координацію всіх мережевих операцій. ONOS контролер підтримує глобальний погляд на мережеву топологію, що дозволяє приймати оптимальні рішення на основі повної інформації про стан мережі. Ключовою особливістю ONOS контролера є його здатність працювати в кластерному режимі, забезпечуючи високу доступність та масштабованість. Система використовує розподілені алгоритми для синхронізації стану між вузлами кластера та автоматичного перерозподілу навантаження в разі відмови окремих компонентів. Контролер забезпечує консистентність даних через використання розподілених структур даних та протоколів консенсусу. Архітектура ONOS базується на мікросервісах, де кожна функція реалізована як окремий сервіс з чітко визначеними інтерфейсами. Це забезпечує модульність системи та можливість незалежної розробки, тестування та розгортання окремих компонентів. Контролер підтримує динамічне завантаження та вивантаження модулів без переривання роботи системи.

Площина даних (Data Plane). Площина даних представлена мережевими пристроями, такими як OpenFlow-сумісні комутатори, маршрутизатори та точки

доступу. Ці пристрої виконують безпосередню обробку та пересилання мережевого трафіку згідно з правилами, отриманими від контролера. На відміну від традиційних мережевих пристроїв, обладнання в SDN-мережі не приймає самостійних рішень щодо маршрутизації, а діє як виконавчий механізм для інструкцій контролера. Кожен пристрій площини даних підтримує таблиці потоків (flow tables), де зберігаються правила обробки пакетів, отримані від ONOS контролера.

Протокол OpenFlow. Протокол OpenFlow забезпечує комунікацію між контролером ONOS та пристроями площини даних. Цей протокол дозволяє контролеру програмно налаштовувати поведінку мережевих пристроїв, встановлювати правила обробки трафіку та отримувати статистичну інформацію про стан мережі. OpenFlow підтримує різні версії протоколу, при цьому ONOS забезпечує сумісність з найпоширенішими версіями, що гарантує широку підтримку мережевого обладнання від різних виробників.

Northbound API. Northbound API являє собою програмний інтерфейс, через який зовнішні додатки та сервіси взаємодіють з ONOS контролером. Цей інтерфейс надає можливість розробникам створювати мережеві додатки, які можуть керувати поведінкою мережі, впроваджувати нові сервіси та інтегруватися з існуючими системами управління. ONOS підтримує як REST API для HTTP-взаємодії, так і більш спеціалізовані інтерфейси для високопродуктивних додатків.

Елементи Traffic Engineering. Елементи Traffic Engineering в контексті ONOS включають компоненти для оптимізації використання мережевих ресурсів та забезпечення якості обслуговування. Ці елементи дозволяють динамічно керувати потоками трафіку, балансувати навантаження між різними шляхами в мережі та забезпечувати резервування пропускної здатності для критично важливих сервісів. ONOS інтегрує TE функціональність через спеціалізовані додатки та модулі, які аналізують поточний стан мережі та оптимізують маршрути відповідно до встановлених політик та вимог до якості обслуговування.

Система моніторингу та телеметрії. Система моніторингу та телеметрії забезпечує збір, аналіз та візуалізацію даних про роботу мережі. ONOS включає

потужні механізми для відстеження продуктивності мережі, виявлення аномалій та генерації звітів. Ця система дозволяє адміністраторам отримувати детальну інформацію про використання ресурсів, затримки, втрати пакетів та інші ключові метрики, які є критично важливими для ефективного управління SDN-мережею та впровадження елементів Traffic Engineering.

Висновки

Результатом проведеного аналізу теоретичних засад було досліджено концепцію Software Defined Networking (SDN) як сучасний підхід, що суттєво змінює принципи побудови мереж. Виявлено, що фундаментальний поділ на площину управління (Control Plane) та площину даних (Data Plane) дозволяє вирішити проблеми масштабованості та складності адміністрування, забезпечуючи централізоване та програмне керування інфраструктурою.

До сильних сторін обраного контролера ONOS можна віднести його чітку орієнтацію на потреби сервіс-провайдерів та підтримку кластерної архітектури. Це забезпечує системі високу доступність, відмовостійкість та можливість горизонтального масштабування, що вигідно відрізняє ONOS від інших рішень і є критично важливим фактором для промислових мереж, де відмова контролера неприпустима.

Варто відмітити актуальність мікросервісної архітектури ONOS, яка дозволяє гнучко налаштовувати функціональність та інтегрувати елементи Traffic Engineering. Наявність розвинених інтерфейсів (Northbound та Southbound API) разом із вбудованими системами моніторингу та інтент-фреймворком створює необхідне технічне підґрунтя для автоматизації управління трафіком та впровадження нових сервісів без переривання роботи мережі.

Отже, можна зробити висновок про доцільність використання платформи ONOS як основи для подальшої розробки системи, оскільки вона надає повний набір інструментів для реалізації централізованого контролю, динамічної оптимізації маршрутів та забезпечення якості обслуговування (QoS).

РОЗДІЛ 2

АРХІТЕКТУРА ТА ПРИЗНАЧЕННЯ ОСНОВНИХ МОДУЛІВ ONOS

2.1 Загальна концепція та багатошарова архітектура

Архітектура Open Network Operating System побудована на фундаментальних принципах модульності, масштабованості та високої доступності, які реалізуються через багатошарову організацію системи з чіткими інтерфейсами взаємодії між компонентами. Центральною ідеєю архітектури є концепція "Network Operating System", яка передбачає створення уніфікованого програмного інтерфейсу для взаємодії з мережевими ресурсами аналогічно до того, як традиційні операційні системи надають абстракцію над апаратними ресурсами комп'ютера. Реалізація цієї концепції базується на мікросервісній архітектурі з використанням фреймворку OSGi, що забезпечує динамічне управління життєвим циклом модулів та їх ізоляцію, багатошаровій структурі, де кожен шар відповідає за специфічні аспекти управління мережею від низькорівневої взаємодії з пристроями до високорівневих мережевих застосунків, та чітко визначених північних і південних інтерфейсах для взаємодії відповідно із зовнішніми системами управління та гетерогенним мережевим обладнанням. Така архітектурна організація дозволяє створювати складні мережеві рішення через композицію простих сервісів, забезпечує незалежність від конкретних виробників обладнання та протоколів управління, а також підтримує еволюцію системи через додавання нових компонентів без модифікації існуючої функціональності [1], [9].

Загальна концепція архітектури. Архітектура Open Network Operating System (ONOS) побудована на принципах модульності, масштабованості та високої доступності, що робить її однією з найбільш прогресивних платформ для управління програмно-визначеними мережами. ONOS використовує багатошарову архітектуру, яка логічно розділяє різні аспекти управління мережею та забезпечує чіткі інтерфейси між компонентами системи. Основною філософією архітектури є створення

операційної системи для мереж, яка надає абстракцію над складними мережевими протоколами та пристроями, дозволяючи розробникам створювати мережеві застосунки так само просто, як звичайні програми для операційних систем.

Центральною ідеєю архітектури ONOS є концепція "Network Operating System", яка передбачає надання уніфікованого програмного інтерфейсу для взаємодії з мережевими ресурсами. Це досягається через створення абстракційних рівнів, які приховують складність базової мережевої інфраструктури та надають простий і потужний API для розробки мережевих застосунків. Архітектура забезпечує повну ізоляцію між застосунками, що дозволяє незалежну розробку та розгортання різних мережевих сервісів без взаємного впливу.

Ключовою особливістю архітектури ONOS є її орієнтація на сервіс-провайдерів та корпоративні мережі, що відображається в архітектурних рішеннях, спрямованих на забезпечення високої продуктивності, надійності та масштабованості. Система розроблена з урахуванням вимог до обробки великих обсягів трафіку, підтримки тисяч мережевих пристроїв та забезпечення мінімальних затримок у прийнятті рішень щодо маршрутизації.

Мікросервісна архітектура та OSGi Framework. ONOS побудована на основі мікросервісної архітектури, де кожен функціональний компонент реалізований як незалежний сервіс з чітко визначеними інтерфейсами взаємодії. Ця архітектурна парадигма забезпечує винятковий рівень модульності, дозволяючи розробникам створювати, тестувати та розгортати окремі компоненти незалежно від решти системи. Кожен мікросервіс інкапсулює специфічну функціональність, таку як управління топологією, обробка правил потоків, моніторинг мережі або реалізація конкретних протоколів.

Модульна архітектура ONOS базується на фреймворку OSGi (Open Services Gateway Initiative), який забезпечує динамічне управління життєвим циклом модулів. OSGi дозволяє завантажувати, оновлювати та

вивантажувати пакети (модулі) під час роботи системи без необхідності перезапуску контролера. Це критично важливо для промислових мереж, де простої можуть призвести до значних фінансових втрат. Кожен пакет має власний завантажувач класів, що забезпечує ізоляцію та запобігає конфліктам між різними версіями бібліотек.

Архітектура мікросервісів у ONOS реалізована через систему внутрішніх API та інтерфейсів, які забезпечують стандартизовану взаємодію між компонентами. Основними типами інтерфейсів є Service API для надання функціональності іншим компонентам, Administrative API для управління життєвим циклом сервісу та Event API для асинхронного обміну повідомленнями. Така структура дозволяє створювати складні мережеві застосунки шляхом композиції простих сервісів [8].

Багатошарова архітектура системи. Архітектура ONOS організована у вигляді багатошарової структури, де кожен шар відповідає за специфічні аспекти управління мережею та надає абстракцію для вищих шарів. Найнижчий шар - це Protocol Abstraction Layer, який відповідає за безпосередню взаємодію з мережевими пристроями через різні південні протоколи. Цей шар включає реалізації OpenFlow, NETCONF, P4Runtime, gNMI, SNMP та інших протоколів, що дозволяє ONOS взаємодіяти з широким спектром мережевого обладнання від різних виробників. Взаємозв'язок та ієрархію цих рівнів у загальній структурі системи наведено на рисунку 2.1.

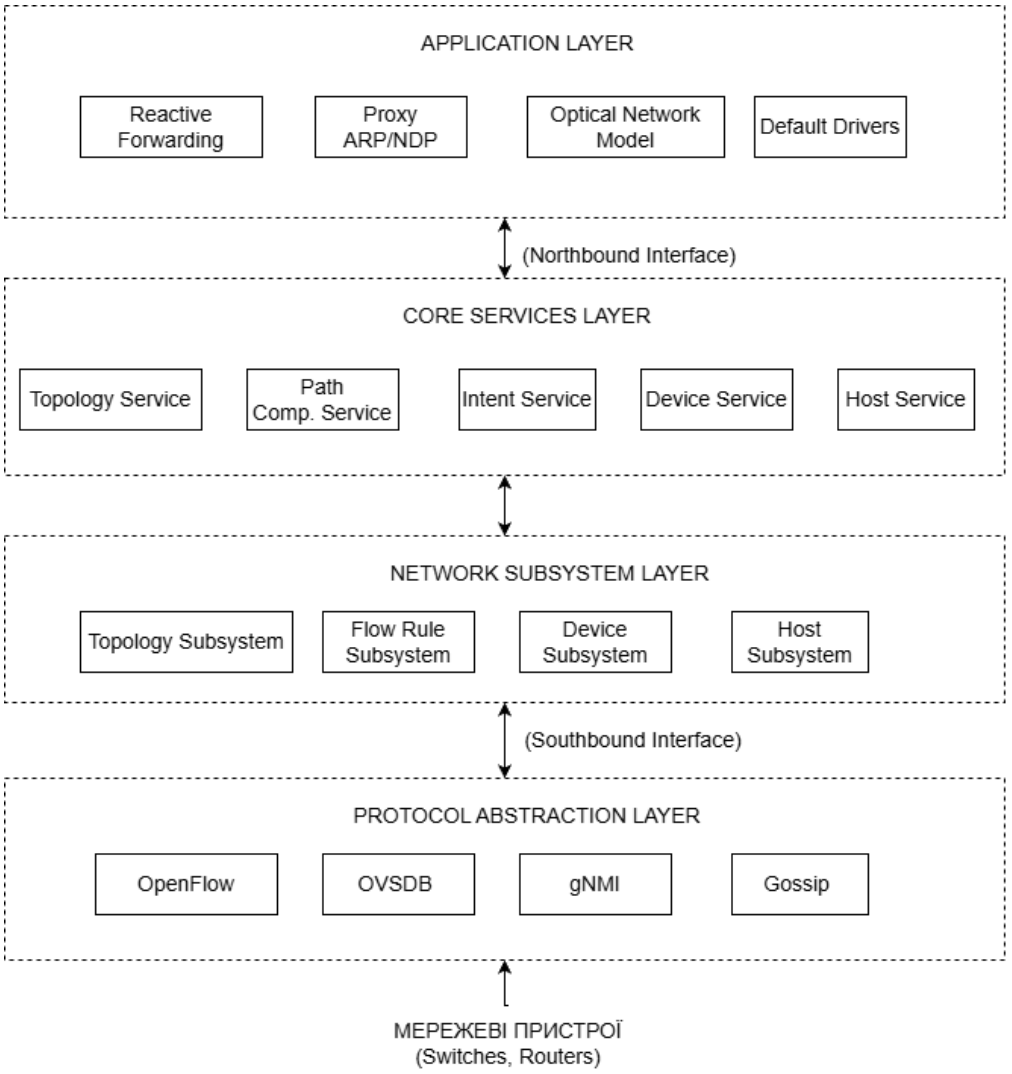

Рис. 2.1. Багатошарова архітектура контролера ONOS

Network Subsystem Layer створює уніфіковану абстракцію над різноманітними мережевими пристроями та протоколами. Цей шар містить основні підсистеми для представлення топології мережі, конфігурації пристроїв, політик безпеки та інших мережевих ресурсів. Основними компонентами цього шару є Device Subsystem для представлення мережевих пристроїв, Topology Subsystem для опису зв'язків між пристроями, Flow Rule Subsystem для представлення правил обробки трафіку та Host Subsystem для відстеження кінцевих пристроїв.

Core Services Layer надає фундаментальні сервіси управління мережею, які використовуються всіма мережевими застосунками. До цих сервісів належать Topology Service для управління топологією, Path Computation

Service для обчислення маршрутів, Intent Service для високорівневого опису мережевих політик, Device Service для управління пристроями та Host Service для відстеження кінцевих вузлів. Ці сервіси забезпечують базову функціональність, необхідну для ефективного управління програмно-визначеною мережею.

Application Layer містить мережеві застосунки, що реалізують конкретну бізнес-логіку та функціональність. Цей шар включає вбудовані застосунки, такі як Reactive Forwarding для базової маршрутизації, Proxy ARP для обробки ARP запитів, DHCP Relay для пересилання DHCP трафіку, та спеціалізовані застосунки для інженерії трафіку, балансування навантаження, забезпечення безпеки тощо [8].

Північні та південні інтерфейси. Northbound Interface Layer забезпечує взаємодію зовнішніх застосунків та систем управління з контролером ONOS через стандартизовані API. Архітектура включає RESTful API для взаємодії на базі HTTP, WebSocket API для зв'язку у реальному часі, GraphQL endpoint для гнучких запитів даних, та gRPC interface для високопродуктивних застосунків. Кожен інтерфейс надає повний доступ до функціональності ONOS з урахуванням специфічних вимог різних типів клієнтських застосунків.

Southbound Interface Layer відповідає за взаємодію з мережевими пристроями через різні протоколи управління. Архітектура підтримує OpenFlow для програмно-визначених комутаторів, NETCONF/YANG для управління конфігурацією, P4Runtime для програмованих площин даних, gNMI для моніторингу мережі, SNMP для підтримки застарілих пристроїв, та власні REST API для інтеграцій, специфічних для виробників. Кожен південний протокол реалізований як окремий постачальник, що забезпечує модульність та можливість легкого додавання підтримки нових протоколів.

Система включає також абстракції, незалежні від протоколів, які дозволяють верхнім шарам працювати з універсальними мережевими примітивами незалежно від специфічних південних протоколів. Це

забезпечує незалежність від виробників та спрощує розробку мережевих застосунків, які можуть працювати з гетерогенною мережевою інфраструктурою без модифікацій для кожного типу обладнання [9].

2.2 Розподілена система управління станом та кластеризація

Забезпечення високої доступності та масштабованості в ONOS досягається через архітектуру розподіленої системи, де множина екземплярів контролера працюють як єдиний логічний кластер з узгодженим представленням стану мережі. Критично важливим компонентом цієї архітектури є платформа Atomix, яка надає фундаментальні примітиви для побудови розподілених систем на основі протоколу консенсусу Raft, гарантуючи строгу консистентність операцій модифікації стану та автоматичне відновлення після відмов окремих вузлів. Архітектура кластеризації ONOS реалізує активно-активну конфігурацію, де всі вузли одночасно беруть участь в обробці запитів та управлінні мережею через розподіл відповідальності за різні мережеві пристрої між вузлами з автоматичним балансуванням навантаження та механізмами швидкого перемикання при відмовах. Основні сервіси координації, включаючи Leadership Service для визначення лідерських ролей, Cluster Service для управління членством у кластері, та Storage Service для надання розподілених структур даних, забезпечують необхідну інфраструктуру для узгодженої роботи розподіленої системи. Така організація дозволяє системі масштабуватися горизонтально через додавання нових вузлів до кластера, витримувати відмови окремих компонентів без втрати управління мережею, та забезпечувати передбачувану продуктивність навіть при управлінні великими мережами з тисячами пристроїв.

Atomix - розподілена система управління станом. Критично важливим компонентом архітектури ONOS є Atomix - потужна платформа для побудови розподілених систем, яка забезпечує консистентне зберігання та синхронізацію даних у кластерному середовищі. Atomix являє собою окремий проект, який ONOS використовує як основу для управління

розподіленим станом та координації між вузлами кластера. Ця система надає високорівневі примітиви для розподілених обчислень, такі як розподілені відображення, множини, блокування, лічильники та бар'єри.

Atomix базується на протоколі консенсусу Raft, який гарантує строгу консистентність для всіх операцій модифікації стану. Система автоматично обирає головний вузол, який відповідає за прийняття рішень та координацію реплікації даних на підлеглі вузли. У випадку відмови головного вузла, Atomix автоматично проводить вибори нового лідера, забезпечуючи безперервність роботи системи. Для покращення продуктивності читання Atomix дозволяє підлеглим вузлам обслуговувати запити лише для читання без консультації з головним вузлом.

Архітектура Atomix включає також систему управління членством кластера, яка автоматично відстежує стан вузлів та керує процесами приєднання та відключення вузлів від кластера. Система використовує протокол поширення чуток для обміну інформацією про стан вузлів та детектор відмов для швидкого виявлення недоступних вузлів. Це забезпечує високу доступність та автоматичне відновлення після мережевих розділень або апаратних відмов.

Архітектура кластеризації. Архітектура кластеризації ONOS базується на принципах активно-активної конфігурації, де всі вузли кластера активно беруть участь в обробці запитів та управлінні мережею. Підсистема управління кластером відповідає за координацію між вузлами, управління членством в кластері та забезпечення консистентності розподіленого стану. Підсистема використовує Atomix для координації на основі консенсусу та включає механізми для автоматичного розподілу навантаження між вузлами кластера.

Система реалізує складну стратегію балансування навантаження, яка розподіляє відповідальність за управління різними частинами мережі між вузлами кластера. Кожен мережевий пристрій має первинний головний вузол, який відповідає за безпосереднє управління цим пристроєм, та

декілька резервних головних вузлів, які готові перебрати управління у випадку відмови первинного. Цей підхід забезпечує як високу доступність, так і ефективне використання ресурсів кластера.

Архітектура включає також механізми для автоматичного перемикання при відмовах та відновлення, які мінімізують вплив відмов окремих вузлів на роботу мережі. Система використовує розподілений протокол контролю серцебиття для відстеження стану вузлів та складні алгоритми виявлення відмов для швидкого виявлення проблем. У випадку відмови вузла система автоматично переназначає його відповідальність іншим вузлам та ініціює процедури відновлення для відновлення безперервності сервісу.

Основні сервіси координації. Core Services складають основу архітектури ONOS та забезпечують базову функціональність, необхідну для роботи всіх компонентів системи. Ці сервіси реалізовані як набір взаємопов'язаних модулів, кожен з яких відповідає за специфічний аспект управління мережею. Application Service відповідає за управління життєвим циклом мережевих застосунків, включаючи їх реєстрацію, активацію, деактивацію та взаємодію з іншими компонентами системи. Цей сервіс забезпечує ізоляцію між застосунками та контролює доступ до системних ресурсів відповідно до політик безпеки.

Component Configuration Service надає механізм для динамічного налаштування параметрів компонентів без необхідності перезапуску системи. Цей сервіс дозволяє адміністраторам змінювати конфігурацію як окремих модулів, так і всієї системи через уніфікований інтерфейс. Сервіс підтримує ієрархічні схеми конфігурації, правила перевірки коректності та механізми автоматичного відкату у випадку некоректних змін конфігурації. Leadership Service забезпечує координацію між вузлами кластера для визначення лідерських ролей при управлінні різними аспектами мережі, використовуючи алгоритми розподіленого консенсусу для процесів вибору лідера та перемикання у випадку відмов.

Cluster Service відповідає за управління членством у кластері та

координацію між різними екземплярами ONOS. Цей сервіс відстежує стан вузлів кластера, забезпечує автоматичне виявлення вузлів, керує процесами приєднання та від'єднання від кластера та підтримує консистентні метадані кластера. Storage Service надає абстракцію над різними типами сховищ даних, включаючи кеші в пам'яті, постійні сховища та розподілені структури даних, забезпечуючи уніфікований інтерфейс для всіх компонентів системи для збереження та отримання даних.

2.3 Сервіси управління мережевими ресурсами

Управління мережевими ресурсами в ONOS організоване через систему взаємопов'язаних сервісів, які створюють уніфіковану абстракцію над гетерогенною мережевою інфраструктурою та забезпечують централізований контроль за її станом і поведінкою. Підсистема управління пристроями (Device Subsystem) формує фундаментальний рівень абстракції, надаючи єдине представлення мережевих пристроїв незалежно від їх типу або південного протоколу підключення, автоматично відстежуючи їх доступність, можливості та операційний стан через шаблон постачальника, що дозволяє легко інтегрувати нові типи обладнання. Сервіси мережевої підсистеми, зокрема Link Service та Host Service, доповнюють цю картину інформацією про міжз'єднання між пристроями та розташування кінцевих вузлів у мережі, формуючи повне уявлення про фізичну та логічну структуру мережі. Topology Service інтегрує інформацію з усіх цих джерел для створення комплексного графічного представлення мережі з підтримкою ефективних графових алгоритмів для обчислення шляхів та аналізу досяжності, тоді як підсистема правил потоків (Flow Rule Subsystem) перетворює високорівневі рішення про маршрутизацію в конкретні інструкції для програмування поведінки площини даних через абстракції, незалежні від специфіки реалізації таблиць потоків на різних типах обладнання. Ця багаторівнева система абстракцій забезпечує як ефективне управління ресурсами на низькому рівні, так і зручні інтерфейси для розробки мережевих застосунків на високому рівні.

Підсистема управління пристроями. Device Subsystem є одним з ключових компонентів архітектури ONOS, який забезпечує абстракцію над мережевими пристроями та управління їх життєвим циклом. Ця підсистема включає Device Manager для централізованого управління пристроями, Device Provider для взаємодії з конкретними типами обладнання через різні південні протоколи, та Device Store для зберігання інформації про стан пристроїв. Підсистема підтримує функціональність підключення та використання, автоматично виявляючи нові пристрої в мережі та інтегруючи їх в загальну топологію.

Device Service являє собою центральний компонент для управління мережевими пристроями в екосистемі ONOS. Цей сервіс забезпечує уніфікований погляд на всі пристрої у мережі, незалежно від їх типу або південного протоколу, через який вони підключені до контролера. Device Service включає Device Manager для централізованого управління, Device Registry для зберігання метаданих пристроїв, Device Event Manager для обробки подій, пов'язаних зі змінами стану пристроїв, та Device Provider Registry для управління різними типами постачальників пристроїв. Сервіс автоматично відстежує доступність пристроїв, їх можливості, конфігураційні параметри та операційний стан.

Архітектура Device Subsystem використовує шаблон постачальника, який дозволяє легко додавати підтримку нових типів мережевого обладнання без модифікації основного коду. Кожен Device Provider реалізує специфічну логіку взаємодії з певним типом пристроїв або протоколом, наприклад OpenFlowDeviceProvider для комутаторів OpenFlow, NetconfDeviceProvider для пристроїв, сумісних з NETCONF, або RestDeviceProvider для пристроїв з REST API. Це забезпечує модульність та розширюваність системи.

Device Subsystem також включає механізми для управління конфігурацією пристроїв, моніторингу їх стану та збору телеметричних даних. Система автоматично відстежує зміни в конфігурації пристроїв, синхронізує локальну модель з реальним станом обладнання та генерує події

про важливі зміни стану. Для забезпечення надійності підсистема включає механізми логіки повторних спроб, відновлення з'єднання та поступової деградації у випадку проблем з підключенням до пристроїв.

Сервіси мережевої підсистеми. Link Service відповідає за виявлення, управління та моніторинг міжз'єднань між мережевими пристроями. Цей сервіс автоматично виявляє фізичні та логічні з'єднання у мережі, відстежує їх стан, пропускну здатність, затримку та інші характеристики. Link Service включає фреймворк Link Provider для інтеграції з різними механізмами виявлення з'єднань, Link Store для постійного зберігання інформації про з'єднання та Link Event Manager для генерації подій про зміни у зв'язності. Сервіс також підтримує різні типи з'єднань, включаючи прямі фізичні підключення, тунелі, віртуальні з'єднання та міжрівневі залежності.

Host Service забезпечує відстеження та управління кінцевими пристроями у мережі, включаючи фізичні пристрої, віртуальні машини, контейнери та інші мережеві кінцеві точки. Цей сервіс автоматично виявляє нові вузли через різні механізми, такі як моніторинг ARP/NDP, відстеження DHCP, інспекція пакетів та інтеграція з зовнішніми базами даних вузлів. Host Service підтримує багаті метадані вузлів, включаючи MAC-адреси, IP-адреси, призначення VLAN, інформацію про розташування, статус автентифікації та відстеження мобільності. Сервіс генерує події при появі, переміщенні або зникненні вузлів, що дозволяє мережевим застосункам автоматично адаптуватися до змін у мережевих кінцевих точках.

Сервіс топології. Topology Service є одним з найважливіших компонентів ONOS, який відповідає за створення, підтримку та надання доступу до уніфікованого представлення топології мережі. Цей сервіс агрегує інформацію з різних джерел, включаючи Device Service, Link Service та Host Service, для створення комплексного графічного представлення мережі. Topology Service включає Topology Manager для координації процесу побудови топології, Topology Store для ефективного зберігання та отримання топологічних даних, фреймворк Topology Provider для інтеграції з

зовнішніми джерелами топології та Topology Event Manager для розповсюдження подій про зміни топології.

Сервіс підтримує множинні представлення топології, включаючи фізичну топологію, логічну топологію, накладні мережі та специфічні для застосунків представлення. Фізична топологія представляє фактичну апаратну зв'язність, логічна топологія включає віртуальні мережі, VLAN та інші логічні конструкції, а накладна топологія може представляти погляд застосунків програмно-визначених мереж на мережу. Topology Service забезпечує автоматичне виявлення топології через інтеграцію з південними протоколами та зовнішніми системами управління, оновлення топології в реальному часі при змінах у мережі та інтелектуальні механізми кешування для високопродуктивних запитів топології.

Topology Service також включає розширені можливості для аналізу топології та обчислення шляхів. Сервіс підтримує різні графові алгоритми для обчислення найкоротших шляхів, виявлення альтернативних шляхів, аналізу досяжності мережі та ідентифікації вузьких місць. Ці можливості використовуються іншими сервісами та застосунками для прийняття інтелектуальних рішень щодо маршрутизації, балансування навантаження, інженерії трафіку та оптимізації мережі. Сервіс забезпечує масштабовану обробку топології навіть для великих мереж з тисячами пристроїв та десятками тисяч з'єднань.

Підсистема правил потоків та управління трафіком. Flow Rule Subsystem відповідає за управління правилами обробки трафіку в мережевих пристроях та є центральним компонентом для реалізації політик маршрутизації в програмно-визначеній мережі. Ця підсистема включає Flow Rule Manager для централізованого управління правилами потоків, Flow Rule Provider для взаємодії з пристроями, Flow Rule Store для постійного зберігання правил та Flow Statistics Manager для збору статистики використання правил. Підсистема забезпечує абстракцію над різними моделями таблиць потоків, підтримуючи як специфічні для OpenFlow

концепції, так і більш загальні моделі обробки пакетів.

Flow Rule Service відповідає за централізоване управління правилами потоків у всіх мережевих пристроях та є критично важливим компонентом для реалізації логіки пересилання програмно-визначених мереж. Цей сервіс забезпечує уніфікований інтерфейс для встановлення, модифікації та видалення правил потоків, незалежно від базових можливостей пристроїв або південних протоколів. Сервіс підтримує складне управління життєвим циклом правил потоків, включаючи автоматичне встановлення правил на цільових пристроях, виявлення та вирішення конфліктів між конфліктуючими правилами, автоматичне оновлення правил з обмеженим часом життя, та інтелектуальне очищення правил для оптимізації використання таблиць.

Архітектура Flow Rule Subsystem використовує концепцію цілей потоків - високорівневі абстракції, які описують бажану поведінку без прив'язки до специфічних деталей реалізації на конкретних пристроях. Система автоматично перекладає цілі потоків в конкретні правила потоків відповідно до можливостей цільових пристроїв. Це дозволяє застосункам працювати з універсальними примітивами, не зважаючи на гетерогенність мережевої інфраструктури.

Flow Objective Service надає абстракцію вищого рівня через концепцію цілей потоків - незалежних від виробника описів бажаної поведінки пересилання. Цей сервіс автоматично перекладає цілі потоків у специфічні для пристроїв правила потоків, дозволяючи застосункам працювати з переносними примітивами пересилання. Сервіс підтримує різні типи цілей, включаючи Filtering Objectives для класифікації пакетів, Forwarding Objectives для вибору наступного вузла та Next Objectives для специфікації дій. Це забезпечує незалежність від пристроїв та спрощує розробку мережевих застосунків.

Flow Rule Subsystem включає також механізми для оптимізації використання ресурсів таблиць потоків, автоматичного управління часом

життя правил та збирання сміття для видалення застарілих записів. Система відстежує використання кожного правила та автоматично видаляє неактивні правила для звільнення місця в таблицях. Для критично важливих правил підсистема забезпечує механізми проактивного оновлення та резервних стратегій для гарантування безперервності обслуговування.

2.4 Високорівневі сервіси та інтерфейси управління

Високорівневі сервіси ONOS надають потужні абстракції для декларативного опису мережевих політик та автоматизації складних операцій управління, значно спрощуючи розробку мережевих застосунків та підвищуючи їх переносимість між різними мережевими середовищами. Intent Framework представляє сучасний підхід до програмування мережі, дозволяючи описувати бажану поведінку через високорівневі декларативні заяви замість низькорівневого імперативного програмування правил потоків, автоматично перекладаючи наміри в конкретні конфігурації з урахуванням поточного стану мережі, доступних ресурсів та динамічних змін топології. Application Management Service забезпечує повний життєвий цикл мережевих застосунків від розгортання до моніторингу з механізмами безпечної ізоляції, контролю доступу та управління ресурсами, тоді як Security Service надає комплексну систему автентифікації, авторизації та аудиту для захисту від несанкціонованого доступу та зловмисних дій. Network Configuration Service централізує управління всіма типами конфігурацій з підтримкою версіонування, перевірки коректності та автоматичного розповсюдження змін, а моніторинг та аналітичні сервіси забезпечують спостережність системи через збір телеметричних даних, виявлення аномалій та генерацію звітів про стан мережі. Інтеграція цих високорівневих сервісів створює потужну платформу для побудови інтелектуальних самокеруючих мереж, здатних автоматично адаптуватися до змінюваних умов, оптимізувати використання ресурсів та забезпечувати необхідні гарантії якості обслуговування [10].

Фреймворк намірів. Intent Framework являє собою один з найбільш

інноваційних компонентів архітектури ONOS, який дозволяє описувати мережеві політики та вимоги на високому рівні абстракції без необхідності специфікації технічних деталей реалізації. Intent представляє декларативний опис бажаної поведінки мережі, який система автоматично перекладає в конкретні конфігурації пристроїв та правила потоків. Архітектура включає Intent Manager для управління життєвим циклом намірів, Intent Compiler для трансляції високорівневих намірів в виконувані мережеві операції, та Intent Store для постійного зберігання намірів.

Intent Service являє собою сучасний підхід до програмування мережі, який дозволяє описувати поведінку мережі через високорівневі декларативні заяви замість низькорівневого імперативного програмування. Цей сервіс включає Intent Manager для управління життєвим циклом намірів, фреймворк Intent Compiler для перекладу намірів у виконувані мережеві операції, Intent Store для постійного зберігання намірів та Intent Monitor для безперервної перевірки задоволення намірів. Intent Service забезпечує автоматичну компіляцію намірів, вирішення конфліктів між конкуруючими намірами та динамічну перекомпіляцію при зміні мережевих умов.

Система підтримує різні типи намірів, включаючи Connectivity Intent для встановлення зв'язності між кінцевими точками, Path Intent для примусового маршрутизування через певний шлях, Bandwidth Intent для резервування пропускної здатності та Protection Intent для створення резервних шляхів. Кожен тип наміру має власний компілятор, який знає як перекласти високорівневі вимоги в конкретні мережеві операції з урахуванням поточного стану мережі та доступних ресурсів. Система підтримує багату таксономію типів намірів для різних випадків використання мережі, включаючи Point-to-Point Intent для виділеної зв'язності зі специфічними вимогами до якості обслуговування, Multi-Point Intent для групової або широкомовної зв'язності, та Host-to-Host Intent для автоматичної адаптації до мобільності вузлів.

Intent Framework включає також механізми безперервного моніторингу

та автоматичної перекомпіляції, які забезпечують адаптацію мережі до змін в топології або відмов компонентів. Система постійно відстежує виконання кожного наміру та автоматично перекомпілює їх у випадку змін, які впливають на можливість їх реалізації. Intent Monitor безперервно перевіряє, що мережа дійсно задовольняє вимоги намірів через моніторинг стану мережі в реальному часі та аналіз статистики потоків. У випадку виявлення порушень намірів або змін у мережі, які впливають на задоволення намірів, система автоматично запускає процеси перекомпіляції та переустановлення. Це забезпечує самовідновлювальну поведінку мережі та автоматичну адаптацію до відмов, змін топології або конфліктів ресурсів.

Управління застосунками та сервіси безпеки. Application Management Service відповідає за комплексне управління життєвим циклом мережевих застосунків в екосистемі ONOS. Цей сервіс включає Application Store для зберігання метаданих, Application Registry для відстеження застосунків у часі виконання, Permission Manager для контролю доступу, та Application Event Manager для міжзастосункової комунікації. Сервіс забезпечує безпечну ізоляцію застосунків через механізми пісочниці, примусове виконання квот ресурсів для запобігання їх вичерпанню, та управління залежностями для складних розгортань з множинними застосунками.

Сервіс підтримує динамічні операції життєвого циклу застосунків, включаючи гаряче розгортання нових застосунків без перезапуску системи, активацію та деактивацію застосунків у часі виконання для операційної гнучкості, версіонування застосунків та можливості відкату для безпечних оновлень, та комплексний моніторинг застосунків для аналізу продуктивності. Application Management Service також забезпечує складну систему дозволів, де застосунки можуть запитувати специфічні можливості, а система автоматично забезпечує контроль доступу на основі наданих дозволів.

Security Service забезпечує комплексну систему безпеки для всіх

аспектів операцій ONOS. Цей сервіс включає Authentication Service для перевірки ідентичності користувачів та застосунків, Authorization Service для забезпечення контролю доступу, Certificate Management для операцій з інфраструктурою відкритих ключів, та Security Event Manager для виявлення інцидентів безпеки та реагування на них. Сервіс підтримує інтеграцію з зовнішніми постачальниками ідентичності через стандартні протоколи, такі як LDAP, Active Directory, SAML та OAuth, забезпечуючи безшовну інтеграцію з існуючою корпоративною інфраструктурою безпеки.

Сервіс мережевої конфігурації. Network Configuration Service надає уніфіковану платформу для управління всіма типами мережевих конфігурацій у середовищі ONOS. Цей сервіс забезпечує централізоване сховище конфігурацій, механізми перевірки коректності, відстеження змін та автоматичне розповсюдження оновлень конфігурації до відповідних компонентів. Сервіс включає Configuration Store для постійного зберігання, Configuration Event Manager для повідомлень про зміни, Configuration Validator для перевірки коректності та Configuration Synchronizer для підтримки консистентності між вузлами кластера.

Сервіс підтримує ієрархічні схеми конфігурації з підтримкою різних доменів конфігурації, таких як конфігурації пристроїв, мережеві політики, налаштування застосунків та системні параметри. Configuration Service забезпечує атомарні оновлення конфігурації для консистентного стану системи, версіонування конфігурації для відстеження змін та можливості відкату, та генерацію конфігурації на основі шаблонів для ефективного масового розгортання конфігурації. Сервіс також включає складний механізм перевірки коректності, який перевіряє консистентність та сумісність конфігурації перед застосуванням.

Network Configuration Service інтегрується з Intent Service для автоматичного перекладу змін конфігурації у відповідні наміри, з Device Service для автоматичної синхронізації конфігурації пристроїв, та з Application Service для динамічної реконфігурації застосунків. Це забезпечує

безшовний досвід управління конфігурацією, де зміни в одному домені автоматично поширюються до пов'язаних доменів, зберігаючи консистентність та стабільність системи.

Моніторинг та аналітичні сервіси. Network Monitoring Service забезпечує комплексну спостережність мережі через збір, обробку та аналіз різних типів мережевих даних. Цей сервіс включає двигун збору метрик для збору даних про продуктивність, сервіс агрегації статистики для обробки даних, диспетчер тривог для сповіщень на основі порогів, та сервіс панелі інструментів для візуалізації даних. Сервіс автоматично збирає статистику від мережевих пристроїв, моніторить використання потоків, відстежує зміни топології та аналізує шаблони трафіку для надання інформації про поведінку мережі.

Модуль моніторингу продуктивності відстежує ключові показники продуктивності, такі як використання з'єднань, втрати пакетів, затримка, джитер та пропускна здатність по всій мережевій інфраструктурі. Сервіс підтримує моніторинг у реальному часі з налаштовуваними частотами вибірки, збереження історичних даних для аналізу тенденцій, та складні механізми сповіщень для проактивного виявлення проблем. Дані моніторингу використовуються застосунками інженерії трафіку для прийняття рішень щодо оптимізації та інструментами планування мережі для прогнозування потужності.

Analytics Service надає розширені можливості обробки даних для витягування дієвих висновків з телеметричних даних мережі. Цей сервіс включає двигун машинного навчання для розпізнавання шаблонів та виявлення аномалій, модуль прогнозної аналітики для прогнозування поведінки мережі, та двигун звітності для генерації комплексних мережевих звітів. Сервіс підтримує інтеграцію з зовнішніми аналітичними платформами та фреймворками обробки великих даних для масштабування аналітичних можливостей за межі того, що може бути оброблено на одному кластері ONOS.

Висновки

У ході аналізу архітектурної побудови ONOS було встановлено, що система базується на передових принципах мікросервісної архітектури та використанні фреймворку OSGi. Такий підхід забезпечує фундаментальну модульність та гнучкість, дозволяючи розробляти та впроваджувати нові сервіси управління трафіком без необхідності зупинки чи модифікації ядра системи. Багатошарова структура з чітким розділенням на рівні абстракції (від Protocol Abstraction Layer до Application Layer) дозволяє ефективно приховувати складність гетерогенного обладнання та надавати уніфіковані інтерфейси для верхніх рівнів.

Критичною перевагою платформи визначено її здатність до кластеризації та забезпечення високої доступності (High Availability). Використання розподіленої системи управління станом Atomix на базі протоколу консенсусу Raft гарантує сувору консистентність даних та автоматичне відновлення після відмов, що є обов'язковою вимогою для сервіс-провайдерів. Активно-активна конфігурація кластера дозволяє масштабувати систему горизонтально, забезпечуючи надійне управління мережами з тисячами пристроїв.

Окрему увагу в структурі компонентів заслуговує Intent Framework, який реалізує сучасний підхід до програмування мережі. Можливість декларативного опису бізнес-політик (намірів) замість імперативного налаштування окремих правил потоків значно спрощує розробку додатків Traffic Engineering. Наявність механізмів автоматичної компіляції та моніторингу намірів забезпечує самовідновлювану поведінку мережі та її адаптацію до змін топології в реальному часі.

Підсумовуючи, архітектура ONOS, завдяки своїм розвиненим сервісам (Topology Service, Flow Rule Service) та потужним API, створює ідеальну екосистему для розробки запланованого стартап-проекту. Вона надає необхідні інструменти для створення інтелектуальних алгоритмів маршрутизації та гарантування якості обслуговування (QoS), спираючись на надійну та масштабовану платформу.

РОЗДІЛ 3

АНАЛІЗ ПРОТОКОЛІВ ВЗАЄМОДІЇ В ONOS

3.1 Загальні принципи протокольної взаємодії

Ефективне функціонування кластерної архітектури ONOS забезпечується комплексною системою протоколів взаємодії, які організовують комунікацію між вузлами контролера, синхронізують розподілений стан та координують управління мережевими пристроями. На відміну від монолітних систем, де всі компоненти працюють в єдиному адресному просторі, розподілена природа ONOS вимагає складної інфраструктури для забезпечення узгодженості даних, виявлення відмов та розподілу відповідальності.

Протоколи взаємодії в ONOS можна класифікувати за кількома критеріями. За рівнем архітектури розрізняють внутрішньокластерні протоколи для координації між вузлами контролера та зовнішні протоколи для взаємодії з мережевими пристроями та додатками. За призначенням виділяють протоколи консенсусу для досягнення згоди щодо критичних рішень, протоколи синхронізації стану для поширення інформації, протоколи виявлення відмов для забезпечення надійності та протоколи управління для конфігурації мережевих пристроїв.

Ключовою особливістю протокольної екосистеми ONOS є багаторівневий підхід до забезпечення узгодженості даних. Для критичних операцій, таких як призначення відповідальності за пристрої або зміна конфігурацій, використовуються протоколи строгого консенсусу, що гарантують атомарність та узгодженість. Для менш критичних даних, як-от статистика або метрики, застосовуються протоколи поступової узгодженості, які жертвують миттєвою консистентністю заради продуктивності та масштабованості.

3.2 Протоколи внутрішньокластерної координації

Координація між вузлами кластера ONOS забезпечується комплексом протоколів різного призначення, які спільно гарантують узгодженість розподіленого стану, високу доступність сервісів та швидке виявлення відмов. Протокольна екосистема організована за принципом диференційованого підходу: для критичних операцій (призначення майстрів пристроїв, зміна конфігурацій)

використовуються протоколи строгого консенсусу на базі Raft, що гарантують атомарність та лінеаризовність навіть за умов мережевих відмов, тоді як для менш критичних даних (статистика, метрики) застосовуються протоколи поступової узгодженості на основі Gossip, які жертвують миттєвою консистентністю заради масштабованості. Цей багаторівневий підхід дозволяє балансувати між вимогами до узгодженості та операційною ефективністю, забезпечуючи надійність критичних операцій та високу пропускну здатність для масових оновлень стану.

Raft Protocol. У розподіленій архітектурі ONOS протокол Raft забезпечує фундаментальний механізм досягнення консенсусу між вузлами для критичних операцій, що вимагають строгої узгодженості. Комунікаційна модель Raft організована навколо трьох типів повідомлень, які забезпечують координацію між вузлами: RequestVote для процесу виборів лідера, AppendEntries для реплікації записів лога та Heartbeat для підтримки авторитету лідера.

Процес комунікації під час виборів лідера відбувається наступним чином. Коли послідовник перестає отримувати heartbeat-повідомлення протягом election timeout (зазвичай 150-300 мс), він ініціює новий term (термін) та надсилає RequestVote повідомлення всім іншим вузлам. Кожне RequestVote містить номер терміну кандидата та інформацію про його лог. Вузли-отримувачі аналізують ці дані: вони надають голос лише якщо термін кандидата не менший за їх власний, і лог кандидата принаймні настільки ж актуальний, як їх власний. Кандидат, що отримав голоси від більшості вузлів, стає лідером та починає надсилати періодичні AppendEntries повідомлення для підтвердження свого статусу. Механізм досягнення консенсусу та реплікації логів візуалізовано на рисунку 3.1.

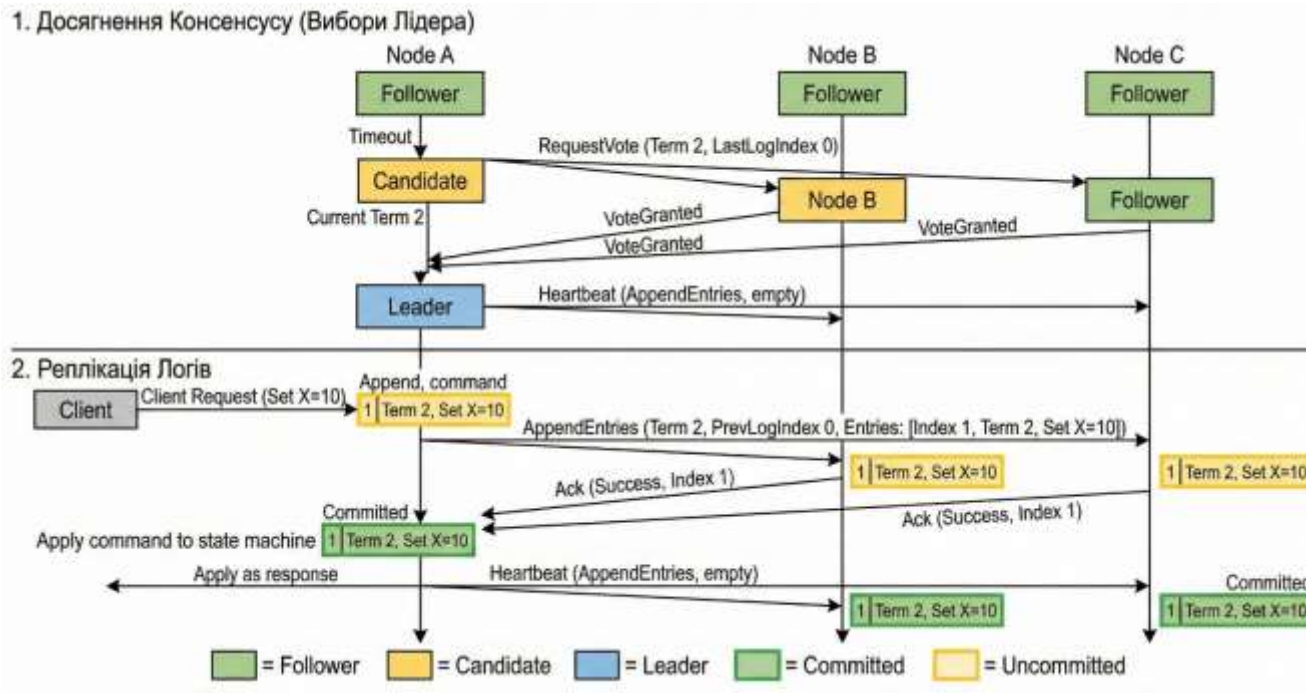

Рис. 3.1. Механізм досягнення консенсусу та реплікації логів у протоколі Raft.

Реплікація даних через протокол Raft використовує механізм AppendEntries повідомлень. Коли лідер отримує запит на зміну стану (наприклад, оновлення конфігурації інтента), він спочатку додає цей запис до свого локального лога, але не застосовує його. Потім лідер надсилає AppendEntries повідомлення всім послідовникам, які додають запис до своїх логів та відповідають підтвердженням. Коли лідер отримує підтвердження від більшості вузлів, він вважає запис committed (зафіксованим), застосовує зміну до свого стану та інформує послідовників, що вони теж можуть застосувати цей запис. Така послідовність гарантує, що дані ніколи не будуть втрачені навіть у разі відмови лідера, оскільки щонайменше більшість вузлів мають копію кожного зафіксованого запису.

Протокол включає механізми обробки мережевих розділень та конфліктів. Якщо вузол отримує повідомлення з терміном, меншим за його власний, він відкидає таке повідомлення. Якщо послідовник виявляє, що його лог неузгоджений з логом лідера (наприклад, через попереднє розділення мережі), лідер автоматично відкочує лог послідовника до точки узгодженості та повторно надсилає всі наступні записи. Це забезпечує, що всі вузли в кінцевому підсумку досягають ідентичного стану.

Протоколи синхронізації на базі Atomix. Платформа Atomix надає ONOS

набір високорівневих протоколів для синхронізації розподілених структур даних, абстрагуючи складність базових алгоритмів консенсусу. Комунікаційна архітектура Atomix побудована на концепції розподілених примітивів, кожен з яких реалізує специфічний протокол взаємодії відповідно до семантики структури даних.

Для розподілених відображень (distributed maps) Atomix використовує протокол, заснований на Raft, з оптимізаціями для операцій читання. Операції запису (put, remove) завжди проходять через лідера партиції та реплікуються на більшість вузлів перед підтвердженням. Однак для операцій читання Atomix підтримує кілька режимів консистентності через різні протоколи комунікації: linearizable reads вимагають звернення до лідера та перевірки його актуальності через heartbeat механізм, bounded-staleness reads дозволяють читання з будь-якого вузла з гарантією максимального часу застарівання даних, а eventual-consistency reads можуть обслуговуватися локально без міжвузлової комунікації.

Протокол транзакцій в Atomix забезпечує атомарні операції над множиною ключів. Клієнт спочатку надсилає prepare повідомлення лідерам всіх задіяних партицій, які блокують відповідні ключі та перевіряють можливість виконання операції. Якщо всі партиції відповідають позитивно, клієнт надсилає commit повідомлення, і лідери застосовують зміни через стандартний Raft протокол. У випадку відмови будь-якої партиції, клієнт надсилає abort повідомлення для відкату транзакції. Цей двофазний протокол гарантує ACID властивості для розподілених операцій.

Для оптимізації продуктивності Atomix використовує протокол пакетування (batching) запитів. Замість обробки кожної операції окремо, система накопичує множину запитів протягом короткого часового вікна та обробляє їх як єдину Raft-транзакцію. Це значно зменшує кількість міжвузлових повідомлень та дисковий ввід-вивід, підвищуючи пропускну здатність системи. Лідер періодично (зазвичай кожні 10-50 мс) збирає накопичені запити, створює пакет та реплікує його на послідовників.

Gossip Protocol. Gossip Protocol забезпечує децентралізований механізм

розповсюдження інформації в кластері ONOS, особливо ефективний для даних, що не вимагають строгої узгодженості, але потребують високої доступності та стійкості до відмов. Протокол працює за принципом епідемічного поширення: інформація розповсюджується від вузла до вузла подібно до того, як поширюються чутки в соціальних мережах.

Комунікаційний цикл протоколу організований наступним чином. Кожен вузол підтримує локальну базу даних стану (state database), що містить інформацію про членство в кластері, призначення майстрів для пристроїв та метадані додатків. Періодично (зазвичай кожні 1-2 секунди) вузол обирає випадковий набір з k інших вузлів (типово k=3) та ініціює з ними gossip exchange. Обмін відбувається в три фази: спочатку вузол надсилає digest повідомлення, що містить summary своєї бази даних (зазвичай версії або хеші записів), потім отримувач порівнює цей digest зі своєю базою та відповідає delta повідомленням, що містить записи, яких немає у відправника, або які новіші, і нарешті відправник оновлює свою базу та може надіслати власні delta записи назад. Діаграму послідовності обміну повідомленнями для синхронізації стану наведено на рисунку 3.2.

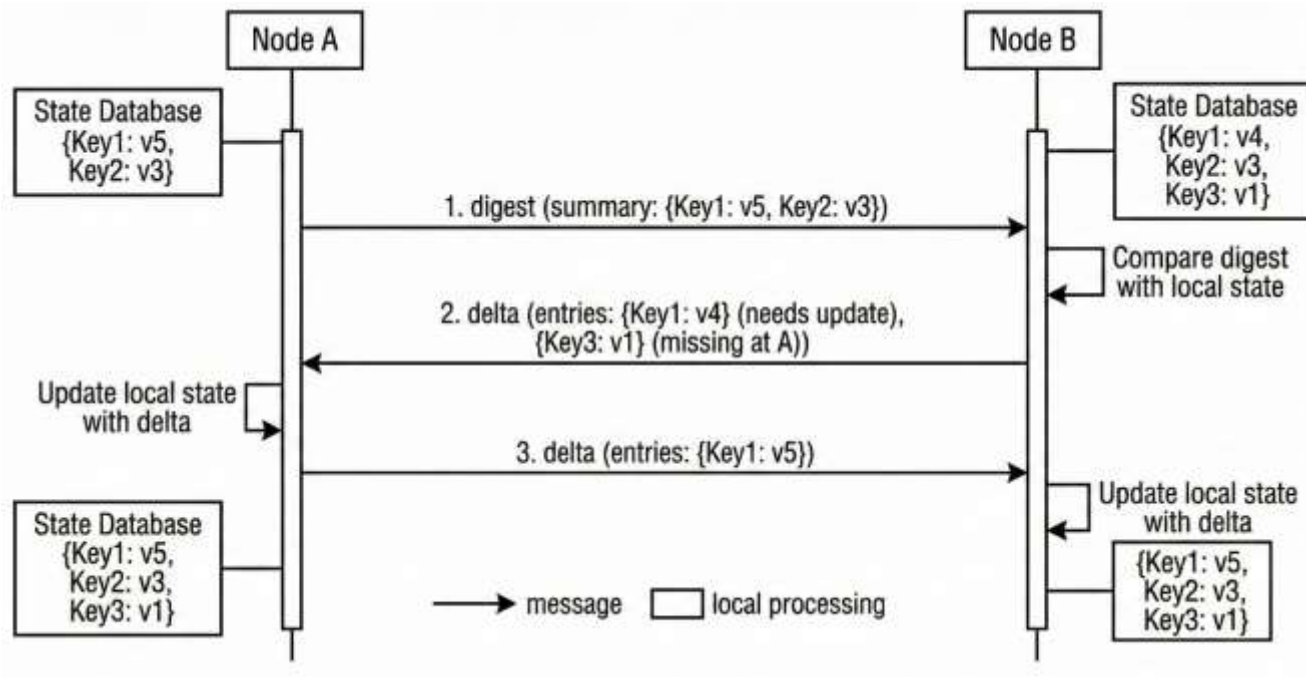

Рис. 3.2. Діаграма послідовності обміну повідомленнями в протоколі Gossip для синхронізації стану.

Для cluster membership information протокол працює наступним чином. Коли

новий вузол приєднується до кластера, він спочатку контактує з одним або кількома seed nodes (вузли-насіння), отримує повний список членів кластера та починає брати участь у gossip обміні. Кожен запис про вузол містить його ідентифікатор, адресу, heartbeat counter та версію інформації. Коли вузол отримує через gossip інформацію про новий вузол або оновлення існуючого, він перевіряє версію: якщо отримана версія новіша, запис оновлюється. Периодичний gossip забезпечує, що інформація про новий вузол за логарифмічний час (O(log N) раундів) поширюється на всі вузли кластера.

Device mastership information синхронізується через спеціалізований gossip протокол з додатковими гарантіями. Кожне призначення майстра для пристрою має term number та timestamp. Коли вузол стає майстром пристрою через Master Election протокол, він збільшує term number та розповсюджує цю інформацію через gossip. Інші вузли, отримуючи повідомлення з вищим term number, визнають нового майстра. Протокол включає механізм anti-entropy для виправлення розбіжностей: окрім регулярного gossip, вузли періодично виконують повну синхронізацію з випадково обраним партнером, обмінюючись повними snapshot своїх баз даних.

Протокол також включає механізми оптимізації для великих кластерів. Натомість надсилання повних digest, використовується Bloom filters для компактного представлення наявних записів, що зменшує розмір повідомлень. Для даних, що рідко змінюються, застосовується rate limiting - якщо запис не змінювався протягом певного часу, його version включається в digest рідше. Це зменшує непотрібний трафік при збереженні швидкості поширення нових даних.

Master Election Protocol. Master Election Protocol використовує специфічну послідовність повідомлень для координації призначення відповідальності за пристрої. Протокол заснований на концепції Leadership Election примітиву Atomix, який реалізує Raft-based розподілений лок з додатковою логікою для fair scheduling.

Комунікаційна послідовність протоколу виглядає наступним чином. Спочатку всі вузли, які бажають керувати певним пристроєм, надсилають compete повідомлення до Atomix leadership service, вказуючи device ID, свій ідентифікатор та priority score (обчислений на основі поточного навантаження та топологічної

близькості). Leadership service, працюючи на Raft-групі, збирає ці запити та використовує детерміністичний алгоритм для вибору майстра. Типово обирається вузол з найвищим priority score, але система також враховує поточний розподіл навантаження для балансування. Обраний майстер отримує leadership grant повідомлення, після чого починає активно управляти пристроєм. Всі інші вузли отримують standby role повідомлення з їх позицією в черзі резервних майстрів.

Протокол підтримки майстерства використовує lease-based механізм. Майстер повинен періодично (зазвичай кожні 5-10 секунд) надсилати renew lease повідомлення до leadership service, підтверджуючи свою працездатність. Якщо майстер не продовжує lease протягом timeout періоду (зазвичай 15-30 секунд), leadership service автоматично ініціює reelection. Перший резервний вузол отримує promote повідомлення та стає новим майстром. Інші резервні вузли отримують updated standby list повідомлення з оновленими позиціями в черзі.

Протокол включає механізм демократичного перепризначення для балансування навантаження. Адміністратор або система балансування може ініціювати rebalance request, який запускає новий цикл виборів для групи пристроїв. Поточні майстри отримують graceful handover повідомлення, що дає їм час завершити поточні операції перед передачею управління. Новообрані майстри отримують full state transfer від попередніх майстрів через dedicated synchronization channel, що гарантує безшовний перехід без втрати даних про конфігурації або поточні операції.

У випадку мережевого розділення кластера протокол використовує quorum-based approach. Leadership service може працювати лише в partition, що містить більшість вузлів Raft-групи. Вузли в minority partition автоматично відмовляються від майстерства та припиняють активні операції з пристроями. Після відновлення зв'язку протокол виконує reconciliation: порівнює версії leadership assignments з обох партицій, обирає ті, що мають вищі term numbers, та синхронізує стан між усіма вузлами.

Event Distribution Protocol. Система розповсюдження подій в ONOS побудована на event-driven архітектурі, що забезпечує асинхронну та слабко

зв'язану комунікацію між компонентами на різних вузлах кластера. Архітектура Event Distribution Protocol базується на Distributed Event Bus - розподіленій шині подій, що реалізує publish-subscribe модель поверх Atomix.

Процес публікації події відбувається наступним чином. Коли компонент (наприклад, Device Service) виявляє зміну стану (нове з'єднання пристрою, зміна конфігурації), він створює об'єкт події та викликає publish operation на локальному Event Bus proxy. Проксі сериалізує подію, призначає їй sequence number через Atomix counter та надсилає її до Atomix distributed topic. Atomix, використовуючи свій внутрішній Raft протокол, реплікує подію на всі вузли, що підписані на цей topic, гарантуючи total order delivery - всі підписники отримують події в ідентичній послідовності.

Механізм підписки працює через registration protocol. Коли компонент хоче отримувати певні типи подій, він надсилає subscribe request до локального Event Bus proxy, вказуючи event topic та опціонально event filter (предикат для фільтрації подій). Проксі реєструє підписку в Atomix distributed set, після чого починає отримувати відповідні події. Підписки автоматично синхронізуються між вузлами - якщо компонент на вузлі A підписується на топологічні події, всі вузли дізнаються про цю підписку і можуть ефективно маршрутизувати події.

Для критичних подій, таких як зміни топології або конфігурацій, протокол використовує reliable delivery mechanism. Кожна подія має версію та delivery confirmation requirement. Після отримання події підписник обробляє її та надсилає acknowledgment повідомлення назад до Event Bus. Якщо acknowledgment не отримано протягом timeout періоду, система автоматично повторює доставку. Події зберігаються у persistent queue до отримання підтвердження від усіх підписників, що гарантує, що жодна критична подія не буде втрачена навіть у випадку тимчасових відмов вузлів.

Протокол також включає механізм event ordering guarantees на кількох рівнях. Per-source ordering гарантує, що події від одного джерела обробляються в порядку їх генерації. Causal ordering забезпечує, що якщо подія B причинно залежить від події A, то B буде оброблена після A на всіх вузлах. Total ordering гарантує

глобальну послідовність для всіх подій в певному topic. Для оптимізації продуктивності менш критичні події можуть використовувати weakened ordering guarantees.

Event Bus підтримує event batching для оптимізації пропускної здатності. Замість надсилання кожної події окремо, система накопичує події протягом короткого часового вікна (зазвичай 10-50 мс) та надсилає їх пакетами. Це зменшує кількість міжвузлових повідомлень та overhead від Raft протоколу. Для event-sensitive додатків можливе налаштування максимальної затримки батчингу або повна його відмова на користь мінімальної латентності.

Протоколи виявлення та обробки відмов. Система виявлення відмов в ONOS організована як багаторівнева ієрархія протоколів, кожен з яких забезпечує специфічний аспект детектування та реагування на збої. Ця архітектура дозволяє системі швидко виявляти різні типи відмов - від повної втрати вузла до часткової деградації сервісів - та ініціювати відповідні процедури відновлення.

На найнижчому рівні працює Heartbeat Protocol для базового моніторингу живучості вузлів. Кожен вузол підтримує heartbeat sender, який періодично (зазвичай кожні 1-2 секунди) надсилає heartbeat повідомлення всім іншим вузлам кластера через UDP multicast або direct TCP connections. Heartbeat повідомлення містить ідентифікатор вузла, часову мітку, sequence number та список інших вузлів, від яких відправник нещодавно отримував heartbeat (gossip-style). Кожен вузол підтримує failure detector, який відстежує час останнього отриманого heartbeat від кожного партнера. Якщо heartbeat не отримано протягом detection threshold (зазвичай 5-10 секунд), вузол позначається як suspected.

Для підвищення точності детектування ONOS використовує φ-accrual failure detector algorithm. На відміну від простого timeout-based детектора, який використовує фіксований поріг, φ-accrual адаптується до мережевих умов. Алгоритм підтримує історію arrival times попередніх heartbeat повідомлень та обчислює статистичний розподіл інтервалів між ними. На основі цього розподілу він динамічно оцінює ймовірність того, що вузол відмовив. Значення φ (phi) представляє впевненість у відмові: φ=1 означає 10% ймовірність помилки, φ=2 -

1%, φ=3 - 0.1% і так далі. Адміністратор може налаштувати threshold φ, балансуючи між швидкістю детектування та кількістю хибних спрацьовувань.

Application-Level Health Checking Protocol забезпечує більш глибоку перевірку працездатності критичних сервісів. Окрім перевірки, що вузол "живий", система регулярно виконує synthetic health checks - тестові операції, що перевіряють функціональність ключових компонентів. Наприклад, health checker періодично намагається прочитати та записати тестові дані в Atomix distributed map, виконати query до Device Service, або отримати топологічну інформацію. Кожна операція має timeout (зазвичай 2-5 секунд), і якщо вона не завершується успішно, health checker позначає відповідний сервіс як degraded. Результати health checks розповсюджуються через Gossip Protocol, щоб всі вузли знали про стан сервісів на інших вузлах.

Network Partition Detection Protocol вирішує складну проблему split-brain scenarios, коли кластер розділяється на дві або більше ізольовані групи. Протокол базується на quorum-based approach: тільки partition, що містить більшість вузлів (strict majority), може продовжувати активні операції. Детектування розділення відбувається через аналіз connectivity graph: кожен вузол підтримує список вузлів, з якими він може комунікувати (на основі heartbeat). Якщо вузол виявляє, що може зв'язатися лише з меншістю кластера, він переходить у passive mode - припиняє приймати нові запити та відмовляється від ролей майстрів для пристроїв. Atomix автоматично забезпечує, що його Raft-групи можуть працювати лише в majority partition, що гарантує консистентність критичних даних.

Після виявлення відмови вузла або сервісу ініціюється Failure Recovery Protocol. Процес відновлення включає кілька фаз. Спочатку відбувається notification phase: система генерує failure events, які розповсюджуються через Event Bus на всі зацікавлені компоненти. Потім Master Reelection Service ініціює reelection для всіх пристроїв, майстром яких був відмовлений вузол. Leadership Service автоматично призначає нових майстрів з резервних вузлів через Master Election Protocol. Наступна фаза - state recovery: новообрані майстри синхронізують стан пристроїв, відновлюють з'єднання та перепрограмовують flow rules якщо потрібно.

Нарешті, відбувається load rebalancing: система аналізує поточний розподіл навантаження та може ініціювати перепризначення деяких пристроїв для оптимального балансування між вузлами, що залишилися.

Протокол включає також механізм false positive mitigation для зменшення впливу хибних спрацьовувань. Коли вузол підозрюється у відмові, система не одразу ініціює recovery, а спочатку виконує secondary verification: намагається встановити alternative communication channels з підозрюваним вузлом або запитує думку інших вузлів через gossip. Тільки якщо множинні незалежні джерела підтверджують відмову, ініціюється повноцінна процедура відновлення. Це особливо важливо в умовах нестабільних мережевих з'єднань, де тимчасові проблеми зв'язку не повинні призводити до масштабних перебудов кластера.

3.3 Southbound протоколи управління мережевими пристроями

Southbound інтерфейс ONOS реалізує взаємодію контролера з гетерогенною інфраструктурою мережевих пристроїв через множину спеціалізованих протоколів, кожен з яких оптимізований для специфічних аспектів управління площиною даних. Архітектура побудована за принципом розділення відповідальності: OpenFlow Protocol забезпечує динамічне програмування таблиць потоків та реактивну обробку трафіку, OVSDB Protocol надає можливості управління віртуальною мережевою інфраструктурою (мости, порти, тунелі), а gNMI Protocol представляє нове покоління інтерфейсів на базі gRPC та YANG моделей для високоефективної конфігурації та телеметрії в реальному часі. Така багатопротокольна архітектура забезпечує гнучкість інтеграції, дозволяючи ONOS ефективно керувати як класичними OpenFlow комутаторами, так і сучасними whitebox пристроями з підтримкою програмних інтерфейсів [3].

OpenFlow Protocol. OpenFlow Protocol є фундаментальним southbound протоколом, який забезпечує комунікацію між контролером ONOS та мережевими пристроями площини даних. Протокол реалізує чітке розділення між control plane, де приймаються рішення про обробку трафіку, та data plane, де ці рішення виконуються в апаратних або програмних комутаторах. ONOS використовує OpenFlow для динамічного програмування поведінки мережевих пристроїв через

встановлення, модифікацію та видалення правил потоків.

Комунікаційна сесія OpenFlow встановлюється через TCP з'єднання (зазвичай на порту 6653) між комутатором та контролером. Протокол handshake починається з того, що комутатор надсилає OFPT_HELLO повідомлення, вказуючи підтримувані версії OpenFlow. ONOS відповідає власним OFPT_HELLO, після чого обидві сторони погоджують спільну версію протоколу. Далі контролер надсилає OFPT_FEATURES_REQUEST, на яке комутатор відповідає OFPT_FEATURES_REPLY, що містить datapath ID (унікальний ідентифікатор), кількість буферів, кількість таблиць потоків, підтримувані capabilities та список портів з їх характеристиками.

Після встановлення з'єднання ONOS конфігурує комутатор через серію SET_CONFIG повідомлень, визначаючи режим обробки фрагментованих пакетів та максимальний розмір Packet-In повідомлень. Контролер також встановлює початкові flow rules через OFPT_FLOW_MOD повідомлення для базової функціональності, наприклад table-miss entry - правило за замовчуванням, яке надсилає невідомі пакети на контролер.

Протокол управління потоками використовує OFPT_FLOW_MOD повідомлення з різними командами. OFPFC_ADD додає нове правило в таблицю потоків, включаючи match fields (критерії відповідності пакетів: MAC адреси, IP адреси, VLAN теги, TCP/UDP порти), priority (пріоритет правила), actions (дії: пересилання на порт, модифікація заголовків, відкидання), та timeout values (idle_timeout для неактивних потоків, hard_timeout для абсолютного часу життя). OFPFC_MODIFY дозволяє змінювати існуючі правила, OFPFC_DELETE видаляє правила відповідно до критеріїв пошуку, а OFPFC_DELETE_STRICT видаляє правило з точним співпадінням match та priority.

Reactive flow programming реалізується через Packet-In/Packet-Out механізм. Коли пакет надходить на комутатор і не знаходить відповідного правила, комутатор надсилає OFPT_PACKET_IN повідомлення контролеру, включаючи buffer_id (якщо пакет збережено в буфері комутатора), in_port (порт надходження), reason (причина: table-miss, action explicit send), та payload (повний пакет або його

заголовок). ONOS аналізує пакет, приймає рішення про маршрутизацію та може надіслати OFPT_PACKET_OUT для негайного пересилання пакета з вказаними actions. Одночасно контролер встановлює відповідне правило потоку через FLOW_MOD, щоб наступні пакети цього потоку оброблялися безпосередньо комутатором без залучення контролера. Загальна схема обміну повідомленнями протоколу OpenFlow представлена на рисунку 3.3.

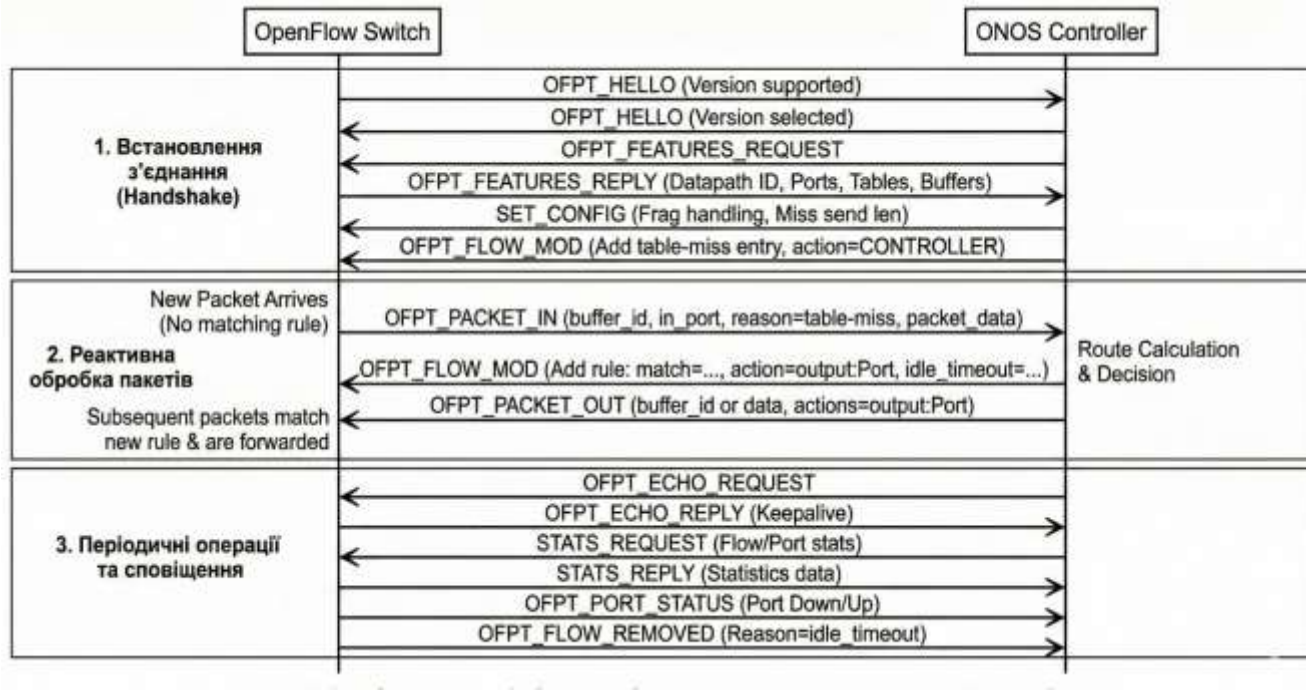

Рис. 3.3. Діаграма обміну повідомленнями протоколу OpenFlow

Statistics collection здійснюється через серію STATS_REQUEST/STATS_REPLY повідомлень. OFPST_FLOW_REQUEST запитує статистику по конкретних flow rules (кількість пакетів, байтів, тривалість існування правила). OFPST_PORT_REQUEST отримує статистику портів (отримано/передано пакетів, помилки, відкинуті пакети). OFPST_TABLE_REQUEST запитує інформацію про заповнення таблиць потоків. ONOS періодично (зазвичай кожні 5-30 секунд) опитує комутатори для збору цих метрик, які використовуються для моніторингу мережі та traffic engineering.

Протокол також включає асинхронні сповіщення про події. OFPT_PORT_STATUS надсилається комутатором при зміні стану порту (підключення/відключення кабелю, зміна швидкості). OFPT_FLOW_REMOVED інформує контролер про видалення правила потоку через спрацювання timeout або

заповнення таблиці. OFPT_ERROR повідомляє про помилки обробки команд контролера. Ці асинхронні повідомлення дозволяють ONOS підтримувати актуальне бачення стану мережі без постійного опитування.

Для підтримки keepalive та виявлення відмов з'єднань використовуються OFPT_ECHO_REQUEST/OFPT_ECHO_REPLY повідомлення. ONOS періодично надсилає echo requests, і якщо не отримує відповіді протягом timeout періоду, вважає з'єднання з комутатором втраченим та ініціює процедури відновлення.

OVSDB Protocol. OVSDB Protocol забезпечує інтерфейс для конфігурації та моніторингу віртуальних комутаторів Open vSwitch, доповнюючи OpenFlow можливостями управління самою інфраструктурою комутатора. Тоді як OpenFlow керує потоками даних, OVSDB дозволяє ONOS керувати структурою віртуальної мережі: створювати bridges, порти, тунелі та інші об'єкти.

Протокол базується на JSON-RPC 2.0 і працює через постійне TCP з'єднання (зазвичай на порту 6640). Комунікація може бути ініційована як контролером (active mode), так і комутатором (passive mode). ONOS зазвичай використовує active mode, встановлюючи з'єднання з OVSDB server, що працює на хості з Open vSwitch.

OVSDB організована як реляційна база даних з чітко визначеною схемою. Основні таблиці включають Open_vSwitch (глобальна конфігурація), Bridge (віртуальні мости), Port (порти мостів), Interface (мережеві інтерфейси), Controller (налаштування підключення до OpenFlow контролерів). Кожен запис має UUID та набір колонок відповідно до схеми.

Протокол підтримує кілька типів RPC операцій. transact method виконує атомарні транзакції над базою даних, які можуть включати множину операцій: insert для додавання нових записів, delete для видалення, update для модифікації існуючих записів, select для запитів даних, mutate для атомарних змін (наприклад, додавання елемента в множину). monitor method встановлює підписку на зміни в таблицях - ONOS отримує notifications кожного разу, коли відбувається зміна в моніторюваних таблицях, що дозволяє відстежувати стан інфраструктури в реальному часі.

Типовий сценарій створення VXLAN тунелю виглядає наступним чином.

ONOS надсилає transact запит, який включає insert операцію для таблиці Interface з параметрами type="vxlan", options={"remote_ip": "...", "key": "..."}, далі insert для таблиці Port, що посилається на створений Interface, і нарешті mutate для таблиці Bridge, що додає новий Port до списку портів мосту. Вся ця послідовність виконується атомарно - або всі операції успішні, або жодна не застосовується.

Протокол включає механізм schema negotiation. При встановленні з'єднання ONOS надсилає get_schema запит, отримуючи повний опис схеми бази даних, включаючи всі таблиці, колонки, типи даних та обмеження. Це дозволяє контролеру адаптуватися до різних версій OVS, які можуть мати відмінності в підтримуваних features.

Monitor updates надсилаються від OVSDB server до ONOS як JSON-RPC notifications з методом update. Кожне повідомлення містить ідентифікатор монітору та set of changes: нові записи (initial для першої передачі повного snapshot, insert для нових записів), модифіковані записи (modify з old та new значеннями змінених колонок), видалені записи (delete). ONOS обробляє ці updates, синхронізуючи своє внутрішнє представлення топології віртуальної мережі.

Для забезпечення надійності протокол підтримує echo method (аналогічно OpenFlow echo) для keepalive перевірок. ONOS періодично надсилає echo запити, і якщо не отримує відповіді, вважає з'єднання втраченим та намагається перепідключитися.

gNMI. gNMI (gRPC Network Management Interface) представляє нове покоління southbound протоколів, що використовує сучасний gRPC фреймворк для високоефективної комунікації з мережевими пристроями. Протокол базується на HTTP/2 для транспорту та Protocol Buffers для серіалізації даних, що забезпечує значно вищу продуктивність порівняно з текстовими протоколами попереднього покоління.

gRPC з'єднання встановлюється через TLS-захищений канал, де ONOS виступає як gNMI client, а мережевий пристрій - як gNMI target. Протокол підтримує двосторонню автентифікацію через certificates та опціонально username/password credentials. Після встановлення з'єднання ONOS може викликати

один з чотирьох основних RPC methods.

Capabilities RPC дозволяє ONOS запитати можливості пристрою через CapabilityRequest. Пристрій відповідає CapabilityResponse, що містить список підтримуваних YANG моделей (name, organization, version), підтримувані encoding formats (JSON, Protobuf, ASCII), та версію gNMI протоколу. Ця інформація використовується для адаптації комунікації до конкретних можливостей пристрою.

Get RPC виконує одноразове читання даних конфігурації або стану. GetRequest містить prefix (загальний префікс для всіх шляхів), path (список YANG шляхів до запитуваних даних), type (CONFIG для конфігураційних даних, STATE для операційного стану, OPERATIONAL для обох), та encoding. Пристрій відповідає GetResponse зі списком notifications, кожна з яких містить timestamp та update записи з path та value для кожного запитуваного елемента даних.

Set RPC виконує атомарні зміни конфігурації. SetRequest може включати множину операцій: delete для видалення конфігураційних елементів за вказаними шляхами, replace для заміни всієї конфігурації за шляхом, update для часткової модифікації. Всі операції в одному запиті виконуються атомарно - або всі успішні, або жодна не застосовується. SetResponse повертає результати для кожної операції та timestamp їх застосування.

Subscribe RPC є найбільш потужною можливістю gNMI, що дозволяє отримувати streaming telemetry. SubscribeRequest встановлює підписку з одним з трьох режимів: ONCE для одноразової передачі поточного snapshot даних, POLL для передачі даних на вимогу за кожним SubscribeRequest, та STREAM для безперервного потоку оновлень. STREAM режим підтримує три sub-modes: TARGET_DEFINED (частота визначається пристроєм), SAMPLE (періодична передача з вказаним sample_interval), ON_CHANGE (передача лише при зміні даних).

Типовий streaming subscription workflow виглядає наступним чином. ONOS надсилає SubscribeRequest зі списком subscription entries, кожна з яких вказує path до даних (наприклад, інтерфейси, BGP peers, QoS counters), mode (ON_CHANGE або SAMPLE), та sample_interval для SAMPLE режиму. Пристрій підтверджує

підписку та починає надсилати потік SubscribeResponse повідомлень. Кожне повідомлення містить update з timestamp, prefix та набором update записів з новими значеннями даних. При ON_CHANGE режимі пристрій надсилає update лише коли дані змінюються, що мінімізує трафік та латентність сповіщень. Архітектуру стеку та процес підписки на телеметрію зображено на рисунку 3.4.

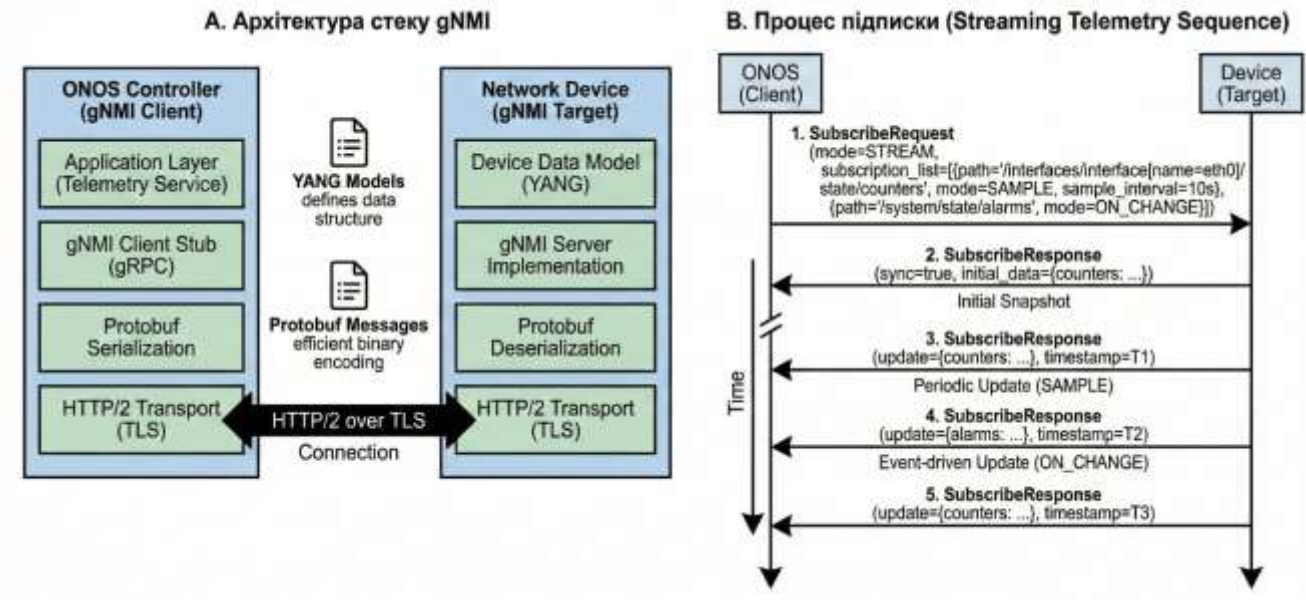

Рис. 3.4. Архітектура протоколу gNMI та процес підписки на потокову телеметрію (Streaming Telemetry)

gNMI підтримує також sophisticated error handling. Кожна відповідь може містити gRPC status codes (OK, INVALID_ARGUMENT, NOT_FOUND, PERMISSION_DENIED) та детальні error messages. Для Set операцій можливе часткове виконання з детальною інформацією про те, які операції успішні, а які завершилися помилкою.

Протокол включає механізми оптимізації bandwidth. Використання Protocol Buffers замість JSON значно зменшує розмір повідомлень. HTTP/2 multiplexing дозволяє паралельну обробку множини запитів через одне з'єднання без head-of-line blocking. gRPC streaming усуває overhead повторних HTTP requests для кожного update.

3.4 Northbound інтерфейс REST API

REST API (Representational State Transfer) виступає фундаментальним протоколом "північного" інтерфейсу (Northbound Interface), який забезпечує

уніфіковану взаємодію зовнішніх додатків, систем оркестрації та веб-інтерфейсів з ядром контролера ONOS.

На відміну від південних протоколів, орієнтованих на низькорівневе керування обладнанням, REST API надає абстрагований рівень доступу до ресурсів мережі, дозволяючи розробникам інтегрувати ONOS з хмарними платформами (наприклад, OpenStack) без необхідності заглиблення у внутрішню Java-архітектуру контролера.

Комунікаційна модель базується на стандартному протоколі HTTP/HTTPS (зазвичай на порту 8181) та використовує формат JSON для обміну даними. Архітектура REST в ONOS реалізована з використанням фреймворку JAX-RS (Jersey), що забезпечує мапінг HTTP-запитів на внутрішні методи сервісів OSGi.

Процес обробки запиту відбувається за чітко визначеною послідовністю. Коли зовнішній клієнт надсилає HTTP-запит, вбудований веб-сервер (Jetty) маршрутизує його до відповідного Web Resource класу. Цей клас діє як шлюз: він десеріалізує вхідні дані, звертається до відповідного сервісу ядра (наприклад, `DeviceService` або `FlowRuleService`) для виконання операції, отримує результат та серіалізує його назад у JSON-формат для відповіді клієнту.

Взаємодія базується на чотирьох основних HTTP методах, які реалізують CRUD-логіку (Create, Read, Update, Delete) над мережевими ресурсами.

Метод **GET** використовується для отримання інформації про стан мережі без внесення змін. Типовий запит на отримання топології ініціює звернення до `TopologyService`, який повертає граф мережі, список комутаторів та лінків. ONOS підтримує фільтрацію вихідних даних безпосередньо в запиті, що дозволяє, наприклад, отримати потоки лише для конкретного пристрою або статистику конкретного порту.

Методи **POST** та **PUT** використовуються для створення та модифікації ресурсів. Найбільш поширеним сценарієм є інсталяція інтентів (Intents). Клієнт формує JSON-об'єкт з описом бажаного з'єднання (точки входу/виходу, вимоги до смуги пропускання) та надсилає його через POST запит. `IntentService` валідує запит, компілює його у набір низькорівневих правил потоків та інсталює їх на

пристрої, повертаючи клієнту унікальний ідентифікатор (Intent ID) та HTTP статус 201 (Created). Процес обробки такого запиту показано на рисунку 3.5.

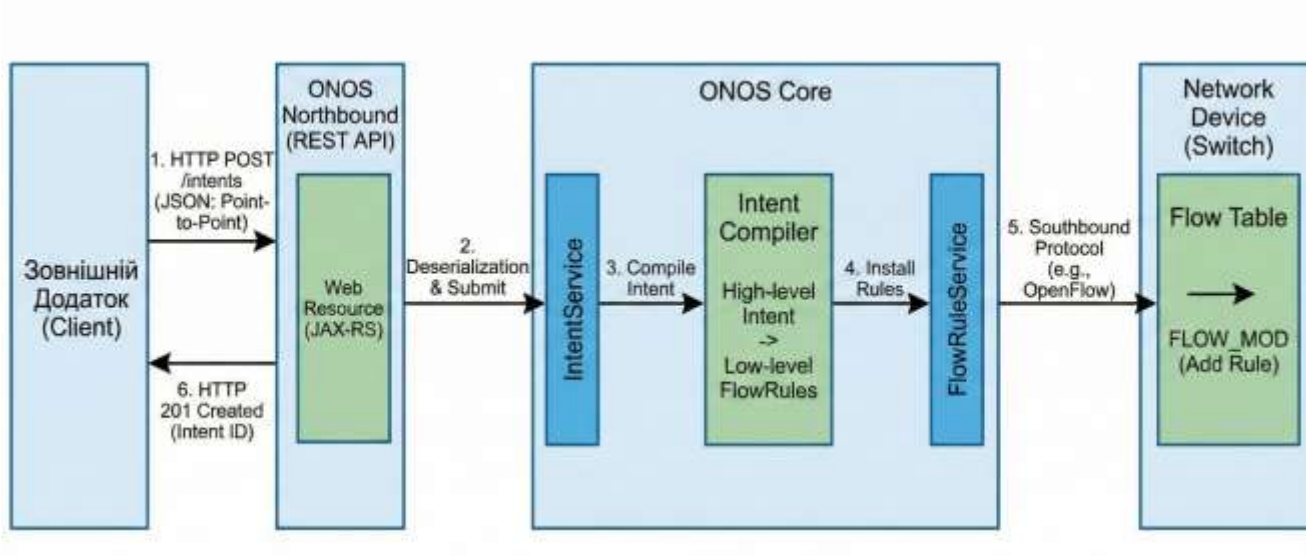

Рис. 3.5. Процес обробки запиту REST API та компіляції інтенту в правила потоків.

Метод **DELETE** забезпечує видалення ресурсів. При отриманні такого запиту із вказаним ідентифікатором ресурсу, контролер виконує процедуру відкликання (withdraw) для інтентів або видалення правил з таблиць комутаторів, гарантуючи очищення ресурсів мережі.

Критичним елементом архітектури REST API в ONOS є система **JSON Codecs**. Оскільки внутрішні об'єкти ONOS є складними Java-класами, система використовує спеціалізовані кодеки для їх трансформації. Кожен тип ресурсу (Device, Host, Link, Flow) має свій власний кодек, який визначає, як саме поля об'єкта мають бути представлені в JSON-структурі. Це забезпечує гнучкість API та стабільність структури даних незалежно від змін у внутрішній реалізації класів.

Для забезпечення безпеки REST API підтримує механізми автентифікації та авторизації. У базовій конфігурації використовується HTTP Basic Auth, проте для продуктових середовищ підтримується інтеграція з токенами доступу та шифрування трафіку через TLS/SSL. Це запобігає несанкціонованому доступу до керування мережею.

Документування API реалізовано через інтеграцію з **Swagger (OpenAPI)**.

ONOS автоматично генерує специфікацію API на основі анотацій у коді, надаючи розробникам інтерактивний веб-інтерфейс для тестування запитів та ознайомлення зі структурою JSON-об'єктів безпосередньо під час роботи контролера.

REST API також підтримує асинхронну модель роботи для довготривалих операцій, повертаючи клієнту статус обробки, що дозволяє уникнути блокування клієнтського додатку під час виконання складних перерахунків маршрутів або масштабних змін конфігурації [3], [9].

Висновки

У результаті аналізу протокольної екосистеми ONOS було визначено, що ефективність функціонування контролера базується на гібридній моделі узгодженості даних. Система застосовує диференційований підхід: для критичних операцій (зміна конфігурації, призначення майстрів) використовується протокол суворого консенсусу Raft, що гарантує атомарність змін, тоді як для високошвидкісного обміну метриками застосовується протокол Gossip, який забезпечує масштабованість через механізм поступової узгодженості. Такий розподіл є архітектурною перевагою для розроблюваного проекту, оскільки дозволяє уникнути "вузьких місць" у продуктивності при масштабуванні мережі.

Важливим аспектом надійності системи є реалізація адаптивного детектора відмов на основі алгоритму φ-accrual. На відміну від традиційних фіксованих таймаутів, цей механізм динамічно адаптується до умов мережі, знижуючи ймовірність хибних спрацьовувань. Разом із quorum-based захистом від сценаріїв розщеплення мережі (split-brain), це створює надійний фундамент для побудови відмовостійких рішень класу Carrier Grade.

Аналіз Southbound-інтерфейсів показав, що для задач Traffic Engineering (TE) критично важливим є використання протоколу gNMI. На відміну від класичного OpenFlow, який фокусується на правилах маршрутизації, gNMI забезпечує потокову передачу телеметрії (streaming telemetry) в реальному часі через режим Subscribe. Це дозволяє додатку TE отримувати дані про завантаження каналів миттєво, без затримок, властивих періодичному опитуванню, що є необхідною умовою для динамічного балансування навантаження.

Також встановлено, що Northbound інтерфейс на базі REST API надає достатній рівень абстракції для інтеграції розроблюваного додатка. Можливість декларативного управління інтентами через стандартні HTTP-запити спрощує взаємодію із зовнішніми системами та дозволяє зосередитися на реалізації логіки оптимізації трафіку, а не на низькорівневих деталях комунікації.

Отже, досліджені протоколи взаємодії формують повний технологічний стек, необхідний для реалізації стартап-проекту: gNMI забезпечує надходження даних для аналітики, Raft гарантує узгодженість рішень кластера, а REST API дозволяє зручно керувати політиками мережі.

РОЗДІЛ 4

МЕТОДИ TRAFFIC ENGINEERING

4.1 Концепція Traffic Engineering в SDN-мережах

Traffic Engineering (TE) являє собою комплексний підхід до оптимізації використання мережевих ресурсів через інтелектуальне управління потоками трафіку [4]. В контексті програмно-визначених мереж на базі ONOS, Traffic Engineering набуває нових можливостей завдяки централізованому управлінню та глобальному огляду мережевої топології (Global Network View).

На відміну від традиційних мереж, де кожен маршрутизатор приймає рішення локально, SDN-контролер має повне уявлення про стан всієї мережі через Topology Service, що дозволяє приймати глобально оптимальні рішення щодо розподілу трафіку.

Основною метою Traffic Engineering є максимізація ефективності використання мережевих ресурсів при одночасному забезпеченні якості обслуговування. Це включає запобігання перевантаженню каналів, мінімізацію затримок для критичних додатків та забезпечення гарантованої пропускної здатності.

В ONOS ці завдання вирішуються через взаємодію базових сервісів ядра та спеціалізованих додатків. Архітектура Traffic Engineering базується на трьох складових: моніторингу топології та ресурсів; механізмах обчислення шляхів (Path Computation); та системі застосування рішень через Flow Rules та Groups [4].

4.2 Алгоритми обчислення шляхів

Обчислення оптимальних шляхів є центральною задачею Traffic Engineering, оскільки вибір маршруту безпосередньо впливає на продуктивність мережі, балансування навантаження та відмовостійкість з'єднань. ONOS надає інфраструктуру для обчислення маршрутів через PathService та TopologyService, використовуючи множину алгоритмів різної складності та призначення, кожен з яких оптимізований для специфічних сценаріїв управління трафіком. Базовий алгоритм Дейкстри забезпечує пошук найкоротших шляхів у графі мережі з гарантією оптимальності за заданою метрикою, модифікована версія дозволяє

виявляти всі рівноцінні альтернативи однакової мінімальної довжини для рівномірного розподілу навантаження, тоді як алгоритм Єна знаходить k шляхів різної довжини у порядку зростання для багаторівневої відмовостійкості. Вибір конкретного алгоритму залежить від вимог до балансування навантаження, гарантій якості обслуговування та стратегій резервування шляхів при відмовах мережевих компонентів [6].

Базовий алгоритм Дейкстри. Алгоритм Дейкстри використовується як базовий метод для знаходження найкоротших шляхів у графі мережі. ONOS реалізує оптимізовану версію цього алгоритму, що враховує вагу ребер графа (метрики), яка може представляти кількість переходів (hop-count) або затримку (latency).

Класичний алгоритм Дейкстри працює на основі жадібної стратегії, поступово розширюючи множину вершин з відомою мінімальною відстанню від джерела. Для кожної вершини v графа G = (V, E, w) алгоритм зберігає поточну оцінку найкоротшої відстані dist[v] та попередника prev[v] на оптимальному шляху. Початково всі відстані встановлюються рівними нескінченності, окрім вершини-джерела s, для якої

$$dist[s] = 0. \quad (4.1)$$

На кожній ітерації алгоритм вибирає вершину u з мінімальною поточною відстанню серед невідвіданих вершин та "розслаблює" (relaxes) всі її вихідні ребра. Розслаблення ребра (u, v) означає перевірку умови: чи буде шлях через u коротшим за поточний відомий шлях до v. Математично це виражається як:

$$alt = dist[u] + w(u, v). \quad (4.2)$$

Якщо alt < dist[v], то знайдено коротший шлях, і dist[v] оновлюється значенням alt, а prev[v] встановлюється рівним u. Процес продовжується доти, доки не буде оброблено всі досяжні вершини або не буде знайдено шлях до цільової вершини.

Часова складність класичного алгоритму Дейкстри залежить від реалізації черги з пріоритетом. При використанні бінарної купи складність становить O(|E| +

|V| log |V|), де |E| — кількість ребер, |V| — кількість вершин. У щільних графах, типових для мережевих топологій дата-центрів, де $|E| \approx |V|^2$, ця складність близька до $O(|V|^2 \log |V|)$.

Модифікований алгоритм Дейкстри для знаходження множинних шляхів. В ONOS реалізовано модифіковану версію алгоритму Дейкстри, яка суттєво відрізняється від класичної у способі зберігання інформації про попередників. Замість одного попередника prev[v] для кожної вершини зберігається **множина попередників** predecessors[v]. Це дозволяє виявляти всі можливі найкоротші шляхи однакової мінімальної довжини.

Ключова відмінність виникає на етапі розслаблення ребер. Якщо знайдено шлях через вершину u до вершини v, що має довжину

$$alt = dist[u] + w(u, v), \quad (4.3)$$

можливі три сценарії:

1. alt < dist[v] — знайдено строго коротший шлях. У цьому випадку dist[v] оновлюється, а множина попередників повністю замінюється:

$$predecessors[v] = \{u\}. \quad (4.4)$$

Всі раніше знайдені шляхи до v виявились довшими і стають неактуальними.

2. alt = dist[v] — знайдено альтернативний шлях тієї ж довжини. Вершина u додається до існуючої множини попередників:

$$predecessors[v] = predecessors[v] \cup \{u\}. \quad (4.5)$$

Це критичний момент, що відрізняє модифікований алгоритм від класичного.

3. alt > dist[v] — знайдений шлях довший за вже відомий мінімум, тому ніяких дій не виконується.

Після завершення фази розслаблення всіх ребер структура predecessors[] містить повну інформацію про всі найкоротші шляхи. Для кожної вершини v множина predecessors[v] включає всі вершини, через які проходить хоча б один оптимальний маршрут до v.

Для перетворення структури попередників у конкретні послідовності вершин

використовується модифікований пошук у глибину (DFS). Алгоритм починає з цільової вершини d та рекурсивно рухається назад до джерела s, досліджуючи всі можливі комбінації попередників.

На кожному рівні рекурсії для поточної вершини vertex алгоритм перевіряє, чи досягнуто джерела. Якщо vertex = s, то поточний шлях currentPath (що будувався у зворотному порядку) реверсується та додається до множини результатів allPaths. Якщо vertex ≠ s, алгоритм ітерується по всіх елементах predecessors[vertex], додаючи кожного попередника pred на початок currentPath та рекурсивно викликаючи себе для pred.

Важливим обмеженням є параметр maxPaths, що визначає максимальну кількість шляхів, які повертає система. Це критично для запобігання переповненню пам'яті у випадку дуже зв'язних графів, де кількість найкоротших шляхів може зростати експоненційно. Наприклад, у повному графі K_n кількість найкоротших шляхів (довжини 1) між будь-якою парою вершин дорівнює 1, але в топології fat-tree між leaf-комутаторами різних pod'ів може існувати k^2 рівноцінних шляхів, де k — кількість spine-комутаторів.

Просторова складність модифікованого алгоритму становить $O(|V| \cdot avg\_pred)$, де avg_pred — середня кількість попередників на вершину. У найгіршому випадку це може досягати $O(|V|^2)$ для повністю зв'язних підграфів. Часова складність відновлення шляхів залежить від структури графа та дорівнює $O(\min(paths\_count, maxPaths) \cdot L)$, де paths_count — загальна кількість існуючих найкоротших шляхів, L — довжина шляху.

Критична властивість модифікованого алгоритму полягає в тому, що всі повернуті шляхи мають строго однакову довжину — мінімальну можливу. Це не k-shortest paths різної довжини, а множина рівноцінних альтернатив за метрикою ваги ребер.

Алгоритм Єна для знаходження k-найкоротших шляхів. Для забезпечення відмовостійкості та балансування, коли потрібні не тільки шляхи однакової мінімальної довжини, але й альтернативи різної довжини, використовується алгоритм Єна (Yen's algorithm). ONOS підтримує реалізацію

цього алгоритму через клас KShortestPathsSearch для знаходження k найкоротших шляхів у порядку зростання їх довжини.

Алгоритм Єна працює ітеративно, знаходячи шляхи один за одним. На першій ітерації використовується класичний алгоритм Дейкстри для знаходження найкоротшого шляху π_1 між вершинами s та d. Цей шлях зберігається як перший елемент результуючого масиву

$$K[0] = \pi_1. \quad (4.6)$$

Для знаходження кожного наступного i-го шляху ($i \geq 1$) алгоритм виконує складну процедуру з тимчасовою модифікацією графа. Ідея полягає в систематичному "блокуванні" ребер вже знайдених шляхів для пошуку нових альтернатив. Процес включає ітерацію по кожній вершині v_j попереднього шляху K[i-1].

Для кожної такої вершини v_j визначається корінь шляху (root path) — частина шляху від джерела до v_j:

$$rootPath = K[i-1][0 : j]. \quad (4.7)$$

Потім алгоритм видаляє з графа всі ребра, що виходять з v_j і належать шляхам, які мають той самий корінь. Математично це означає: для кожного раніше знайденого шляху $p \in K$, якщо $p[0 : j] = rootPath$, то ребро (v_j, v_{j+1}) тимчасово видаляється з графа.

Після видалення ребер виконується пошук найкоротшого шляху від v_j до d у модифікованому графі (spurPath). Якщо такий шлях існує, формується кандидат на i-й найкоротший шлях:

$$totalPath = rootPath + spurPath. \quad (4.8)$$

Цей кандидат додається до множини потенційних шляхів. Після обробки поточної вершини v_j всі видалені ребра відновлюються у графі, і процес переходить до наступної вершини.

Коли оброблено всі вершини попереднього шляху, з множини кандидатів вибирається шлях з мінімальною довжиною — це і буде K[i]. Якщо множина

кандидатів порожня, алгоритм завершується, оскільки більше альтернативних шляхів не існує.

Ключова відмінність алгоритму Єна від модифікованого Дейкстри полягає в тому, що він знаходить шляхи різної довжини у порядку зростання:

$$L(\pi_1) \leq L(\pi_2) \leq ... \leq L(\pi_k), \quad (4.9)$$

де нерівності можуть бути строгими. Наприклад, π_1 може мати довжину 3, π_2 та π_3 — довжину 4, а π_4 — довжину 5.

Часова складність алгоритму Єна становить $O(k \cdot |V| \cdot (|E| + |V| \log |V|))$, оскільки для кожного з k шляхів виконується до |V| запусків алгоритму Дейкстри (по одному для кожної вершини попереднього шляху). Це робить алгоритм Єна значно більш ресурсоємним порівняно з модифікованим Дейкстрою, особливо для великих значень k та складних топологій [6].

Просторова складність визначається необхідністю зберігати k повних шляхів та множину кандидатів, що у найгіршому випадку може містити $O(k \cdot |V|)$ елементів. Додатково потрібна пам'ять для тимчасового зберігання інформації про видалені ребра на кожній ітерації.

Практичне застосування алгоритмів в ONOS. Важливо розуміти, що у базовій конфігурації ONOS використовує модифікований алгоритм Дейкстри (DijkstraGraphSearch), а не алгоритм Єна. Це означає, що за замовчуванням TopologyService.getPaths() повертає всі найкоротші шляхи однакової мінімальної довжини, що оптимально для більшості сценаріїв Traffic Engineering, де потрібне рівномірне балансування між рівноцінними альтернативами.

Алгоритм Єна через KShortestPathsSearch потребує явної активації і використовується у специфічних випадках, коли необхідно мати заздалегідь підготовлені резервні шляхи різної довжини. Це критично важливо для реалізації багаторівневих схем відмовостійкості, коли після відмови основного шляху система послідовно намагається використати другий, третій і подальші варіанти, навіть якщо вони довші за оптимум.

Для пошуку шляхів з обмеженнями (Constrained Shortest Path First)

використовуються розширення базових алгоритмів. Система може шукати маршрут, який задовольняє вимогу щодо мінімальної доступної пропускної здатності (Bandwidth Constraint), використовуючи дані про поточні алокації ресурсів. У цьому випадку вагова функція w(e) динамічно змінюється або ребра з недостатньою пропускною здатністю виключаються з графа перед запуском алгоритму пошуку шляху [7].

Крім того, підтримується пошук непересічних шляхів (disjoint paths) через алгоритм Suurballe, що критично важливо для TE, оскільки дозволяє системі мати готові альтернативні маршрути для миттєвого перемикання у разі аварій та забезпечувати багаторівневу відмовостійкість з різними компромісами між довжиною шляху та доступністю з'єднання.

4.3 Механізми управління трафіком

Управління трафіком в ONOS реалізується через комплекс взаємопов'язаних механізмів, що забезпечують оптимальний розподіл навантаження, гарантії якості обслуговування та запобігання перевантаженням мережі. На відміну від статичних конфігурацій традиційних мереж, де правила маршрутизації змінюються рідко, програмно-визначена архітектура ONOS дозволяє динамічно адаптувати стратегії управління трафіком у відповідь на зміни навантаження, відмови обладнання або нові вимоги додатків. Балансування навантаження використовує групи OpenFlow для розподілу потоків між альтернативними шляхами без участі контролера в обробці кожного пакету, резервування ресурсів через ResourceService гарантує пропускну здатність для критичних додатків шляхом контролю допуску нових з'єднань, а реактивні механізми виявлення перевантажень дозволяють перемаршрутизовувати трафік в обхід проблемних ділянок мережі. Інтеграція цих механізмів з Intent Framework надає високорівневий декларативний інтерфейс для формулювання політик Traffic Engineering, приховуючи складність низькорівневого програмування flow rules та координації розподілених операцій.

Балансування навантаження. Балансування навантаження в середовищі ONOS реалізується переважно з використанням груп OpenFlow (Group Tables типу SELECT). Це дозволяє розподіляти трафік між декількома шляхами без участі

контролера в обробці кожного окремого пакету.

Equal-Cost Multi-Path (ECMP) є базовою технікою, яка розподіляє трафік між кількома шляхами з однаковою вартістю. ONOS виявляє наявність рівноцінних шляхів та налаштовує групи на комутаторах, де саме мережевий пристрій виконує хешування заголовків пакетів для вибору вихідного порту. Це забезпечує збереження порядку пакетів у потоці (flow consistency).

Для нерівномірного розподілу трафіку (Weighted load balancing) використовуються групи з "ваговими коефіцієнтами" (buckets with weights). ONOS дозволяє налаштовувати групи так, щоб, наприклад, 70% нових потоків направлялися на високошвидкісний канал, а 30% — на резервний.

Динамічне (Adaptive) балансування реалізується на рівні додатків, які періодично опитують статистику портів (PortStatistics) та при виявленні дисбалансу змінюють конфігурацію груп або перемаршрутизовують окремі потоки.

Резервування ресурсів та гарантії QoS. Резервування ресурсів в ONOS забезпечується сервісом ResourceService, який веде облік використаної та доступної пропускної здатності на кожному лінку.

Основним механізмом є Bandwidth Allocation. Додатки можуть запитувати гарантовану смугу для потоку. Якщо на всьому шляху є достатньо ресурсів, ONOS "бронює" їх у своїй базі даних і встановлює відповідні правила. Це дозволяє реалізувати механізм контролю допуску (Admission Control): якщо ресурсів недостатньо, новий інтент не буде встановлено, захищаючи якість обслуговування існуючих потоків.

Забезпечення пріоритетів (Priority-based QoS) реалізується через маніпуляцію полями пакетів та використання OpenFlow Meters. ONOS може встановлювати DSCP мітки для класифікації трафіку. Безпосереднє управління чергами (Queuing) зазвичай виконується на рівні конфігурації пристроїв (через OVSDB або NETCONF), тоді як OpenFlow-правила від ONOS направляють трафік у відповідні черги згідно з політиками.

Такий підхід дозволяє відокремити критичний трафік від фонового та застосувати до нього політики Rate Limiting через Meters для запобігання впливу

агресивних потоків на мережу.

Управління перевантаженнями (Congestion Management). Управління перевантаженнями в ONOS базується на реактивній моделі та активному моніторингу. Система збирає статистику лічильників портів та потоків в реальному часі для оцінки завантаженості каналів.

Congestion detection реалізується шляхом аналізу метрик використання пропускної здатності (link utilization). Додаток моніторингу може бути налаштований на спрацювання тригерів при перевищенні заданих порогів (thresholds).

Основним методом реагування є Traffic Rerouting. При виявленні перевантаженого лінка, TE-додаток може ініціювати перерахунок маршрутів для частини інтентів29. ONOS використовує механізм Intent Framework для перевстановлення шляхів: система обчислює новий маршрут в обхід проблемної ділянки та атомарно оновлює правила потоків, мінімізуючи втрати пакетів.

Також застосовується Traffic Policing за допомогою OpenFlow Meters, які дозволяють відкидати або перемарковувати пакети, що перевищують встановлений ліміт швидкості, захищаючи мережу від перенасичення.

Мультишляхова маршрутизація. Мультишляхова маршрутизація дозволяє агрегувати пропускну здатність паралельних каналів. ONOS підтримує це через абстракцію груп.

Реалізація ECMP в ONOS автоматизує процес створення груп для розгалуження трафіку33. Це особливо ефективно в топологіях типу Leaf-Spine або Fat-Tree.

Важливою особливістю є підтримка Resilient Hashing (стійкого хешування) на рівні сумісних комутаторів. Це дозволяє при зміні кількості активних лінків (наприклад, при падінні одного з портів) перерозподіляти лише ті потоки, що йшли через проблемний шлях, залишаючи інші потоки без змін, що значно зменшує вплив на мережу порівняно з традиційним перерахунком хешу [7].

Інтеграція Traffic Engineering з Intent Framework. Intent Framework в ONOS є високорівневим інструментом для реалізації політик TE. Він дозволяє

оператору або додатку формулювати "намір" (що потрібно з'єднати і як), приховуючи складність низькорівневої конфігурації.

Bandwidth Intent дозволяє явно вказати необхідну швидкість каналу. При компіляції такого інтенту ONOS автоматично звертається до Resource Service для перевірки та резервування ресурсів.

Protected Transport Intent (або Protection Intent) автоматизує створення відмовостійких з'єднань. Компілятор інтентів обчислює два непересічні шляхи (primary та backup) і встановлює відповідні групи типу FAST_FAILOVER на комутаторах39. Це забезпечує апаратне перемикання на резервний шлях за мілісекунди без участі контролера при втраті сигналу на порту.

Multi-constraint Intent дозволяє комбінувати вимоги, наприклад, запитувати шлях з пропускною здатністю не менше X та затримкою не більше Y, уникаючи при цьому транзиту через певний набір вузлів.

4.4 Реактивне переадресування в ONOS

Додаток Simple Reactive Forwarding (fwd) демонструє практичну реалізацію принципів Traffic Engineering через реактивний підхід до встановлення правил переадресування, який принципово відрізняється від проактивних методів попереднього конфігурування мережі. Замість встановлення всіх можливих правил заздалегідь, що призводить до надмірного використання обмеженої пам'яті таблиць потоків комутаторів, реактивне переадресування створює flow rules динамічно лише для активних потоків у відповідь на надходження пакетів до контролера. Цей підхід забезпечує ефективне використання ресурсів комутаторів та природну адаптацію до змінюваних патернів трафіку через механізми автоматичного видалення неактивних правил за таймаутом, водночас вимагаючи ретельної оптимізації для мінімізації затримок, пов'язаних з обробкою перших пакетів нових потоків контролером. Аналіз архітектури та алгоритмів додатку fwd дозволяє зрозуміти компроміси між реактивними та проактивними стратегіями управління, механізми взаємодії з сервісами топології та хостів, а також методи забезпечення відмовостійкості через автоматичне виявлення та усунення застарілих правил при змінах мережевої конфігурації.

Архітектура та принципи роботи. Архітектура додатку базується на компоненті ReactivePacketProcessor, який обробляє packet-in повідомлення від комутаторів. Кожен вхідний пакет проходить багатоетапну фільтрацію: відкидаються контрольні протоколи (LLDP, BDDP), multicast трафік (за налаштуванням), та пакети, що вже оброблені іншими компонентами. Після фільтрації система намагається визначити місцеположення хоста призначення через HostService.

Якщо хост невідомий, виконується інтелектуальний flooding: пакет розповсюджується тільки якщо поточна точка підключення визначена топологічним сервісом як broadcast point, що запобігає зайвим копіям та петлям у мережі. Для відомих хостів додаток обчислює найкоротший шлях через TopologyService, отримуючи множину рівноцінних маршрутів однакової мінімальної довжини.

Механізм обчислення та вибору маршруту. Процес вибору маршруту в додатку fwd реалізовано через виклик методу topologyService.getPaths(), який за замовчуванням використовує модифікований алгоритм Дейкстри (DijkstraGraphSearch). Цей метод повертає всі найкоротші шляхи строго однакової мінімальної довжини, а не k шляхів різної довжини.

Робота алгоритму відбувається у два етапи. На першому етапі виконується модифікований алгоритм Дейкстри, який відрізняється від класичної реалізації тим, що для кожної вершини графа зберігається не один попередник, а множина попередників. Це досягається через перевірку умови (4.3): якщо альтернативний шлях до вершини має таку саму довжину, як і вже знайдений мінімум, вершина-джерело цього шляху додається до множини попередників згідно з формулою (4.5). Таким чином, після завершення алгоритму система має інформацію про всі можливі найкоротші маршрути.

На другому етапі використовується пошук у глибину для відновлення конкретних шляхів із збережених множин попередників. Алгоритм рекурсивно проходить від вершини призначення до вершини джерела, формуючи всі можливі комбінації. Кількість повернутих шляхів обмежується параметром maxPaths для

запобігання переповненню пам'яті у випадку дуже зв'язних графів.

Критично важливою особливістю є те, що базова конфігурація ONOS використовує одиничні ваги для всіх ребер графа

$$w(e) = 1, \forall e \in E, \quad (4.10)$$

тому метрикою оптимальності виступає кількість переходів (hop count), а не пропускна здатність чи затримка. Це означає, що шлях через три гігабітні лінки та шлях через три перевантажені канали по 100 Мбіт/с будуть розглядатись як рівноцінні, якщо обидва мають довжину 3 переходи.

Альтернативний підхід реалізований через алгоритм Єна (KShortestPathsSearch), який здатен знаходити k найкоротших шляхів різної довжини згідно з формулою (4.9). Алгоритм Єна працює ітеративно: спочатку знаходиться найкоротший шлях класичним Дейкстрою, потім для кожної вершини цього шляху алгоритм тимчасово видаляє ребра, що використовувались у вже знайдених маршрутах, та обчислює новий найкоротший шлях від цієї вершини до призначення згідно з формулами (4.7)-(4.8). Це дозволяє знайти другий, третій і наступні за довжиною маршрути. Однак алгоритм Єна не використовується за замовчуванням в додатку fwd і потребує явної активації через налаштування DefaultTopology.

Встановлення правил переадресування. Встановлення правил відбувається інкрементно: кожен packet-in призводить до створення flow rule лише на одному комутаторі — тому, з якого надійшов пакет. Це означає, що повний шлях від джерела до призначення формується поступово, у міру проходження першого пакету через кожен проміжний вузол. Додаток підтримує гнучку конфігурацію селекторів правил: від простого matching за MAC-адресою призначення до детального узгодження полів L3/L4 (IP-адреси, TCP/UDP порти, ICMP параметри, DSCP мітки).

Критичною особливістю є те, що ARP пакети ніколи не призводять до встановлення flow rules — вони завжди обробляються через packet-out, генеруючи додаткове навантаження на контролер. Встановлені правила мають обмежений час

життя (за замовчуванням 10 секунд), після чого автоматично видаляються, що дозволяє адаптуватися до змін у патернах трафіку.

Механізм виявлення та усунення "чорних дір". Додаток включає механізм виявлення та усунення "чорних дір" через компонент InternalTopologyListener. При відмові з'єднання система ідентифікує всі flow rules, що використовували видалене ребро графа, та виконує backtracking по альтернативних шляхах від точки відмови до джерела. Алгоритм послідовно видаляє застарілі правила на кожному проміжному комутаторі, зупиняючись при виявленні альтернативного маршруту. Нові правила встановлюються автоматично при наступних спробах передачі даних.

Система збору метрик та аналіз ефективності. Для аналізу продуктивності додаток надає систему збору метрик на основі EventuallyConsistentMap, що зберігає статистику для кожної унікальної MAC-адреси джерела. Ключові показники включають кількість packet-in подій, встановлених правил, відкинутих пакетів та packet-out повідомлень.

Коефіцієнт встановлення правил визначається як

RIR = forwardedPacket / inPacket, (4.12)

де forwardedPacket — кількість встановлених flow rules, inPacket — загальна кількість пакетів, що надійшли до контролера. Для ідеального реактивного переадресування RIR ≈ 1 для перших пакетів нових потоків та RIR ≈ 0 після встановлення правил, коли подальші пакети обробляються безпосередньо комутаторами.

Коефіцієнт відкидання пакетів

DR = droppedPacket / inPacket (4.13)

дозволяє оцінити частку відфільтрованого трафіку. Високий DR може вказувати на велику кількість контрольного трафіку (LLDP, BDDP), multicast трафік при увімкненому ignoreIpv4McastPackets, або проблеми з конфігурацією IPv6.

Аналіз співвідношень між цими метриками дозволяє оцінити ефективність реактивного переадресування та виявити проблеми конфігурації.

Висновки

У ході дослідження методів інженерії трафіку (Traffic Engineering) в середовищі ONOS було встановлено, що платформа реалізує комплексний підхід до оптимізації мережевих ресурсів, який базується на глобальному баченні топології. Ключовим елементом архітектури є підсистема обчислення шляхів, яка за замовчуванням використовує модифікований алгоритм Дейкстри. Його особливістю є здатність знаходити всі рівноцінні найкоротші шляхи однакової довжини шляхом збереження множини попередників для кожної вершини. Це робить його ідеальним інструментом для реалізації стратегій ECMP, забезпечуючи рівномірне балансування навантаження без додаткових обчислювальних витрат.

Водночас для задач забезпечення відмовостійкості та багаторівневого резервування критично важливим є використання алгоритму Єна, який дозволяє знаходити k найкоротших шляхів у порядку зростання їх. Хоча цей алгоритм є більш ресурсоємнимі потребує явної активації в ONOS, саме він надає необхідну базу для створення сервісів із гарантованим перемиканням на резервні маршрути при деградації основного каналу.

Аналіз механізмів управління трафіком показав ефективність дворівневої моделі: на нижньому рівні використовуються групи OpenFlow для апаратного розподілу потоків та Meters для обмеження швидкості , а на верхньому — Intent Framework для декларативного опису політик. Така абстракція дозволяє реалізовувати складні сценарії, такі як Bandwidth Intent для гарантування смуги пропускання з використанням механізму Admission Control, що запобігає перевантаженню мережі новими запитами.

Розгляд реактивного підходу на прикладі додатку fwd виявив його переваги в гнучкості та економії ресурсів таблиць потоків, оскільки правила встановлюються лише для активних з'єднань.

РОЗДІЛ 5

ПОБУДОВА СЕГМЕНТУ SDN-МЕРЕЖІ НА БАЗІ КОНТРОЛЕРУ ONOS

5.1 Логічна схема та основні складові віртуального макету ONOS мережі

Для моделювання та дослідження принципів роботи програмно-конфігурованих мереж розроблено віртуальний макет, архітектура якого представлена на рис. 5.1. Схема відображає ієрархічну структуру взаємодії компонентів мережі під управлінням контролера ONOS [2].

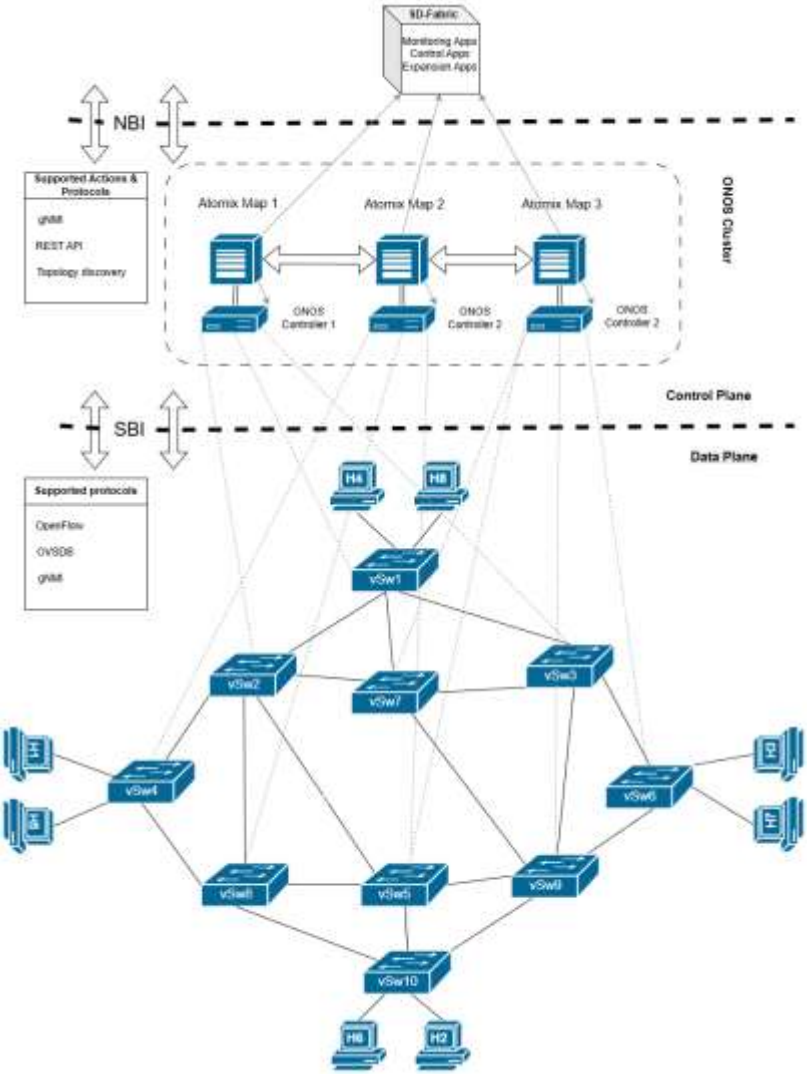

Рис. 5.1 Схема віртуального макету для моделювання елементів ONOS мережі

Архітектура спроєктованої мережі складається з наступних функціональних рівнів та компонентів:

SD-Fabric (Верхній рівень):

Загальний програмний шар, до складу якого входять додатки для моніторингу, управління та розширення функціоналу мережі. Взаємодія з кластером ONOS здійснюється через північний інтерфейс (Northbound Interface, NBI).

Northbound Interface (NBI):

Інтерфейс, що забезпечує комунікацію між бізнес-додатками SD-Fabric та рівнем управління (ONOS-кластером). Підтримує протоколи gNMI (gRPC Network Management Interface), REST API та механізми виявлення топології (Topology discovery).

Рівень управління (Control Plane):

Центральна частина схеми, що представлена ONOS-кластером.

Елементи кластера: Три контролери ONOS (Controller 1, 2, 3), які забезпечують відмовостійкість системи та балансування навантаження.

Atomix Map: Розподілене сховище даних, що забезпечує механізм синхронізації станів між контролерами, гарантуючи доступ кожного вузла управління до актуальної інформації про мережу.

Southbound Interface (SBI):

Інтерфейс, що забезпечує взаємодію між рівнем управління (ONOS) та рівнем передачі даних (Data Plane). Для комунікації використовуються протоколи OpenFlow, OVSDB (Open vSwitch Database Management Protocol) та gNMI.

Рівень передачі даних (Data Plane):

Нижній рівень схеми, який відповідає за безпосередню пересилку пакетів.

Комутатори (vSw): Віртуальні пристрої (vSw1–vSw9, vSw10), що моделюють роботу мережевого обладнання. Вони з'єднані між собою каналами передачі даних (зелені лінії) та мають канали управління з контролерами (сірі лінії).

Кінцеві вузли (H1–H8): Віртуальні хости, що емулюють користувацькі пристрої або сервери. Їхньою функцією є генерація трафіку та взаємодія в межах мережі.

Запропонований макет розгортається у віртуальному середовищі (наприклад, Mininet) і дозволяє здійснювати верифікацію сценаріїв SDN, зокрема аналіз

відмовостійкості та масштабованості кластера ONOS [2], [12].

5.2 Порядок налаштування та розгортання ONOS Controller

Процес розгортання віртуального стенда розпочинається зі встановлення середовища віртуалізації. Для цього використовується програмне забезпечення Oracle VirtualBox, яке необхідно завантажити з офіційного ресурсу (рис. 5.2).

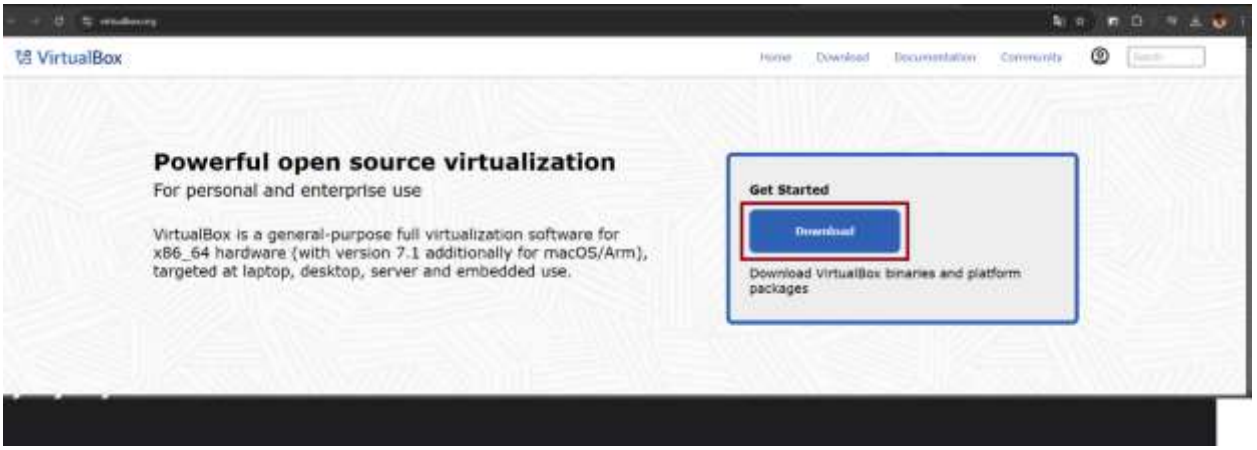

Рис. 5.2 Завантаження VirtualBox

В якості платформи управління SDN-мережею обрано контролер версії onos-tutorial-1.15.0 [11]. Даний образ містить попередньо налаштоване середовище для відображення топології та проведення тестувань. Взаємодія з контролером реалізується через веб-інтерфейс (GUI) [10].

Після завантаження образу віртуальної машини (формат .ova) виконується його імпорт у середовище VirtualBox (рис. 5.3).

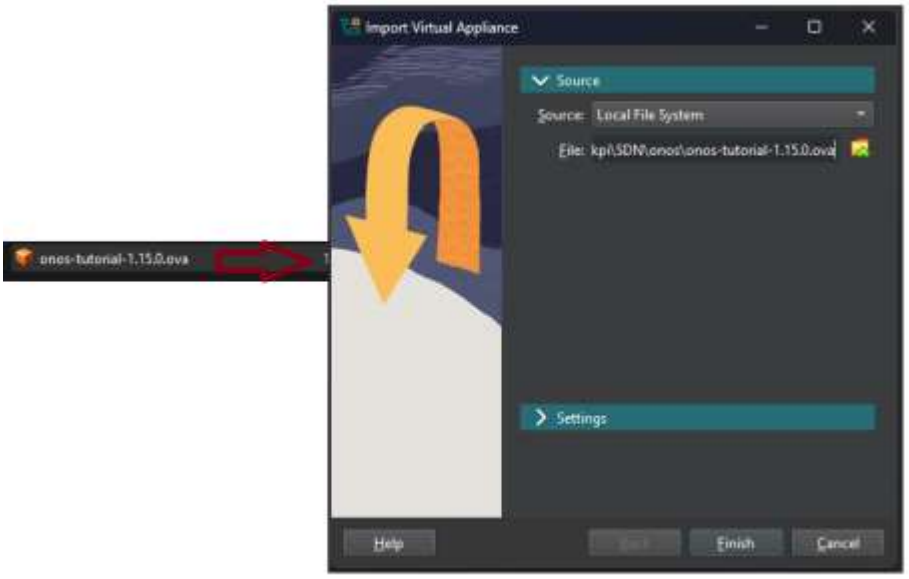

Рис. 5.3 Процес імпорту віртуального образу

На етапі конфігурації рекомендується оптимізувати виділення апаратних ресурсів (CPU та RAM) відповідно до можливостей хост-системи для забезпечення стабільної роботи кластера (рис. 5.4).

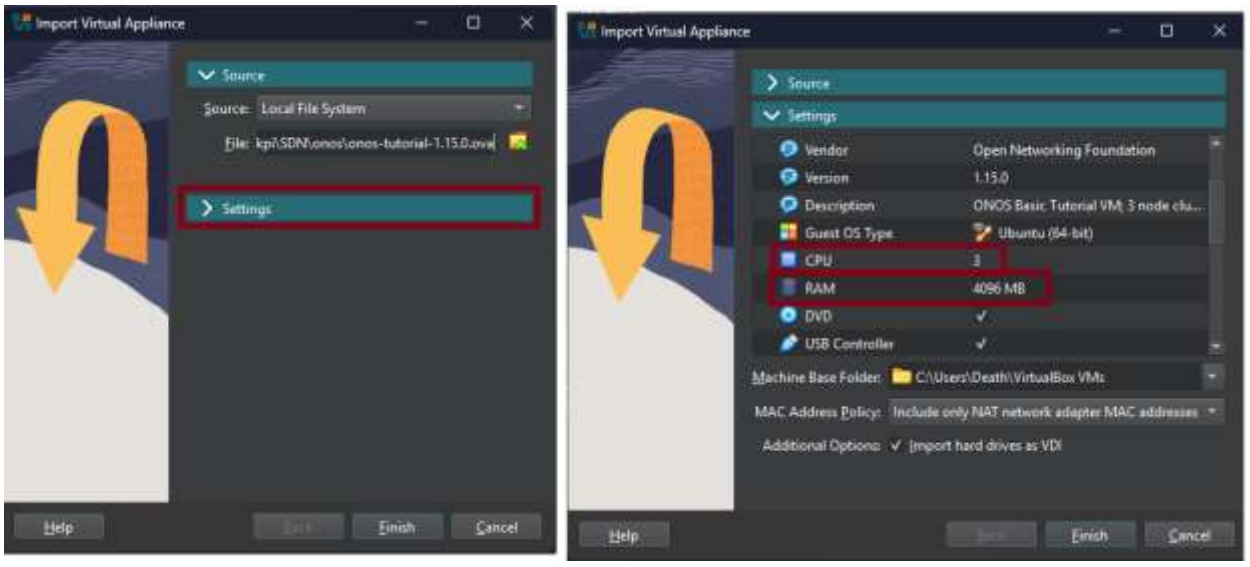

Рис. 5.4 Конфігурація ресурсів імпортованого образу ONOS

Альтернативний метод імпорту передбачає використання меню «File» -> «Import Appliance» із вибором відповідного файлу образу (рис. 5.5).

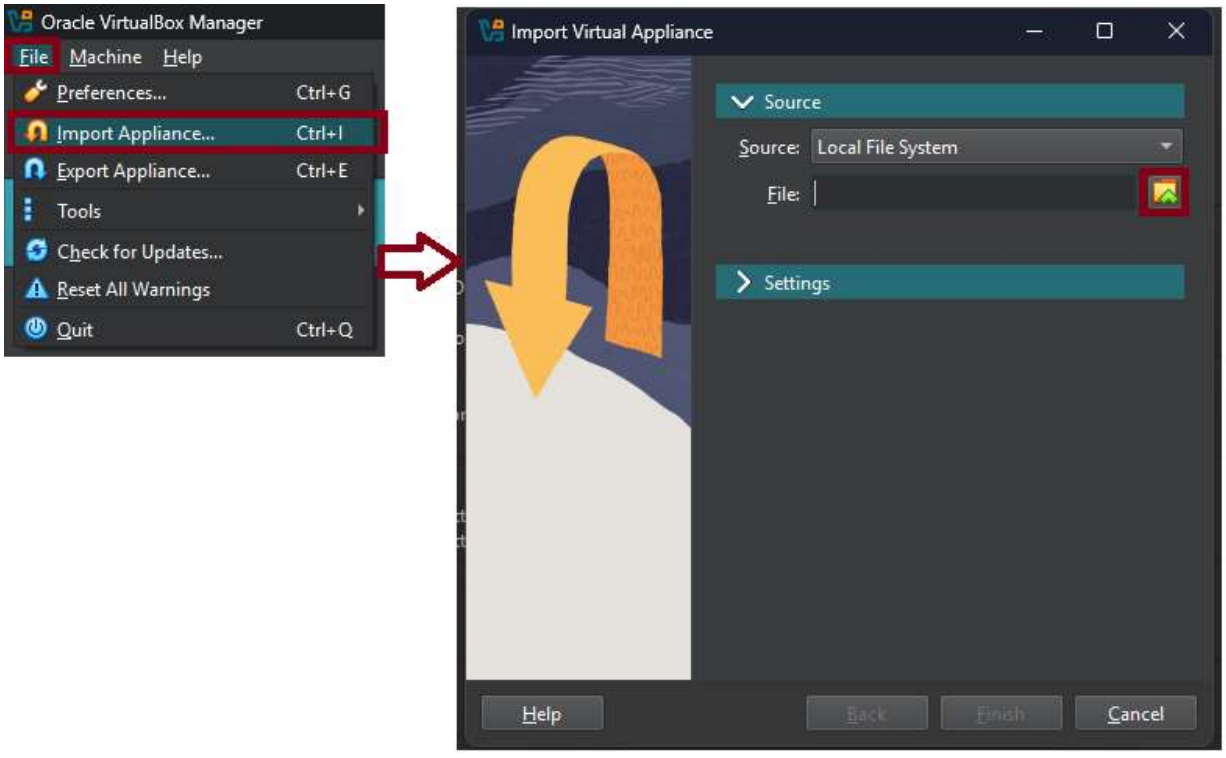

Рис. 5.5 Альтернативний спосіб імпорту віртуальної машини

Після успішного завершення імпорту віртуальна машина з'явиться у списку

доступних систем менеджера VirtualBox (рис. 5.6).

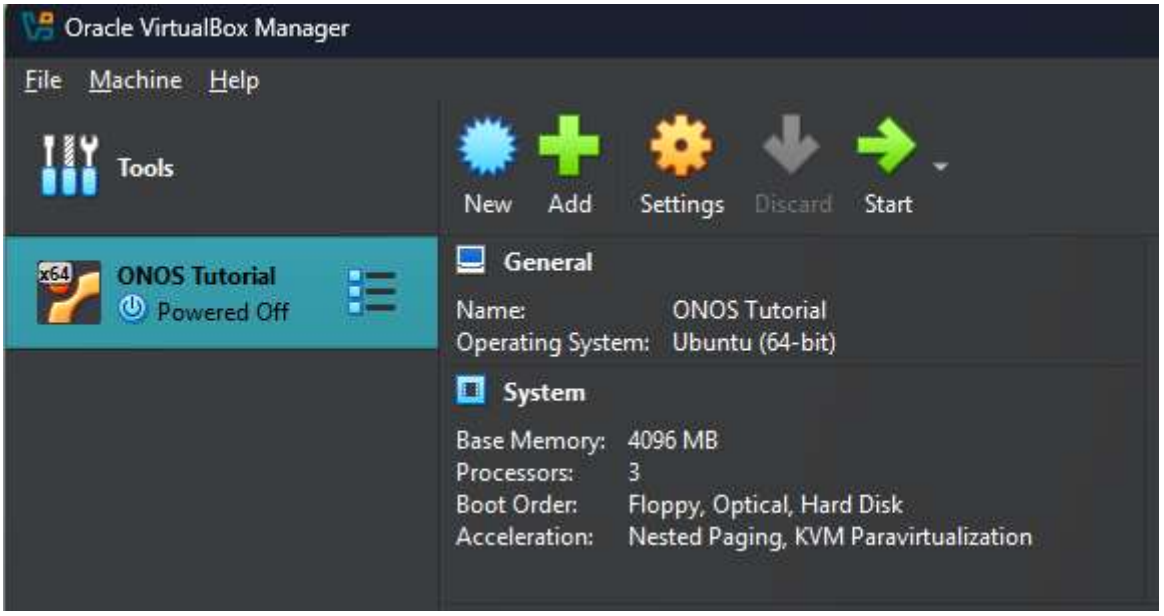

Рис. 5.6 Відображення віртуальної машини в менеджері VirtualBox

Важливим етапом є налаштування мережевих інтерфейсів. У меню налаштувань («Settings» -> «Network») для адаптера №2 необхідно змінити тип підключення з «Host-only Adapter» на «Bridged Adapter» (Мережевий міст), що забезпечить прямий доступ віртуальної машини до фізичної мережі (рис. 5.7).

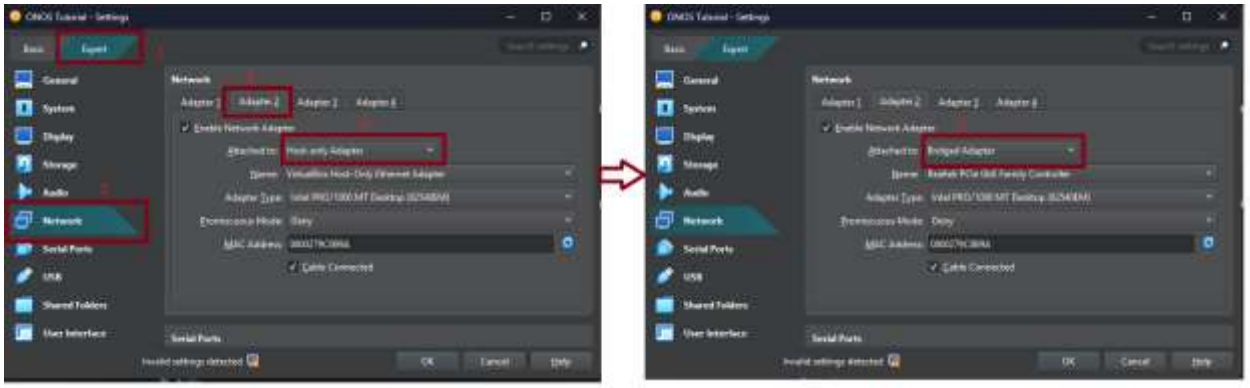

Рис. 5.7 Налаштування мережевого адаптера в режим «Bridge Adapter»

Після запуску віртуальної машини виконується вхід у систему (рис. 5.8). Для авторизації використовується пароль за замовчуванням: rocks.

Рис. 5.8 Вікно входу в операційну систему віртуальної машини

Примітка: У разі виникнення системних повідомлень про помилки під час завантаження середовища, їх можна ігнорувати, натиснувши «Cancel».

Ініціалізація кластера контролерів ONOS здійснюється шляхом запуску скрипта Setup ONOS Cluster, розміщеного на робочому столі. Виконання даного скрипта активує термінал, у якому відбувається процес завантаження та налаштування сервісів ONOS (рис. 5.9).

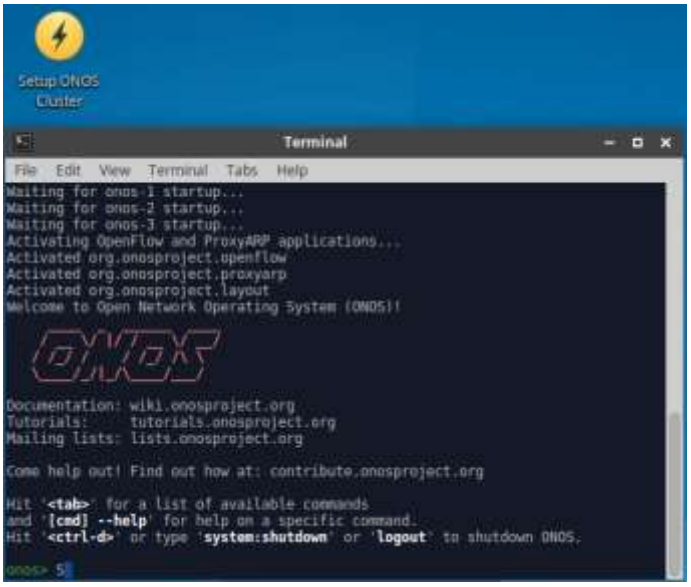

Рис. 5.9 Процес запуску та ініціалізації контролерів ONOS

Верифікація стану вузлів кластера виконується командою nodes у консолі ONOS CLI. Коректна робота системи підтверджується статусом state=READY для всіх вузлів (рис. 5.10).

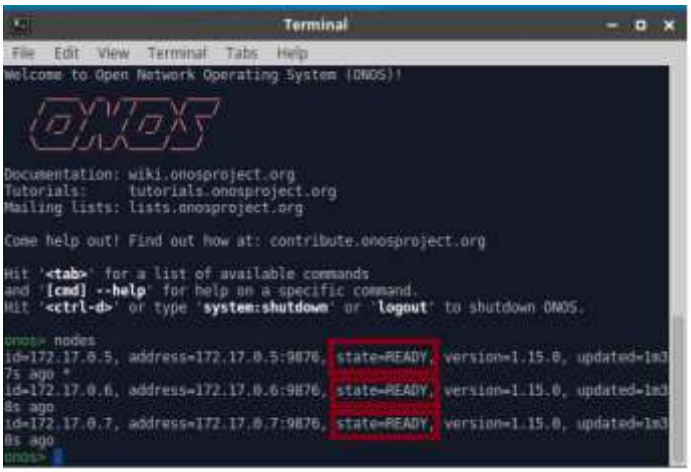

Рис. 5.10 Перевірка статусу вузлів кластера ONOS

Примітка: Якщо статус відображається як Active, але не Ready, рекомендується перевірити налаштування інтернет-з'єднання хост-машини (наприклад, використати VPN або альтернативну точку доступу).

Подальше конфігурування та моніторинг мережі здійснюється через графічний веб-інтерфейс (ONOS GUI). Доступ до нього забезпечується через вбудований браузер за допомогою ярлика ONOS GUI (рис. 5.11) або шляхом прямого введення адреси http://172.17.0.5:8181/onos/ui (рис. 5.12).

Рис. 5.11 Ярлик для запуску веб-інтерфейсу ONOS

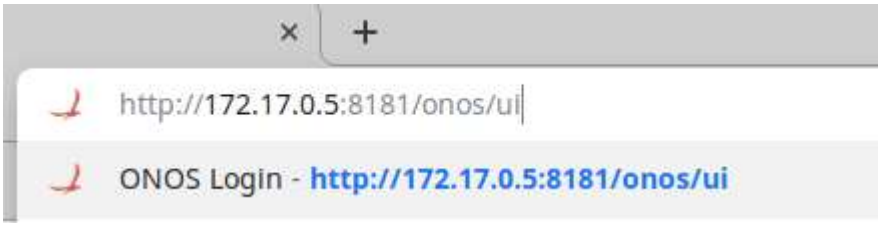

Рис. 5.12 Введення URL-адреси для доступу до графічного веб-інтерфейсу контролера ONOS

Для авторизації у веб-інтерфейсі (рис. 5.13) використовуються облікові дані:

Логін: onos

Пароль: rocks

Рис. 5.13 Форма авторизації у веб-інтерфейсі

Після входу відкривається головна панель управління (Topology View), яка на початковому етапі не відображає пристроїв, оскільки топологія мережі ще не ініціалізована (рис. 5.14).

Рис. 5.14 Головний інтерфейс контролера (Topology View)

5.3 Тестування SDN мережі на основі ONOS кластера

Наступним етапом є розгортання та тестування топології мережі. Для цього повторно ініціюється налаштування кластера за допомогою скрипта Setup ONOS Cluster (рис. 5.15).

Рис. 5.15 Запуск скрипта налаштування кластера

Для демонстрації роботи обирається одна із попередньо встановлених топологій, наприклад, Spine Leaf Topology (рис. 5.16).

Рис. 5.16 Вибір топології Spine Leaf Topology

Після вибору сценарію емулятор Mininet автоматично виконує побудову віртуальної мережі, створюючи комутатори, хости та канали зв'язку (рис. 5.17).

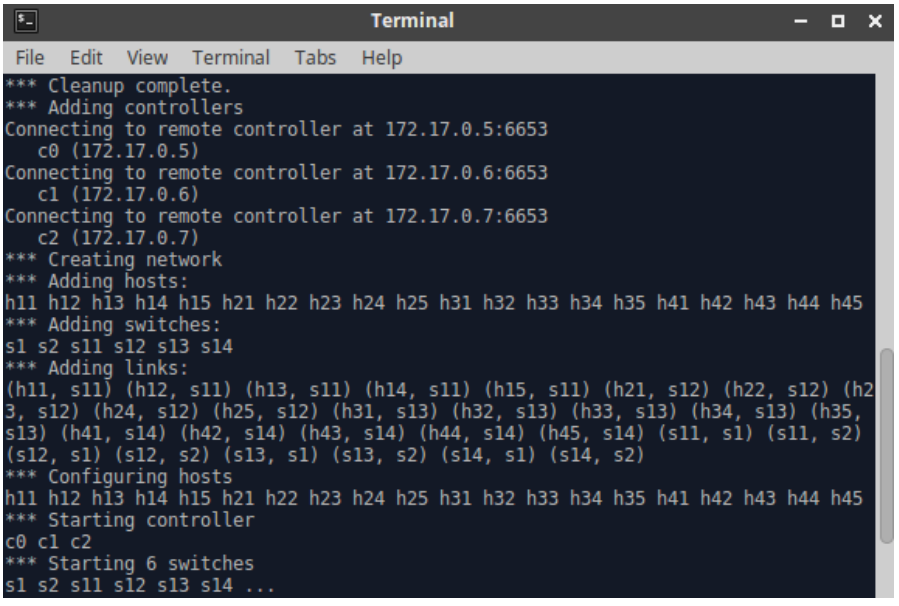

Рис. 5.17 Ініціалізація топології Spine Leaf у Mininet

Для забезпечення маршрутизації пакетів необхідно активувати додаток Reactive forwarding. Це виконується в терміналі ONOS CLI командою:

app activate fwd

Додаток fwd відповідає за встановлення правил пересилки трафіку (flows) на комутаторах на основі запитів від хостів (рис. 5.18).

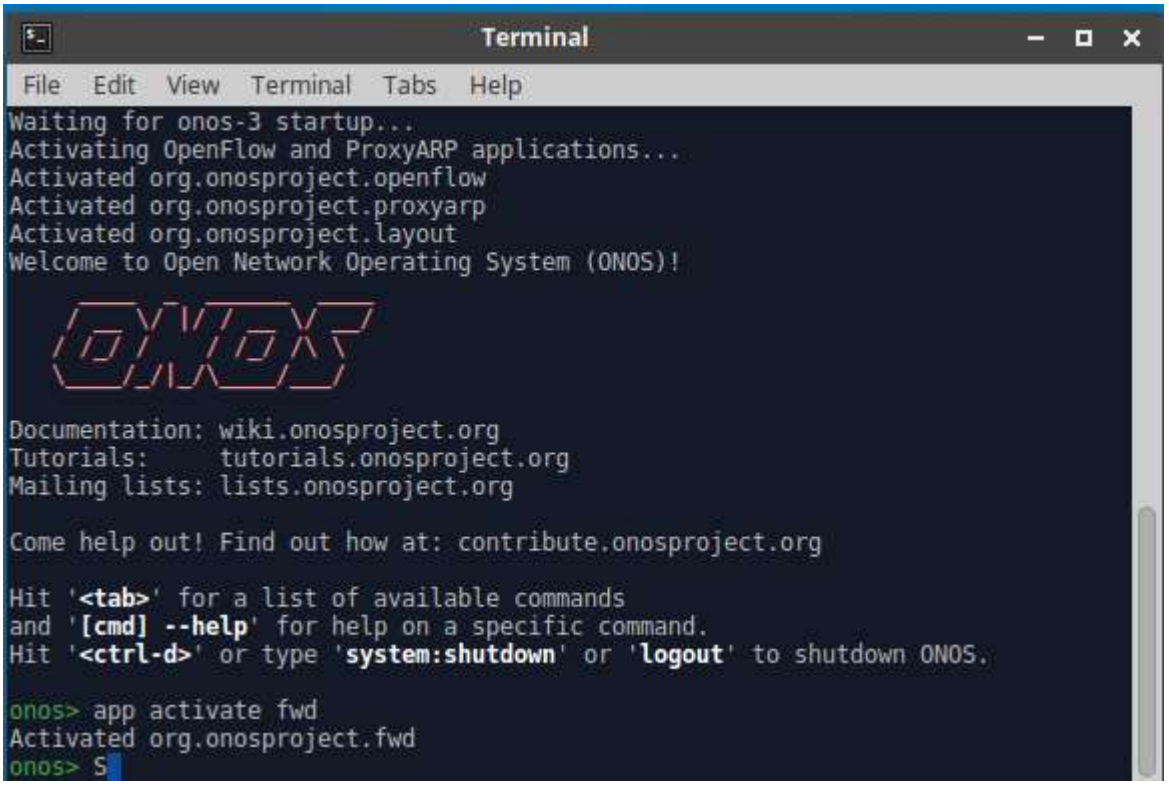

Рис. 5.18 Активація додатку forwarding (fwd)

Перевірка мережевої зв'язності між усіма створеними хостами здійснюється в консолі Mininet за допомогою команди pingall (рис. 5.19). Успішне виконання команди свідчить про коректну роботу комутації.

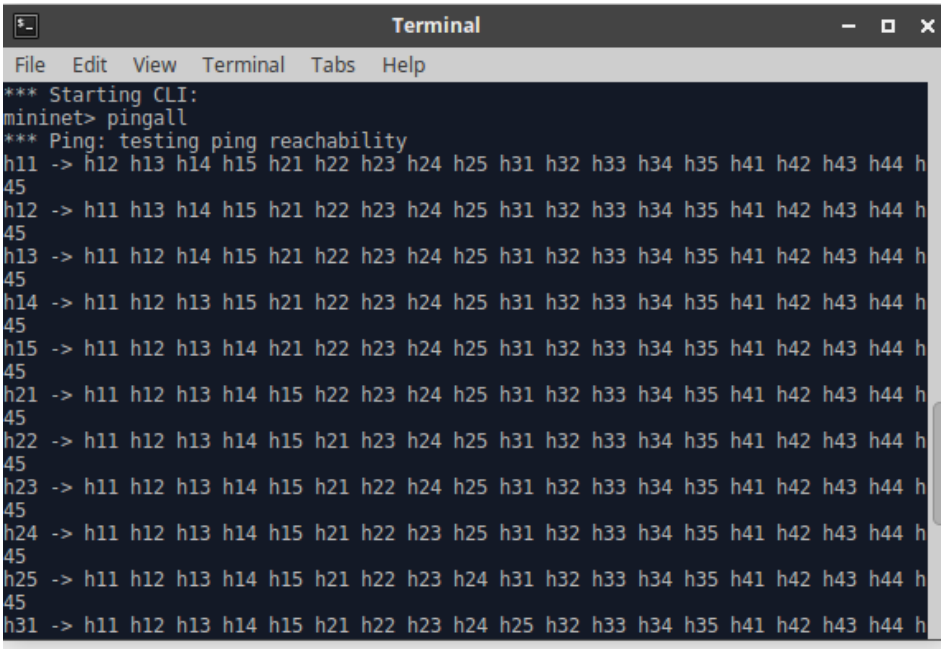

Рис. 5.19 Результати перевірки доступності вузлів (pingall)

Візуалізація топології доступна у веб-інтерфейсі ONOS. Для відображення кінцевих вузлів (хостів) необхідно натиснути клавішу H. Позиціювання елементів на схемі здійснюється за допомогою комбінації клавіш Alt + прокручування/перетягування мишею (рис. 5.20).

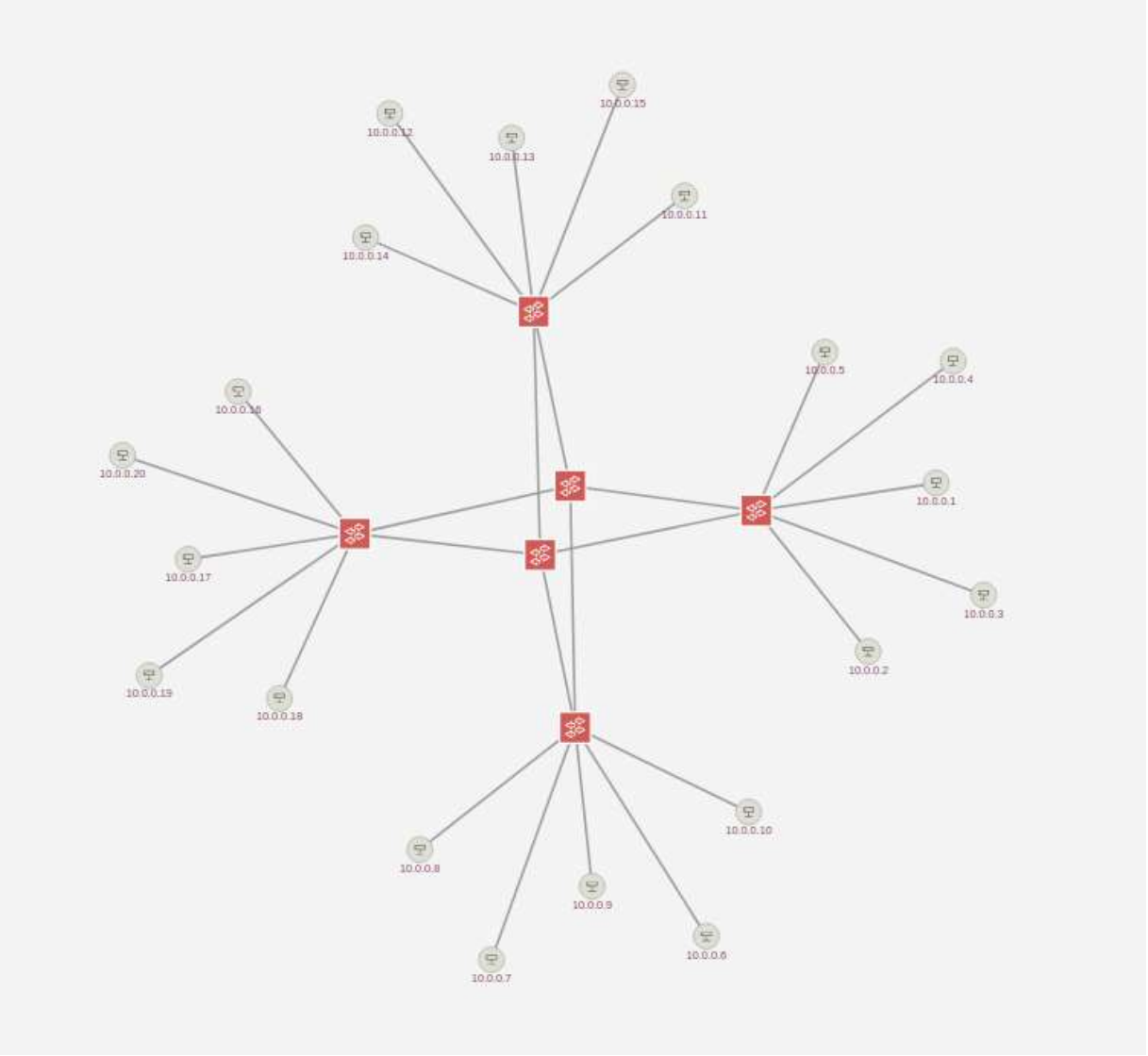

Рис. 5.20 Візуалізація топології Spine Leaf в GUI

Для оптимізації розподілу навантаження між контролерами кластера виконується команда балансування (рис. 5.21):

balance-masters

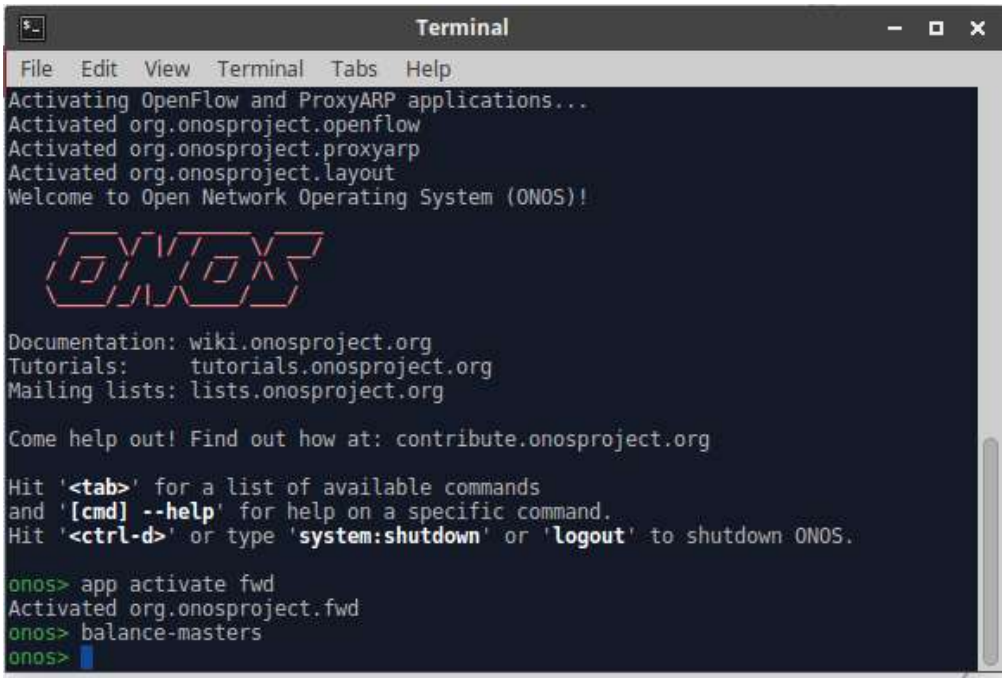

Рис. 5.21 Виконання команди балансування навантаження

Результат балансування відображається у GUI зміною кольорового маркування комутаторів, що відповідає приналежності до певного майстер-контролера (рис. 5.22).

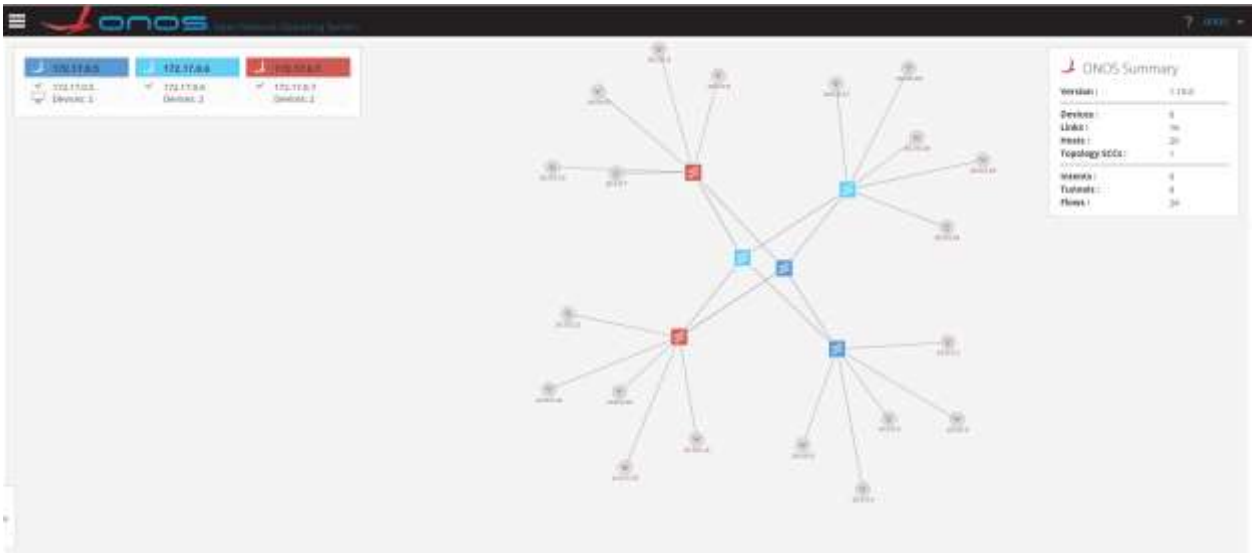

Рис. 5.22 Відображення розподілу комутаторів за контролерами

Перед переходом до налаштування складніших сценаріїв необхідно коректно завершити роботу поточної емуляції. Вихід з Mininet здійснюється командою exit або комбінацією Ctrl+D. Очищення конфігурації контролера виконується в ONOS CLI командою (рис. 5.23):

wipe-out please

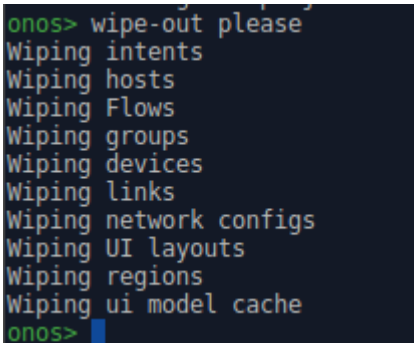

Рис. 5.23 Очищення конфігурації контролера

Додатково проводиться очищення середовища Mininet від залишкових процесів та тимчасових файлів командою в системному терміналі [12]:

sudo mn -c

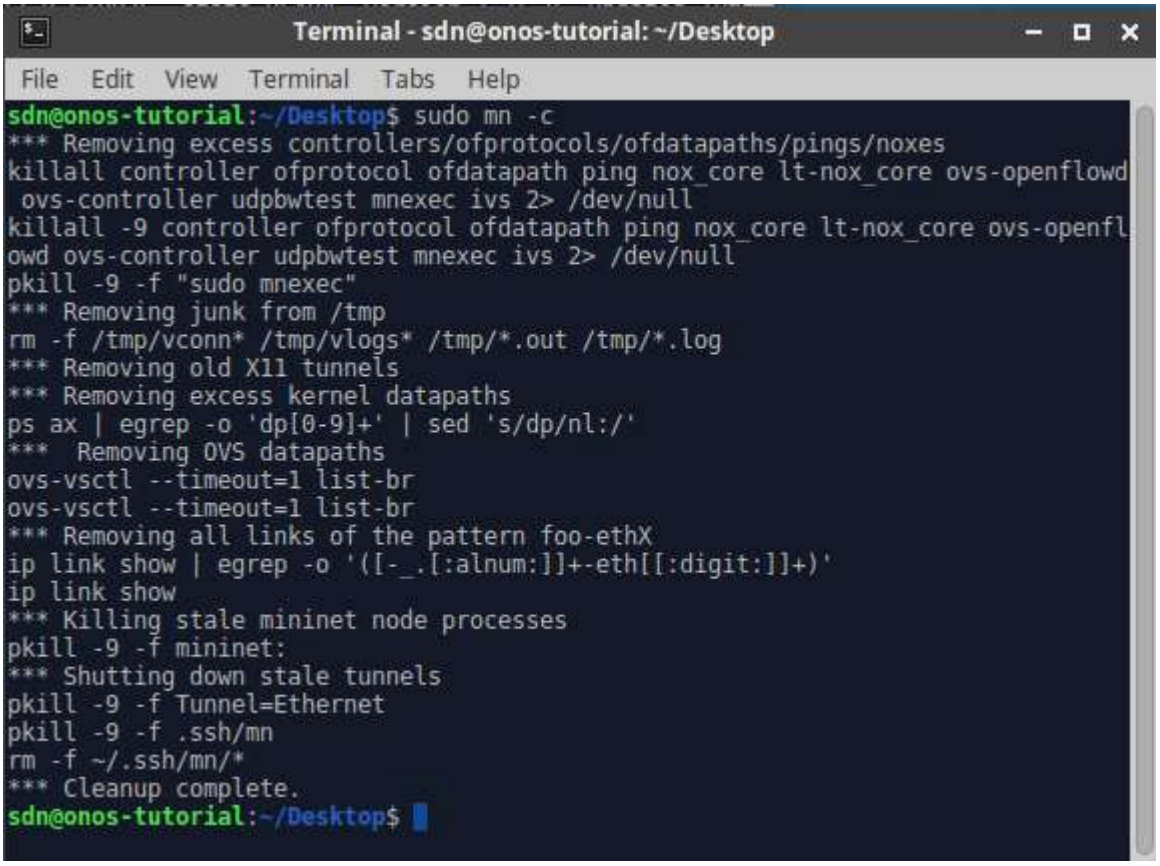

Рис. 5.24 Очищення середовища Mininet

5.4 Налаштування кастомної топології та керування маршрутизацією трафіку

Дослідження можливостей керування маршрутизацією в ONOS проводилось

на базі кастомної топології з множинними шляхами між вузлами. Експеримент включав створення та ініціалізацію топології через Python-скрипт, організацію передачі UDP-трафіку за допомогою утиліти iperf, виявлення перевантаження каналів та усунення втрат пакетів. ONOS надає два рівноцінні методи налаштування маршрутів: через командний інтерфейс (CLI) з використанням команд створення інтентів та через графічний інтерфейс (GUI) з візуальним вибором хостів, що дозволяє обрати оптимальний спосіб залежно від сценарію використання.

В рамках дослідження реалізовано кастомну топологію мережі, схема якої наведена на рис. 5.25. Вона передбачає наявність множинних шляхів між вузлами для дослідження алгоритмів маршрутизації.

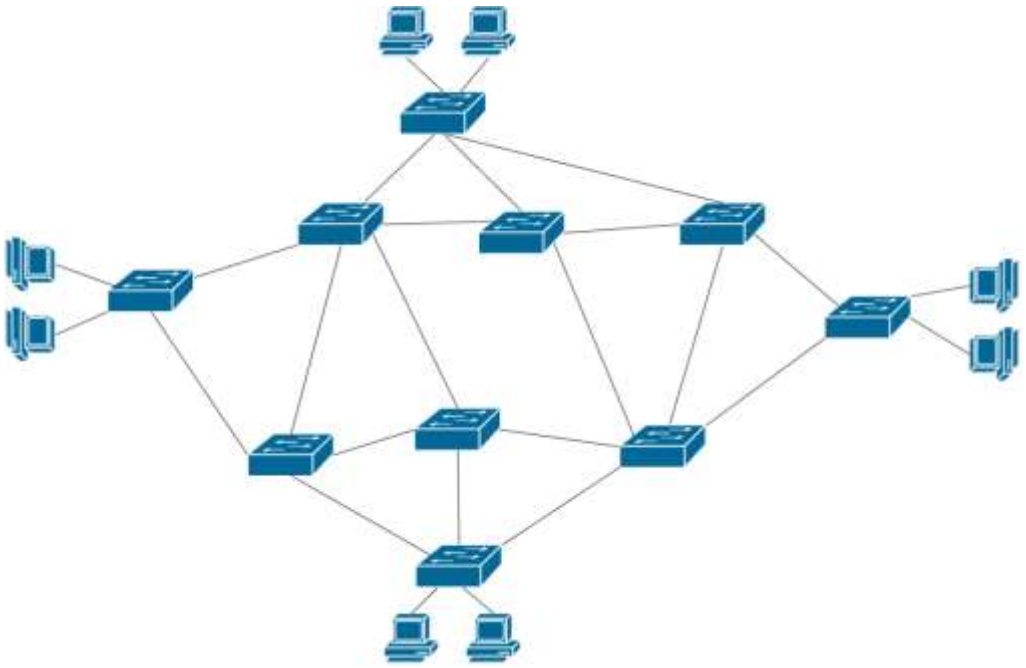

Рис. 5.25 Схема кастомної топології мережі

Опис топології реалізовано мовою Python у файлі onos_net.py додаток А. Скрипт містить оголошення комутаторів, хостів та визначення ліній зв'язку між ними (рис. 5.26) [13].

Ініціалізація мережі виконується шляхом запуску створеного скрипта з терміналу, відкритого у відповідній директорії, для перевірки прописується команда (рис. 5.27):

ls

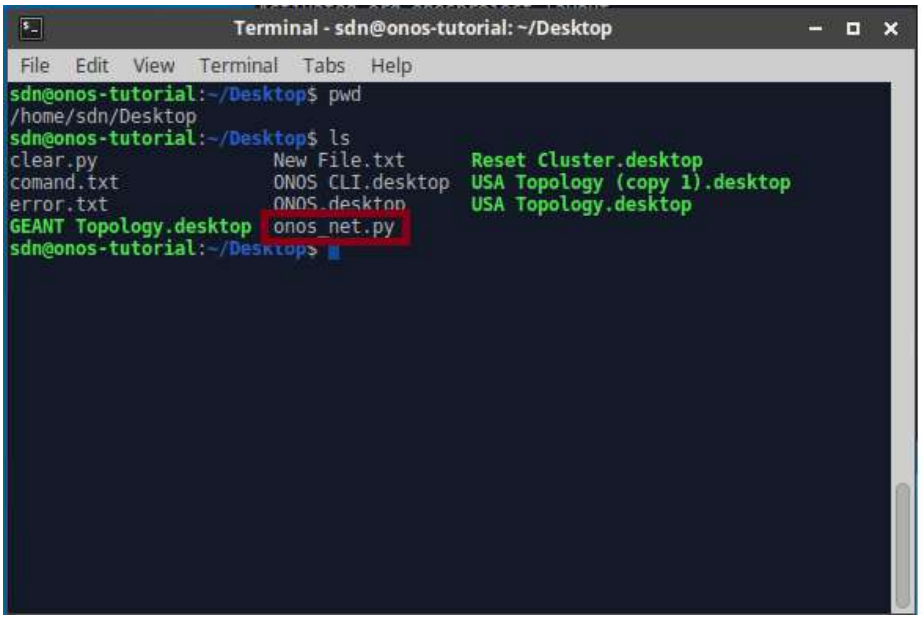

Рис. 5.27 Перевірка наявності файлу топології

Запуск емуляції здійснюється командою з правами суперкористувача (рис. 5.28):

sudo python onos_net.py

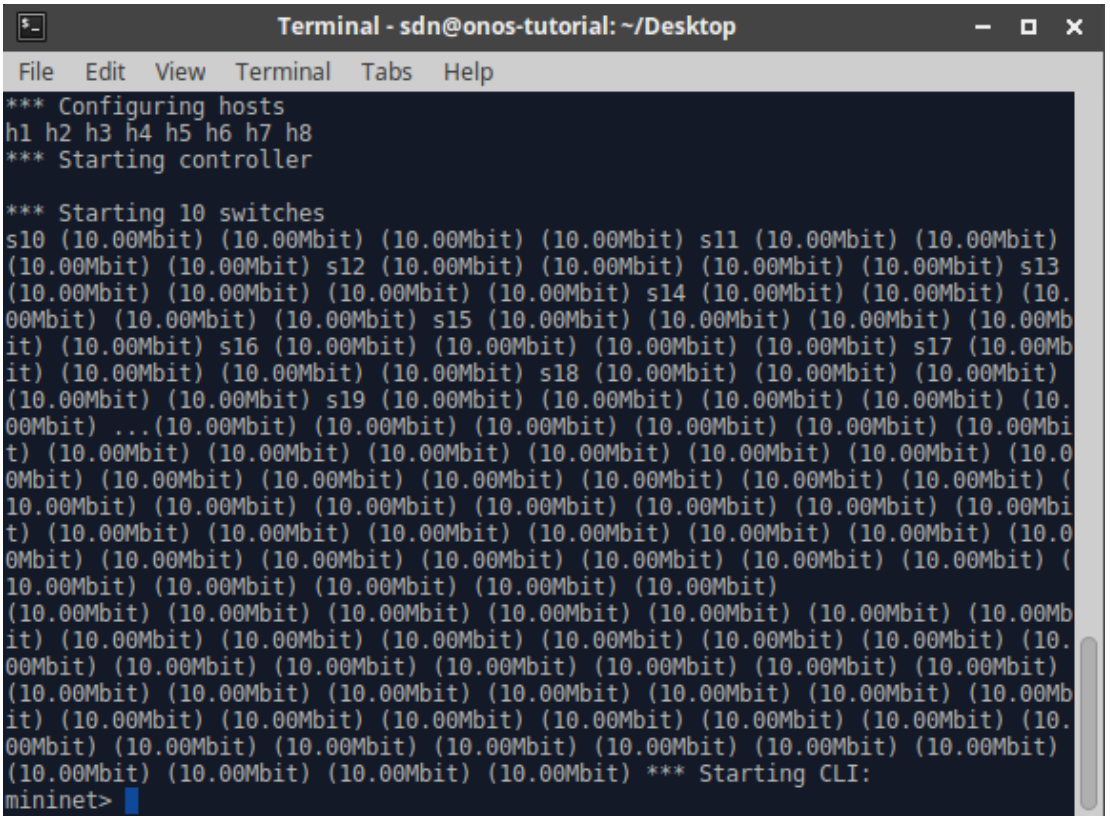

Рис. 5.28 Запуск скрипта ініціалізації мережі

Після розгортання топології необхідно повторно активувати додаток

маршрутизації в ONOS CLI (рис. 5.29):

app activate fwd

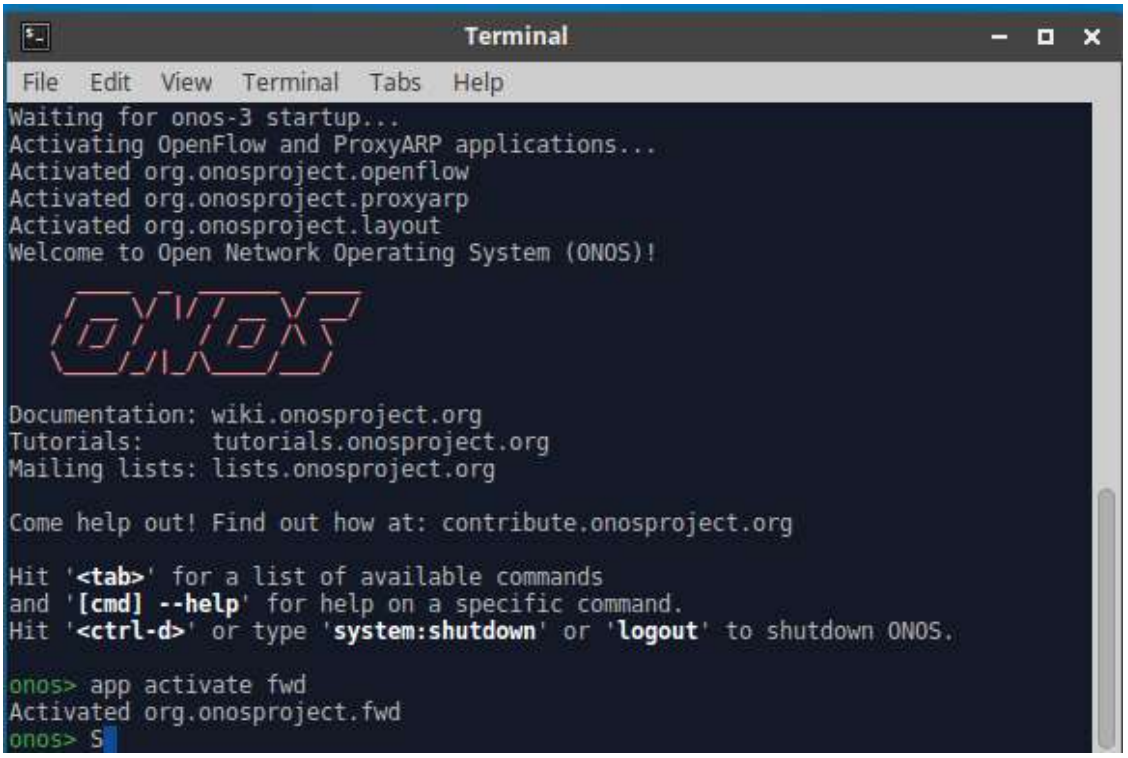

Рис. 5.29 Повторна активація додатку fwd

Перевірка зв'язності виконується командою pingall у консолі Mininet (рис. 5.30).

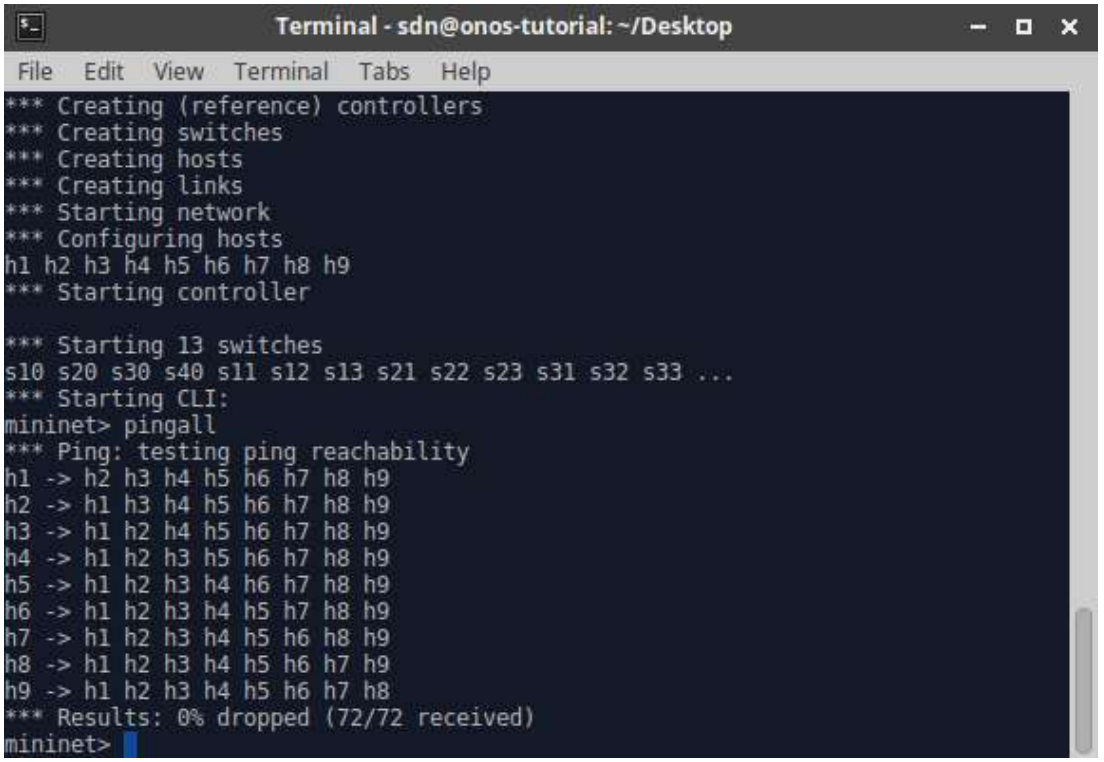

Рис. 5.30 Перевірка зв'язності у кастомній топології

Візуалізація новоствореної топології доступна в графічному інтерфейсі ONOS (рис. 5.31).

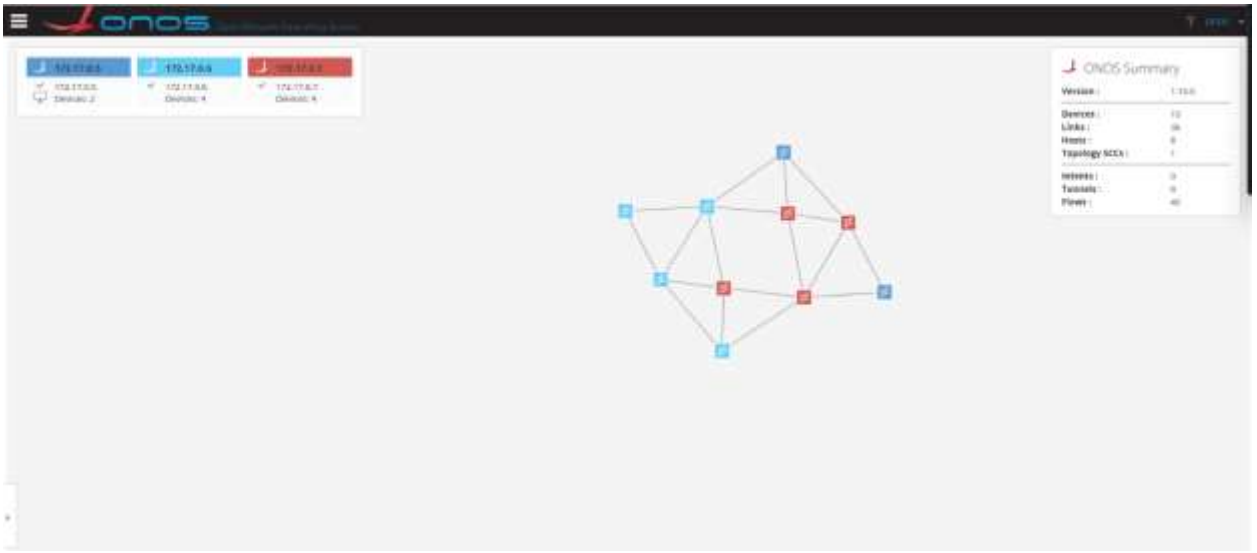

Рис. 5.31 Відображення кастомної топології (тільки комутатори)

Для відображення хостів використовується клавіша H (рис. 5.32).

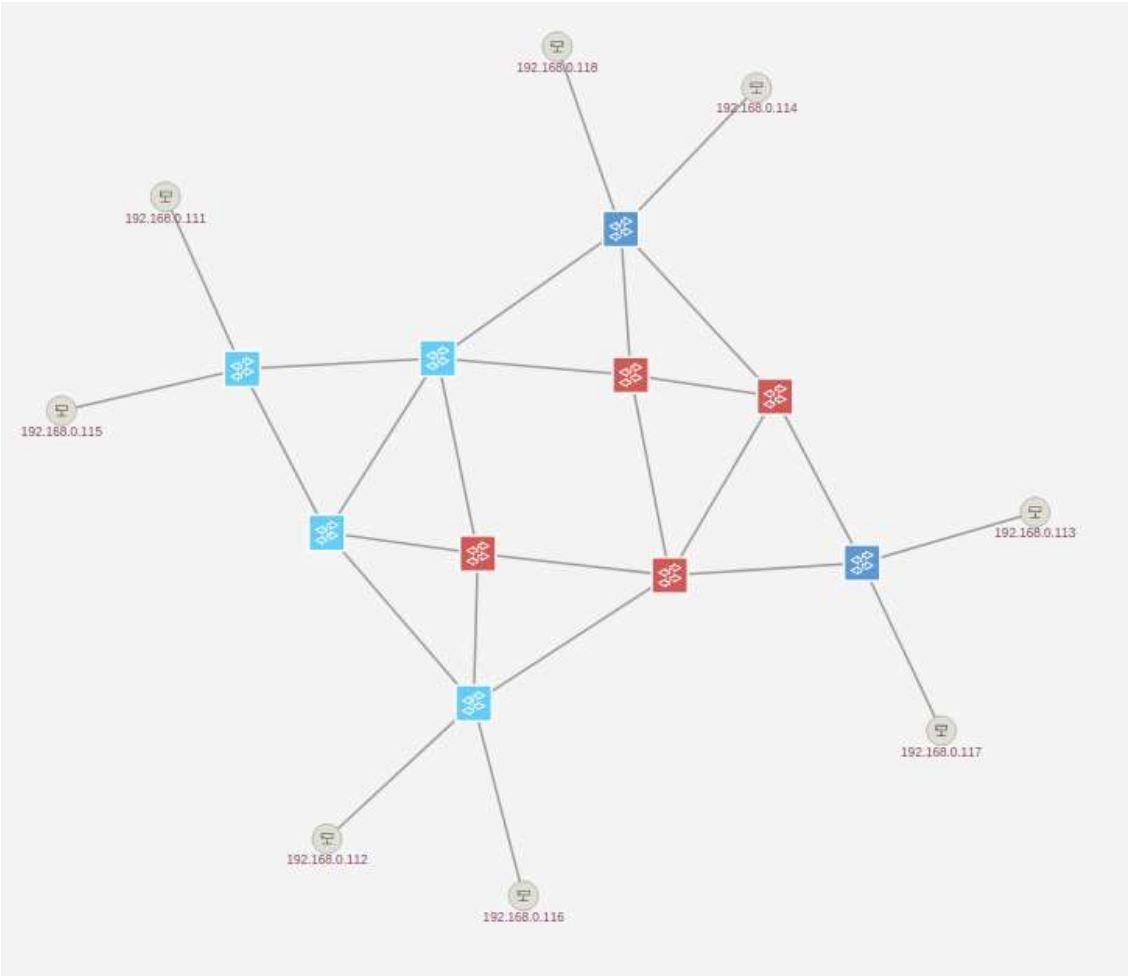

Рис. 5.32 Повна візуалізація топології з хостами

Для рівномірного розподілу навантаження на площину управління виконується команда балансування (рис. 5.33): balance-masters

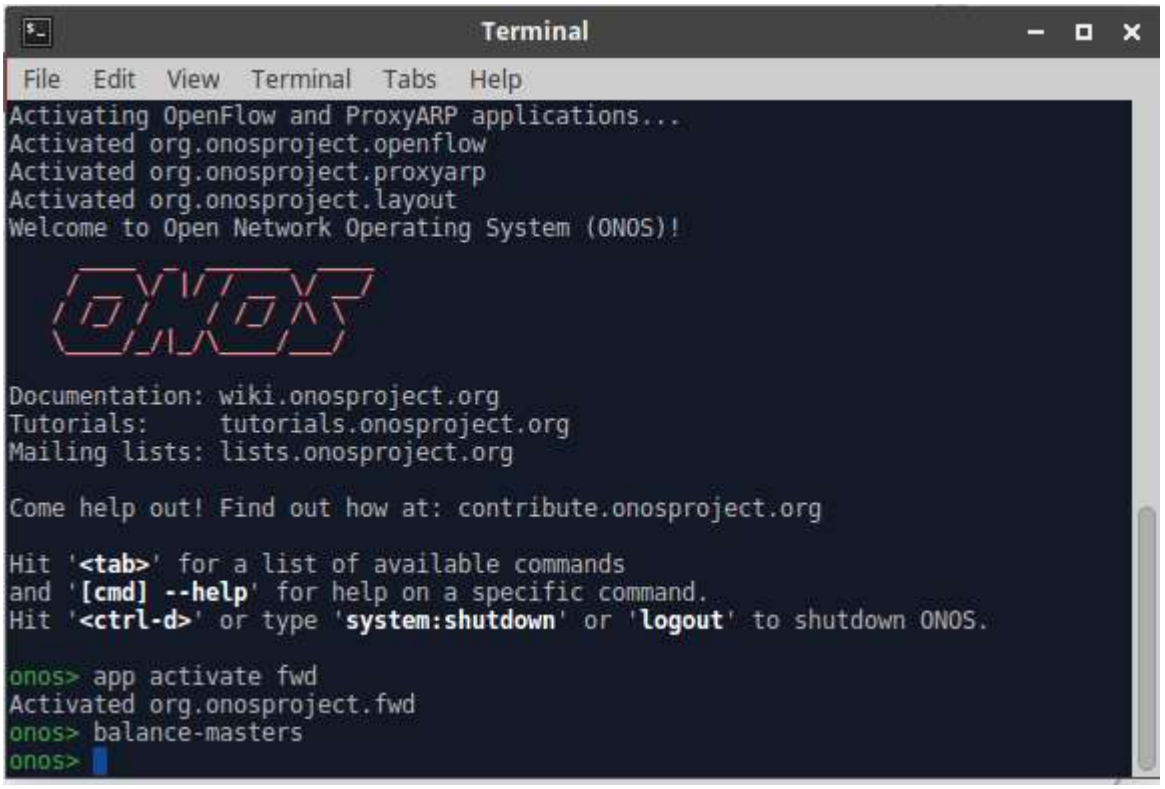

Рис. 5.33 Команда балансування майстер-контролерів

Результат розподілу комутаторів між контролерами представлено на рис. 5.34.

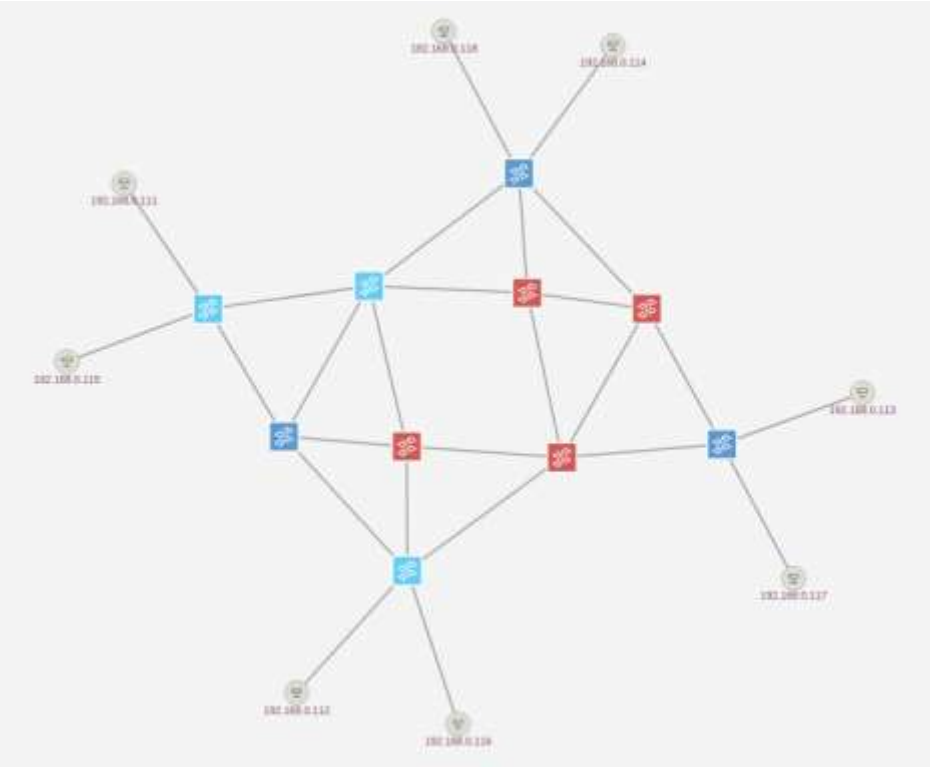

Рис. 5.34 Розподіл комутаторів між контролерами після балансування

Для дослідження поведінки мережі під навантаженням організовано передачу даних за протоколом UDP за схемою «клієнт-сервер» за допомогою утиліти iperf.

На хості H3 ініціалізується серверна частина. Для цього через термінал емулятора (xterm h3) виконується команда:

iperf -s -u -p 10 -i 0.5

де -s – режим сервера, -u – протокол UDP, -p – порт, -i – інтервал звіту (рис. 5.35).

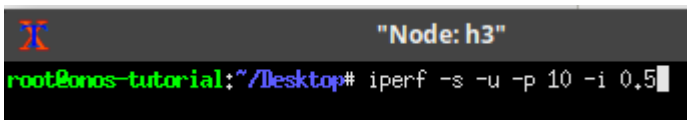

Рис. 5.34 Ініціалізація UDP-сервера iperf на хості H3

На хості H1 запускається клієнтська частина для генерації потоку даних до сервера H3:

iperf -c 192.168.0.113 -u -b 8M -p 10 -t 100000

де -b 8M – смуга пропускання 8 Мбіт/с (рис. 5.36).

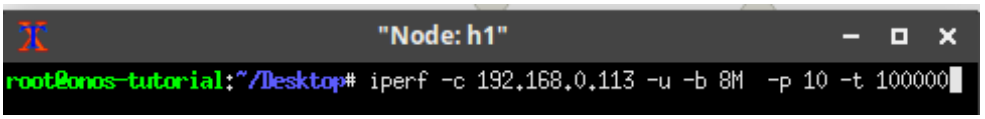

Рис. 5.36 Запуск UDP-клієнта iperf на хості H1

Аналогічні дії виконуються для пари хостів H5 (клієнт) та H7 (сервер).

У консолях хостів відображається статистика передачі пакетів (рис. 5.37).

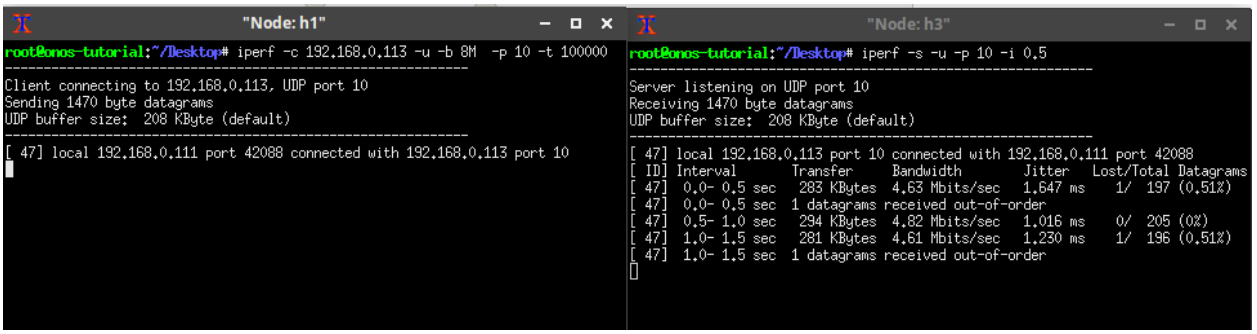

Рис. 5.37 Лог передачі даних між клієнтом та сервером

Моніторинг завантаженості каналів у реальному часі здійснюється в ONOS GUI (активація режиму відображення трафіку клавішею A). На рис. 5.38 продемонстровано візуалізацію потоків даних.

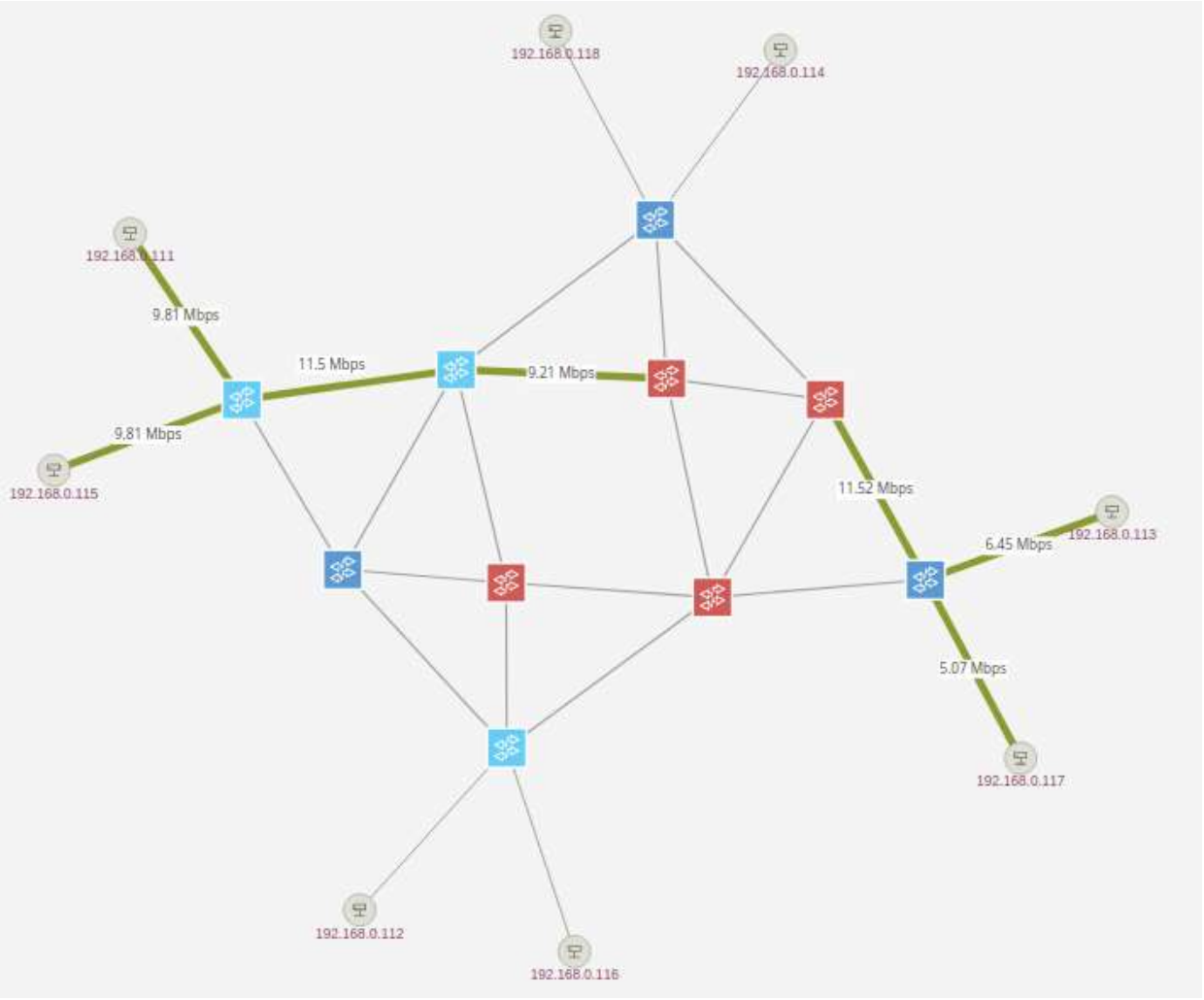

Рис. 5.38 Графічне відображення передачі трафіку

В ході експерименту зафіксовано перевантаження каналу зв'язку, оскільки сумарний обсяг трафіку (16 Мбіт/с) перевищує пропускну здатність окремого лінку (10 Мбіт/с). Це призводить до значних втрат пакетів (рис. 5.39). Автоматичні спроби мережі перебудувати маршрут не дають результату через обмеженість ресурсів єдиного обраного шляху.

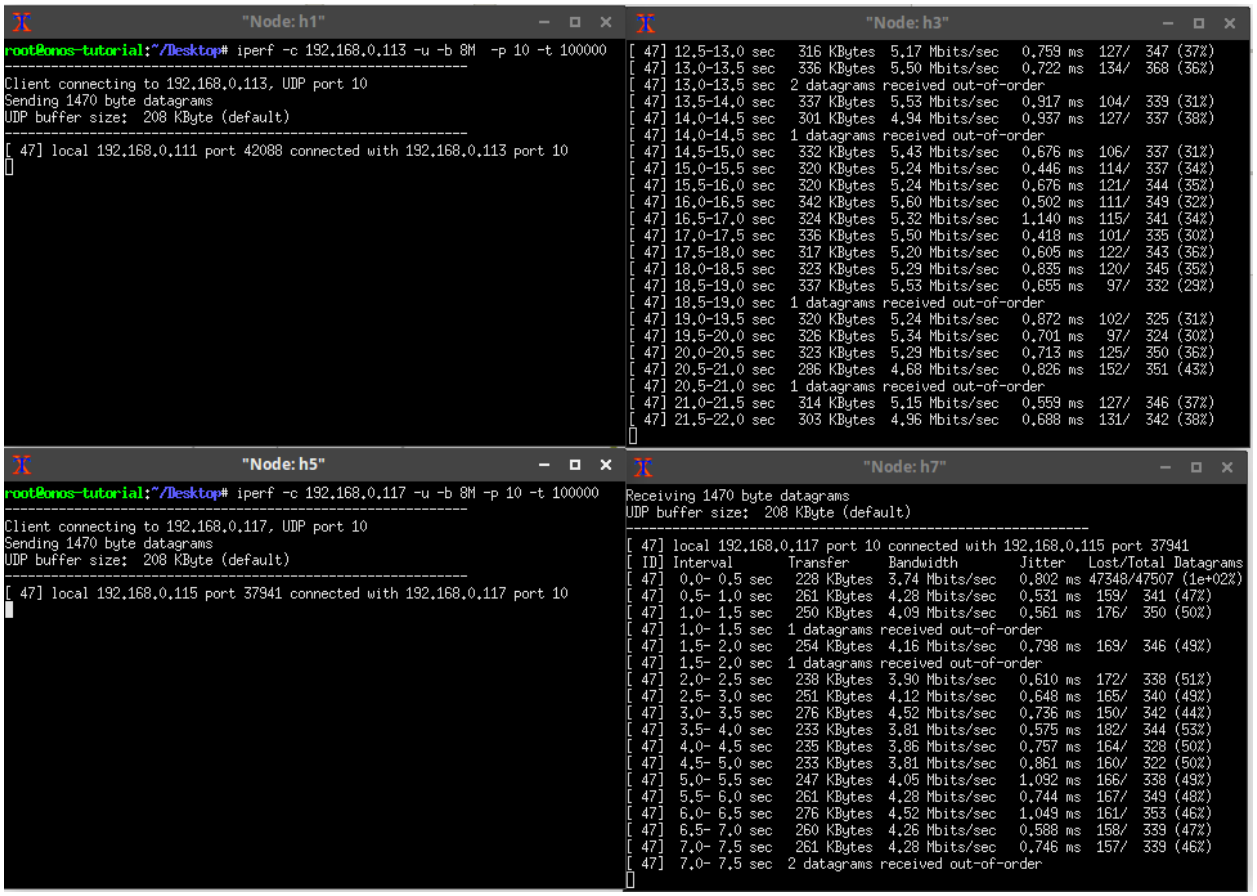

Рис. 5.39 Демонстрація втрати пакетів

Для вирішення проблеми перевантаження пропонується застосування механізму інтентів для примусового розділення потоків трафіку.

Спочатку деактивується додаток реактивної маршрутизації fwd для уникнення конфліктів правил (рис. 5.40):

app deactivate fwd

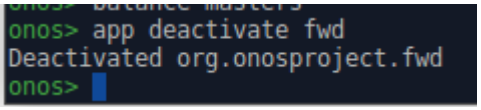

Рис. 5.40 Деактивація додатку fwd

Після відключення fwd передача даних припиняється, що підтверджується відсутністю активності на каналах в GUI (рис. 5.41).

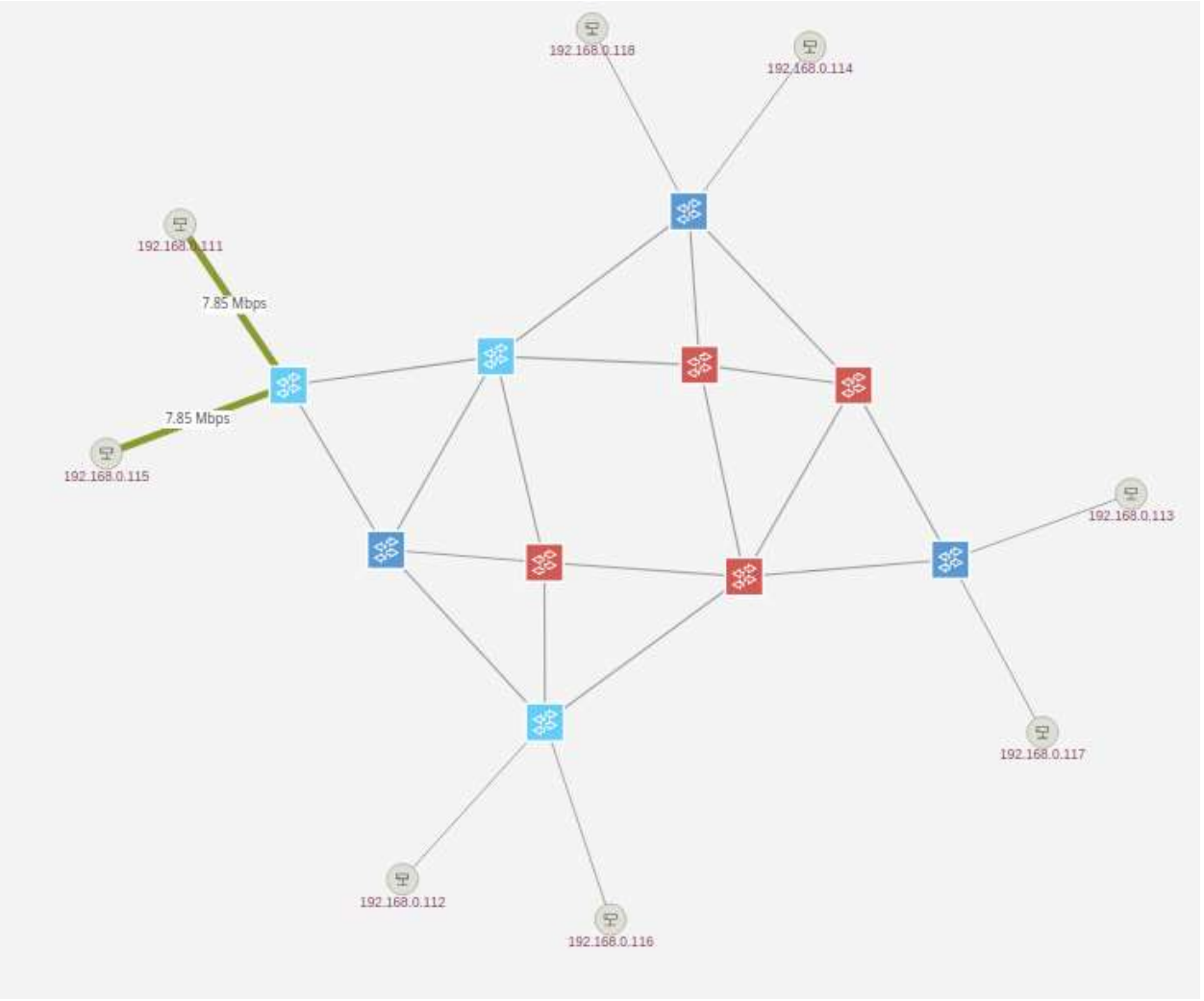

Рис. 5.41 Ізоляція трафіку в межах локального комутатора

Відновлення зв'язності та налаштування маршрутизації здійснюється шляхом створення інтентів. Розглянуто два методи: через командний рядок (CLI) та графічний інтерфейс (GUI).

Налаштування маршрутів через ONOS CLI. Для створення інтента необхідно ідентифікувати унікальні ідентифікатори хостів. Список усіх хостів виводиться командою hosts (рис. 5.42).

```
onos> hosts
id=00:00:00:00:00:01/None, mac=00:00:00:00:00:01, locations=[of:0000000000000010
/3], vlan=None, ip(s)=[192.168.0.111], innerVlan=None, outerTPID=unknown, provid
er=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:00:02/None, mac=00:00:00:00:00:02, locations=[of:0000000000000022
/4], vlan=None, ip(s)=[192.168.0.112], innerVlan=None, outerTPID=unknown, provid
er=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:00:03/None, mac=00:00:00:00:00:03, locations=[of:0000000000000020
/3], vlan=None, ip(s)=[192.168.0.113], innerVlan=None, outerTPID=unknown, provid
er=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:00:04/None, mac=00:00:00:00:00:04, locations=[of:0000000000000023
/4], vlan=None, ip(s)=[192.168.0.114], innerVlan=None, outerTPID=unknown, provid
er=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:00:05/None, mac=00:00:00:00:00:05, locations=[of:0000000000000010
/4], vlan=None, ip(s)=[192.168.0.115], innerVlan=None, outerTPID=unknown, provid
er=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:00:06/None, mac=00:00:00:00:00:06, locations=[of:0000000000000022
/5], vlan=None, ip(s)=[192.168.0.116], innerVlan=None, outerTPID=unknown, provid
er=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:00:07/None, mac=00:00:00:00:00:07, locations=[of:0000000000000020
/4], vlan=None, ip(s)=[192.168.0.117], innerVlan=None, outerTPID=unknown, provid
er=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:00:08/None, mac=00:00:00:00:00:08, locations=[of:0000000000000023
/5], vlan=None, ip(s)=[192.168.0.118], innerVlan=None, outerTPID=unknown, provid
er=of:org.onosproject.provider.host, configured=false
```

Рис. 5.42 Вивід команди hosts

Співставлення IP-адрес та ідентифікаторів здійснюється на основі даних топології (рис. 5.43).

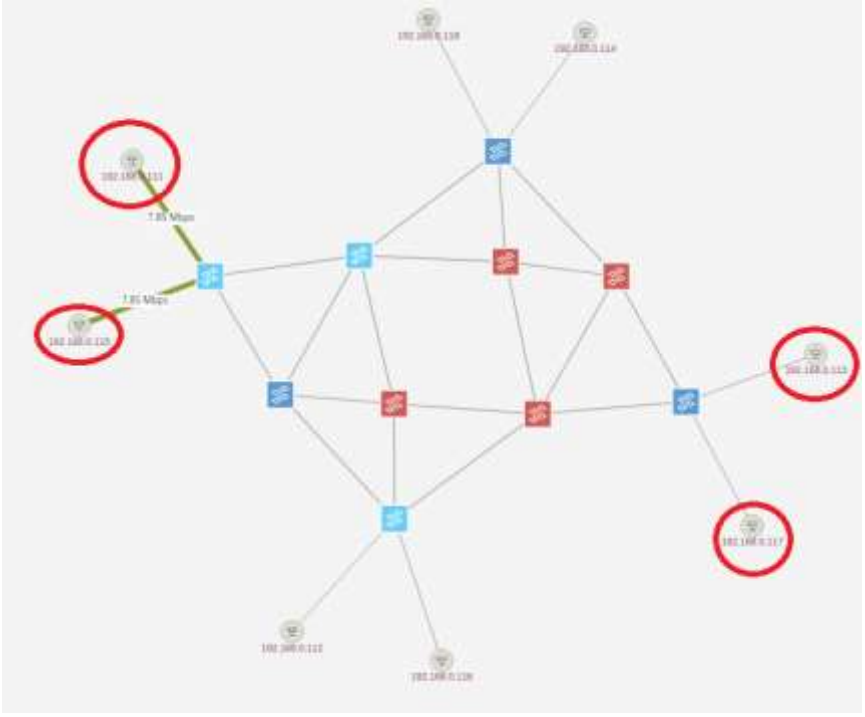

Рис. 5.43 Ідентифікація хостів у середовищі ONOS GUI

Необхідні ID мають формат MAC/VLAN ID (наприклад, 00:00:00:00:00:01/None) (рис. 5.44).

Рис. 5.44 Приклад необхідних ID у списку hosts

Створення Host-to-Host інтентів для пар H1-H3 та H5-H7 виконується командами:

add-host-intent <ID-H1> <ID-H3>

add-host-intent <ID-H5> <ID-H7>

Результат виконання команд представлено на рис. 5.45.

Рис. 5.45 Результат виконання команд створення інтентів

Після застосування інтентів передача трафіку відновлюється (рис. 5.46).

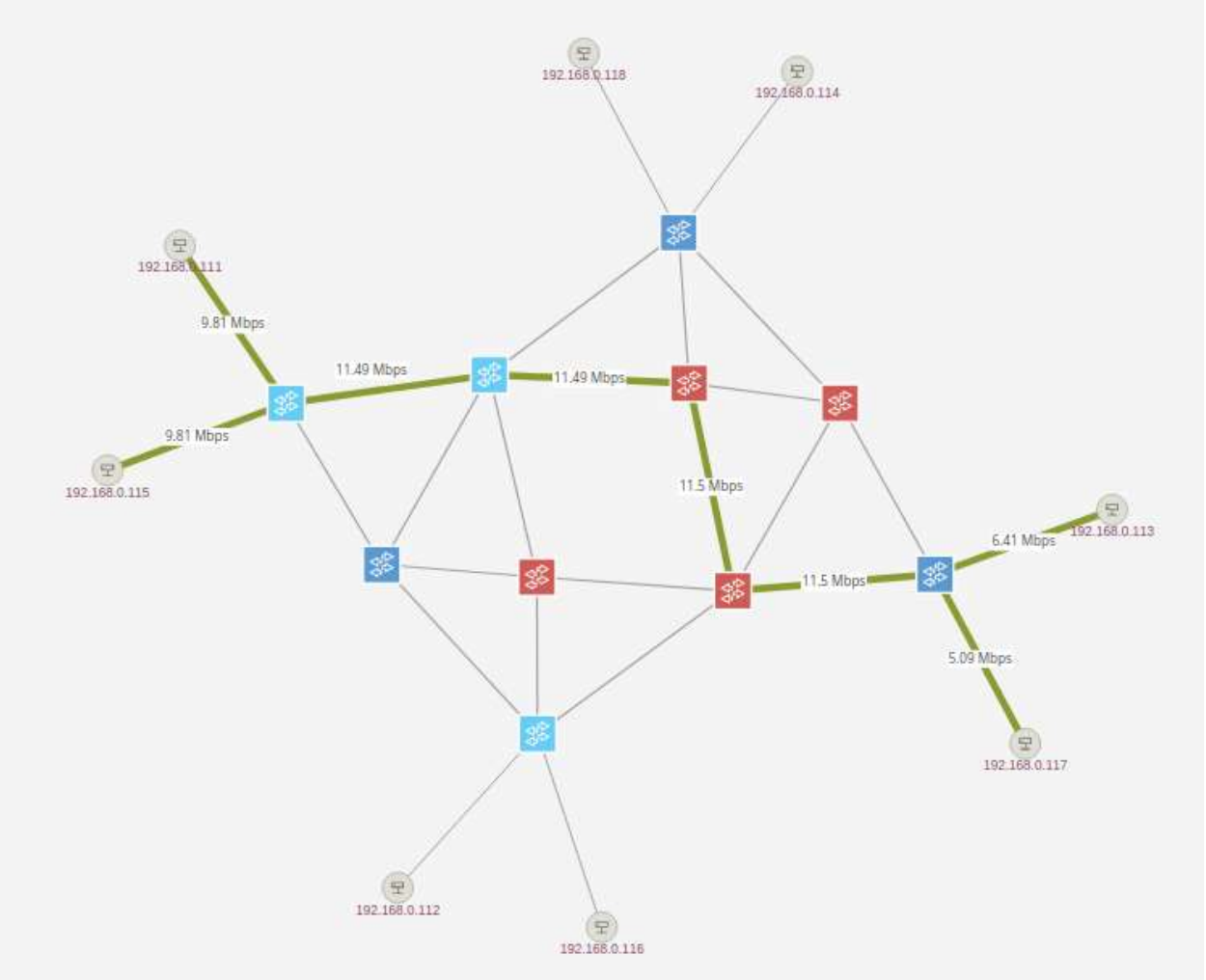

Рис. 5.46 Відновлення передачі трафіку між хостами

Однак, якщо система автоматично обрала маршрути, що перетинаються, втрати пакетів можуть зберігатися (рис. 5.47).

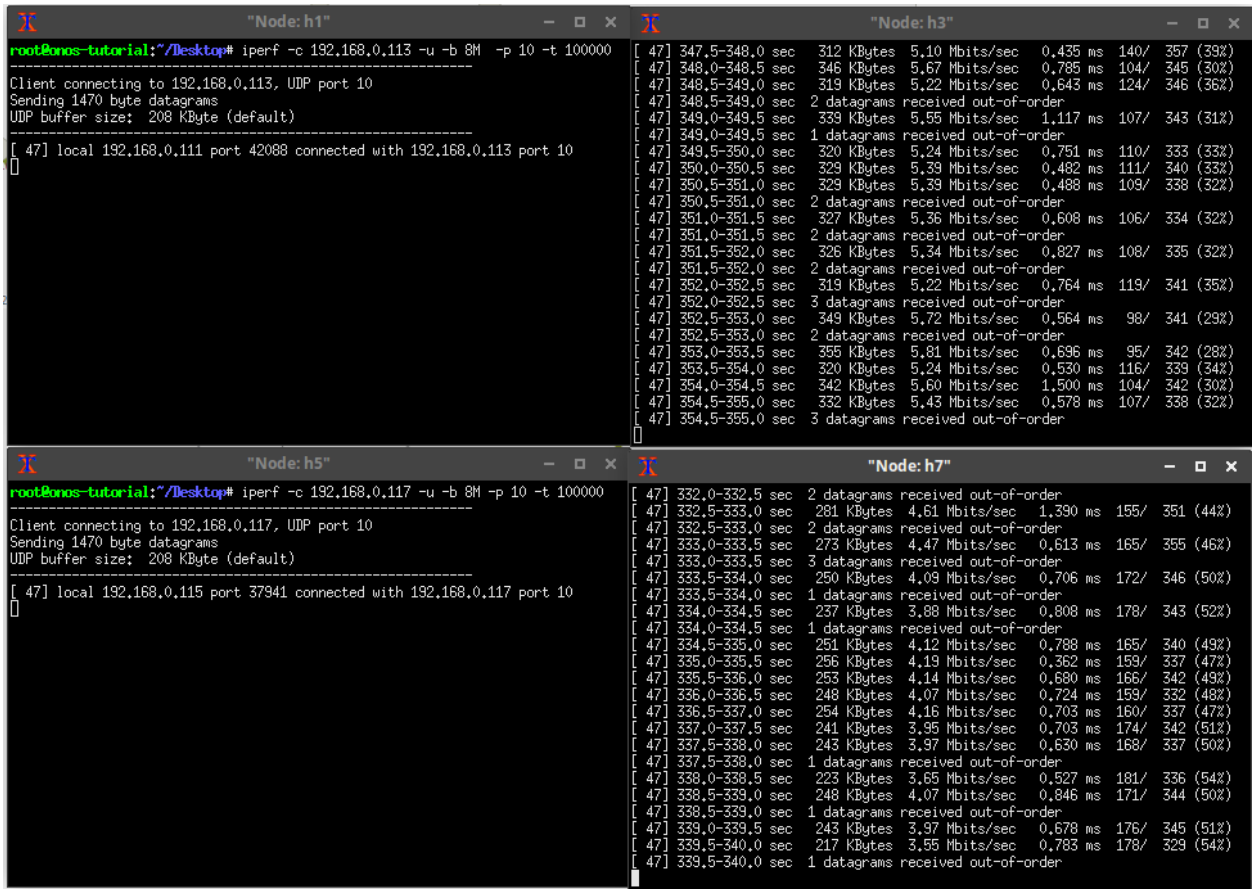

Рис. 5.47 Втрати при передачі трафіку через перевантаження каналу

Для усунення колізій необхідно домогтися прокладання маршрутів по незалежних шляхах. Повторне відправлення команди створення інтента змушує контролер перерахувати маршрут. Після перебудови топології потоків (рис. 5.48) досягається розвантаження каналів.

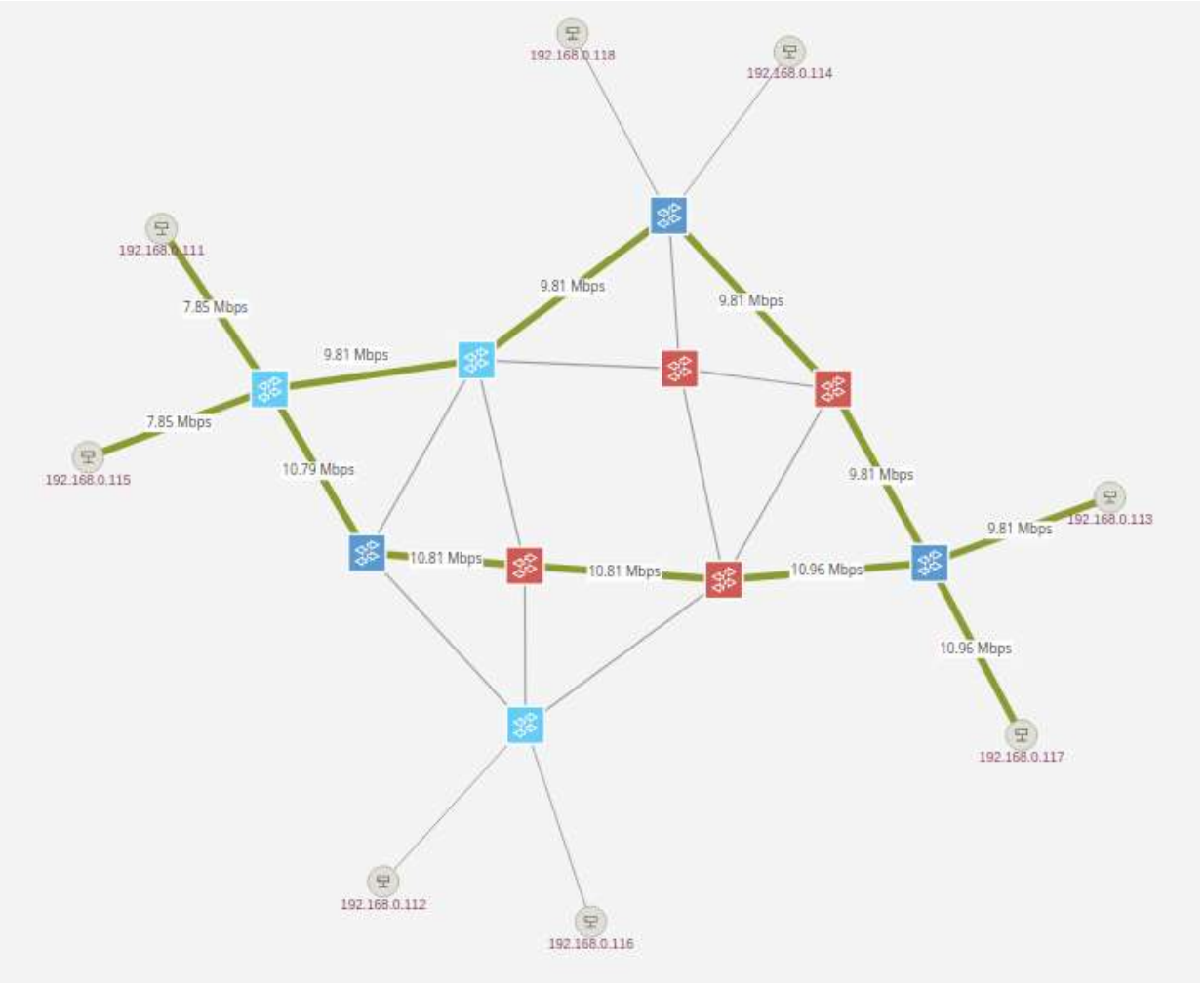

Рис. 5.48 Маршрутизація трафіку по незалежних шляхах

У результаті оптимізації маршрутів втрати пакетів усуваються, що підтверджується логами iperf (рис. 5.49).

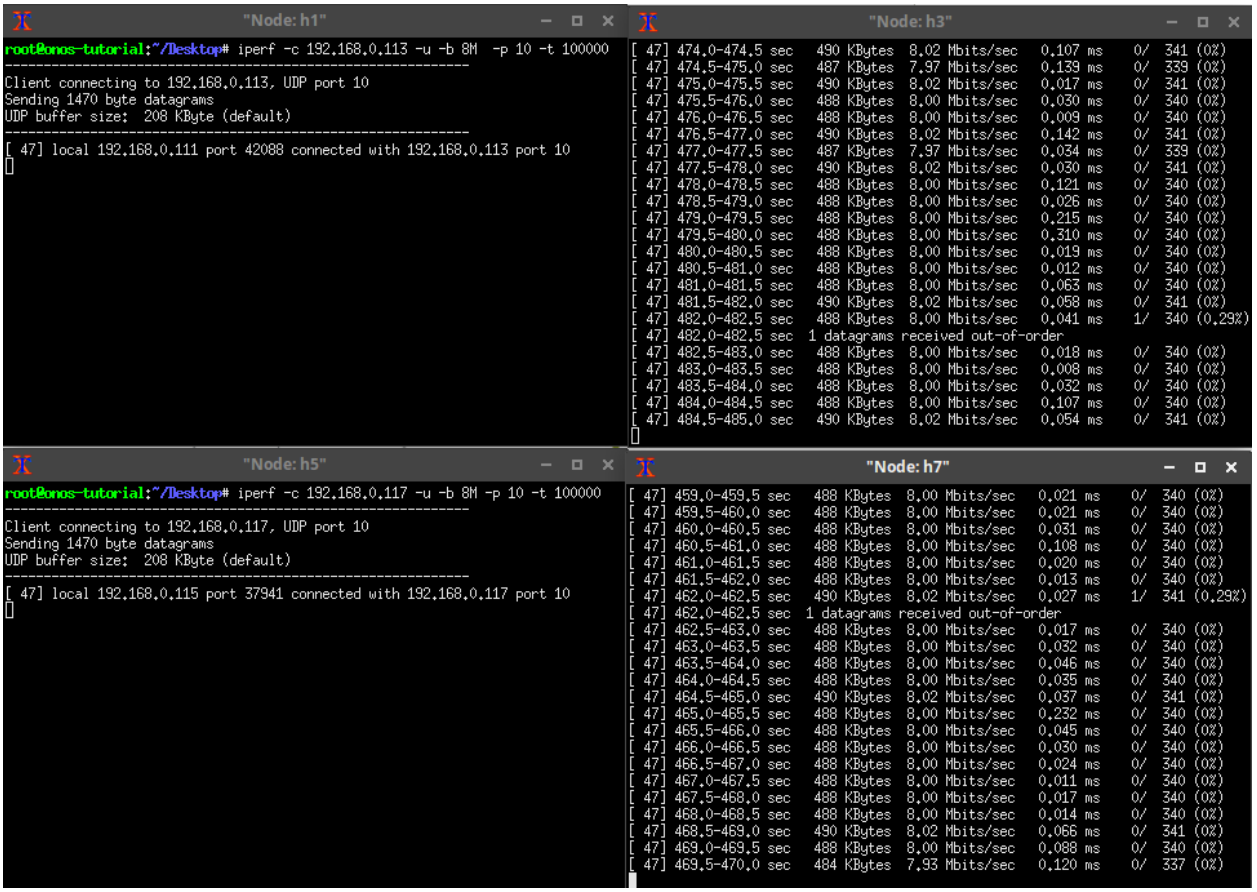

Рис. 5.49 Передача даних без втрат після оптимізації маршрутів

Налаштування маршрутів через ONOS GUI.

Альтернативний спосіб створення маршрутів передбачає використання графічного інтерфейсу. Для цього необхідно, утримуючи клавішу Shift, виділити два цільові хости. У меню «Selected items» обрати опцію Create host-to-host flow (рис. 5.50).

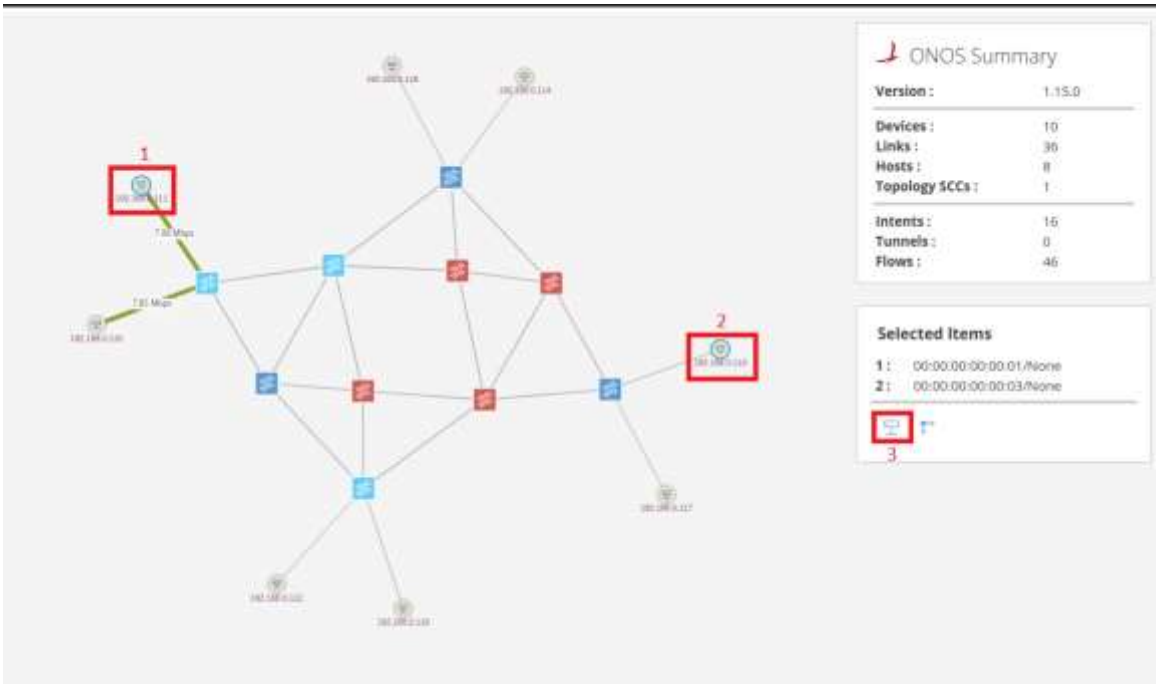

Рис. 5.50 Процес вибору хостів та створення потоку в ONOS GUI

Примітка: Перед виконанням цього пункту попередні інтенти, створені через CLI, були видалені.

Система візуалізує прокладений шлях пунктирною лінією (рис. 5.51).

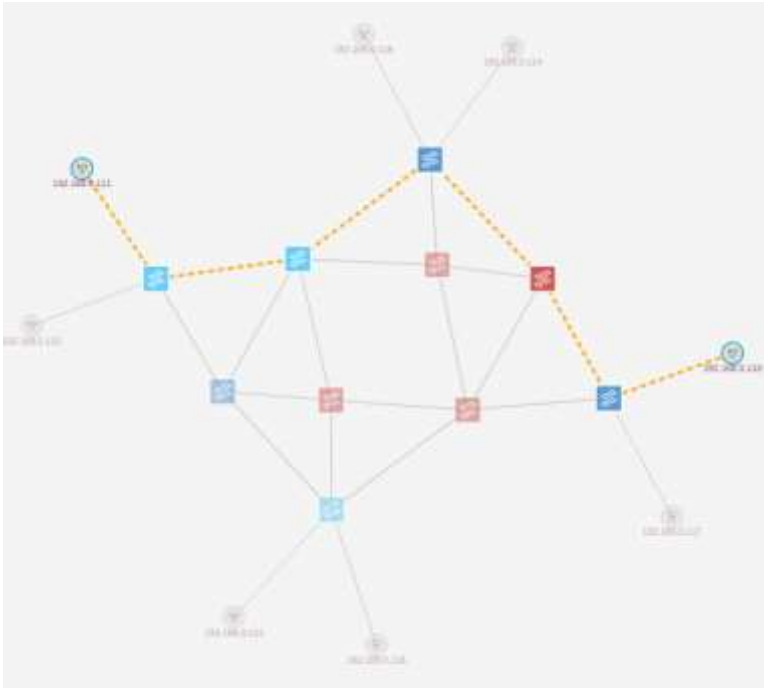

Рис. 5.51 Візуалізація прокладеного маршруту між хостами

Для оновлення інформації про трафік у реальному часі необхідно перезапустити режим моніторингу (клавіша A) (рис. 5.52).

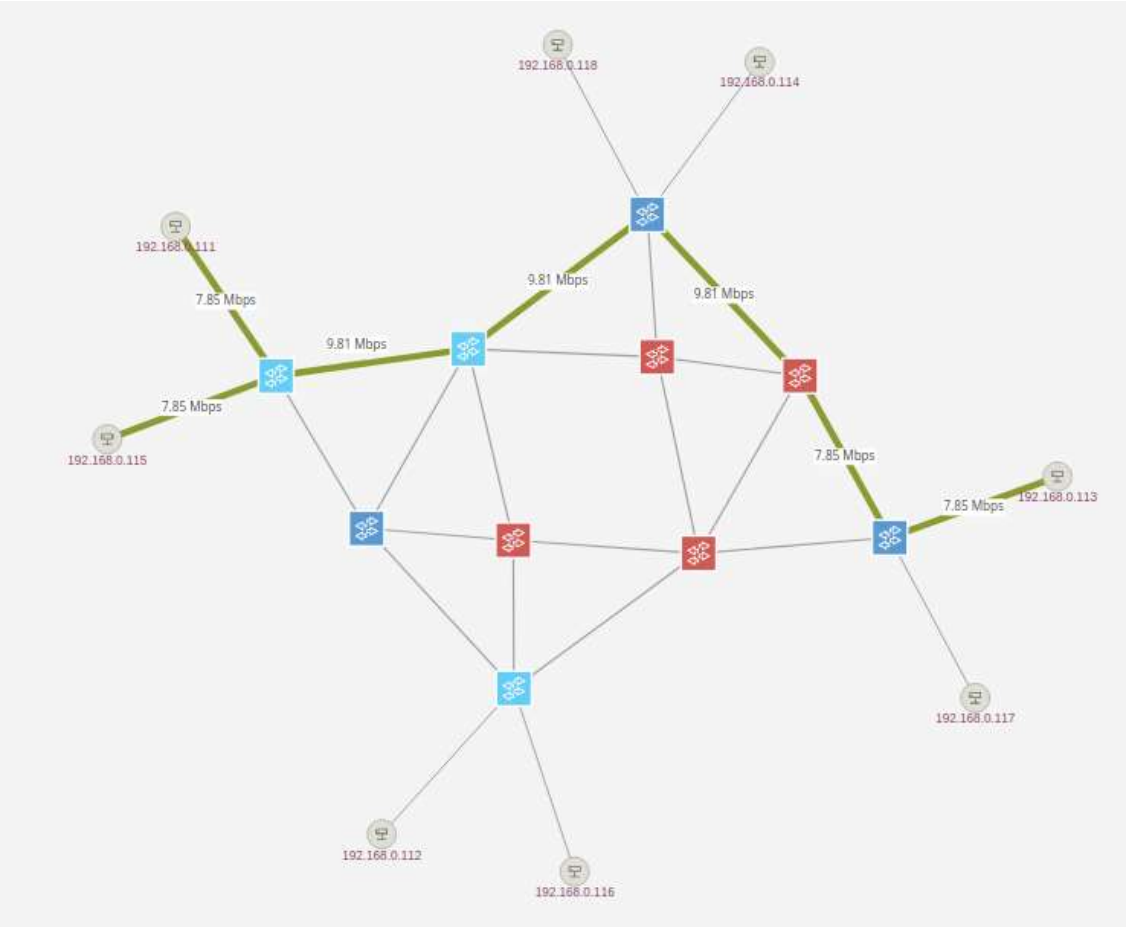

Рис. 5.52 Графічне відображення навантаження на канали зв'язку

Аналогічна процедура виконується для другої пари хостів. На рис. 5.53 продемонстровано оптимальний варіант маршрутизації, коли потоки даних проходять незалежними шляхами.

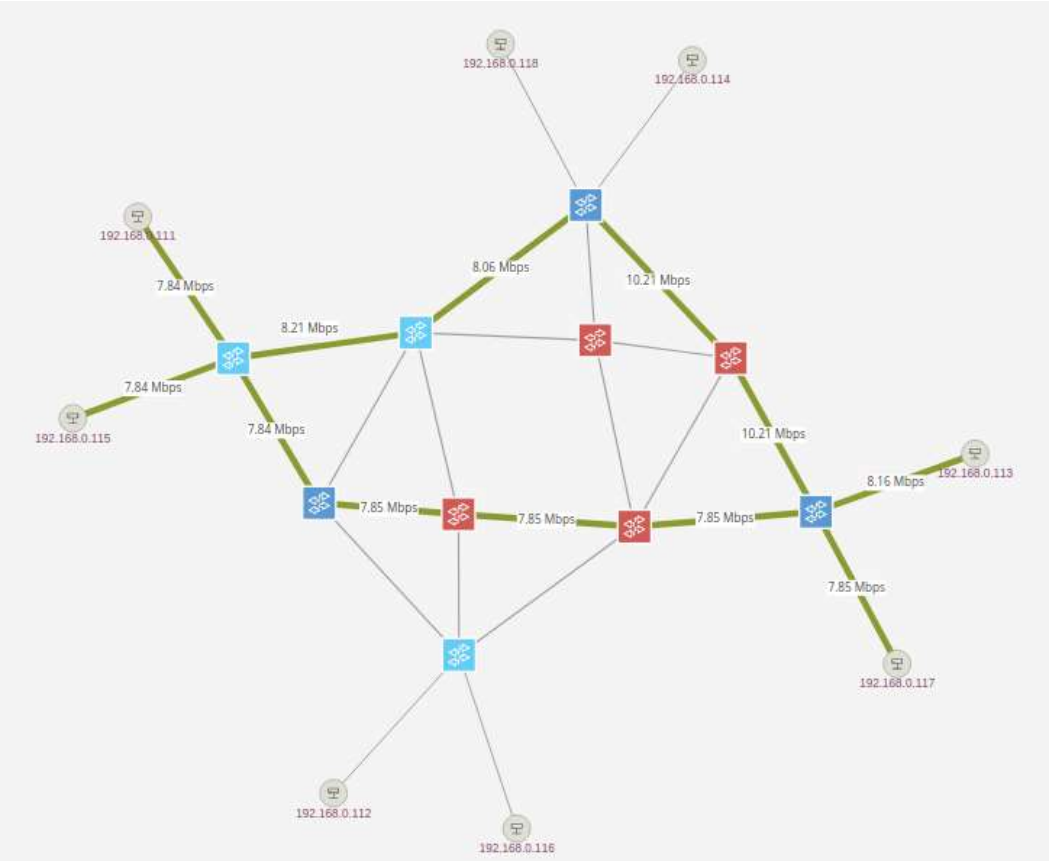

Рис. 5.53 Маршрутизація потоків по незалежних шляхах

За такої конфігурації втрати пакетів відсутні (рис. 5.54).

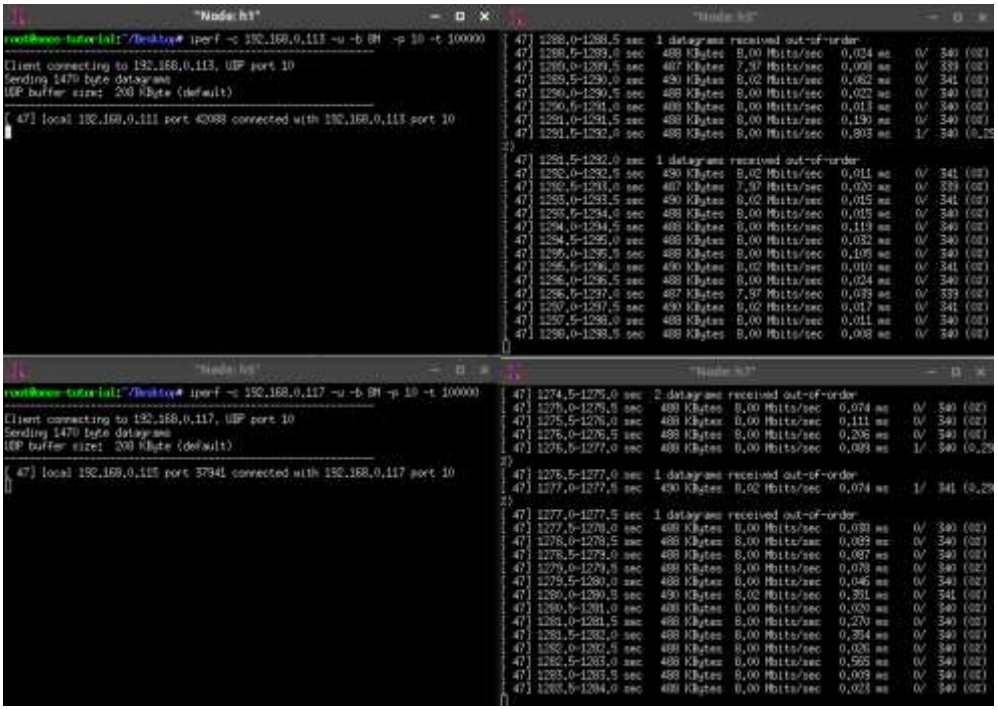

Рис. 5.54 Лог передачі даних без втрат

Проте, можливі випадки, коли автоматично побудовані маршрути мають

спільні сегменти (рис. 5.55).

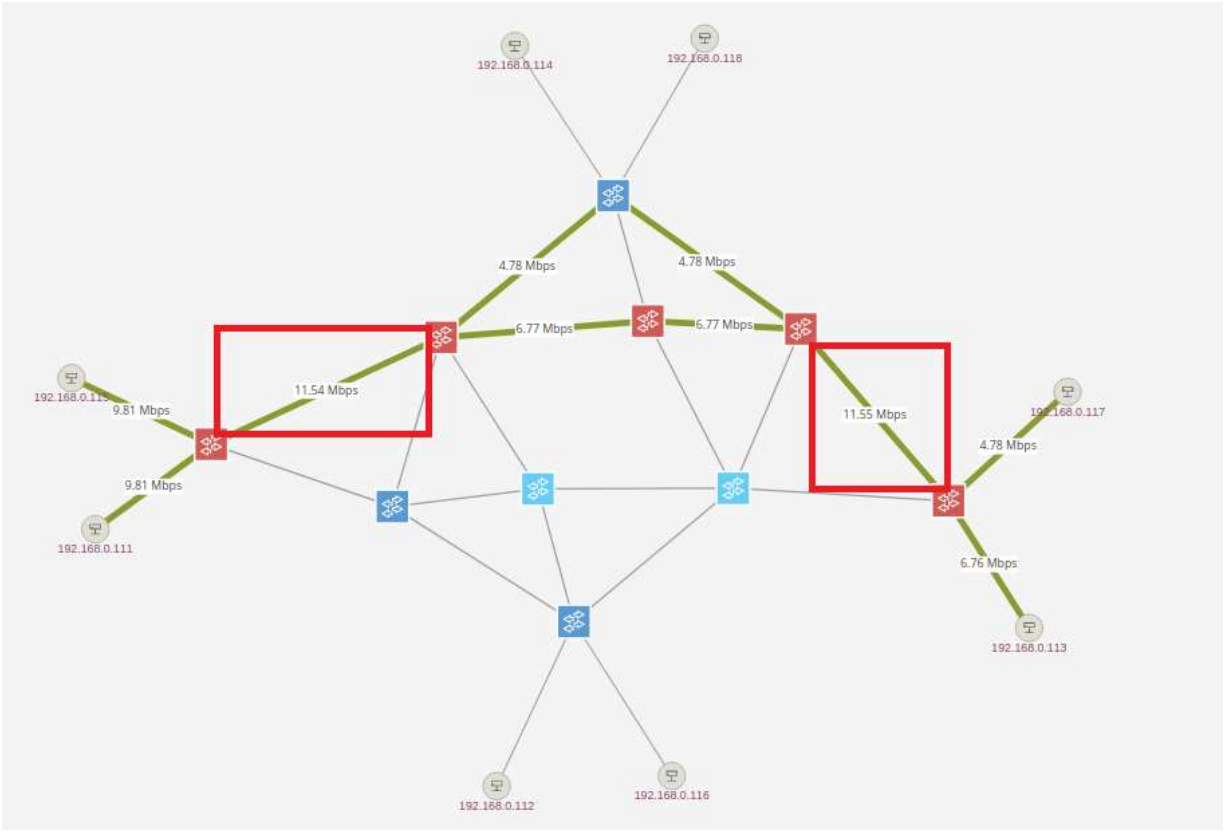

Рис. 5.55 Колізія маршрутів на спільних ділянках мережі

У разі виникнення перенавантаження на спільній ділянці мережі спостерігаються втрати пакетів через перевищення пропускної здатності каналу (рис. 5.56). Зміна маршруту здійснюється повторним створенням потоку (Host-to-host flow) до моменту обрання альтернативного шляху.

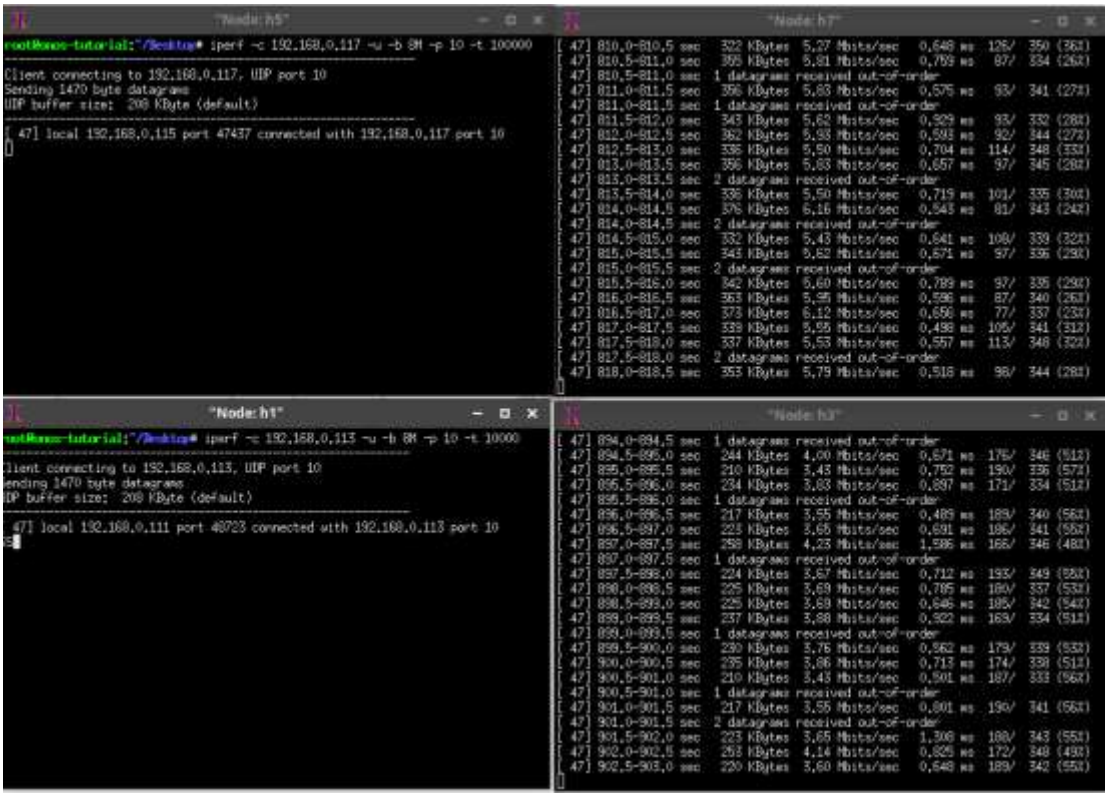

Рис. 5.56 Демонстрація втрат при перетині маршрутів

Висновки

У ході практичної реалізації сегмента SDN-мережі було розгорнуто віртуальний макет на базі кластера контролерів ONOS та середовища емуляції Mininet. Експериментально підтверджено працездатність ієрархічної архітектури системи, де відокремлений рівень управління (Control Plane) з трьох контролерів забезпечує стабільну координацію мережевих пристроїв та підтримує балансування навантаження на площину управління (Control Plane Load Balancing) через механізм майстер-контролерів.

Ключовим результатом експериментів стало дослідження поведінки мережі в умовах перевантаження каналів. Тестування за допомогою генератора трафіку iperf виявило недоліки базового механізму реактивної маршрутизації (додаток fwd), який при виборі найкоротших шляхів не враховує поточну утилізацію каналів, що призводить до колізій та значних втрат пакетів (понад 50%) при агрегації потоків на спільних лінках.

Практично доведено ефективність застосування Intent Framework для вирішення задач Traffic Engineering. Деактивація реактивного пересилання та

перехід до проактивного управління через встановлення host-to-host інтентів дозволили реалізувати логічне розділення трафіку. Примусове розведення потоків по незалежних фізичних маршрутах забезпечило повне усунення втрат даних та стабілізацію пропускної здатності, що підтверджує переваги централізованого підходу SDN над традиційними методами маршрутизації у задачах управління якістю обслуговування.

Реалізація кастомної топології за допомогою Python-скриптів продемонструвала гнучкість платформи для моделювання складних мережевих сценаріїв. Отримані результати свідчать про те, що комбінація візуальних засобів (ONOS GUI) та інструментів командного рядка (CLI) надає оператору повний контроль над станом мережі, дозволяючи оперативно виявляти "вузькі місця" та динамічно переконфігуровувати маршрути без фізичного втручання в інфраструктуру.

РОЗДІЛ 6

РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

Стартап як форма малого ризикового (венчурного) підприємництва впродовж останнього десятиліття набула широкого розповсюдження у світі через зниження бар'єрів входу в ринок і вважається однією із наріжних складових інноваційної економіки, оскільки за рахунок мобільності, гнучкості та великої кількості стартап-проектів загальна маса інноваційних ідей зростає. Опис ідеї проекту(таблиця 6.1,6.2)

Таблиця 6.1

Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Програмний додаток (ONOS Application) для інтелектуального управління трафіком (Traffic Engineering). Рішення використовує Intent Framework та модулі аналітики контролера ONOS для динамічної оптимізації мережевих потоків, балансування навантаження та гарантування якості обслуговування (QoS).	Сервіс-провайдери, великі центри обробки даних (Data Centers) та корпоративні мережі, які використовують або планують впровадження SDN-мереж на базі ONOS.	Максимізація ефективності мережі: Автоматична оптимізація шляхів та уникнення перевантажень. Гарантована якість сервісу (QoS): Резервування пропускної здатності для критичних додатків. Автоматизація: Спрощення управління мережею через високорівневі "інтенти" (політики) замість ручного налаштування пристроїв.

6.1 Технологічний аудит ідеї проекту

Таблиця 6.2

Технологічна здійсненність ідеї проекту

Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
Впровадження адаптивних алгоритмів балансування навантаження та обчислення шляхів.	Використання Path Computation Service (алгоритми Дейкстри, k-shortest paths) та Flow Objective Service контролера ONOS.	ONOS є платформою з відкритим кодом. Базові сервіси (Topology, Path, Flow) є в ядрі системи.	Архітектура ONOS (OSGi, мікросервіси) спеціально розроблена для створення нових додатків. Алгоритми TE потребують кастомної реалізації як модуль ONOS.
Технологія високорівневого управління політиками (Intents) для TE.	Використання Intent Framework для трансляції бізнес-вимог (напр., Bandwidth Intent, Protection Intent) у конкретні правила потоків.	Intent Framework є ключовим компонентом ONOS.	Технологія доступна через Northbound API контролера (напр., REST API), що дозволяє створити власний інтерфейс управління.
Модулі взаємодії з системами моніторингу та аналітики.	Інтеграція з Network Monitoring Service ONOS та використання сучасних протоколів телеметрії (напр., gNMI) для збору даних у реальному часі.	ONOS підтримує gNMI та має вбудовані сервіси моніторингу.	Технології доступні, але потребують налаштування та інтеграції з аналітичним ядром стартапу.
Підтримка користувачів та консалтинг з питань впровадження SDN та TE.	Побудова системи для роботи відділу з питань підтримки користувачів.	Найм співробітників у відділ з питань підтримки користувачів.	Рішення є доступним.

За попередніми оцінками, ринок систем управління SDN-мережами є дуже привабливим, але має ряд обмежень, вимог до високої надійності та потребує кваліфікованих кадрів.

Таблиця 6.3

Характеристики потенційних клієнтів стартап-проекту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
Потреба в оптимізації використання дорогих мережевих ресурсів, автоматизації управління складними мережами та гарантуванні якості обслуговування (QoS).	1. Сервіс-провайдери (ISPs, Telecom). 2. Великі центри обробки даних (Data Centers). 3. Великі корпоративні та кампусні мережі.	Сервіс-провайдери вимагають жорстких гарантій QoS та резервування пропускної здатності. Data Centers фокусуються на мультишляховій маршрутизації (ECMP) та мінімізації затримок. Корпорації потребують простого управління політиками (через Intents) для пріоритезації трафіку (напр., відеодзвінків).	1. Надійність: Висока доступність (HA) та інтеграція з кластером ONOS. 2. Продуктивність: Доведена ефективність у балансуванні навантаження та уникненні перевантажень. 3. Інтеграція: Підтримка Northbound API (REST) для інтеграції з існуючими OSS/BSS системами.

Таблиця 6.4.

Фактори загроз

Фактор	Зміст загрози	Можливі шляхи реакції компанії
Технологічна конкуренція	Поява альтернативних SDN-контролерів (напр., OpenDaylight) або вбудованих TE-рішень від великих вендорів.	Постійна інновація; фокус на унікальних перевагах ONOS (продуктивність, масштабованість) та глибині власних алгоритмів TE.
Цінова конкуренція	Коливання цін на послуги конкурентів.	Пошук шляхів зниження вартості послуг.

Продовження таблиці 6.4

Зниження доходів потенційних споживачів	Зниження купівельної спроможності клієнтів.	Вимушене зменшення обсягів виробництва.
Рівень інфляції	"Знецінювання коштів, ріст різниці в курсах валют."	Отримання довгострокового кредиту.

Даний проект має принципові можливості для роботи на ринку з огляду на конкурентну ситуацію.

Сильні сторони: Створення вузькоспеціалізованого, але потужного рішення для Traffic Engineering, що не має прямих аналогів серед відкритих додатків ONOS. Концентрація на автоматизації та QoS.

Можливості: Зростання ринку SDN; попит на автоматизацію мереж для зниження операційних витрат (OpEx). Стабілізація економічного стану сприятиме впровадженню інновацій.

Таблиця 6.5

Обґрунтування факторів конкурентоспроможності

Фактор конкурентоспроможності	"Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)"
Ціна та змінні витрати	Гнучка модель ліцензування (наприклад, підписка, плата за керований пристрій або за вузол кластера ONOS).
Розміри закупівель замовників	"Розміри закупівель легко масштабуються, оскільки продуктом є програмне рішення" (додаток ONOS), що працює на кластері будь-якого розміру.
Доступ до ресурсів у конкурентів	"Так як потенційні конкуренти (великі вендори) вже працюють на ринку, вони мають... більше ресурсів". Стартап повинен конкурувати гнучкістю та глибокою експертизою в ONOS.
Гнучкість цін на послуги	"Ціноутворення є гнучким", можливі різні пакети: (1) Базове балансування навантаження, (2) Просунутий TE з QoS, (3) Повний пакет з аналітикою.
Рівень концентрації послуг	"Спектр послуг... є більш вузьким". Висока концентрація виключно на ONOS Traffic Engineering.

Таблиця 6.6

Порівняльний аналіз сильних і слабих сторін

Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з запропонованим рішенням						
		-3	-2	-1	0	1	2	3
Ціна та змінні витрати	16							+
Розміризакупівель замовників	14		+					
Гнучкість цін на послуги	15							+
Рівень концентрації послуг	8	+						

Таблиця 6.7

Вибір цільових груп потенційних споживачів

Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті
Сервіс-провайдери (ISPs)	Висока. ONOS був розроблений для потреб сервіс-провайдерів.	Високий. Постійна потреба в оптимізації трафіку та QoS.	Висока (з боку пропрієтарних рішень), але низька для кастомних додатків ONOS.
Великі центри обробки даних (Data Centers)	Висока. Багато ЦОД вже використовують SDN.	Високий. Потреба в ефективному балансуванні навантаження (ECMP) та оптимізації енергоспоживання.	Висока.
Великі корпоративні та кампусні мережі	Середня. Поступовий перехід на SDN.	Середній. Зацікавленість у простоті управління (через Intents) та пріоритезації трафіку.	Середня.
Науково-освітні мережі (NRENs)	Висока. Часто є ранніми послідовниками технологій (приклад – топологія GEANT).	Низький/Середній. Потреба у специфічних, гарантованих шляхах для великих наукових потоків даних (Bandwidth Intents).	Низька.

На підставі ринкової стратегії обрано використання стратегії диференційованого маркетингу.

Таблиця 6.8

Визначення базової стратегіх розвитку

Обрана альтернатива розвитку	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Флангова атака (або Нішева стратегія)	Стратегія диференційованого маркетингу	Глибока експертиза в ONOS. Передові алгоритми ТЕ. Використання Intent Framework для автоматизації.	Стратегія диференціації

Висновки

Результатом проведеного аналізу та оцінки ризиків, було виявлено, що даний проект (інтелектуальний додаток ТЕ для ONOS) має можливість для ринкової комерціалізації.

До сильних сторін проекту можна віднести майже повну відсутність конкуренції на вітчизняному ринку у ніші відкритих додатків для ONOS та наявність лише великих, менш гнучких конкурентів (вендорів обладнання) на світовому ринку.

Також перевагою проекту є його наукова місткість та новизна (використання адаптивних алгоритмів та Intent-компіляторів). Варто відмітити актуальність теми автоматизації мереж (SDN) та тенденції до підвищення вимог до якості обслуговування (QoS).

Для описаного стартап-проекту є гарні перспективи входження в ринок, не зважаючи на наявність певних перешкод, таких як висока наукоємність галузі, потреба у висококваліфікованих SDN-інженерах та можлива інфляція.

Отже, можна зробити висновок про доцільність подальшої роботи над імплементацією даного проекту.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

У дипломній роботі вирішено актуальне науково-прикладне завдання підвищення ефективності функціонування програмно-конфігурованих мереж шляхом розробки та імплементації методів інженерії трафіку на базі контролера ONOS. Основні результати роботи полягають у наступному:

Проведено аналіз архітектури SDN-мереж, в результаті якого встановлено, що використання контролера ONOS дозволяє вирішити проблему масштабованості та надійності, властиву раннім SDN-рішенням. Визначено, що кластерна архітектура та мікросервісний підхід забезпечують високу доступність сервісів (High Availability), що є критичним для мереж операторського класу.

Досліджено екосистему протоколів взаємодії, що дозволило виявити ефективність гібридної моделі узгодженості даних в ONOS. Встановлено, що комбінація протоколу Raft для критичних операцій (конфігурація, майстерність) та протоколу Gossip для синхронізації стану (метрики, топологія) забезпечує оптимальний баланс між швидкістю реакції системи та надійністю зберігання даних. Для задач телеметрії обґрунтовано переваги протоколу gNMI, який дозволяє отримувати дані про стан каналів у реальному часі.

Виконано порівняльний аналіз методів інженерії трафіку, який показав обмеженість базових алгоритмів найкоротшого шляху (Dijkstra) для забезпечення QoS у навантажених мережах. Доведено, що використання Intent Framework у поєднанні з алгоритмами пошуку k-найкоротших шляхів (алгоритм Єна) дозволяє реалізовувати складні сценарії резервування смуги пропускання та обходу перевантажених ділянок без необхідності низькорівневого маніпулювання правилами потоків.

Експериментально підтверджено неефективність реактивних методів маршрутизації (додаток fwd) в умовах високого навантаження, де зафіксовано втрати пакетів понад 50% через колізії на спільних каналах. Натомість запропонований підхід із використанням проактивних інтентів (host-to-host intents) дозволив примусово розділити потоки трафіку по незалежних фізичних маршрутах, що повністю усунуло втрати пакетів та стабілізувало роботу мережі.

Побудовано повнофункціональний віртуальний макет мережі в середовищі Mininet під управлінням кластера ONOS. Розроблено комплект скриптів на Python для автоматизації розгортання кастомних топологій та сценаріїв тестування, що може бути використаний як інструментарій для подальшого проектування та верифікації мережевих рішень перед їх впровадженням у фізичну інфраструктуру.

Розроблено стартап-проект комерціалізації створеного програмного модуля Traffic Engineering. Проведений маркетинговий аналіз підтвердив наявність попиту на рішення для автоматизації управління трафіком серед інтернет-провайдерів та центрів обробки даних, а розраховані показники вказують на економічну доцільність проекту за умови використання стратегії диференціації.

У підсумку, запропоновані в роботі методи та практичні рекомендації дозволяють створювати відмовостійкі, керовані та ефективні SDN-мережі, здатні динамічно адаптуватися до змін навантаження та забезпечувати високу якість обслуговування користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Romanov, O., Nesterenko, M., Mankivskyi, V., Zhuk, O. Principles of Building Modular Control Plane in Software-Defined Network//Lecture Notes in Networks and Systems, 2023, 548, pp. 333–355, DOI: 10.1007/978-3-031-16368-5_17, https://link.springer.com/chapter/10.1007/978-3-031-16368-5_17
2. Romanov, O., Nesterenko, M., Marinov, A., Skolets, S., Burlaka, H. SDN network modeling using the GUI MiniEdit/Proceedings - 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering, TCSET 2022, 2022, стр. 637–642
3. Romanov, O., Korniienko, N., Burlaka, H. Construction of the SDN Transport Network Model using the T-API Interface/2021 IEEE 4th International Conference on Advanced Information and Communication Technologies, AICT 2021 - Proceedings, 2021, стр. 220–224
4. Romanov, O., Siemens, E., Nesterenko, M., Mankivskyi, V. Mathematical description of control problems in SDN networks/Proceedings of International Conference on Applied Innovation in IT, 2021, 9(1), стр. 33–39
5. Romanov O. et al. Optical Network Management by ONOS-Based SDN Controller // Radiotekhnika. 2022. No. 210. P. 188–196.
6. Yen J. Y. Finding the K Shortest Loopless Paths in a Network // Management Science. 1971. Vol. 17, No. 11. P. 712–716.
7. Cherevatenko O. et al. Creation of the method of multipath routing using known paths in software-defined networks // Technology Audit and Production Reserves. 2022. Vol. 4, No. 2(66). P. 19–24.
8. Geeksforgeeks. [Електронний ресурс]. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/software-engineering/open-networking-operating-system-onos-in-software-defined-networks/
9. ONOS wiki Documentation. [Електронний ресурс]. – Режим доступу до ресурсу: https://wiki.onosproject.org/display/ONOS/ONOS+Documentation
10. ONOS wiki Guide. [Електронний ресурс]. – Режим доступу до ресурсу: https://wiki.onosproject.org/display/ONOS/Developer+Guide

11. ONOS wiki Tutorial. [Електронний ресурс]. – Режим доступу до ресурсу: https://wiki.onosproject.org/display/ONOS/Basic+ONOS+Tutorial
12. Mininet documentation. [Електронний ресурс]. – Режим доступу до ресурсу: https://github.com/mininet/mininet/wiki/Documentation
13. Python documentation. [Електронний ресурс]. – Режим доступу до ресурсу: https://www.python.org/doc/

ДОДАТКИ

Додаток А

Лістинг Python-скрипта

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.node import Controller, OVSSwitch, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel

def multiControllerNet():
net = Mininet( controller=RemoteController, switch=OVSSwitch,
link=TCLink )
print '*** Creating (reference) controllers'
c1 = RemoteController( 'c1', ip='172.17.0.5', port=6653 )
c2 = RemoteController( 'c2', ip='172.17.0.6', port=6653 )
c3 = RemoteController( 'c3', ip='172.17.0.7', port=6653 )

print '*** Creating switches'
s10 = net.addSwitch('s10', dpid='0000000000000010',
mac='00:00:00:00:SW:10')
s11 = net.addSwitch('s11', dpid='0000000000000020',
mac='00:00:00:00:SW:11')
s12 = net.addSwitch('s12', dpid='0000000000000030',
mac='00:00:00:00:SW:12')
s13 = net.addSwitch('s13', dpid='0000000000000040',
mac='00:00:00:00:SW:13')
s14 = net.addSwitch('s14', dpid='0000000000000011',
mac='00:00:00:00:SW:14')
s15 = net.addSwitch('s15', dpid='0000000000000012',
mac='00:00:00:00:SW:15')
s16 = net.addSwitch('s16', dpid='0000000000000013',
mac='00:00:00:00:SW:16')
s17 = net.addSwitch('s17', dpid='0000000000000021',
mac='00:00:00:00:SW:17')
s18 = net.addSwitch('s18', dpid='0000000000000022',
mac='00:00:00:00:SW:18')
s19 = net.addSwitch('s19', dpid='0000000000000023',
mac='00:00:00:00:SW:19')

print '*** Creating hosts'
h1 = net.addHost('h1', ip='192.168.0.111/24',
mac='00:00:00:00:00:01')
h2 = net.addHost('h2', ip='192.168.0.112/24',
mac='00:00:00:00:00:02')
h3 = net.addHost('h3', ip='192.168.0.113/24',
mac='00:00:00:00:00:03')
h4 = net.addHost('h4', ip='192.168.0.114/24',
mac='00:00:00:00:00:04')
```

```
h5 = net.addHost('h5', ip='192.168.0.115/24',
mac='00:00:00:00:00:05')
h6 = net.addHost('h6', ip='192.168.0.116/24',
mac='00:00:00:00:00:06')
h7 = net.addHost('h7', ip='192.168.0.117/24',
mac='00:00:00:00:00:07')
h8 = net.addHost('h8', ip='192.168.0.118/24',
mac='00:00:00:00:00:08')

print '*** Creating links'
net.addLink(s10, s14, bw=10)
net.addLink(s10, s17, bw=10)
net.addLink(s17, s14, bw=10)
net.addLink(s17, s16, bw=10)
net.addLink(s17, s18, bw=10)
net.addLink(s18, s16, bw=10)
net.addLink(s18, s15, bw=10)
net.addLink(s16, s14, bw=10)
net.addLink(s16, s15, bw=10)
net.addLink(s14, s13, bw=10)
net.addLink(s14, s19, bw=10)
net.addLink(s19, s13, bw=10)
net.addLink(s19, s12, bw=10)
net.addLink(s13, s12, bw=10)
net.addLink(s13, s15, bw=10)
net.addLink(s15, s11, bw=10)
net.addLink(s15, s12, bw=10)
net.addLink(s11, s12, bw=10)

net.addLink(s10, h1, bw=10)
net.addLink(s18, h2, bw=10)
net.addLink(s11, h3, bw=10)
net.addLink(s19, h4, bw=10)
net.addLink(s10, h5, bw=10)
net.addLink(s18, h6, bw=10)
net.addLink(s11, h7, bw=10)
net.addLink(s19, h8, bw=10)

print '*** Starting network'
net.start()
c1.start()
c2.start()
c3.start()
s10.start([c1, c2, c3])
s11.start([c1, c2, c3])
s12.start([c1, c2, c3])
s13.start([c1, c2, c3])
s14.start([c1, c2, c3])
s15.start([c1, c2, c3])
s16.start([c1, c2, c3])
s17.start([c1, c2, c3])
s18.start([c1, c2, c3])
s19.start([c1, c2, c3])
```

```
CLI( net )
net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )  # for CLI output
    multiControllerNet()
```