Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(освітній рівень)

на тему: "Централізоване управління політиками безпеки у SDN із використанням механізмів фільтрації доступу "

Виконав: студент VI курсу, групи СБм-61
Спеціальності:
125 Кібербезпека та захист інформації
(шифр і назва напряму підготовки, спеціальності)
Стрихарський Володимир Миколайович
підпис (прізвище та ініціали)

Керівник Оробчук О. Р.
підпис (прізвище та ініціали)

Нормоконтроль Стадник М. А.
підпис (прізвище та ініціали)

Завідувач кафедри Загородна Н.В.
підпис (прізвище та ініціали)

Рецензент
підпис (прізвище та ініціали)

м. Тернопіль – 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Загородна Н.В.
(підпис) (прізвище та ініціали)
«__» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека та захист інформації
(шифр і назва спеціальності)

Студенту Стрихарському Володимиру Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Централізоване управління політиками безпеки у SDN із використанням механізмів фільтрації доступу

Керівник роботи Оробчук Олександра Романівна, доктор філософії, старший викладач кафедри КБ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 24 » 11 2025 року № 4/7-1024

2. Термін подання студентом завершеної роботи 12.12.2025

3. Вихідні дані до роботи Архітектура програмно-визначеної мережі, перелік її критичних активів і каналів взаємодії, а також вимога забезпечити централізоване управління політиками безпеки з використанням механізмів фільтрації доступу.

4. Зміст роботи (перелік питань, які потрібно розробити)
Формування моделі активів програмно-визначеної мережі, аналіз загроз і вразливостей для відповідних площин та каналів взаємодії, розроблення методики кількісної оцінки ризиків, проєктування архітектури централізованого механізму фільтрації доступу, створення моделі станів з'єднання для побудови правил OpenFlow та реалізацію і експериментальну перевірку прототипу SDN-фаєрвола із перевіркою стану.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Актуальність дослідження. Мета, об'єкт, предмет дослідження.
Наукова новизна та практичне значення.
Завдання дослідження.
Архітектурні особливості SDN. Схема конвеєра обробки даних на базі OpenFlow.
Архітектура фаєрвола для SDN. Схема роботи оркестратора в SDN.
Модель SDN-еквіваленту скінченного автомату. Реактивна та проактивна поведінка фаєрвол-додатка в SDN. Загальна схема стенда тестування SDN фаєрвола. Середній час встановлення з'єднань за різних факторів. Частка повторно переданих контрольних TCP-пакетів
Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Г.М., к.т.н., доцент		
Безпека в надзвичайних ситуаціях	Теслюк В.М., проректор з адміністративно-господарської роботи та будівництва		

7. Дата видачі завдання 19.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	20.09 – 22.09	Виконано
2.	Підбір джерел для аналізу в галузі дослідження	25.09 – 10.10	Виконано
3.	Опрацювання джерел в галузі дослідження	11.10 – 15.10	Виконано
4.	Налаштування симуляційного середовища	16.10 – 25.10	Виконано
5.	Оформлення розділу «Аналіз уразливостей у програмно-визначених мережах»	30.10 – 05.11	Виконано
6.	Оформлення розділу «Архітектура SDN-фаєрвола з перевіркою стану з’єднання»	06.11 – 10.11	Виконано
7.	Оформлення розділу «Оцінка ефективності SDN-фаєрвола та оркестрація політик»	11.11 – 25.11	Виконано
8.	Виконання завдання до підрозділу «Охорона праці та безпека в надзвичайних ситуаціях»	26.11-01.12	Виконано
9.	Оформлення кваліфікаційної роботи	02.12 – 10.12	Виконано
10.	Нормоконтроль	08.12 – 10.12	Виконано
11.	Перевірка на плагіат	12.12 – 14.12	Виконано
12.	Попередній захист кваліфікаційної роботи	15.12 – 20.12	Виконано
13.	Захист кваліфікаційної роботи	23.12.2025	

Студент _____ Стрихарський В. М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Оробчук О. Р.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Централізоване управління політиками безпеки у SDN із використанням механізмів фільтрації доступу // ОР «Магістр» // Стрихарський Володимир Миколайович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБм-61 // Тернопіль, 2025 // С. 96, рис. – 19, табл. – 1, кресл. – 15, додат. – 3.

Ключові слова: SDN, cybersecurity, stateful firewall, OpenFlow, RYU, CVSS, AHP, policy orchestration, risk analysis, virtualized environment.

У кваліфікаційній роботі магістра досліджено питання підвищення рівня кібербезпеки у програмно-визначених мережах (SDN) шляхом створення моделі централізованого керування політиками безпеки та реалізації SDN-фаєрвола із перевіркою стану з'єднання. Обґрунтовано актуальність переходу від статичних механізмів фільтрації до динамічних систем, здатних адаптуватися до поточного стану мережі та контексту трафіку. Розроблено архітектуру та програмну реалізацію SDN-фаєрвола із перевіркою стану з'єднання, побудованого на базі протоколу OpenFlow і контролера RYU. Механізм перевірки стану реалізовано у вигляді розширеного скінченного автомата, який відстежує життєвий цикл сесій, контролює ініціацію, підтримку та завершення з'єднань, а також реагує на аномалії. Запропоновано оркестрацію політик безпеки, що забезпечує централізовану синхронізацію правил між контролером і комутаторами. Виконано експериментальну оцінку ефективності запропонованого рішення на віртуалізованому стенді з використанням Mininet, Open vSwitch та RYU. Проведено моделювання атак типу SYN-Flood і легітимного трафіку, виміряно метрики часу обробки пакетів, коефіцієнта блокування атак, навантаження на ресурси та пропускної здатності. Отримані результати показали зменшення затримок обробки трафіку та підвищення точності виявлення атак у порівнянні з традиційними ACL-механізмами.

ABSTRACT

Centralized security policy management in SDN using access-filtering mechanisms // Thesis of educational level "Master"// Volodymyr Strykharskyi // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cybersecurity, group СБм-61 // Ternopil, 2025 // p. 96, figs. 19, tbls. 1, drws. 14, apps. 3.

Keywords: SDN, cybersecurity, stateful firewall, OpenFlow, RYU, CVSS, AHP, policy orchestration, risk analysis, virtualized environment.

The master's qualification work investigates the problem of enhancing cybersecurity in Software-Defined Networks (SDN) by developing a model of centralized security policy management and implementing an SDN firewall with connection state inspection. The relevance of transitioning from static filtering mechanisms to dynamic systems capable of adapting to the current network state and traffic context is substantiated. An architecture and software implementation of an SDN stateful firewall have been developed based on the OpenFlow protocol and the RYU controller. The state inspection mechanism is implemented as an extended finite state machine (EFSM) that tracks the session life cycle, controls the initiation, maintenance, and termination of connections, and responds to anomalies. A policy orchestration mechanism is proposed, ensuring centralized synchronization of access rules between the controller and switches. An experimental evaluation of the proposed solution was carried out on a virtualized testbed using Mininet, Open vSwitch, and RYU. Simulations of SYN-Flood attacks and legitimate traffic were conducted, and metrics such as packet processing time, attack blocking ratio, resource utilization, and throughput were measured. The obtained results demonstrate reduced traffic processing delays and improved accuracy of attack detection compared to traditional ACL-based mechanisms.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

SDN	—	Software-Defined Networking
IDS	—	Intrusion Detection System
SDNEFSM	—	Software-Defined Network Element Finite State Machine
CVSS	—	Common Vulnerability Scoring System
AHP	—	Analytic Hierarchy Process
OSI	—	Open Systems Interconnection
MAC	—	Media Access Control
SYN	—	Synchronize
ACK	—	Acknowledgment
FIN	—	Finish
RST	—	Reset
DPI	—	Deep Packet Inspection
QoS	—	Quality of Service
FSM	—	Finite State Machine
EFSM	—	Extended Finite State Machine
OVS	—	Open vSwitch
SIEM	—	Security Information and Event Management
PPT	—	Packet Processing Time
ABAC	—	Attribute-Based Access Control
OPT	—	Orchestration Processing Time
FAPT	—	Firewall Application Processing Time
CPT	—	Controller Processing Time
IPT	—	Infrastructure Processing Time
PPTT	—	Policy Processing Total Time
OSR	—	Orchestrator Setup Rate
FSR	—	Firewall Setup Rate
ISR	—	Infrastructure Setup Rate

ВСТУП

Актуальність теми. Стрімка цифровізація критичних інфраструктур і сервісів у хмарних та віртуалізованих середовищах посилює залежність організацій від мережевих платформ з програмно-визначеним керуванням (SDN). Разом із гнучкістю оркестрації політик доступу це створює нові площини атак: від зловживань у площині керування до виснаження ресурсів віртуальних комутаторів і помилок пріоритизації правил. Класичні підходи, такі як статичні фільтри пакетів та сигнатурні IDS демонструють обмежену ефективність у динамічних топологіях та під час багатовекторних атак. Тому актуальним є розроблення централізованого керування політиками безпеки та реалізації SDN-фаєрвола із перевіркою стану з'єднання, що враховують специфіку SDN-архітектур.

Мета і задачі дослідження. Підвищити узгодженість та ефективність забезпечення мережевої безпеки в SDN за рахунок централізованого керування політиками доступу та впровадження фаєрвола із перевіркою стану з'єднання.

Для досягнення цієї мети необхідно вирішити такі задачі:

- дослідити активи і загрози у SDN;
- реалізувати прототип SDN-фаєрвола із перевіркою стану з'єднання;
- забезпечити централізовану оркестрацію політик безпеки;
- розробити сценарії моделювання атак і легітимного трафіку;
- виміряти ключові показники ефективності системи безпеки.

Об'єкт дослідження. Процеси забезпечення мережевої безпеки у програмно-визначених мережах з централізованим керуванням політиками доступу.

Предмет дослідження. Методи й засоби забезпечення виконання політик безпеки в SDN, зокрема фаєрвол із перевіркою стану з'єднання.

Наукова новизна одержаних результатів кваліфікаційної роботи. Удосконалено методику формалізації станів з'єднання для SDN-фаєрвола у вигляді спеціалізованого скінченного автомата (SDNEFSM), який інтегрується з правилами OpenFlow і підтримує контекстні події (ініціація, встановлення,

завершення, аномалії таймаутів). Удосконалено механізм оркестрації політик безпеки у SDN, що дозволяє централізовано синхронізувати зміни між контролером, віртуальними комутаторами й модулями фаєрвола.

Практичне значення одержаних результатів. Запропонований SDN-фаєрвол із перевіркою стану з'єднання та оркестрація політик дає змогу підвищити точність фільтрації в порівнянні зі статичними ACL у динамічних топологіях, зменшити поверхню атак, пов'язаних із неправильними таймаутами/пріоритетами потоків, скоротити час реакції на інциденти завдяки централізованому керуванню. Результати можуть бути впроваджені у приватних дата-центрах, кампусних мережах та сегментах критичної інфраструктури, що переходять на SDN/NFV.

Апробація результатів магістерської роботи. Основні результати дослідження були представлені на XIII науково-технічній конференції «Інформаційні моделі, системи та технології» (ТНТУ, Тернопіль, Україна, 17-18 грудня 2025 р).

Публікації. Основні результати кваліфікаційної роботи опубліковано у працях конференції (див. Додаток А).

РОЗДІЛ 1 АНАЛІЗ УРАЗЛИВОСТЕЙ У ПРОГРАМНО-ВИЗНАЧЕНИХ МЕРЕЖАХ

1.1 Сучасні архітектурні особливості SDN та виклики безпеки

Сучасний розвиток мережевих технологій та швидке зростання обсягів трафіку вимагають нових підходів до побудови інфраструктури зв'язку. Традиційна архітектура комп'ютерних мереж, заснована на вертикальній інтеграції рівнів керування, пересилання даних та обробки застосунків, виявилася недостатньо гнучкою для потреб хмарних сервісів, мобільних обчислень, Інтернету речей та інших масштабованих середовищ. У такій архітектурі кожен мережевий пристрій поєднує у собі апаратне забезпечення, програмну логіку та локальний контроль, що призводить до складності конфігурації, високої вартості модернізації та неможливості швидкого впровадження інновацій. Залежність від постачальників устаткування, різнорідність протоколів і обмежені можливості централізованого керування стали серйозними обмеженнями для розвитку інформаційних сервісів.

Перехід до програмно-визначених мереж (SDN) став відповіддю на цю проблему. Основна ідея SDN полягає у відокремленні площини керування від площини даних, що дозволяє перенести інтелектуальні рішення з окремих мережевих елементів у централізований програмний контролер [1]. Такий підхід усуває жорстку прив'язку між апаратною частиною та функціями керування, відкриваючи можливість програмного налаштування мережевої поведінки відповідно до поточних потреб користувачів або сервісів. Унаслідок цього мережа перетворюється на динамічну платформу, яку можна керовано перебудовувати без фізичного втручання у пристрої.

Програмна визначеність означає, що рішення щодо маршрутизації, пріоритезації трафіку чи застосування політик безпеки приймаються в одному логічному центрі, а мережеві елементи лише виконують отримані інструкції. Концептуальна різниця між класичною архітектурою та SDN показана на рисунку 1.1

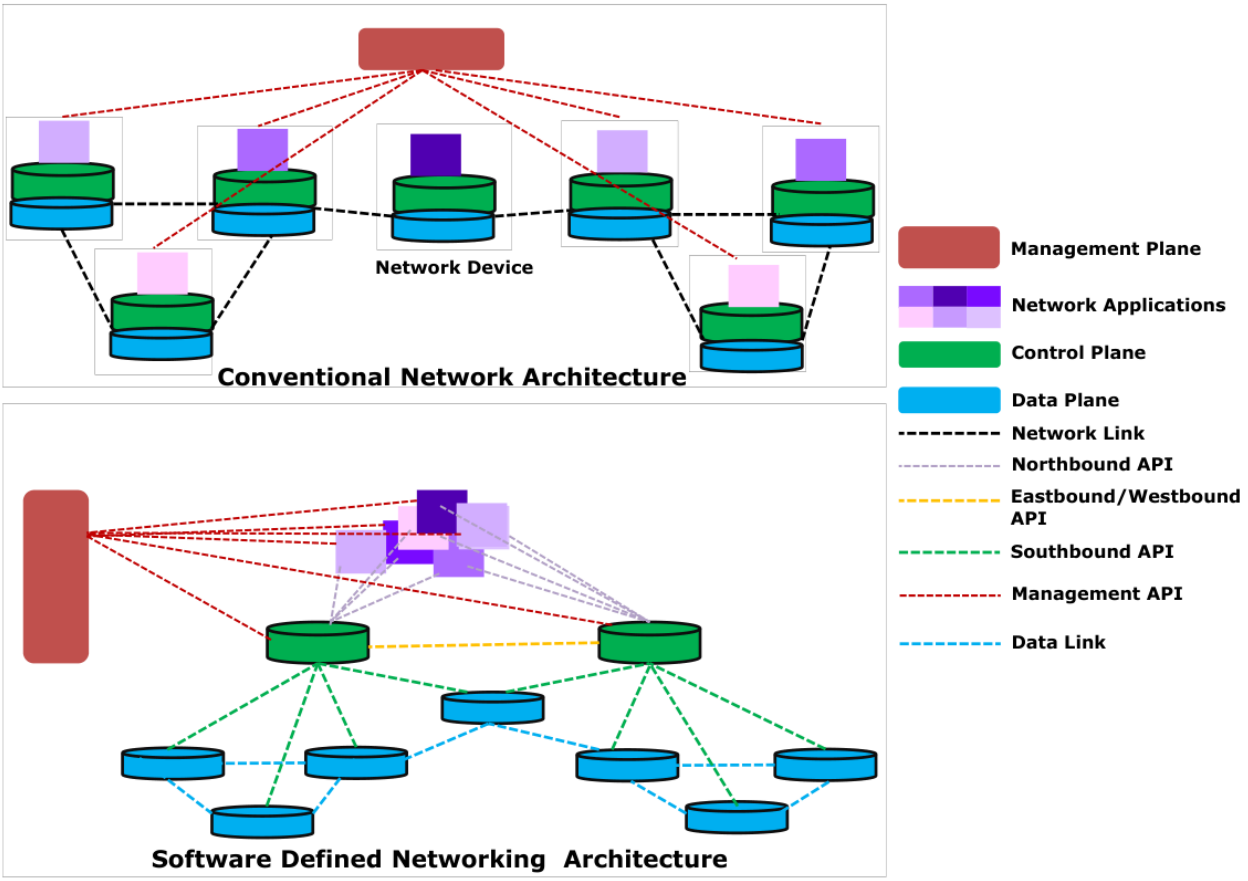


Рисунок 1.1 – Класична архітектура та SDN

У традиційній моделі логіка контролю й пересилання інформації зосереджені у кожному пристрої, тоді як у SDN ці функції відокремлені та взаємодіють через стандартизовані програмні інтерфейси.

Такий підхід створює основу для переходу від фрагментованої мережевої екосистеми до гнучкої, відкритої та керованої програмно інфраструктури. Контролер SDN забезпечує глобальне бачення мережі, збираючи інформацію про стан усіх вузлів і каналів, тоді як прикладні сервіси можуть безпосередньо взаємодіяти з ним через північний інтерфейс (Northbound API) [2]. Завдяки цьому мережа перетворюється на своєрідну операційну систему, у якій можна реалізовувати політики маршрутизації, балансування навантаження або безпеки, використовуючи єдині абстракції.

Архітектура SDN складається з трьох основних площин - даних, керування і застосунків, а також допоміжної площини управління ресурсами [3]. Кожна з них виконує окрему функцію та взаємодіє з іншими через відкриті інтерфейси.

Площина даних включає фізичні та віртуальні комутатори, маршрутизатори й інші пристрої, що відповідають за пересилання трафіку. Вони реалізують набір простих функцій таких, як прийом, обробку й передачу пакетів відповідно до правил, які надходять від контролера. Самі по собі ці елементи не мають інформації про глобальний стан мережі й не приймають рішень щодо маршрутизації, що значно спрощує їхню структуру й знижує ймовірність помилок у налаштуванні.

Площина керування реалізується програмним контролером, який є центральним компонентом SDN та формує політику функціонування мережі. Контролер збирає інформацію про топологію, стан каналів і пристроїв, приймає рішення про маршрутизацію, балансування навантаження, пріоритети трафіку та реалізацію політик безпеки. На практиці контролер може бути як фізично централізованим, так і розподіленим, але логічно він завжди виступає єдиним джерелом керування.

Площина застосунків забезпечує взаємодію користувацьких сервісів з контролером. У цьому шарі працюють програми, що реалізують аналітику, моніторинг, безпеку або оптимізацію мережевих ресурсів. Вони взаємодіють із контролером через північний інтерфейс, використовуючи високорівневі команди для формування політик і сценаріїв поведінки.

Для з’єднання між шарами використовуються відкриті інтерфейси. Південний інтерфейс (Southbound API) з’єднує контролер із пристроями площини даних, передаючи їм інструкції у вигляді таблиць потоків або правил маршрутизації. Найпоширенішим стандартом для цієї взаємодії є OpenFlow, який визначає структуру таблиць потоків і механізм обміну повідомленнями між контролером і мережевими елементами [4].

На рисунку 1.2 показано спрощену схему конвеєра обробки пакетів на базі OpenFlow. Пакет, що надходить на елемент мережі, послідовно проходить через набір таблиць, де його заголовок порівнюється з полями правил. Знайдене співпадіння визначає набір дій таких, як передати пакет, змінити його, відкинути або передати контролеру.

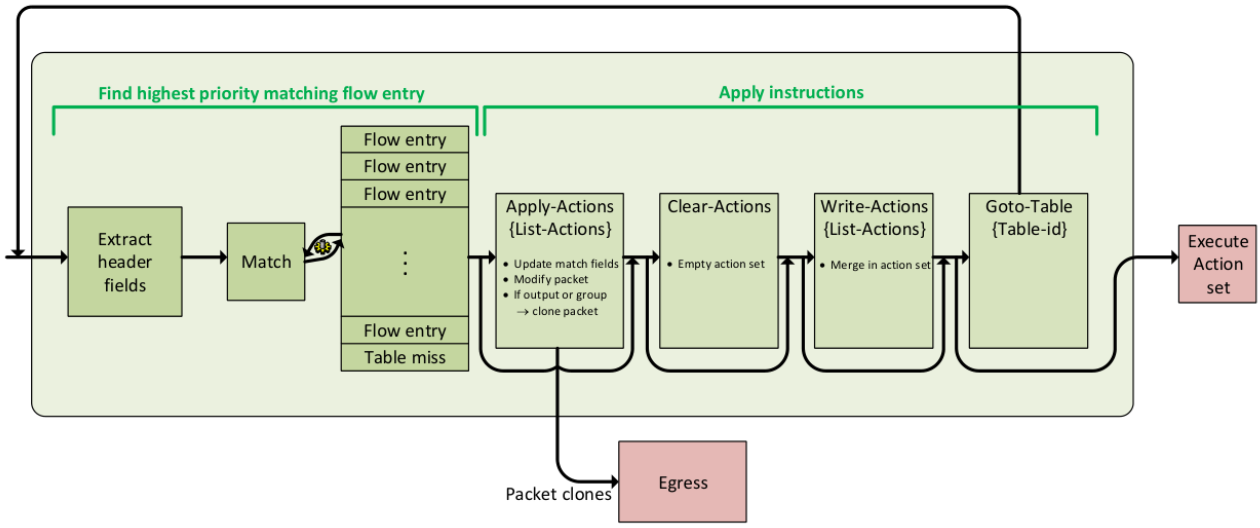


Рисунок 1.2 – Схема конвеєра обробки на базі OpenFlow

Якщо жодне правило не збігається, спрацьовує так зване table-miss правило, яке зазвичай пересилає пакет на обробку контролеру. Така модель дає змогу централізовано формувати політики керування потоками та забезпечує передбачувану реакцію мережі на події.

Перевагою SDN є також можливість програмованості. Завдяки відкритим інтерфейсам мережеві застосунки можуть задавати поведінку мережі за допомогою високорівневих абстракцій, а контролер транслює ці абстракції у конкретні команди для пристроїв. Такий підхід забезпечує гнучкість, можливість швидкого розгортання нових сервісів і централізований контроль над усією інфраструктурою. Ще однією принциповою особливістю є федеративність. Вона означає відкритість архітектури й стандартизацію інтерфейсів, що дозволяє різним постачальникам розробляти сумісні контролери, мережеві елементи та застосунки. Це ліквідує ефект “vendor lock-in” - залежність від одного виробника і створює умови для широкої інтеграції нових технологій [5].

Перехід до SDN приніс низку переваг порівняно з традиційними мережами. По-перше, централізація керування забезпечує глобальне бачення стану мережі, що дозволяє приймати узгоджені рішення щодо маршрутизації й безпеки. Контролер володіє повною інформацією про потоки трафіку, що дає змогу оптимізувати використання ресурсів, зменшувати затримки й уникати конфліктів між політиками. По-друге, автоматизація конфігурації знижує

кількість помилок, пов’язаних із людським фактором. У традиційних мережах адміністратори вручну налаштовують десятки пристроїв, тоді як у SDN зміна політики може бути здійснена централізовано одним викликом API, після чого контролер самостійно оновлює таблиці правил на всіх вузлах. По-третє, відкритість архітектури сприяє інноваціям. Розробники можуть створювати нові сервіси без необхідності змінювати апаратну частину. Наприклад, можна реалізувати алгоритми розподілу навантаження або системи виявлення вторгнень як звичайні програмні модулі, інтегровані в контролер. Нарешті, SDN забезпечує гнучкість і масштабованість. Мережу можна легко розширювати, додаючи нові пристрої без зміни її логічної структури, оскільки кожен мережевий елемент підключається до контролера за стандартним протоколом.

Попри очевидні переваги, перехід до програмно-визначених мереж створює нові виклики для кібербезпеки. Розділення площин керування й даних, введення додаткових рівнів абстракції та відкритих інтерфейсів значно розширюють площину потенційних атак. Одним із головних ризиків є централізація контролю. Контролер SDN стає критичним елементом інфраструктури та єдиною логічною точкою, від якої залежить робота всієї мережі. Компрометація або перевантаження контролера може призвести до втрати керування всією системою. На відміну від традиційних мереж, де вихід з ладу одного маршрутизатора не впливає на інші, у SDN атака на контролер потенційно вражає всі підлеглі вузли. Другий аспект стосується розширення площини атаки через відкриті інтерфейси. Southbound API, Northbound API та східно-західний (East/West API) інтерфейси забезпечують взаємодію між різними площинами, але водночас відкривають можливості для зловживань [6]. Якщо зловмисник отримує доступ до Southbound-каналу, він може змінювати таблиці потоків і, таким чином, перенаправляти або блокувати трафік. Недостатня автентифікація на Northbound API дозволяє маніпулювати політиками або вводити фальшиві правила безпеки. У розподілених конфігураціях компрометація East/West-каналів між контролерами створює ризик порушення узгодженості даних і дублювання правил. Особливу небезпеку становить динамічність оновлення таблиць потоків. У SDN правила пересилання постійно змінюються відповідно

до поточного стану трафіку. Це підвищує ефективність використання ресурсів, але водночас ускладнює контроль коректності цих змін. Затримка або помилка у синхронізації таблиць може спричинити короткочасні вікна вразливості, коли пакети пересилаються без перевірки політик безпеки. Відкритість архітектури створює ще один ризик - компрометацію прикладних сервісів. Застосунки, що взаємодіють із контролером через Northbound API, мають високі привілеї, адже вони можуть безпосередньо впливати на маршрутизацію та політики доступу. Якщо такий застосунок містить вразливості або є зловмисним, він отримує повний контроль над поведінкою мережі. Не менш критичною є проблема аутентифікації та авторизації в межах самої архітектури. У багатьох реалізаціях SDN канали зв'язку між контролером і мережевим елементом захищаються слабкими або нестандартними механізмами. Відсутність обов'язкового шифрування або двофакторної перевірки дозволяє реалізовувати атаки типу "man-in-the-middle" та перехоплювати службові повідомлення [7]. Ще один виклик - масштабованість контролера. Коли кількість мережевих пристроїв зростає, навантаження на контролер може перевищувати його можливості. Зловмисник може навмисно створити велику кількість нових потоків або підроблених запитів для атаки "Packet-in Flooding" щоб перевантажити процесорні ресурси контролера й викликати затримки або відмову обслуговування. Проблеми безпеки виникають і на рівні віртуалізації мережевих функцій. У SDN часто використовується віртуалізація для ізоляції сегментів трафіку або розгортання сервісів безпеки [9-11]. Якщо гіпервізор або середовище віртуалізації має вразливості, порушується межа між сегментами, що дозволяє здійснювати атаки через суміжні середовища. Ще одна категорія ризиків пов'язана з помилками політик. Централізований контроль спрощує адміністрування, але робить систему вразливою до помилкових або неконсистентних правил. Неправильно сформульована політика на рівні контролера автоматично поширюється на всі пристрої, створюючи потенційно глобальний збій.

У загальному випадку усі ці проблеми демонструють, що перехід від децентралізованої моделі до централізованої структури не усуває ризики, а лише

змінює їхню природу. Якщо у традиційних мережах головним об’єктом атаки є окремий пристрій або протокол маршрутизації, то в SDN мішенню стають логічні зв’язки та програмні інтерфейси, які керують усією інфраструктурою.

Кожен із чотирьох базових принципів SDN - externalization, centralization, federation і programmability має як переваги, так і потенційні ризики.

Принцип externalization, тобто винесення функцій керування з мережевих елементів у контролер, забезпечує гнучкість, але створює залежність від зовнішнього каналу зв’язку. Якщо цей канал перехопити або зруйнувати, мережа втрачає здатність реагувати на зміни трафіку. Centralization підвищує узгодженість рішень і спрощує моніторинг, однак формує єдину точку відмови. Навіть короткочасне виведення контролера з ладу може призвести до зупинки обробки трафіку або до втрати інформації про стан мережі. Federation сприяє здатності різних систем, пристроїв або програмного забезпечення взаємодіяти між собою, обмінюватися даними та коректно працювати спільно, навіть якщо вони створені різними виробниками або на різних технологічних платформах, але водночас ускладнює забезпечення довіри між компонентами від різних постачальників. Відкрите API, що дає змогу інтегрувати нові сервіси, потребує суворих механізмів перевірки автентичності та ізоляції. Принцип programmability забезпечує найвищий рівень гнучкості, але є потенційним джерелом нових вразливостей. Кожен програмний модуль, який має доступ до функцій контролера, може змінювати поведінку мережі. Неправильна реалізація логіки або помилки у коді можуть створити умови для несанкціонованих дій, що особливо небезпечно для політик безпеки та QoS.

Одним із ключових висновків сучасних досліджень є те, що безпека в SDN не може розглядатися як зовнішня надбудова. Вона повинна бути інтегрована в саму архітектуру. Усі рівні від площини даних до площини застосунків повинні проектуватися з урахуванням принципів захищеності. Для площини даних це означає перевірку достовірності джерел команд і захист каналів обміну. Для площини керування - розмежування доступу між модулями контролера та забезпечення відмовостійкості. Для площини застосунків - обмеження привілеїв і сувора автентифікація API-викликів. Захищене проектування передбачає також

створення механізмів валідації політик перед їх застосуванням, моніторинг стану таблиць потоків, багаторівневий контроль доступу до інтерфейсів і виявлення аномалій у поведінці контролера [12,13].

1.2 Методологія ідентифікації активів та цілей безпеки

Побудова моделі безпеки програмно-визначеної мережі потребує системного опису її складових, визначення ролі кожної з них у забезпеченні стабільності функціонування та виявлення властивостей, порушення яких призводить до появи загроз. У традиційних мережах активи, що підлягають захисту, здебільшого мають матеріальну природу - сервери, маршрутизатори, лінії зв’язку, термінали користувачів. У програмно-визначених мережах такий підхід стає недостатнім, адже значна частина функціональності переміщується у логічну площину. Компоненти, які формують поведінку мережі, тепер не обмежуються фізичними пристроями, а представлені програмними модулями, внутрішніми структурами даних, каналами комунікації між площинами та абстракціями політик. Відповідно, поняття активу набуває нового змісту: активом вважається будь-яка сутність, компрометація або відмова якої може вплинути на роботу мережі, порушити узгодженість управлінських рішень або змінити хід виконання політик. Першим етапом методології є ідентифікація активів, що становлять основу архітектури SDN. Вона починається з аналізу функціонального поділу мережі на три основні площини - даних, керування та застосунків, між якими встановлено логічні канали взаємодії. Кожна з площин містить власні об’єкти управління та інформаційні потоки, а взаємозалежність між ними створює цілісну інфраструктуру. Для площини даних активами є елементи мережі, таблиці потоків, буфери, лічильники, інтерфейси вводу-виводу, тобто всі елементи, що фізично чи логічно реалізують пересилання пакетів. Для площини керування - це контролер, його модулі прийняття рішень, бази даних станів, механізми синхронізації, а також канали зв’язку, через які передаються інструкції до мережевих елементів. Для площини застосунків активами виступають політики, аналітичні алгоритми, сервіси моніторингу та

програмні інтерфейси, через які користувач або адміністратор впливають на поведінку мережі. Результати такого розподілу наведені у таблиці активів SDN (див. таблиця 1.1), де для кожного рівня визначено ключові об’єкти, їхній функціональний внесок і зв’язок з іншими компонентами.

Таблиця 1.1 – Активи SDN

№	**Категорія / Рівень**	**Актив (Asset)**	**Короткий опис функції**	**Основна ціль безпеки**
1	Площина застосунків (Application Plane)	Application Modules	Застосунки, що формують політики управління трафіком і безпекою, взаємодіють із контролером через Northbound API.	Конфіденційність / Цілісність
2		Security Applications	Модулі безпеки, що контролюють доступ, IDS/IPS, виявлення аномалій.	Цілісність / Доступність
3		Monitoring and Analytics	Служби моніторингу стану мережі та статистики трафіку.	Цілісність
4	Площина керування (Control Plane)	Controller Core	Центральний компонент, який приймає рішення та керує потоками.	Доступність / Цілісність

Продовження таблиці 1.1

№	Категорія / Рівень	Актив (Asset)	Короткий опис функції	Основна ціль безпеки
5		Topology Database	Зберігає інформацію про структуру мережі та зв’язки між вузлами.	Цілісність / Достовірність
6		Flow-Rule Manager	Відповідає за створення, зміну й видалення правил у таблицях потоків.	Цілісність
7		Policy Repository	База даних, у якій зберігаються політики мережевого керування.	Конфіденційність / Цілісність
8		Event Handler / Scheduler	Механізм синхронізації дій між модулями контролера.	Доступність
9	Площина даних (Data Plane)	OpenFlow Switches / Devices	Мережеві елементи, що виконують правила, отримані від контролера.	Цілісність / Доступність
10		Flow-Tables	Логічні таблиці, що містять правила пересилання пакетів.	Цілісність
11		Counters and Buffers	Елементи обліку та тимчасового зберігання даних у пристроях.	Достовірність / Доступність

Продовження таблиці 1.1

№	Категорія / Рівень	Актив (Asset)	Короткий опис функції	Основна ціль безпеки
12	Канали взаємодії (Communication Channels)	Southbound Interface	Канал між контролером і площиною даних (наприклад, OpenFlow).	Цілісність / Доступність
13		Northbound Interface	Канал між контролером і застосунками.	Конфіденційність / Цілісність
14		East–West Interface	Зв’язок між кількома контролерами в розподіленому середовищі.	Доступність / Узгодженість
15	Системні служби (Support Services)	Log and Audit Records	Журнали подій, необхідні для відстеження стану безпеки.	Цілісність / Спостережуваність
16		Authentication / Authorization Services	Механізми перевірки користувачів та контролю привілеїв.	Автентичність / Контроль доступу

Ця таблиця демонструє, що в сучасних програмно-визначених мережах більшість активів мають змішану природу та поєднують апаратні ресурси з програмними структурами. Наприклад, маршрутизатор одночасно є фізичним пристроєм і логічною сутністю, у якій розміщено таблиці потоків, що формуються контролером. Контролер, у свою чергу, є програмним модулем, але його робота залежить від стану фізичних каналів і пристроїв. Такий взаємозв’язок ускладнює традиційний поділ активів на «мережеві» й «системні», адже будь-яка зміна в одному з них може вплинути на інші. Важливим аспектом ідентифікації є опис каналів взаємодії між площинами. Вони розглядаються як самостійні активи, оскільки компрометація каналу може бути не менш небезпечною, ніж порушення роботи окремого пристрою. Південний інтерфейс, через який контролер передає правила мережевим елементам, є критичним для

забезпечення цілісності та автентичності команд. Північний інтерфейс, що поєднує контролер із прикладними сервісами, визначає, які політики застосовуються до мережі. У випадку, коли цей канал використовується без належного контролю доступу, існує ризик втручання у логіку управління мережею. Східно-західні канали, що забезпечують взаємодію між кількома контролерами, мають стратегічне значення для забезпечення узгодженості станів у розподіленому середовищі. Отже, у межах SDN активами є не лише вузли, але й канали, що забезпечують їхнє функціонування. Після формування переліку активів необхідно визначити для них відповідні цілі безпеки. Цей етап спрямований на встановлення властивостей, без яких мережа не може функціонувати коректно. Для кожного активу потрібно оцінити, яка з властивостей - конфіденційність, цілісність чи доступність є для нього критичною. Наприклад, для таблиць потоків найважливішою є цілісність, оскільки модифікація навіть одного правила може змінити маршрут пакета або дозволити несанкціонований доступ. Для каналів взаємодії першочерговою є доступність, адже втрата з’єднання між контролером і мережевими елементами зупиняє обмін інструкціями. Для модулів прикладного рівня особливе значення має конфіденційність, тому що витік політик чи аналітичних алгоритмів може розкрити стратегію безпеки всієї мережі. У рамках цієї методології кожен актив оцінюється також за ступенем впливу його відмови на інші компоненти. Існують активи, які мають локальний вплив, та активи, чия компрометація спричиняє каскадний ефект. Для виявлення таких залежностей розглядається схема взаємозв’язків активів, наведена на рисунку 1.3, яка демонструє попередні результати оцінювання взаємного впливу у контексті базових і тимчасових характеристик CVSS [14].

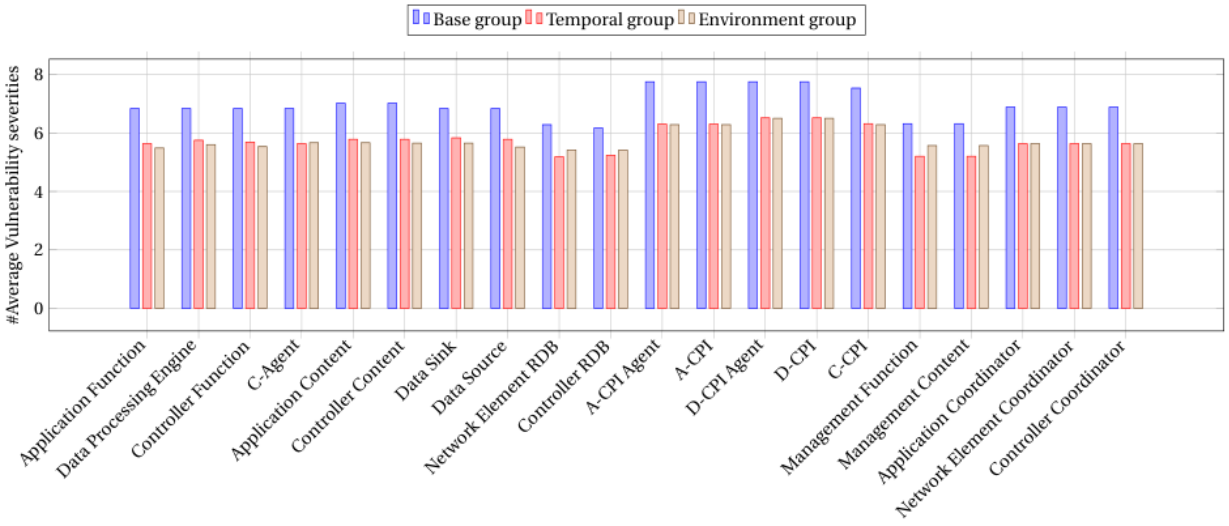


Рисунок 1.3 – Інтегрована оцінка активів, що відображає взаємозалежність та критичність кожного активу в архітектурі SDN

Найчутливішими елементами системи є контролер та його канали зв'язку, оскільки вони одночасно належать до кількох площин і виступають проміжною ланкою між політиками та їх реалізацією.

Визначення активів та цілей безпеки завершується побудовою карти взаємозалежностей, яка відображає, як порушення властивостей одного активу може вплинути на інші. Наприклад, спотворення топологічних даних у модулі керування призведе до формування некоректних таблиць потоків, що вплине на площину даних. Відмова модуля моніторингу у прикладному рівні може призвести до невчасного реагування на аномалії. Тому ідентифікація активів це не лише визначення їхнього набору, а й встановлення характеру їхньої взаємодії.

На основі такої моделі формуються відповідні цілі безпеки. Вони не обмежуються класичною тріадою «конфіденційність, цілісність, доступність», а розглядаються у розширеному контексті. Для SDN актуальними є також автентичність, контроль доступу та спостережуваність. Автентичність забезпечує перевірку джерела кожної команди, контроль доступу регламентує повноваження користувачів і застосунків, а спостережуваність гарантує можливість відстеження подій для подальшого аудиту. Проте в межах цього підрозділу розглядаються лише базові властивості, що утворюють основу для наступних етапів формалізації.

Після визначення активів і цілей безпеки необхідно встановити спосіб оцінювання ступеня їх уразливості. На цьому етапі використовується система CVSS, яка дозволяє надати кількісну характеристику ризику для кожного активу. Методологічно CVSS інтегрується у процес ідентифікації активів через відповідність між цінностями безпеки та групами метрик системи. Базові метрики CVSS описують фундаментальні властивості уразливостей, що не залежать від контексту, тимчасові відображають зміни у часі, а контекстні враховують особливості середовища, у якому функціонує актив. Таким чином, система дозволяє перетворити якісний опис активу на числову оцінку його критичності.

Для активів SDN базові метрики CVSS тлумачаться з урахуванням специфіки архітектури. Показник Attack Vector відображає, наскільки легко зловмисник може досягти активу через відповідну площину. Attack Complexity характеризує складність дій, необхідних для експлуатації уразливості. Privileges Required визначає рівень доступу, який потрібен для впливу на актив. User Interaction вказує, чи потребує атака участі користувача. Метрики Impact на конфіденційність, цілісність і доступність корелюють із визначеними раніше цілями безпеки. Для контролера найважливішим є параметр доступності, оскільки перевантаження або відмова зупиняє роботу всієї мережі. Для мережевих елементів критичними є цілісність та автентичність правил, які вони отримують, а для прикладних модулів - конфіденційність і точність політик.

На рисунку 1.4 наведено попередні результати оцінювання CVSS для активів SDN у базовій, тимчасовій та контекстній групах.

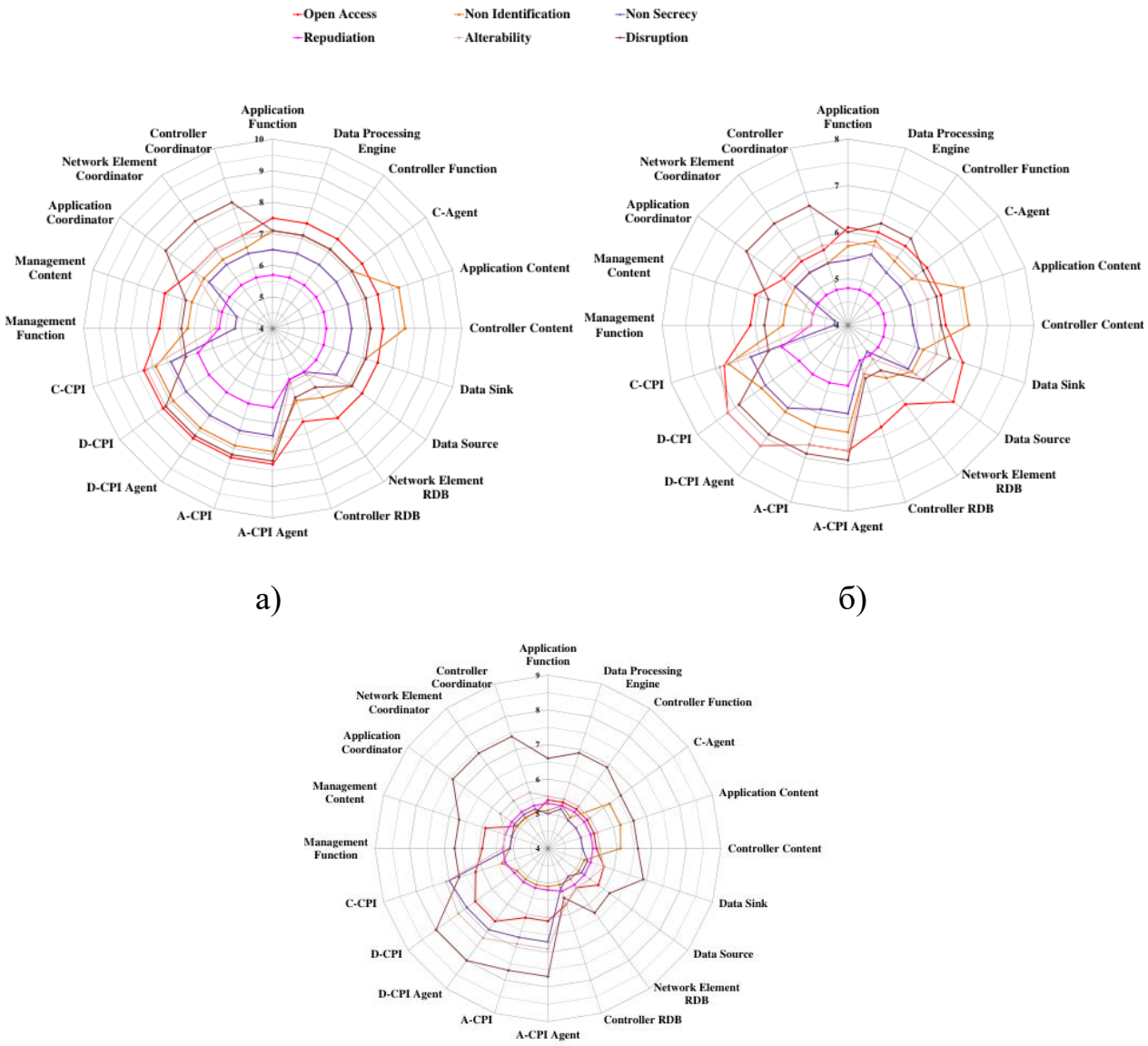


Рисунок 1.4 – Попередні результати оцінювання CVSS для активів SDN у базовій (а), тимчасовій (б) та контекстній групах (в)

Діаграми демонструють відмінності у значеннях між різними класами активів, що дає змогу визначити, які з них є найбільш уразливими у даній архітектурі. У базовій групі найвищі показники мають канали взаємодії та контролер, оскільки вони відкривають шлях до всієї мережі. У тимчасовій групі відносна вага цих активів зменшується через наявність механізмів оновлення та резервування. У контекстній групі найвищі оцінки отримують активи, пов’язані з прикладним рівнем, де вплив залежить від конфігурації та політик доступу. Ці результати підтверджують, що методологія ідентифікації активів повинна

враховувати не лише структурну позицію елемента, але й контекст його використання. CVSS дозволяє встановити пріоритети захисту, але її роль у межах цього підрозділу полягає передусім у підтримці процесу ідентифікації. Вона виступає як інструмент, що кількісно відображає результати якісного аналізу. Наприклад, якщо у таблиці активів зазначено, що для каналу Southbound критичною властивістю є цілісність, то CVSS допомагає визначити числове значення впливу порушення цієї властивості. Це значення не використовується ще для оцінки ризику, цей етап відбувається пізніше, але воно дозволяє ранжувати активи за рівнем важливості. Важливим елементом методології є також принцип незалежності оцінки. CVSS розділяє вплив уразливостей на три незалежні аспекти, що співвідносяться з цілями безпеки. Такий підхід усуває надмірну суб'єктивність, притаманну експертним оцінкам, і забезпечує відтворюваність результатів. У контексті SDN це особливо важливо, оскільки мережа постійно змінюється, а динамічне оновлення правил може призводити до тимчасових змін у критичності активів. Тому для коректного відображення стану системи потрібно, щоб методика ідентифікації залишалася стабільною навіть за умов постійної еволюції архітектури.

Після завершення етапу ідентифікації активів і встановлення цілей безпеки формується аналітична база для подальших досліджень. Визначені активи стають вхідними даними для процесів формалізації уразливостей і кількісного аналізу ризиків. Однак головним результатом цього етапу є не список елементів, а розуміння того, як вони взаємодіють між собою і які властивості є ключовими для їх стабільного функціонування. Саме це розуміння створює основу для побудови інтегрованої моделі безпеки SDN, де кожен актив розглядається у контексті всієї мережевої екосистеми.

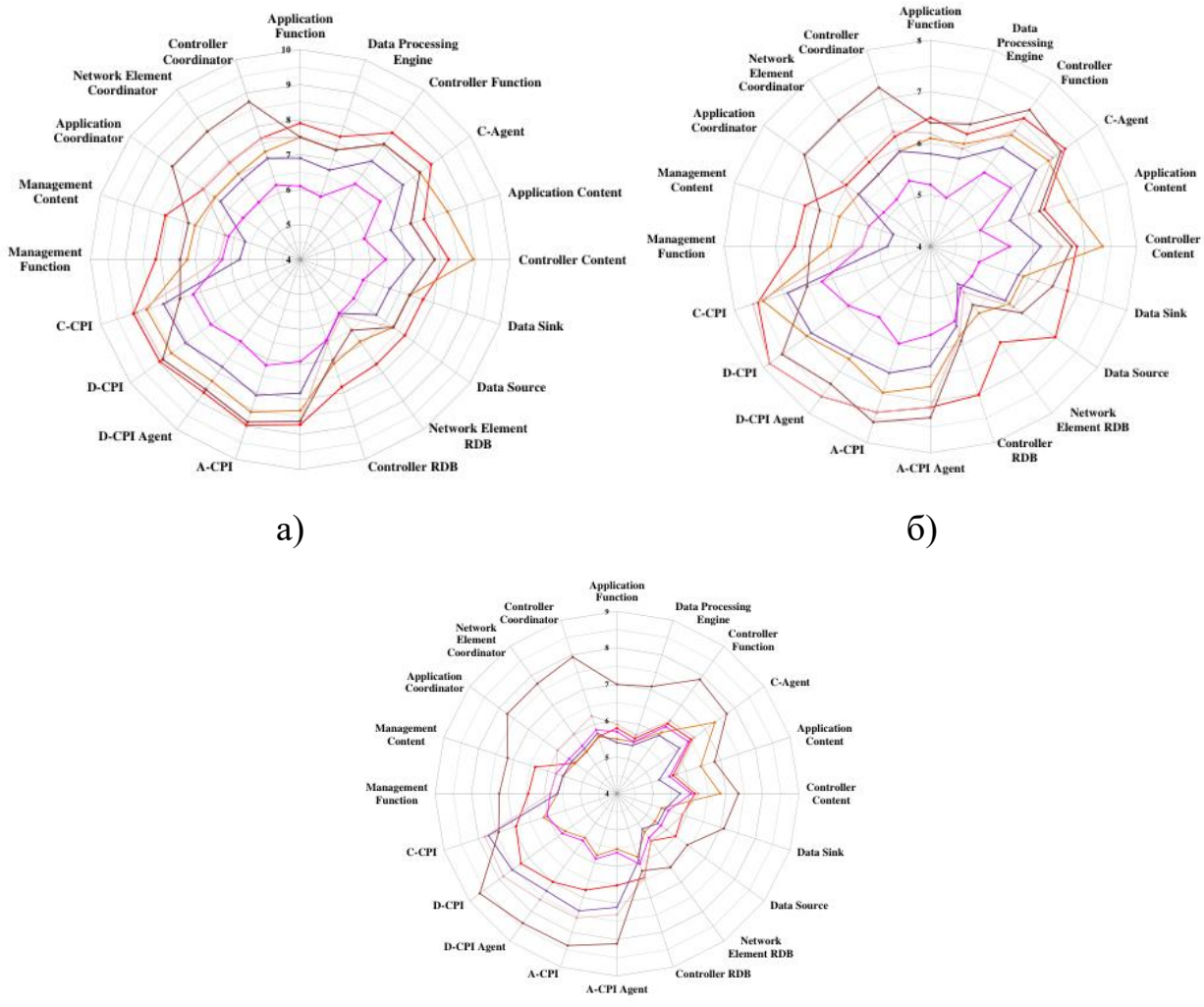
1.3 Формалізація уразливостей через реверсію цілей безпеки

Після ідентифікації активів і встановлення для них базових цілей безпеки постає завдання формалізувати уразливості як порушення цих цілей. Така формалізація необхідна, щоб перейти від описового рівня до кількісного аналізу

ризику. Основою є принцип реверсії цілей безпеки, за яким кожна уразливість розглядається як інверсія певної властивості системи - конфіденційності, цілісності чи доступності. У межах цього підходу замість прямого пошуку вразливих місць у системі дослідник визначає, які умови мають бути порушені, щоб конкретна властивість перестала виконуватися. Це дозволяє описати простір потенційних уразливостей не як набір окремих подій, а як структурований ансамбль відхилень від нормальної поведінки, прив'язаний до цілей безпеки активів. Початковим кроком є побудова карти зв'язків між активами, цінностями безпеки та можливими станами їх порушення. Для кожної властивості встановлюється набір логічних предикатів, що визначають її виконання. Наприклад, цілісність контролера забезпечується, якщо команди, що надходять через північний або південний інтерфейс, відповідають авторизованим джерелам і не зазнають модифікації в процесі передачі. У формалізованому вигляді це можна представити як функцію стану I(c) = true, якщо всі команди мають вірогідне джерело та незмінний вміст. Уразливість виникає, коли цей предикат стає false, тобто з'являється хоча б один невалідний вхід. Таким чином, поняття уразливості безпосередньо виводиться з реверсії умов, що визначають безпечний стан. Для кожної мети безпеки існує власний тип реверсії. У випадку конфіденційності порушення означає втрату контролю над доступом до інформації. Формально це стан, коли об'єкт, який не має необхідних привілеїв, отримує доступ до даних активу. Для цілісності реверсія означає модифікацію інформації або стану без авторизації. Для доступності - це втрата можливості використовувати актив у межах встановленого часу або з потрібним рівнем сервісу. Такий підхід дозволяє задати уразливість не через зовнішній фактор, а через внутрішню логіку системи, що полегшує її подальше картографування у межах CVSS. На етапі формалізації застосовується узагальнена структура взаємозалежностей, у якій кожен актив має асоційовані цілі безпеки, а кожна ціль може бути порушена набором умов. Ці умови можна виразити у вигляді функцій стану або правил логічного висновку. Наприклад, для південного інтерфейсу S, через який контролер взаємодіє з мережевими елементами, умова безпеки може бути подана як множина {A1, A2, A3}, де A1 - автентичність джерела, A2 -

перевірка цілісності, A3 - відсутність затримки або блокування каналу. Уразливість з'являється, якщо будь-який елемент цієї множини порушено. Такий підхід створює підґрунтя для побудови ієрархічної моделі залежностей між активами, де уразливість одного може спричинити порушення цілей іншого, формуючи каскадні ефекти.

На рисунках 1.5 та 1.6 наведено результати розширеної оцінки CVSS, у яких процес формалізації уразливостей пов'язано із зворотним аналізом цілей безпеки.



а) б)

в)

Рисунок 1.5 – Розширені результати оцінювання CVSS для активів SDN у базовій (а), тимчасовій (б) та контекстній групах (в)

Діаграми відображають, як зміни у значеннях метрик впливають на загальний показник критичності активів. У порівнянні з попередніми попередніми результатами (див. рисунок 1.4), де аналіз проводився на рівні базових і тимчасових характеристик, тут додатково враховано взаємозв'язки між властивостями безпеки. Зокрема, рисунок 1.5 (a) показує нормалізовані значення впливу базових метрик для кожної категорії активів, (b) демонструє зміни у тимчасових оцінках із урахуванням оновлення контексту, (c) відображає інтегровані результати після зворотного аналізу. Рисунок 1.6 узагальнює оцінку за всіма активами після застосування моделі реверсії.

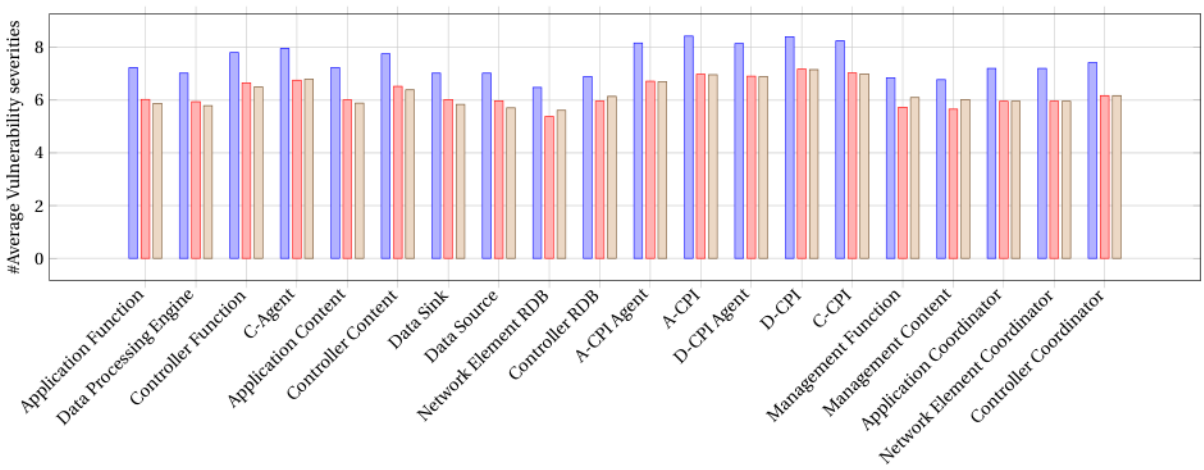


Рисунок 1.6 – Розширена інтегрована оцінка активів, що відображає взаємозалежність та критичність кожного активу в архітектурі SDN

Саме ці діаграми ілюструють, як формалізація уразливостей через реверсію дозволяє кількісно пов'язати порушення властивостей безпеки з поведінкою системи в цілому.

Метод реверсії цілей безпеки має важливу перевагу - він дозволяє описати нові або складні уразливості, які не мають чітких аналогів у відомих базах даних. Замість класифікації за зовнішніми ознаками (наприклад, "DoS", "MITM", "Privilege Escalation") кожна уразливість описується як результат логічного переходу системи з безпечного стану в небезпечний. Це підхід, орієнтований на модель, а не на перелік загроз. Такий опис краще підходить для динамічних

систем, як SDN, де звичайні категорії атак можуть змінюватися залежно від конфігурації чи політик у реальному часі.

У процесі формалізації враховується контекст кожного активу. Наприклад, компрометація контролера призводить до одночасної втрати доступності для прикладних модулів і порушення цілісності таблиць потоків, тому її можна розглядати як складену уразливість. Відповідно, її формальний опис містить кілька цілей безпеки, реверсованих одночасно. Інший приклад - канали взаємодії. Якщо порушено автентичність південного інтерфейсу, це впливає не лише на передачу правил, але й на достовірність аналітики у верхніх рівнях. Таким чином, реверсія однієї властивості породжує вторинні ефекти для інших. Саме це відображено на рисунка 1.6, де активи із сильними міжрівневими зв’язками мають найвищі значення критичності після моделювання реверсій.

Реверсія цілей безпеки також допомагає узгодити якісні висновки з кількісними результатами CVSS. У системі оцінювання CVSS кожна метрика має числовий вплив на показники конфіденційності, цілісності та доступності. Коли певна властивість порушується, її реверсія відображається у збільшенні відповідного показника. Тому процес реверсії можна розглядати як механізм зворотного зв’язку, який перевіряє, наскільки зміна у безпеці активу відображається у його числовому рейтингу. Це дає змогу оцінити адекватність моделі. Якщо формалізована реверсія цілісності не спричиняє зміни у CVSS, значить модель активу не враховує реальні залежності, і її потрібно уточнити.

Іншим наслідком формалізації є можливість побудови матриці відповідності між активами й властивостями безпеки. Така матриця слугує проміжним етапом перед розрахунком ризиків. Її елементи визначаються за допомогою логічних співвідношень. Якщо актив A впливає на актив B і для A порушено ціль X, то для B автоматично знижується стійкість за тією самою ціллю. Цей принцип передбачає, що уразливості мають транзитивний характер, і саме реверсія цілей дозволяє простежити їхнє поширення мережею. Звідси стає можливим побудувати більш точну ієрархію критичності, ніж у класичних схемах CVSS, де активи розглядаються ізольовано.

Формалізація через реверсію також забезпечує узгодження між логічною моделлю уразливостей і процесом управління політиками безпеки. У SDN політики задають бажану поведінку системи, а формалізовані уразливості визначають, які обмеження можуть її порушити. Таким чином, можна не лише оцінювати існуючі загрози, а й передбачати потенційні. Наприклад, якщо політика передбачає оновлення таблиць потоків раз на певний період, а модель реверсії показує, що при затримці цього оновлення понад певний поріг виникає втрата цілісності, це означає, що система має бути доповнена механізмом перевірки часу реакції. Такий рівень інтеграції між формалізацією і політиками робить модель безпеки не статичною, а адаптивною.

На завершення слід підкреслити, що реверсія цілей безпеки - це не просто теоретичний інструмент, а практичний механізм узгодження моделі уразливостей із реальними сценаріями загроз. На відміну від традиційного підходу, який починається з опису відомих атак, метод реверсії починається з визначених цілей і аналізує, які умови повинні бути порушені, щоб ці цілі не виконувалися. Завдяки цьому формується універсальна рамка, придатна для оцінювання будь-яких нових або невідомих уразливостей у динамічному середовищі SDN.

1.4 Інтегрована методика кількісного оцінювання ризиків у SDN на основі CVSS та AHP

Ефективна оцінка ризиків у програмно-визначених мережах потребує не лише якісного опису активів та уразливостей, але й числової моделі, що враховує специфіку архітектури SDN. Стандартна система CVSS, хоч і широко застосовується для класифікації загроз, у початковому вигляді не повною мірою відображає взаємозалежність компонентів і динамічний характер мережевого середовища. Тому для адекватного відображення впливу уразливостей на рівні контролера, каналів і прикладних політик необхідно модифікувати її структуру. У поєднанні з методом аналітичної ієрархії AHP [15] така адаптація дозволяє не лише оцінити критичність активів, але й визначити їхні вагові коефіцієнти у

загальній моделі ризику, що відкриває шлях до кількісної інтерпретації результатів.

Початковою точкою розроблення модифікованої методики стало спостереження, що базові метрики CVSS, такі як Attack Vector, Attack Complexity, Privileges Required та User Interaction - однаково впливають на всі типи систем, не враховуючи архітектурних особливостей SDN. У цьому середовищі атака може відбуватися не безпосередньо на фізичні пристрої, а через логічні інтерфейси або модулі керування. Наприклад, зловмисник може спричинити некоректну поведінку мережі, не маючи прямого доступу до мережевого елемента, а лише маніпулюючи політиками контролера. Тому в модифікованій моделі введено нову підмножину параметрів, які описують специфічні властивості SDN: Control Plane Exposure, API Trust Level, Flow Rule Integrity, Controller Availability Dependency та Policy Consistency Factor. Ці параметри додаються до базових метрик і впливають на результуючий індекс CVSS шляхом коригування вагових коефіцієнтів.

Розширена структура оцінки враховує також динамічні властивості мережі, які не представлені у класичному CVSS. У SDN стан системи постійно змінюється внаслідок оновлення таблиць потоків, змін політик і перерозподілу ресурсів. Тому до тимчасових метрик додано коефіцієнт Update Frequency, що відображає період оновлення політик, і параметр Synchronization Reliability, який показує ступінь узгодженості між контролерами у розподіленій архітектурі. Чим менша частота оновлення або нижча надійність синхронізації, тим вищий інтегрований ризик, навіть за однакових базових умов. Такий підхід дозволяє уникнути заниження оцінки ризику в сценаріях, коли уразливість не є постійною, але її ефект посилюється в часі через накопичення помилок.

На рисунках 15 та 1.6 показано результати розрахунку розширеної інтегрованої оцінки активів, які демонструють, як інтеграція нових параметрів впливає на структуру оцінки ризику. Як видно з діаграм, найвищі оцінки отримують канали взаємодії між контролерами (East–West Interface) і центральні модулі керування, що підтверджує їхню роль як точок консолідації ризику.

Модифікована формула CVSS для SDN має вигляд:

$$CVSS\_SDN = (Base \times \alpha_1 + Temporal \times \alpha_2 + Environmental \times \alpha_3 + SDN\_specific \times \alpha_4) / \Sigma\alpha, \quad (1.1)$$

де α_1–α_4 - нормовані ваги метрик, а SDN_specific - сукупність додаткових параметрів, характерних для даного середовища. Це дозволяє не лише зберегти сумісність із класичною системою CVSS, але й адаптувати її до архітектурної логіки SDN, де цінність активу визначається не стільки його ізольованими властивостями, скільки роллю у мережевій взаємодії.

Другим компонентом інтегрованої методики є визначення вагових коефіцієнтів активів за допомогою методу AHP. На цьому етапі ієрархія складається з трьох рівнів: мета (загальний рівень безпеки SDN), критерії (властивості безпеки: конфіденційність, цілісність, доступність) і альтернативи (ідентифіковані активи). Кожна пара активів порівнюється між собою за ступенем впливу на безпеку системи відповідно до обраного критерію. У такий спосіб створюється матриця парних порівнянь, а нормалізація її власних векторів дозволяє отримати вагові коефіцієнти.

AHP особливо ефективний для SDN, оскільки дозволяє враховувати експертні оцінки, які відображають практичний досвід адміністраторів та аналітиків безпеки. Наприклад, експерти можуть визначити, що порушення доступності контролера критичніше, ніж втрата цілісності окремої таблиці потоків, навіть якщо остання має вищий CVSS-індекс. Такі суб’єктивні корекції компенсують недоліки числових моделей і забезпечують баланс між формалізмом і реальними пріоритетами. У підсумку AHP формує множину ваг $W = \{w_1, w_2, \ldots, w_n\}$, де сума всіх коефіцієнтів дорівнює одиниці, а значення кожного відображає відносну важливість активу для безпеки всієї мережі.

Поєднання CVSS і AHP здійснюється шляхом агрегації оцінок у єдину модель ризику. Для кожного активу розраховується скоригований показник $R_i = w_i \times CVSS\_SDN_i$, який відображає інтегровану критичність з урахуванням ваги активу. Такий підхід дозволяє побачити не лише абсолютний рівень уразливості, але й системний внесок кожного компонента у загальний ризик. Наприклад, навіть якщо окремий мережевий елемент має невисоку оцінку CVSS, але

контролює значну частину трафіку, його ваговий коефіцієнт підвищує вплив на сумарний ризик.

Кількісний аналіз результатів показує, що найбільші інтегровані ризики зосереджуються у компонентах площини контролю, де поєднуються всі три властивості - цілісність, доступність і конфіденційність. Це пояснюється центральною роллю контролера у керуванні топологією, маршрутизацією та політиками безпеки. У випадку його компрометації вся мережа переходить у стан невизначеності, а зміни у правилах можуть миттєво поширюватися на всі вузли. Канали Southbound і East–West демонструють другу за величиною групу ризиків через можливість перехоплення або ін’єкції неправдивих команд. Найменші значення ризику отримують активи площини даних, що не мають автономних функцій прийняття рішень і діють лише в межах отриманих інструкцій.

Інтерпретація результатів інтегрованої оцінки дозволяє зробити кілька концептуальних висновків. По-перше, рівень безпеки SDN визначається не ізольованими уразливостями, а їхнім розподілом у структурі активів. Високий ризик одного центрального елемента може компенсуватися низькими оцінками периферійних компонентів, якщо вони мають обмежені зв’язки. По-друге, введення додаткових метрик у CVSS дає змогу точніше оцінювати складні залежності, зокрема вплив затримок у синхронізації контролерів або частоту оновлень політик. По-третє, інтеграція з AHP додає гнучкості та адаптивності: вагові коефіцієнти можуть динамічно коригуватися залежно від змін у топології або пріоритетів безпеки.

Завдяки об’єднанню CVSS і AHP створюється універсальна модель оцінювання ризику, яка поєднує точність формалізму та експертний контекст. У подальшому ця модель може використовуватися не лише для оцінювання поточного стану безпеки, а й для прогнозування наслідків змін у конфігурації або впровадження нових політик. Таким чином, інтегрована методика стає не просто інструментом оцінки, а основою для побудови системи підтримки прийняття рішень у сфері кібербезпеки SDN. Вона дозволяє автоматизувати процес

визначення пріоритетів, зменшити суб’єктивність оцінок і забезпечити узгодженість між різними рівнями управління мережею.

Отже, запропонована інтегрована методика кількісного оцінювання ризиків у SDN поєднує модифіковану структуру CVSS, що враховує специфіку програмно-визначених мереж, із методом AHP, який дозволяє визначати ваги активів на основі експертних знань. Її застосування забезпечує комплексну оцінку стану безпеки, враховуючи як технічні, так і організаційні фактори, та створює основу для подальшої автоматизації управління ризиками в SDN-середовищі.

1.5 Висновки до розділу 1

У першому розділі було проведено аналіз уразливостей у програмно-визначених мережах, який охопив архітектурні особливості SDN, методологію ідентифікації активів та цілей безпеки, формалізацію уразливостей через реверсію цілей і побудову інтегрованої моделі кількісного оцінювання ризиків. Встановлено, що логічна централізація керування, відкритість інтерфейсів і динамічність політик формують специфічну площину загроз, де ключову роль відіграють контролер та канали взаємодії між площинами. Розроблена методологія дозволила поєднати описові та формальні підходи. Реверсія цілей безпеки забезпечила структуроване подання уразливостей, а модифікована CVSS-модель у зв’язці з AHP дала можливість кількісно оцінити критичність активів. Отримані результати стали підґрунтям для побудови подальших розділів, присвячених проєктуванню політик безпеки, визначенню пріоритетів захисту й розробленню практичних механізмів управління ризиками у SDN-середовищі.

РОЗДІЛ 2 АРХІТЕКТУРА SDN-ФАЄРВОЛА З ПЕРЕВІРКОЮ СТАНУ З'ЄДНАННЯ

2.1 Аналіз існуючих рішень та обмежень класичних фаєрволів

У системах мережевої безпеки фаєрволи історично стали фундаментальним компонентом, який забезпечував ізоляцію між внутрішнім, довіреним сегментом мережі та зовнішнім середовищем. Їх основна функція полягала у виконанні контролю доступу відповідно до наперед визначених політик безпеки, де рішення про пропуск, відхилення або карантин пакета приймалося на основі фіксованих правил. Класичні фаєрволи, або так звані legacy firewalls, виникли в епоху, коли мережеві топології були статичними, а мережеві ролі були чітко розмежованими, тому архітектура таких пристроїв не передбачала гнучкої адаптації до динамічного середовища, притаманного сучасним віртуалізованим системам і програмно-визначеним мережам.

Традиційний фаєрвол діє в межах визначеної моделі безпеки, вважаючи, що вся довірена мережа є внутрішньо безпечною, а загрози надходять лише ззовні. Такий підхід прив'язаний до поняття "захищеного периметра" - кордону, де весь вхідний трафік проходить через єдину точку контролю. На етапі становлення мережевих технологій цей принцип був ефективним: централізована перевірка пакетів дозволяла відносно просто запобігати атакам із зовнішніх джерел. Проте в епоху хмарних технологій, мобільності, мікросервісної архітектури й внутрішніх загроз такий підхід виявився застарілим, оскільки не враховує складної структури внутрішнього трафіку, багатошарових зв'язків між віртуальними елементами та контексту користувачів.

Традиційні фаєрволи реалізують політики доступу на основі набору фільтраційних правил, які описуються як сукупність критеріїв та відповідної дії. Формально політика П складається з множини правил $P_1...P_n$, де кожне правило r_j визначається набором полів заголовка пакета і дією Accept або Deny. В процесі роботи фаєрвол здійснює пошук збігів між атрибутами пакета (IP-адреси, порти, протокол, прапорці TCP тощо) і цими полями, після чого виконує дію, визначену у відповідному правилі. Такий механізм ефективний лише за

умови стабільного набору потоків і передбачуваних шаблонів трафіку, але не придатний для динамічних мереж, де зв’язки змінюються в реальному часі.

Класифікація традиційних фаєрволів тісно пов’язана з еталонною моделлю OSI, яка визначає сім рівнів обробки мережевих даних - від фізичного до прикладного. Саме рівень, на якому працює фаєрвол, визначає глибину його інспекції, рівень контекстної обізнаності та можливість адаптивного реагування [16]. На нижчих рівнях фаєрволи забезпечують швидке, але поверхневе фільтрування; на вищих - глибокий аналіз, проте зі значними обчислювальними витратами.

Початковий тип - канальні (bridge) фаєрволи, що діють на канальному рівні OSI (Layer 2). Вони фільтрують кадри на основі MAC-адрес і фізичних портів маршрутизатора, не аналізуючи жодної інформації про протокол або вміст пакета. Такі фаєрволи фактично інтегруються у комутаційне середовище й забезпечують простий механізм ізоляції сегментів локальної мережі. Вони ефективні для невеликих або ізольованих систем, але не здатні розрізняти легітимний і підроблений трафік, оскільки не мають інформації про джерело чи стан з’єднання. У термінах OSI це означає, що вони діють на рівні, де ще не існує понять «сеансу» чи «додатка».

Другий тип - безстанові (stateless) фаєрволи, які працюють на мережевому рівні OSI (Layer 3). Вони здійснюють фільтрацію на основі IP-заголовків, аналізуючи адресу джерела, адресу призначення, ідентифікатор протоколу (TCP, UDP, ICMP тощо) та іноді поля типу служби (Type of Service). Прийняття рішення ґрунтується виключно на порівнянні цих полів із наборами правил у таблиці доступу. Такі фаєрволи не мають уявлення про стан або контекст з’єднання, розглядаючи кожен пакет як незалежну подію. Унаслідок цього вони вразливі до атак типу spoofing або flooding, коли зловмисник створює потік фальшивих пакетів із підробленими адресами. Оскільки фаєрвол не розрізняє початок, продовження чи завершення сесії, він сприймає такі пакети як потенційно легітимні, що може призвести до перевантаження або обходу правил фільтрації.

Подальший розвиток привів до появи фаєрволів з перевіркою стану з'єднання (stateful), що працюють уже на транспортному рівні OSI (Layer 4). На цьому рівні додається розуміння концепції сеансу зокрема, етапів встановлення (SYN, SYN-ACK, ACK), підтримання та завершення з'єднання (FIN, RST) у протоколі TCP. Stateful-фаєрвол веде таблицю станів з'єднань (Connection Table), у якій зберігає інформацію про поточні сеанси, зокрема IP-адреси, порти, прапорці, тайм-аути та інші параметри. Завдяки цьому фаєрвол може відрізняти пакети, що належать до активних сесій, від спроб встановлення нових або підозрілих з'єднань. Цей тип суттєво підвищив ефективність фільтрації, адже дозволяв відсікати несанкціоновані пакети до їх передачі на рівень додатків. Проте з архітектурної точки зору він залишався децентралізованим, не мав централізованого управління та стикнувся з проблемою масштабування - таблиці станів зростали експоненційно із кількістю одночасних сеансів.

На наступному рівні еволюції з'явилися фаєрволи типу circuit-level gateways, які функціонують між транспортним і сеансовим рівнями (Layer 4–5) моделі OSI. Вони поєднують інспекцію стану з проксі-механізмами, контролюючи процес встановлення TCP-сеансів. Такі шлюзи виступають як посередники, що створюють два логічно відокремлені з'єднання між клієнтом і шлюзом та між шлюзом і сервером. Під час TCP-handshake шлюз перевіряє легітимність з'єднання, після чого ретранслює лише дозволені пакети. Цей рівень уже вводить базову абстракцію контексту - фаєрвол знає не лише, що з'єднання існує, а й хто його ініціював. Проте підвищена складність обробки, зокрема необхідність підтримувати таблиці станів для великої кількості клієнтів, призводить до затримок і збільшення витрат процесорного часу.

Вершиною розвитку класичного підходу стали прикладні (application-level) та багаторівневі фаєрволи з перевіркою стану з'єднання (stateful multilayer inspection firewalls), які діють на рівнях 5–7 OSI - від сеансового до прикладного. Вони виконують глибоку інспекцію пакетів (DPI), аналізуючи не лише заголовки, а й вміст переданих даних. Такі системи можуть інтерпретувати структуру протоколів вищого рівня (HTTP, FTP, SMTP, DNS), виявляти аномалії у запитах або спроби ін'єкцій шкідливого коду. Це дозволяє розпізнавати

загрози, які залишаються непомітними для попередніх фільтрів. Наприклад, у протоколі FTP DPI-фаєрвол здатен визначати спроби передачі файлів у несанкціоновані каталоги, тоді як мережевий фаєрвол не може інтерпретувати зміст команд.

Однак підвищення рівня інтелектуального аналізу неминуче веде до зниження продуктивності. Час обробки пакетів у багаторівневих DPI-системах може зростати на порядок у порівнянні з простим пакетним фільтром. Це пов’язано з необхідністю аналізу кожного пакета на предмет сигнатур, шаблонів і контекстних зв’язків. Крім того, такі фаєрволи вимагають значного обсягу оперативної пам’яті, оскільки повинні зберігати буфери даних для потоку, перш ніж виконати інспекцію. Через це їх реалізація часто потребує спеціалізованих апаратних прискорювачів або ASIC-компонентів, що підвищує вартість володіння і ускладнює інтеграцію в гнучкі віртуалізовані середовища. Якщо розглядати еволюцію фаєрволів крізь призму моделі OSI, стає очевидним, що кожне наступне покоління піднімалося на вищий рівень мережевої абстракції.

У сучасних динамічних мережах, де загрози часто виникають на межі рівнів - між транспортним і прикладним, або між логічним і фізичним рівнем така ізольованість призводить до “сліпих зон” у системі захисту. Саме це концептуальне обмеження класичних моделей стало передумовою для переходу до SDN-архітектури, де поділ на площину даних і площину керування дозволяє створити глобальний контекст безпеки, який виходить за межі статичної OSI-структури.

Сутність проблеми полягає в тому, що архітектура класичного фаєрвола є статичною та децентралізованою. Кожен пристрій працює автономно, не маючи глобальної видимості всієї мережі. У великих корпоративних системах це призводить до дублювання політик, конфліктів між правилами, непослідовності рішень та труднощів в оновленні конфігурацій. Якщо в одній зоні мережі політика дозволяє певний тип трафіку, а в іншій блокує, відсутність координації може створити так звані “дірки безпеки” або, навпаки, призвести до надмірної блокади сервісів. При зміні топології адміністратор змушений вручну оновлювати правила на десятках пристроїв, що підвищує ризик помилок.

Класичні фаєрволи також страждають від жорсткої прив’язки до фізичної інфраструктури. Вони захищають лише сегмент, у якому фізично встановлені, і не здатні відслідковувати маршрутизацію між віртуальними або динамічними мережами. У хмарних середовищах, де віртуальні машини мігрують між вузлами, такий фаєрвол не може забезпечити постійну захищеність, оскільки правила не “переміщуються” разом із сервісом. Це обмеження особливо критичне для сучасних дата-центрів, де автоматизоване розгортання нових екземплярів вимагає миттєвого створення відповідних політик доступу.

Ще одним суттєвим недоліком є відсутність контекстної обізнаності. Класичний фаєрвол не знає станів мережевих з’єднань, користувацьких сесій або ролей пристроїв. Він не може розрізнити легітимного адміністратора, який підключається з нетипового місця, від зловмисника, що використовує ті ж облікові дані. У кращому випадку такі пристрої дозволяють фільтрацію за IP і портом, але не враховують вищі рівні логіки взаємодії. Як наслідок, виникає дисбаланс між безпекою й функціональністю: щоб не блокувати необхідні сервіси, адміністратори змушені відкривати широкі діапазони портів, чим створюють додаткові вразливості.

З точки зору архітектури, класичні фаєрволи об’єднують контрольну площину (control plane) та площину даних (data plane) в одному пристрої. Це означає, що прийняття рішень і пересилання пакетів відбуваються на тому самому рівні, без відокремлення логіки управління. Такий підхід не дозволяє масштабувати процес ухвалення рішень або здійснювати централізований моніторинг. Як наслідок, фаєрвол стає “вузьким місцем” мережі де усі пакети повинні пройти через нього, що обмежує пропускну здатність. На концептуальному рівні SDN-архітектура передбачає чітке розділення цих площин і централізований контроль, чого класичні рішення принципово не мають.

Не менш важливим є питання внутрішньої безпеки (insider threats). Традиційні фаєрволи не здатні протидіяти загрозам ізсередини організації, оскільки всі внутрішні користувачі апріорі вважаються довіреними. Як наслідок, якщо компрометовано один внутрішній вузол, він може вільно взаємодіяти з

іншими ресурсами без жодного контролю. Спроби розв’язати цю проблему через впровадження множинних внутрішніх фаєрволів призводять до нових труднощів - надлишкових витрат, конфліктів політик і падіння продуктивності.

Проблема внутрішніх атак безпосередньо пов’язана з поняттям де-периметизації безпеки (security de-perimeterization), що означає відхід від моделі, де безпека зосереджена лише на кордоні мережі, на користь розподіленого контролю всередині. Проте класичні фаєрволи не підтримують такого підходу: вони не здатні розміщувати логіку безпеки ближче до джерела трафіку, оскільки не мають механізмів динамічного розгортання правил у середині мережі. Як наслідок, усі пакети, навіть ті, що проходять між внутрішніми вузлами, мусять прямувати через центральний шлюз, що створює додаткову затримку та ризик відмови.

Іще одним системним недоліком є відсутність стандартизованої інтеграції з зовнішніми системами управління. Більшість комерційних фаєрволів мають пропрієтарні інтерфейси, не надають API для зовнішньої взаємодії і не підтримують оркестрацію на рівні політик. У результаті будь-яка автоматизація (наприклад, у середовищах OpenStack або Kubernetes) стає практично неможливою. Сучасна тенденція до інтеграції безпеки як коду (Security as Code) [17] вимагає, щоб механізми контролю доступу могли бути запрограмовані, протестовані і автоматично розгорнуті, однак традиційні фаєрволи залишаються ізольованими “чорними скриньками”.

Крім того, у класичних рішеннях відсутня глобальна координація політик. Вони не мають механізму перевірки узгодженості або запобігання конфліктам правил між різними сегментами. Навіть якщо кілька фаєрволів контролюють один і той самий трафік, кожен із них може ухвалити різне рішення через розбіжності у локальних політиках. Це призводить до дублювання обробки, надмірного навантаження на мережу та непередбачуваної поведінки.

Із практичної точки зору, класичний фаєрвол часто є єдиною точкою відмови. Якщо він виходить з ладу, уся мережа стає або відкритою, або заблокованою. Навіть у разі застосування кластеризації або резервування, управління такими конфігураціями складне та вимагає синхронізації таблиць

правил між усіма вузлами. У SDN-середовищах, навпаки, безпекова логіка може бути розподілена між контролером і мережевими елементами, що зменшує ризик повної втрати функціональності.

2.2 Архітектура фаєрвола для SDN

Архітектура фаєрвола у середовищі SDN ґрунтується на принципі логічного відокремлення функцій контролю та обробки трафіку, що дозволяє реалізувати централізований, контекстно-обізнаний і динамічно керований механізм захисту. На відміну від класичних систем, які виконують як контроль, так і пересилання пакетів на одному рівні, SDN-фаєрвол спирається на архітектуру з двома площинами: control plane, де ухвалюються рішення, і data plane, де ці рішення реалізуються шляхом встановлення правил у мережевих елементах. Саме ця ідея закладена в основу концепції, поданої на рисунку 2.1, який демонструє загальну архітектуру SDN Stateful Firewall з трирівневою структурою: рівень оркестрації (Orchestrator), рівень контролера з фаєрвол-додатками (Firewall Applications) та рівень мережевих елементів (Network Elements).

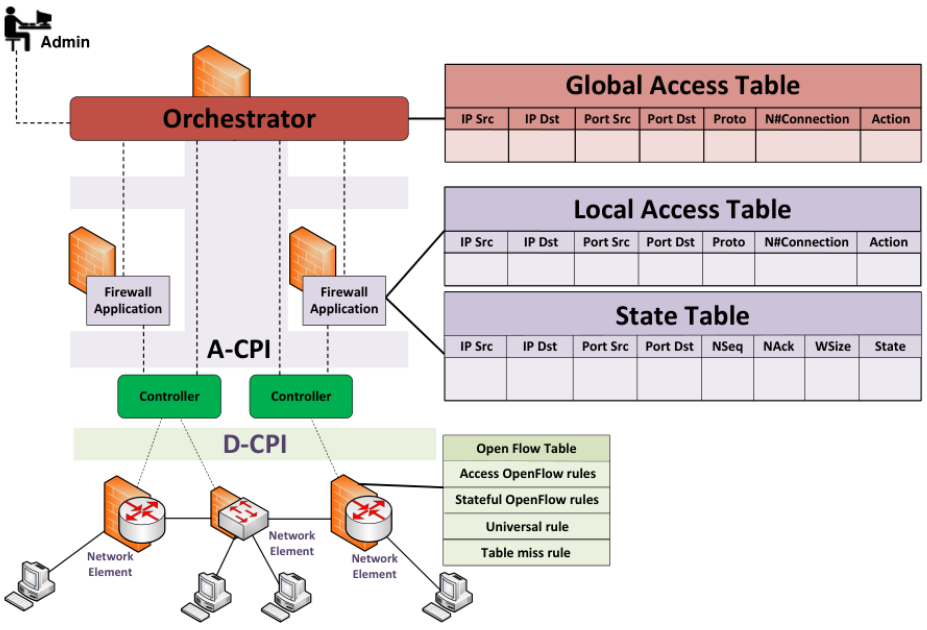


Рисунок 2.1 – Архітектура фаєрвола у середовищі SDN

Такий поділ забезпечує одночасно централізованість управління та децентралізоване виконання політик, що істотно підвищує масштабованість і гнучкість системи. Основна ідея полягає в тому, що функції прийняття рішень щодо пропуску чи блокування потоків переносяться з фізичних пристроїв на рівень програмного контролера. Контролер виступає як центральний інтелект системи, який має повну видимість усієї мережі, таблиць потоків, активних з'єднань і станів кожного вузла. Фаєрвол, реалізований як прикладний модуль у складі контролера, отримує події Packet-In від мережевих елементів, аналізує їх відповідно до поточного контексту з'єднання та формує відповідні Flow Entries, які встановлюються в таблицях мережевих елементів. Таким чином, перевірка трафіку стає не локальною, а системною - усі рішення узгоджуються між собою на основі глобальної політики безпеки.

У центрі архітектури SDN-фаєрвола розташований Orchestrator, який координує взаємодію між прикладними сервісами безпеки та контролерами. На відміну від класичних систем, де конфігурація фаєрвола виконується вручну, оркестратор динамічно керує політиками відповідно до стану мережі, завантаженості елементів і типів додатків. Як показано на рисунку 2.2 Orchestrator виконує роль рівня, який отримує інформацію про стан усіх мережевих об'єктів і розподіляє політики між ними. У цьому контексті фаєрвол не є ізольованим модулем, а частиною загальної системи управління, де політики безпеки узгоджуються з політиками маршрутизації, якості обслуговування (QoS) та віртуалізації ресурсів.

Таке централізоване управління усуває одну з головних проблем класичних фаєрволів - відсутність глобального бачення. Тепер рішення про блокування або дозволи не приймаються у вакуумі, а базуються на поточному стані всієї мережі. Наприклад, якщо оркестратор фіксує підозрілу активність з певного вузла, він може автоматично ініціювати зміну політики фаєрвола на всіх мережевих елементах, які обслуговують цей вузол, без участі адміністратора. Це дозволяє досягти високого ступеня адаптивності - ключової характеристики станової безпеки у SDN.

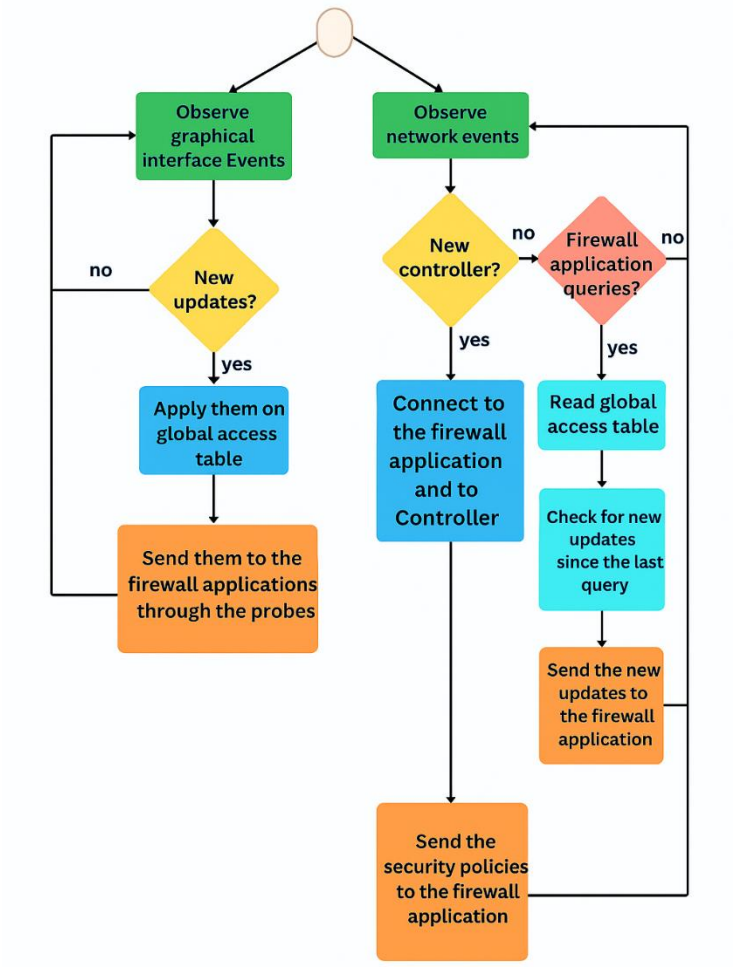


Рисунок 2.2 – Схема роботи оркестратора в SDN

Одним із центральних принципів архітектури є логіка на основі стану (stateful), яка реалізується не на рівні мережевих елементів, а у вигляді програмного автомата в межах контролера. У класичному фаєрволі таблиця станів зберігається локально, а її узгодження між різними пристроями є проблематичним. У SDN-реалізації цю таблицю замінює логічна модель з'єднань, яку веде контролер і яка відображає всі активні потоки. Контролер отримує події про створення, модифікацію й завершення потоків безпосередньо через OpenFlow-інтерфейс, тому може визначати, у якому стані перебуває кожне з'єднання. Таким чином, навіть якщо трафік розподілений між кількома мережевими елементами, його контекст залишається узгодженим у межах системи.

На відміну від класичних фаєрволів, де кожен пакет перевіряється послідовно, SDN Stateful Firewall використовує подієво-орієнтовану модель управління [18]. Коли мережевий елемент не знаходить відповідного правила у

своїй Flow Table, він надсилає подію Packet-In до контролера. Контролер аналізує контекст - тип протоколу, напрям потоку, стан з'єднання, джерело запиту і формує нове правило. Якщо потік дозволено, генерується команда Flow-Mod із параметрами (match, action, timeout), і це правило розповсюджується на відповідні мережеві елементи. Якщо ж потік порушує політику безпеки, контролер формує команду Drop або Redirect. Цей механізм дозволяє реагувати лише на нові потоки, тоді як подальші пакети у межах уже дозволених з'єднань обробляються без участі контролера, що істотно знижує затримку.

Ключовою особливістю архітектури є можливість централізованого управління станами. Таблиці потоків у мережевих елементах не містять детальної інформації про кожне з'єднання, а лише базові критерії (match fields), тоді як розширені стани наприклад, стадія встановлення TCP-сеансу чи кількість переданих пакетів зберігаються у контролері. Це забезпечує узгоджене бачення усіх сеансів і дає змогу контролеру виконувати аналітичні функції, зокрема виявлення аномалій або кореляцію між подіями. Такий підхід дає змогу розглядати фаєрвол не просто як засіб фільтрації, а як інтелектуальний компонент політики безпеки, інтегрований у загальну систему управління мережею. Важливо, що фаєрвол реалізується не як окремий пристрій, а як додаток у середовищі контролера. Це означає, що він може взаємодіяти з іншими додатками SDN - моніторингом, маршрутизацією, балансуванням навантаження або управлінням якістю сервісу. Наприклад, фаєрвол-додаток може використовувати дані з модуля моніторингу для прийняття рішень про блокування або дозволи залежно від трафіку чи затримки. Такий рівень інтеграції неможливий у класичних фаєрволах, де програмна логіка замкнена всередині пристрою. Архітектура SDN, навпаки, забезпечує відкритість і модульність: додатки спілкуються через стандартні API, що дозволяє створювати комплексні рішення без необхідності змінювати базову інфраструктуру.

На рівні мережевих елементів реалізується data plane, який виконує команди, надані контролером. У цій площині не виконується аналіз логіки протоколів чи контенту, лише чітке виконання правил фільтрації. Мережеві елементи є "прозорими виконавцями" - вони отримують Flow Entries з чітко

визначеними умовами (наприклад, IP-адреса джерела, порт призначення, MAC-адреса) і діями (forward, drop, mirror тощо). Уся інтелектуальна частина знаходиться вище у контролері та його прикладних модулях. Саме це відокремлення робить архітектуру SDN придатною для реалізації складних механізмів безпеки без втрати швидкодії на рівні мережевих елементів.

Окрім централізованого управління, архітектура підтримує розподілене виконання. Це означає, що політики, визначені оркестратором, можуть реалізовуватись на десятках мережевих елементів одночасно. Якщо виникає нова загроза або змінюється топологія, контролер оновлює лише ті правила, які стосуються відповідних потоків, не перевантажуючи мережу зайвими оновленнями. Такий підхід дозволяє масштабувати безпекові механізми без лінійного зростання витрат. У класичних системах додавання нового фаєрвола потребувало ручної синхронізації правил; у SDN-фаєрволі достатньо оновити політику у контролері, після чого вона автоматично розповсюджується на всі підлеглі елементи.

Взаємодія між Orchestrator, Firewall Applications і Network Elements описується за допомогою абстрактного інтерфейсу подій. Коли мережевий елемент реєструє новий потік, він генерує подію до контролера. Контролер, у свою чергу, передає цю подію фаєрвол-додатку, який, базуючись на політиках оркестратора, приймає рішення. Якщо потік дозволено, генерується нове правило OpenFlow типу Flow-Mod; якщо ні то потік відхиляється. Цей цикл подій триває постійно, забезпечуючи реактивний контроль нових з’єднань. На відміну від статичних систем, тут рішення приймаються динамічно, а таблиці потоків адаптуються у реальному часі.

Суттєвим компонентом є менеджер таблиць потоків, який підтримує відповідність між логічними станами сеансів і фізичними Flow Entries. Цей менеджер не лише вставляє чи видаляє правила, але й оптимізує їх. Наприклад, якщо два потоки мають однакові атрибути безпеки, система може об’єднати їх в одне правило для зменшення обсягу таблиць. Такі оптимізації підвищують продуктивність і знижують навантаження на мережеві елементи. Окрім того, фаєрвол може використовувати часові політики (idle_timeout, hard_timeout), що

дозволяє автоматично очищати застарілі записи, запобігаючи накопиченню не діючих правил.

Однією з інновацій, яка закладає основу станової поведінки, є уявлення кожного потоку як станового об’єкта, який переходить між станами залежно від подій. Ця ідея, хоч детально формалізується у наступному підрозділі через модель SDN еквіваленту скінченного автомату (FSM) [19], уже на концептуальному рівні інтегрована в архітектуру. Контролер підтримує для кожного потоку набір атрибутів, що відображають поточний стан взаємодії, а фаєрвол приймає рішення з урахуванням цього стану. Наприклад, якщо потік перебуває у фазі встановлення TCP-сеансу, можуть діяти одні правила, а у фазі передавання даних - інші. Це дозволяє реалізувати динамічні політики, що змінюються відповідно до життєвого циклу з’єднання, що було неможливо у класичних системах.

Архітектура також враховує питання стійкості та відмовостійкості. Оскільки контролер є центральною точкою прийняття рішень, його надійність критична. Оркестратор підтримує кілька контролерів, які синхронізують свої таблиці станів. У разі збою одного контролера інший може автоматично підхопити його функції, не порушуючи цілісності системи. Це забезпечує безперервність обслуговування і відповідає принципам fault-tolerant SDN-керування.

Архітектура SDN Stateful Firewall тісно інтегрована з логічною структурою OpenFlow. Комунікація між контролером і комутаторами відбувається за допомогою стандартних OpenFlow-повідомлень, таких як Packet-In, Flow-Mod, Packet-Out, Barrier, Stats. Фаєрвол-додаток аналізує метадані з цих повідомлень, зокрема порт входу, довжину кадру, IP-протокол і прапорці TCP, щоб відновити повний контекст сеансу. Потім він формує рішення у вигляді Flow-Mod команд, які вносяться у Flow Table мережевогоелемента. Таким чином, архітектура використовує вже існуючу інфраструктуру SDN без необхідності модифікувати OpenFlow-протокол, що забезпечує сумісність з типовими контролерами (ONOS, POX, OpenDaylight).

Синергія між оркестратором, контролером і мережевими елементами створює замкнутий контур управління безпекою, у якому політика формується зверху, передається вниз для виконання, а потім контрольні події піднімаються назад для оцінки. У цьому контурі реалізується головна перевага SDN-підходу - можливість навчання та адаптації. Контролер не лише виконує команди, а й накопичує інформацію про поведінку трафіку, що згодом може бути використано для оптимізації правил або прогнозування аномалій. Таким чином, фаєрвол набуває когнітивних властивостей, здатності “пам’ятати” попередні рішення й адаптуватися до нових сценаріїв.

2.3 Модель SDN-еквіваленту скінченного автомату

Архітектура фаєрволів з перевіркою стану з’єднання у середовищі SDN ґрунтується на принципі формального опису поведінки мережевих протоколів за допомогою скінченних автоматів (FSM) та їхнього відображення у вигляді еквіваленту SDN-рівня - SDNEFSM. Така модель є ключовою ланкою між традиційним підходом до контролю з’єднань і централізованим управлінням потоками, притаманним SDN-парадигмі. SDNEFSM дозволяє перетворити правила протоколу на набір керованих станів, якими може маніпулювати контролер, а також формалізувати поведінку фаєрвола в єдиній математичній структурі, що уможливлює перевірку, автоматизацію й синхронізацію дій у масштабах усієї мережі.

Скінченний автомат є абстрактною моделлю, яка описує систему через обмежену множину станів та переходів між ними. Кожен стан відповідає конкретній конфігурації мережевого з’єднання, а перехід - події або дії, що змінює цю конфігурацію. У контексті мережевих протоколів, зокрема TCP, FSM визначає послідовність стандартних кроків: ініціація (SYN), підтвердження (SYN-ACK), встановлення (ACK), передача даних (ESTABLISHED) і завершення (FIN, RST) [20-22]. У класичному фаєрволі така модель використовується лише для пасивного спостереження: система перевіряє, чи відповідає потік цій послідовності. У SDN-середовищі ж, де логіка керування

винесена на контролер, FSM отримує нову роль - він стає активним механізмом прийняття рішень. Саме для цього створюється SDNEFSM - еквівалент скінченного автомата, який інтегрує класичну модель протоколу у функціональну структуру SDN-контролера. Фаєрвол-додаток ухвалює рішення шляхом обробки скінченного автомата протоколів, орієнтованих на з'єднання, приймає як вхідні дані передумови, стани, переходи дій і післяумови, застосовує до них SDN-функцію та генерує як результат SDN-еквівалент скінченного автомата. Отже, SDNEFSM є формальним перетворенням класичного FSM протоколу в його функціональний аналог, адаптований до програмно-визначеного середовища.

Формально SDNEFSM можна подати у вигляді множини:

$$A_{SDN}=(S,\Sigma,\delta,s_0,F,C,f_{SDN}) \quad (2.1)$$

де S - множина станів з'єднання, Σ - множина вхідних подій (пакети, сигнали, таймери), δ - функція переходів, s_0 - початковий стан, F - множина фінальних станів, C - множина контекстних умов, що відображають стан мережі, f_{SDN} - функція SDN-перетворення, яка забезпечує відповідність між подіями автомата і командами контролера.

Таким чином, на відміну від звичайного FSM, у якому перехід визначається лише вхідним сигналом, у SDNEFSM він залежить також від контексту - політик доступу, топології, QoS-параметрів і стану таблиць OpenFlow.

Рисунок 2.3 демонструє цей принцип на прикладі реактивної поведінки фаєрвола. Коли мережевий елемент надсилає до контролера подію Packet-In, SDNEFSM аналізує її як потенційний тригер переходу. Якщо умови збігаються з поточним станом автомата, контролер викликає функцію f_{SDN}, яка трансформує цю подію в конкретну команду - наприклад, створення або оновлення правила Flow-Mod, що дозволяє або блокує трафік.

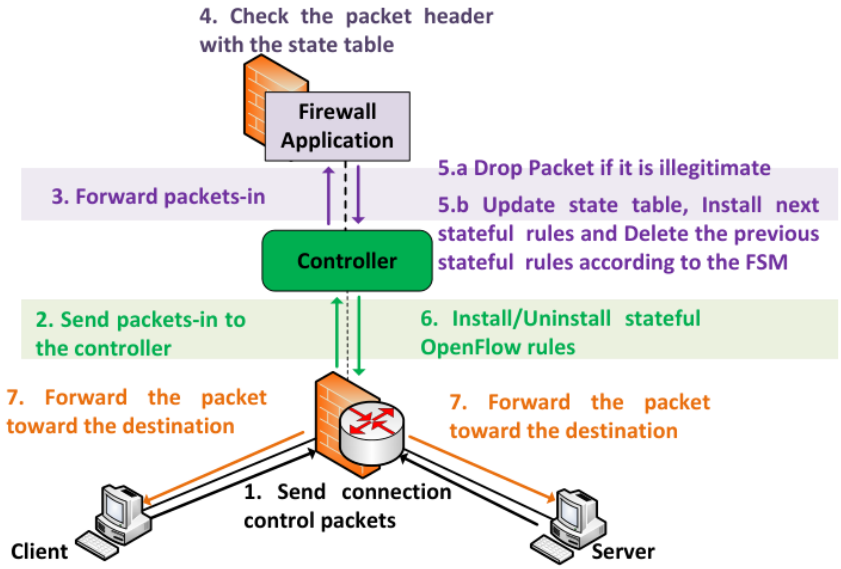


Рисунок 2.3 – Реактивна поведінка фаєрвол-додатка в SDN

Якщо ж умови порушують очікувану послідовність протоколу, SDNEFSM переходить у стан BLOCK або DROP, ініціюючи відповідну дію видалення потоку. Таким чином, кожна зміна стану автомата супроводжується чітко визначеною реакцією в системі керування потоками.

Сутність моделі SDNEFSM полягає у двоетапному відображенні логіки протоколу на структуру SDN-контролера. На першому етапі класичний FSM, що описує поведінку протоколу (наприклад, TCP), перетворюється на абстрактну модель, де кожен перехід має атрибути стану, умови спрацювання та очікувану дію. На другому етапі ці елементи зіставляються з SDN-подіями: умови відповідають характеристикам пакетів і стану топології, а дії відповідно командам OpenFlow, які інсталює контролер. У результаті утворюється динамічна система, у якій кожен перехід FSM реалізується як керована SDN-операція. Перевага SDNEFSM у тому, що вона перетворює фаєрвол із пасивного фільтра на повноцінний механізм оркестрації поведінки з'єднань. Наприклад, коли клієнт ініціює TCP-сеанс, контролер не лише відстежує відповідність послідовності SYN–ACK–ACK, а й контролює створення потокових правил на мережевих елементах, що відповідають кожному стану автомата. Коли з'єднання переходить у стан ESTABLISHED, SDNEFSM вносить зміни у Flow Table, забезпечуючи передачу даних напряму між вузлами без подальших звернень до

контролера. Коли сеанс завершується, автомат переходить у фінальний стан і ініціює видалення відповідних записів, зменшуючи навантаження на ресурси.

Процес формалізації FSM у SDNEFSM здійснюється через дві основні функції - γ і ρ. Перша розширює набір умов FSM додатковими SDN-метаданими (наприклад, інформацією про OpenFlow-версію, порт, політику безпеки чи таймер з'єднання). Друга функція інтерпретує результати цього розширення у вигляді дій контролера, що генерують SDN-правила або виклики до внутрішніх процедур фаєрвол-додатку. У такий спосіб автомат набуває гнучкості - він може враховувати стан мережі в момент ухвалення рішення, а не діяти лише за статичними правилами протоколу. У практичній реалізації SDNEFSM буде існувати як внутрішня таблиця станів у пам'яті фаєрвол-додатку. Для кожного активного з'єднання зберігається унікальний ідентифікатор, стан, набір дозволених переходів і таймер. Контролер синхронізує цю таблицю з локальними таблицями комутаторів, що містять відповідні OpenFlow-правила. Коли на рівні data plane виникає подія, яка збігається з умовами переходу, контролер оновлює свій SDNEFSM і генерує нові команди. Цей механізм забезпечує узгодженість станів між логічною моделлю й фізичною мережею. У порівнянні з класичним FSM, SDNEFSM має ще одну ключову властивість - контекстну керованість. Кожен його стан може бути модифікований зовнішнім впливом інших SDN-додатків або оркестратора. Якщо, наприклад, система моніторингу виявляє надмірне навантаження на певному вузлі, оркестратор може змінити політику для цього потоку, що призведе до корекції стану автомата. SDNEFSM оновить свої переходи так, щоб обмежити нові з'єднання або перенаправити трафік. Це забезпечує динамічну інтеграцію безпеки у процес управління мережею, що неможливо у звичайних фаєрволах із жорсткою архітектурою.

Рисунок 2.4 демонструє логічне продовження концепції SDNEFSM у контексті проактивної поведінки фаєрвола. Тут контролер заздалегідь генерує набір правил для всіх можливих переходів автомата, щоб скоротити час реакції та мінімізувати кількість звернень Packet-In. Проте незалежно від режиму - реактивного чи проактивного центральним механізмом залишається саме

SDNEFSM, який визначає логіку взаємодії між контролером, додатком і мережею.

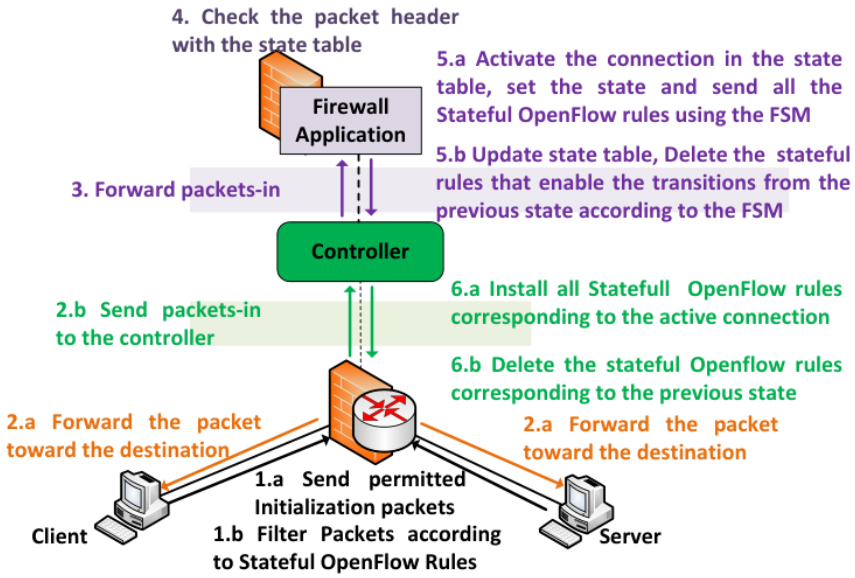


Рисунок 2.4 – Проактивна поведінка фаєрвол-додатка в SDN

На відміну від традиційного становлення FSM, що має жорстко визначений набір станів, SDNEFSM може динамічно розширювати або зменшувати цей набір. Наприклад, під час атаки типу SYN Flood він автоматично додає проміжний стан SYN_CHECK, який дозволяє обмежити кількість запитів від одного клієнта, перш ніж з'єднання переходить у фазу ESTABLISHED. Це є прикладом адаптивного переходу, коли автомат реагує не лише на протокольну послідовність, а й на статистику трафіку. Така поведінка підтверджує, що SDNEFSM не є простою копією FSM у термінах SDN, а його розширенням, здатним до контекстного навчання.

Семантично SDNEFSM можна трактувати як автомат Мілі-Мура з контекстними виходами. Якщо у класичних моделях вихідна функція визначається поточним станом (Мура) або станом і вхідним сигналом (Мілі), то в SDNEFSM вона має вигляд:

$$Y=\lambda(s,x,f_{SDN}(C)) \qquad (2.2)$$

де s - поточний стан автомата, який відображає фазу з'єднання або ситуацію в логіці протоколу (наприклад, LISTEN, SYN-SENT, ESTABLISHED, CLOSE у TCP). Це змінна, що визначає, у якому стані перебуває фаєрвол із перевіркою стану з'єднання в момент обробки події. x - вхідна подія або стимул, який надходить до автомата. У контексті SDN це може бути повідомлення Packet-In, Flow-Removed, Timeout чи інша мережева подія, що вимагає реакції. Саме подія x запускає перехід від стану s до нового стану. $f_{SDN}(C)$ - функція, що враховує контекст середовища. Завдяки цьому система здатна реагувати не лише на події в окремому потоці, а й на глобальні зміни такі,як перевантаження контролера, зміни у топології, оновлення політик безпеки чи QoS-пріоритетів.

Практична користь SDNEFSM полягає також у можливості формальної верифікації. Оскільки всі стани, умови й переходи формалізовані математично, модель можна перевірити на відсутність конфліктів, недосяжних станів або нескінченних циклів. Це спрощує аудит безпеки й дозволяє доводити коректність політик ще до їхнього розгортання. У традиційних фаєрволах це неможливо, адже правила є неформальними, а послідовність дій - не детермінована.

Таким чином, SDNEFSM забезпечує новий рівень інтеграції безпеки та керованості у SDN-мережі. Він поєднує детермінованість автомата з гнучкістю програмно-визначеного середовища, створюючи уніфікований механізм, у якому фаєрвол не лише аналізує пакети, а й управляє їхнім життєвим циклом через стани. SDNEFSM виступає ядром логіки переходів і базовим шаром для реактивної та проактивної поведінки фаєрвола. Його формалізація дозволяє контролеру мислити в термінах станів, а не потоків, що є принциповою відмінністю між SDN-фаєрволом і будь-яким класичним рішенням.

2.4 Висновки до розділу 2

В другому розділі було обґрунтовано перехід від класичних фільтрів до архітектури SDN-фаєрвола із перевіркою стану з'єднання, показано межі традиційних підходів крізь призму моделі OSI та сформовано цілісну

конструкцію з оркестратором, контролером і мережевими елементами. Узгодженість політик досягається завдяки централізованому прийняттю рішень і взаємодії з площиною даних через OpenFlow, тоді як контекстна модель з’єднань описана у вигляді SDN-еквіваленту скінченного автомата, що дає змогу керувати життєвим циклом сеансу, а не лише окремими пакетами. Запропонований SDNEFSM інтегрує стан, подію та функцію контексту, забезпечуючи адаптивні переходи в реактивному й проактивному режимах.

РОЗДІЛ 3 ОЦІНКА ЕФЕКТИВНОСТІ SDN-ФАЄРВОЛА ТА ОРКЕСТРАЦІЯ ПОЛІТИК

3.1 Реалізація фаєрвола в віртуалізованій інфраструктурі

Розроблений фаєрвол функціонує у програмно-конфігурованому мережевому середовищі, де контроль над трафіком реалізується не на рівні фізичних пристроїв, а за допомогою централізованого контролера, що управляє потоками пакетів через протокол OpenFlow. Основна ідея полягає в тому, що всі рішення щодо безпеки - аналіз трафіку, порівняння його зі специфікаціями політик, генерація правил доступу та передача цих правил на мережеві пристрої здійснюються у програмному просторі. Це дає змогу відокремити функції управління від площини пересилання, створюючи гнучку, масштабовану та керовану архітектуру безпеки, адаптовану до вимог віртуалізованих і динамічних інфраструктур.

В основі реалізації фаєрвола лежить контролер RYU, що є відкритим Python-фреймворком з підтримкою компонентної архітектури. Його перевага полягає у можливості створення прикладних модулів безпосередньо поверх ядра контролера, використовуючи стандартизовані API. Завдяки цьому розроблений фаєрвол не потребує модифікації ядра контролера, а функціонує як зовнішній додаток, який взаємодіє з підконтрольними комутаторами через південний інтерфейс OpenFlow. Контролер відповідає за отримання подій з мережі, таких як ініціація нових потоків, запити Packet-In або зміни стану портів, і передає їх у додаток фаєрвола для подальшої обробки.

Архітектура програмного забезпечення, показана на рисунку 3.1, складається з двох взаємопов’язаних пакетів - Orchestrator та Firewall Application. Кожен з них виконує окрему функцію в межах централізованої системи керування політиками безпеки. Перший забезпечує управління політиками, синхронізацію з користувацьким інтерфейсом і глобальну видимість усіх подій у мережі, тоді як другий реалізує безпосередню логіку фільтрації трафіку,

моніторинг протоколів і динамічне створення правил OpenFlow у площині пересилання.

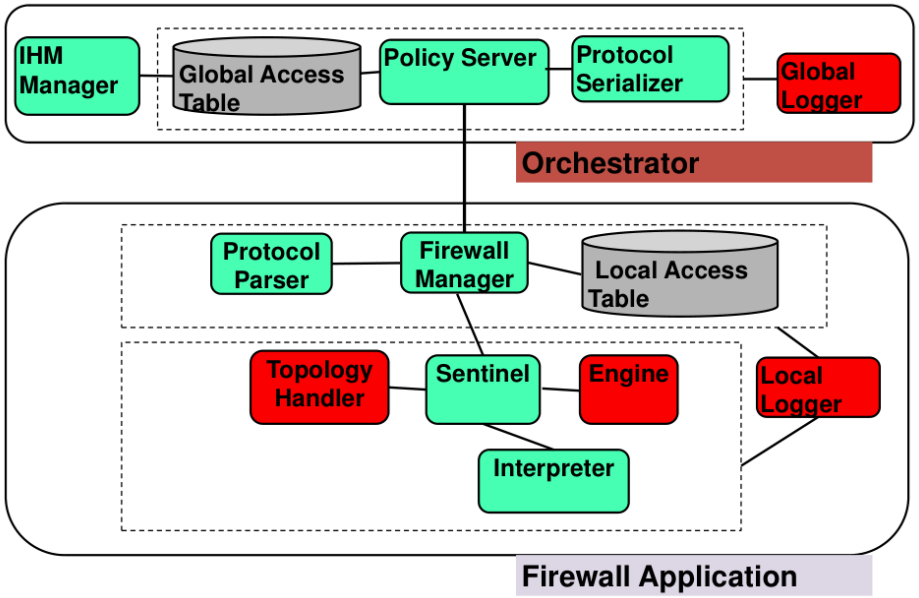


Рисунок 3.1 – Архітектура програмного забезпечення брандмауера

Розміщення цих двох пакетів у різних рівнях архітектури відображає поділ функцій у SDN-парадигмі. Orchestrator розташований у площині управління у менеджмент-шарі, який має глобальний огляд стану мережі, тоді як Firewall Application безпосередньо взаємодіє з контролером RYU [23], утворюючи логіку мережевого рівня. У віртуалізованому середовищі ці компоненти розгортались у вигляді віртуальних машин, що спрощує їх масштабування, оновлення та інтеграцію з іншими сервісами безпеки.

З практичної точки зору реалізація базується на подіях, що надходять з контролера. Коли відбувається нове з’єднання, контролер генерує подію Packet-In, яка передається модулю Sentinel у пакеті Firewall Application. Sentinel аналізує метадані пакета, визначає його протокол, напрямок трафіку та відповідність активним політикам безпеки. Якщо для цього з’єднання ще не створено відповідний механізм контролю, Sentinel ініціалізує новий екземпляр модуля Engine, який реалізує поведінку фаєрвола на основі формалізованої скінченної автоматики. Таким чином, для кожного протоколу зі станом з’єднання

(наприклад, TCP) створюється незалежний об’єкт Engine, який контролює увесь життєвий цикл з’єднання від встановлення сеансу до його завершення.

Використання розширених скінченних автоматів (EFSM) дозволяє забезпечити перевірку стану з’єднання, що є суттєвим відмінним аспектом запропонованого підходу від традиційних безстанових фаєрволів. FSM, що описує конкретний мережевий протокол, перетворюється на EFSM шляхом додавання контекстних змінних та умов переходу, що дозволяє відстежувати як стан, так і параметри сеансу. Модуль Engine виконує це перетворення динамічно, застосовуючи формальний алгоритм, що забезпечує узгодженість між логічним описом протоколу та фактичним його виконанням у системі.

Оркестратор підтримує постійне з’єднання з усіма екземплярами фаєрволів через сокетні канали, які утворюють логічну комунікаційну шину для обміну командами та подіями. Через ці канали оркестратор надсилає команди керування, нові політики або оновлення правил, а також приймає повідомлення про події, виявлені на рівні мережі. Завдяки такій структурі досягається централізоване адміністрування при збереженні автономності окремих фаєрвол-екземплярів, що працюють у різних сегментах мережі або навіть у різних віртуальних доменах.

Ключовим компонентом оркестратора є Policy Server, який відповідає за управління політиками безпеки, збереженими у глобальній таблиці доступу. Політики визначаються адміністратором через інтерфейс користувача, реалізований у модулі IHM Manager, і можуть описувати дозволені або заборонені комбінації атрибутів трафіку - адрес, портів, протоколів і специфічних для протоколу станів. Політики зберігаються у структурованому вигляді в базі даних оркестратора, а перед надсиланням до фаєрвол-додатків проходять перетворення у формат, сумісний із внутрішнім протоколом взаємодії між компонентами. Цю функцію реалізує модуль Protocol Serializer, який трансформує політики у компактні структури даних, що можуть передаватися мережею.

Відповідно, на боці фаєрвола отримані повідомлення обробляються модулем Protocol Parser, який виконує зворотне перетворення даних та

вилучення необхідної інформації для інших модулів. Цей двосторонній процес гарантує, що політики, створені адміністратором на рівні оркестратора, однозначно інтерпретуються фаєрволом, а зміни у протоколі обміну або форматі політик впливають лише на рівень парсера, не потребуючи змін у внутрішніх алгоритмах.

Важливу роль у забезпеченні прозорості та контролю відіграють модулі журналювання. Global Logger на рівні оркестратора акумулює всі події, які надходять від фаєрвол-додатків, включно зі статистикою мережевих пристроїв, повідомленнями про спрацьовування політик, відмови у з'єднаннях або виявлення аномалій. Ці дані візуалізуються у графічному інтерфейсі адміністратора та зберігаються у базі даних для подальшого аудиту. Водночас Local Logger, розташований у пакеті Firewall Application, збирає події локального рівня, наприклад, зміни станів протоколів або внутрішні повідомлення OpenFlow і передає їх до Global Logger для централізованого обліку. Таким чином формується ієрархічна система логування, яка дозволяє одночасно відстежувати локальні процеси та мати глобальний огляд усього середовища.

Реалізація фаєрвола тісно пов'язана з управлінням топологією. Модуль Topology Handler у складі пакета Firewall Application отримує інформацію про мережеві вузли та зв'язки від контролера RYU. Система може працювати у двох режимах: статичному, коли топологія задається адміністратором через оркестратор, і динамічному, коли топологія виявляється автоматично за допомогою OpenFlow-подій. У віртуалізованій інфраструктурі динамічний режим має перевагу, оскільки дозволяє адаптуватися до міграцій віртуальних машин, появи або зникнення контейнерів та оновлення мережевих зв'язків без втручання адміністратора.

З технічного погляду взаємодія між модулями Firewall Application координується через компонент Firewall Manager, який забезпечує зв'язок із Policy Server оркестратора. Він приймає політики, що надходять у вигляді адаптованиї структур, передає їх до відповідних Engine модулів і здійснює зворотне повідомлення про результати перевірки або про виявлені порушення. Firewall Manager також відповідає за маршрутизацію повідомлень між Sentinel та

Engine, керування життєвим циклом екземплярів Engine та розподіл подій у середині програми.

Компонент Sentinel виступає як посередник між фаєрвол-логікою і контролером RYU. Його завдання полягає в тому, щоб перекладати високорівневі правила безпеки у конкретні OpenFlow інструкції, а також перехоплювати події з контролера і спрямовувати їх до відповідних Engine модулів. Коли оркестратор надсилає нову політику, Sentinel викликає модуль Interpreter, який трансформує політику у набір OpenFlow правил. У цьому процесі використовуються карти відповідності між абстрактними атрибутами (наприклад, “дозволити TCP підключення від 10.0.0.1 до 10.0.0.2 на порт 80”) та конкретними полями OpenFlow (тип пакета, IP-адреси, порти, маски та т.ін.). Якщо стандарт OpenFlow оновлюється, достатньо змінити лише карту відповідності у Interpreter, не торкаючись решти системи.

Реалізований фаєрвол підтримує кілька режимів роботи, які можуть задаватися через оркестратор: пасивний моніторинг, активне блокування або змішаний режим, коли трафік аналізується у реальному часі з можливістю автоматичної реакції. Наприклад, у режимі «моніторинг» система лише збирає дані про з’єднання, що надалі використовуються для виявлення аномалій, тоді як у режимі «захист» порушення політики призводить до негайного створення OpenFlow-правил drop. Завдяки цьому адміністратор може адаптувати поведінку фаєрвола залежно від поточних вимог або навантаження мережі.

У контексті віртуалізованої інфраструктури особлива увага приділена взаємодії фаєрвола з віртуальними комутаторами vSwitch, які можуть функціонувати у середовищах KVM, Xen або OpenStack. Кожен віртуальний комутатор виступає як OpenFlow-агент, що підпорядковується контролеру RYU. Фаєрвол отримує повну видимість потоків трафіку між віртуальними машинами або контейнерами, незалежно від їх розміщення на фізичних хостах. Таке рішення усуває проблему «сліпих зон» безпеки, типову для гіпервізорних середовищ, де міжвіртуальний трафік часто не виходить на фізичний рівень і тому не може бути проконтрольований традиційними міжмережевими екранами.

Окрім цього, розроблена архітектура враховує масштабованість і толерантність до збоїв. Оскільки Orchestrator підтримує багатоканальні підключення до кількох Firewall Application екземплярів, у системі можна одночасно розгортати десятки ізольованих фаєрволів для різних сегментів мережі. Кожен з них обслуговує свій набір OpenFlow комутаторів і передає дані до спільного Global Logger. У разі виходу з ладу одного екземпляра інші продовжують роботу, а оркестратор фіксує подію відмови і може автоматично відновити процес через запуск нового контейнера. Завдяки такій децентралізованій, але координованій структурі досягається висока стійкість системи без необхідності створення резервних фізичних вузлів.

З технічного погляду інтеграція з віртуалізованим середовищем здійснюється через стандартні OpenFlow інтерфейси комутаторів OVS. Контролер RYU реєструє події, пов’язані зі зміною стану портів vNIC, міграцією VM або створенням нових тунелів VXLAN, і автоматично актуалізує топологію, з якою працює Topology Handler. Це дозволяє фаєрволу зберігати коректність політик навіть під час динамічного перерозподілу віртуальних машин між хостами. Для пришвидшення реакції використовуються асинхронні канали повідомлень, що мінімізують затримку між моментом події у vSwitch і застосуванням нового правила на комутаторі.

Система також передбачає модульну адаптацію до різних протоколів транспортного рівня. Наприклад, для протоколу TCP реалізовано окремий Engine, що відстежує стани SYN, SYN-ACK та ACK, тоді як для UDP використовується спрощена FSM з двома станами - active та idle. Це дозволяє застосовувати одну й ту саму платформу для контролю різнорідних протоколів, включно з ICMP, HTTP або власними додатками користувача. Під час створення нового Engine для невідомого протоколу адміністратор може додати відповідну FSM-специфікацію через інтерфейс IHM Manager, після чого Orchestrator розповсюдить її серед усіх екземплярів фаєрволів.

Особливістю реалізації є підтримка формалізованого представлення політик через проміжний опис мовою близькою до JSON/XML, який полегшує інтеграцію з зовнішніми системами оркестрації, зокрема OpenStack Heat або

Kubernetes Network Policies. Таким чином, централізований фаєрвол може виступати складовою більшого комплексу безпеки у хмарній інфраструктурі, де політики створюються автоматично під час розгортання сервісів.

Ще одним важливим аспектом є забезпечення ізоляції між різними користувачами та сервісами. Оскільки фаєрвол працює у багатокористувацькому віртуальному середовищі, архітектура передбачає розмежування таблиць доступу на глобальну та локальні. Глобальна таблиця (Global Access Table) зберігається на рівні Orchestrator і визначає базові політики для всієї мережі, тоді як локальні таблиці (Local Access Tables) підтримуються у кожному Firewall Application. Це дозволяє кожному сегменту мати власні правила безпеки, не порушуючи загальну політику мережі. Такий підхід особливо корисний у сценаріях із багатьма орендарями (multi-tenant), де кожен користувач має ізольований віртуальний домен з власними політиками.

У процесі реалізації було особливу увагу приділено ефективності обробки трафіку. Оскільки модулі написані на Python, для мінімізації затримок було використано асинхронні черги подій і неблокуючі сокети. Sentinel отримує повідомлення від контролера у вигляді OpenFlow-пакетів, які перетворюються на події у внутрішній системі черг. Engine модулі працюють у багатопоточному режимі, що дає змогу паралельно обробляти кілька потоків з’єднань. Крім того, обробка логів і взаємодія з базою даних оркестратора виконуються у фоновому режимі, не блокуючи основну логіку фільтрації.

Для зберігання історії подій використовується реляційна база даних, де кожен запис містить часову мітку, тип події, ідентифікатор фаєрвола та метадані пакета. Така структура забезпечує можливість ретроспективного аналізу, статистичної оцінки ефективності політик і виявлення шаблонів атак. Зібрані дані можуть бути інтегровані з системами SIEM або аналітичними платформами для побудови моделей ризику.

3.2 Експериментальна оцінка продуктивності та стійкості до атак

Метою експериментальної оцінки є кількісне порівняння продуктивності SDN-фаєрвола із перевіркою стану з'єднання та його стійкості до SYN-flooding з поведінкою проактивного й реактивного режимів, а також із референсним рішенням на базі NetFilter. Побудована методика відтворює реалістичні навантаження для великої кількості короткоживучих HTTP-з'єднань, варіює обсяг передаваних даних і інтенсивність атаки, а також фіксує низку метрик часу встановлення з'єднання, часу оброблення пакетів і частки повторно пересланих сегментів TCP. Експериментальна платформа побудована в емульованому середовищі Mininet із використанням контролера RYU та комутатора OVS, що відокремлює площини керування й пересилання та дозволяє вимірювати внесок контролера, віртуального комутатора й прикладної логіки фаєрвола за ідентичних умов розміщення й топології. Загальну схему стенду подано на рисунку 3.2, де виділено контур SDN-фаєрвола і контур NetFilter для чистого трасування потоку пакетів у кожному сценарії.

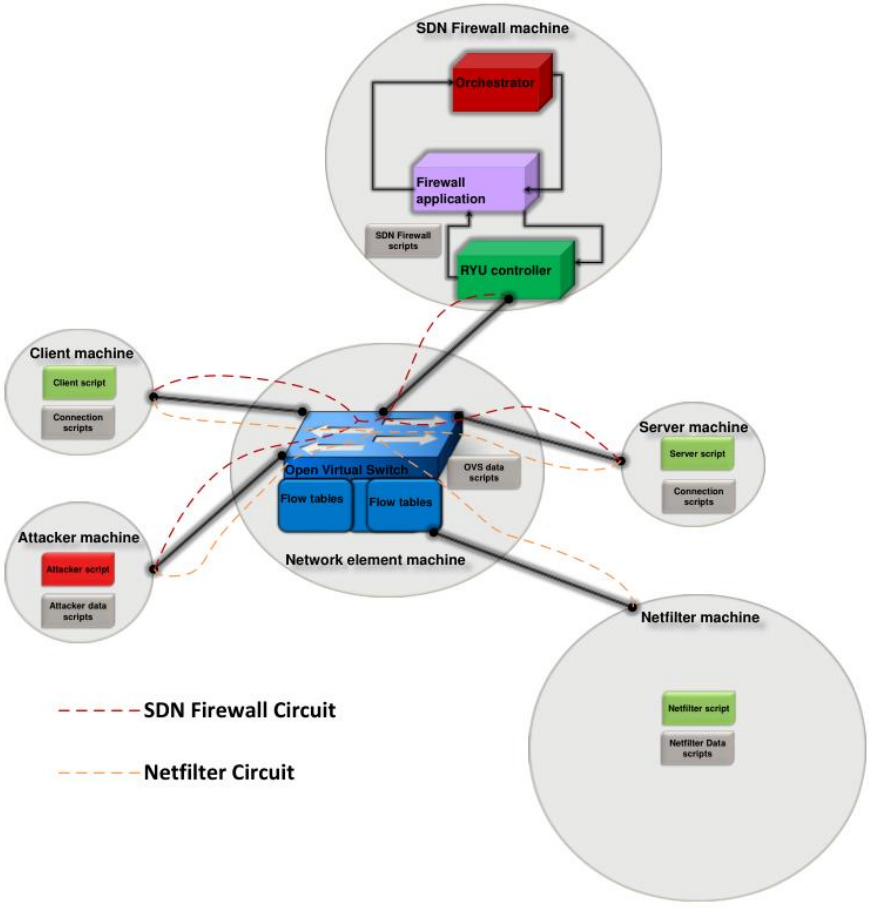


Рисунок 3.2 – Загальна схема стенда тестування SDN фаєрвола

Реалізація експериментального стенда фаєрвола в середовищі OpenFlow виконувалася на базі віртуалізованої інфраструктури, що складалася з одного фізичного сервера під управлінням операційної системи Ubuntu 22.04 LTS (64-bit). Сервер був обладнаний процесором Intel Core i7 із шістьма фізичними ядрами, 32 ГБ оперативної пам’яті DDR4 та твердотільним накопичувачем SSD на 512 ГБ, що забезпечувало стабільну роботу емульованої мережі під час експериментів. Усі компоненти програмного середовища було розгорнуто локально без залучення зовнішніх віртуалізаторів або контейнерних платформ, щоб зберегти контроль над параметрами затримок і навантаження.

Базу середовища становив Mininet версії 2.3 [24], який забезпечував створення віртуальної мережі з одним комутатором Open vSwitch (OVS) і трьома хостами. Комутатор працював у режимі OpenFlow 1.3 і був підключений до контролера RYU через стандартний TCP-порт 6653. Топологія включала вузли h1, h2 і h3, де перший виконував роль клієнта, другий - сервера, а третій - джерела атак. Для зручності керування Mininet запускався безпосередньо зі скрипта (Додаток В), який створював усю топологію, під’єднував контролер та ініціалізував сценарії трафіку.

Контролер SDN реалізовувався за допомогою фреймворку RYU 4.34 на Python 3.10. У його складі було створено прикладний модуль sdn_firewall.py, який безпосередньо реалізував логіку фаєрвола (Додаток Б). Програмний код побудовано на основі базового класу RyuApp із пакета ryu.controller та бібліотек ryu.lib.packet і ryu.ofproto. Розроблений модуль підтримував два режими роботи - реактивний і проактивний. У реактивному режимі правила доступу створювалися динамічно після отримання події Packet-In від комутатора, тоді як у проактивному - встановлювалися наперед, до початку фактичних TCP-сесій. Такий підхід дозволяв оцінити різницю між двома методами управління потоками та їхній вплив на затримки.

Фаєрвол обробляв події від Open vSwitch, формував відповідні OpenFlow-правила та відправляв їх назад комутатору. Основна логіка включала виявлення SYN-пакетів, підрахунок їх частоти та блокування надмірної кількості запитів, що моделювало базовий механізм захисту від SYN-flood атак. Система

використовувала просту таблицю політик, у якій задавалися дозволені TCP-з’єднання, наприклад, між 10.0.0.1 і 10.0.0.2 на порт 80. Усі інші спроби встановлення сесій блокувалися. Завдяки цьому було реалізовано мінімальну, але повноцінну функціональність фаєрвола в SDN-середовищі.

На стороні Mininet-хостів h1 та h2 застосовувалися стандартні утиліти iperf3 та curl для створення легітимного клієнт-серверного трафіку, тоді як h3 імітував атаку за допомогою hping3 та scapy. Сценарії запускалися безпосередньо з Mininet-CLI або через команду всередині скрипта mininet_testbed.py, де реалізовано допоміжні функції для автоматичного запуску HTTP-сервера на вузлі h2 і багатопоточних curl-запитів з боку h1. Для спрощення тестів замість повноцінного Apache-сервера використовувався вбудований модуль Python http.server, який дозволяв швидко піднімати веб-службу на порту 80 без додаткових налаштувань.

Під час експериментів вимірювалися базові часові характеристики, зокрема середній час встановлення з’єднання між клієнтом і сервером та загальна затримка оброблення Packet-In подій у RYU. Вимірювання виконувалися на основі часових міток у логах контролера та результатів роботи tcpdump, запущеного на інтерфейсах віртуальних хостів. Для оцінки ефективності SYN-flood застосовувалися режими з різною інтенсивністю атаки, що дозволяло простежити деградацію швидкодії контролера і реакцію фаєрвола на перевантаження.

Усі компоненти середовища працювали на одному фізичному сервері, тому контрольна та пересильна площини були логічно відокремлені, але фізично розміщені в межах однієї операційної системи. Це спрощувало налагодження та дозволяло виконувати тести без додаткової мережевої інфраструктури. Логи виконання контролера та фаєрвола записувалися у стандартний вихід RYU та в окремі текстові файли для подальшого аналізу результатів.

Описана конфігурація забезпечила повну відтворюваність експериментів. Усі використані програмні компоненти є відкритими і не вимагають ліцензійних обмежень. Застосування Mininet, RYU та Open vSwitch дало змогу реалізувати повноцінний тестовий полігон для перевірки роботи SDN-фаєрвола, відтворити

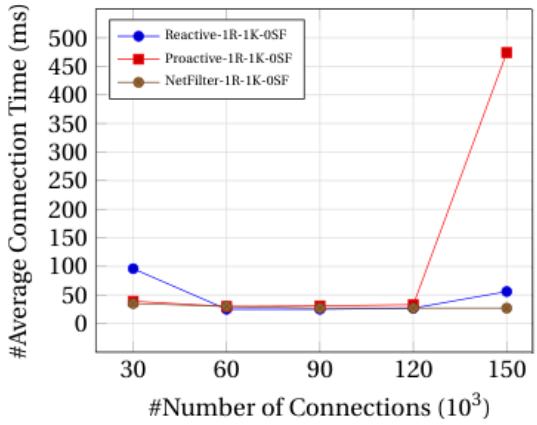
базові сценарії проактивної та реактивної фільтрації, а також дослідити реакцію системи на SYN-flood за реалістичних умов навантаження. Такий набір програмного забезпечення довів свою придатність для подальших досліджень продуктивності та стійкості рішень безпеки у програмно-конфігурованих мережах. У розгортанні не використовувався класичний гіпервізор типу KVM. Емуляція віртуальних вузлів виконувалася самим середовищем Mininet, яке застосовує ізоляцію на рівні network namespaces ядра Linux. Таким чином, усі хости, комутатори та лінки існували в межах одного фізичного сервера, але логічно утворювали окремі мережеві домени з повною підтримкою IP-стека, що забезпечує мінімальні накладні витрати порівняно з традиційною віртуалізацією.

Архітектура випробувань у Mininet передбачає централізований вузол SDN-фаєрвола з оркестратором та контролером RYU, під'єднаних до OVS, а також окремі хости Client, Server та Attacker. Клієнт генерує масив паралельних TCP-сесій через багатопроцесні/багатопотокові фонові завдання всередині вузла Client, піднімаючи кількість активних HTTP-процесів від 100 до 500, що відповідає 30·10^3…150·10^3 одночасних TCP-з'єднань; на боці Server запущено 500 екземплярів HTTP (Apache), кожен із фіксованим розміром відповіді, який змінювався між 1 кБ і 1 МБ залежно від експерименту. Окремий вузол Attacker генерує SYN-flood із керованою швидкістю за допомогою hping3 [25]. Спершу інтенсивність дорівнює нулю, далі становить 10^3 SYN/c і, нарешті, підвищується до 16·10^3 SYN/c, що еквівалентно вичерпанню пропускної здатності інтерфейсу OVS у межах емульованого хоста. Узгодженість і відтворюваність забезпечуються набором скриптів збору телеметрії на кожному вузлі Mininet: фіксуються OpenFlow-події, черги RYU, завантаження CPU та пам'яті OVS і хостів, а також повні дампи трафіку для ретроспективного аналізу.

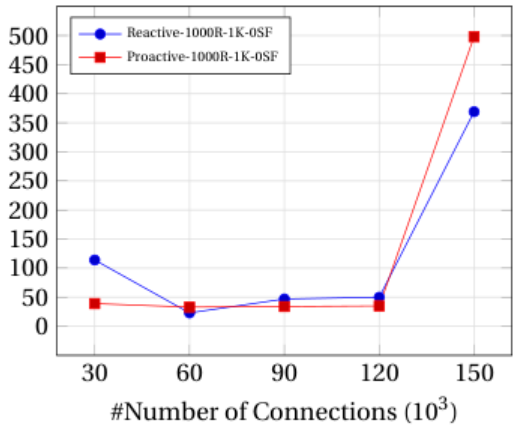
Порівнювані режими включають реактивну конфігурацію SDN-фаєрвола, коли правило доступу встановлюється після перехоплення трафіку у фаєрволі, і проактивну конфігурацію, коли перетворення FSM до EFSM для TCP виконується наперед, а OpenFlow-правила завантажуються до OVS до початку фактичних сесій. Для базового (еталонного) варіанту для порівняння використано NetFilter [26], а в мережевому елементі інстальовано транзитні

потоки «до NetFilter» та «від NetFilter», щоб увесь клієнт-серверний трафік проходив через цей вузол. Задля чистоти порівняння динамічне виявлення топології в SDN-фаєрволі вимкнено, топологія подається оркестратором у статичному режимі, що відповідає настроюванням, наведеним в описі реалізації.

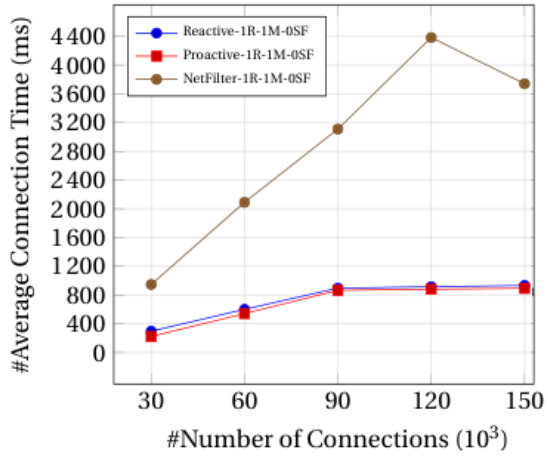
Метрики вимірювань охоплюють середній, мінімальний, медіанний, максимальний і стандартне відхилення часу встановлення одного з'єднання між клієнтом і сервером. Середній час оброблення пакетів керування TCP у фаєрволі та NetFilter; а також відсоток повторно пересланих сегментів керування TCP щодо їх загальної кількості. Сумарну картину за середніми часами встановлення з'єднань за різних факторів подано на рисунку 3.3, тоді як середні часи оброблення пакетів керування і їх деградація для NetFilter за різних навантажень наведені на рисунку 3.4, динаміка часток повторно пересланих пакетів зведена на рисунку 3.5.



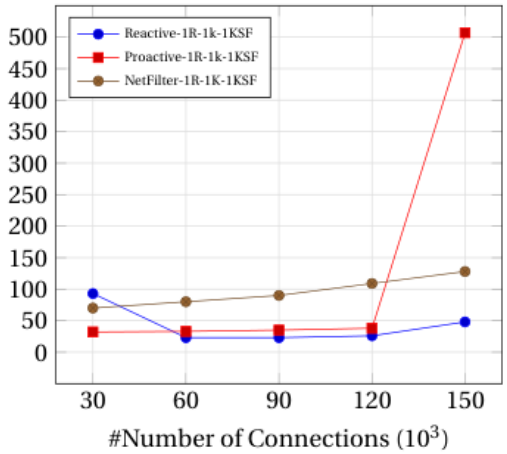
а)



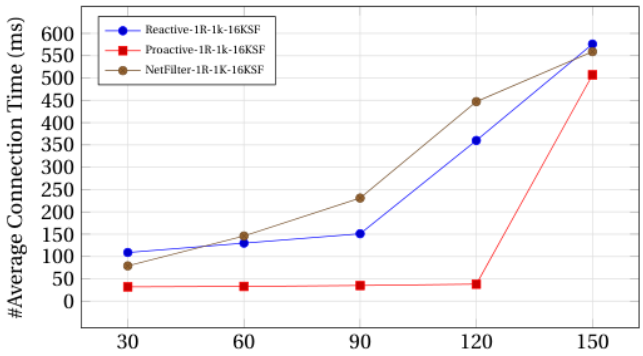
б)



в)

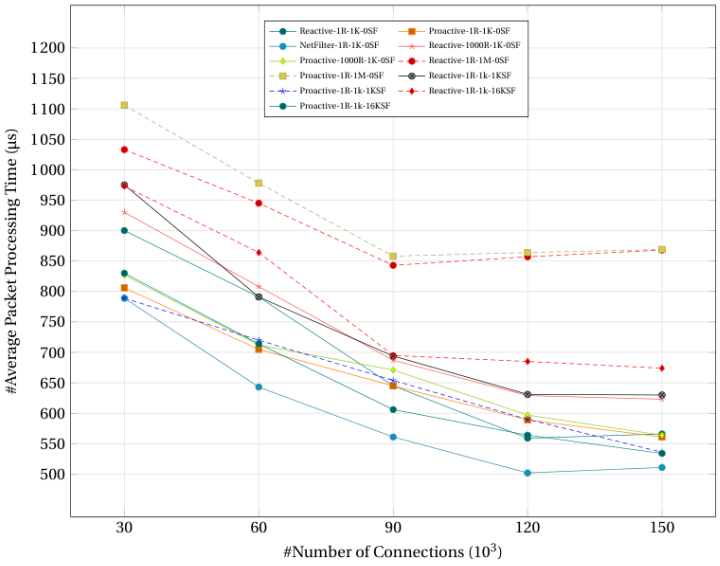


г)

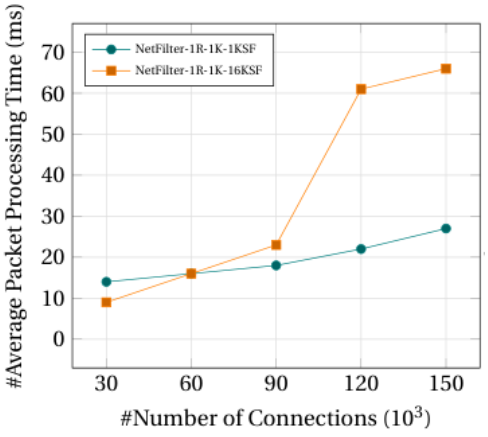


д)

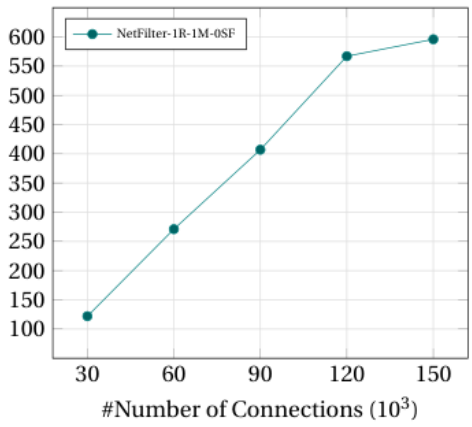
Рисунок 3.3 – Середній час встановлення з’єднань за різних факторів



а)



б)



в)

Рисунок 3.4 – Середній час оброблення пакетів керування і їх деградація за різних навантажень

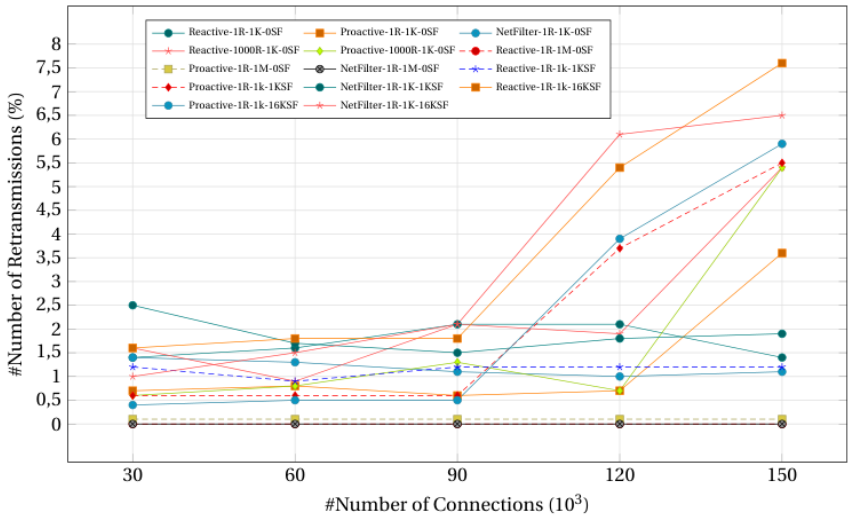


Рисунок 3.5 – Динаміка часток повторно пересланих пакетів

За базових умов з однією політикою доступу, обсягом відповіді 1 кБ і без атак середній час встановлення з'єднання для реактивного режиму становив 96 мс на 30·10^3 паралельних сесій, для проактивного - 39 мс, для NetFilter - 35 мс (див рисунок 3.3а). У цьому прикладі реактивна конфігурація демонструє додаткові накладні витрати, пов'язані з обробленням Packet-In подій і побудовою правил фаєрвола до їх інсталяції в OVS. Водночас картина змінюється із зростанням числа з'єднань: у діапазоні 60·10^3…120·10^3 спостерігається вирівнювання середніх часів для всіх трьох підходів, причому значення зближуються до 25–33 мс залежно від точки. Ця стабілізація пояснюється переходом у стаціонарний режим навантаження на генераторі з'єднань і більш рівномірною завантаженістю процесора OVS та черг подій RYU. За 150·10^3 з'єднань зафіксовано різку деградацію в проактивному режимі 474 мс на тлі лише помірного росту до 56 мс у реактивному й майже сталої затримки в NetFilter. Причиною стрибка є досягнення високих рівнів використання ресурсів у мережевому елементі в проактивному сценарії: середнє завантаження CPU OVS близько 95% та використання пам'яті близько 84% створюють вузькі місця при масовому встановленні заздалегідь згенерованих правил.

Збільшення розміру таблиці доступу до 10^3 правил практично не вплинуло на середні часи у проактивній конфігурації, адже основна робота порівняння виконується апаратним прискоренням на рівні OVS та його механізмів швидкого пошуку по полях заголовків (див рисунок 3.3б). Натомість реактивний режим

відчутно чутливий до росту числа правил, оскільки зіставлення трафіку з політиками виконується у фаєрволі до моменту встановлення потоку в OVS. На 90·10^3 з'єднань середній час зріс до 47 мс, на 120·10^3 - до 50 мс, а на 150·10^3 - до 369 мс. Цей ривок корелює з одноядерними обмеженнями виконання Python-логіки та з тим, що, попри багатонитковість, ефективний паралелізм обмежується, що підвищує латентність при інтенсивних пошуках у великій таблиці правил. Таким чином, проактивний режим краще масштабується за множиною політик, покладаючись на площину пересилання, тоді як реактивний - зазнає зростання накладних витрат у площині керування.

Зміна розміру даних що передаються створює іншу картину. За відповіді 1 МБ на з'єднання всі три підходи демонструють збільшення часу встановлення, однак найбільша деградація відбулася в NetFilter, де середні значення зросли до 946 мс на 30·10^3 і до 4384 мс на 120·10^3 з подальшим зниженням до 3741 мс на 150·10^3 (див рисунок 3.3в). Лінійне зростання в середині діапазону відповідає вичерпанню ресурсів: спершу повне завантаження одного ядра і приблизно 90% пам'яті, далі розподіл обчислень на друге ядро за повністю зайнятої пам'яті, а також заповнення пам'яті OVS уже з 90·10^3 з'єднань. Для проактивного та реактивного SDN-сценаріїв спостерігається лінійний ріст від 221 і 296 мс на 30·10^3 до 865 і 895 мс на 90·10^3, після чого збільшення стає незначним до кінця діапазону. Це результат того, що в SDN-варіантах обсяг корисних даних не проходить через фаєрвол, а лише умови переходів між станами, тоді як NetFilter інспектує трафік ін-лайн.

Аналіз стійкості до SYN-flood підтверджує перевагу проактивних інсталяцій правил у площині пересилання. Для інтенсивності 10^3 SYN/c середні часи у реактивному режимі залишилися близькими до початкових, а у проактивному - практично не змінилися, оскільки потоки атакувальника відсікаються на рівні OVS, не досягаючи контролера (див рисунок 3.3г). За 16·10^3 SYN/c реактивна схема демонструє помірне зростання від 109 мс на 30·10^3 до 151 мс на 90·10^3 і різкий стрибок до 576 мс на 150·10^3, що прямо корелює з нарощуванням споживання пам'яті та CPU як у прикладі фаєрвола, так і в OVS (див рисунок 3.3д). NetFilter виявився найбільш уразливим до SYN-flood

у всьому діапазоні інтенсивностей: при 10^3 SYN/c середні затримки зросли вчетверо відносно базових, а при 16·10^3 SYN/c - до двадцятикратних значень у верхньому діапазоні навантаження. Сукупно це свідчить, що фільтрація трафіку, котрий не відповідає законним переходам EFSM, безпосередньо в мережевому елементі забезпечує не лише нижчі затримки у нормальному режимі, а й стабільність під час атак із насиченням.

Внутрішня телеметрія часу оброблення контрольних пакетів TCP поглиблює картину. За початкових умов середні час оброблення одного контрольного пакета (PPT) перебував в межах сотень мікросекунд для всіх трьох систем: 566–900 мкс для реактивного SDN, 561–806 мкс для проактивного і 511–789 мкс для NetFilter (див рисунок 3.4а). Надалі реактивний режим, як правило, має вищі PPT за проактивний через більший обсяг логіки у фаєрволі. Єдиним винятком є сценарії з транспортом великих даних, де додаткові дії в проактивному потоці призводять до зростання середніх PPT. Для NetFilter поза початковими умовами PPT різко збільшується. Коли SYN-flood значення досягають десятків мілісекунд, а на 16·10^3 SYN/c і 150·10^3 з'єднань - приблизно 66 мс, що у 75 разів перевищує відповідні показники SDN-підходів у тому ж експерименті (див рисунок 3.4б). За великого обсягу даних PPT NetFilter зростає ще суттєвіше від 122 мс на 30·10^3 до 596 мс на 150·10^3 (див рисунок 3.4в). Саме ці великі затримки на оброблення контрольних сегментів зумовлюють різке збільшення середнього часу встановлення з'єднання в NetFilter, фіксуючи причинно-наслідковий зв'язок між інтенсивністю інспекції, вичерпанням ресурсів і кінцевою затримкою для клієнт-серверних сесій.

Додаткове пояснення динаміки дає аналіз частки повторно пересланих контрольних пакетів TCP. На всьому інтервалі 30·10^3…90·10^3 з'єднань відносні показники стабільні, а мінімальні значення спостерігаються в експериментах з великими даними, де у реактивному SDN і NetFilter пересилання практично немає, а в проактивному їхня частка становить близько 0,1% і не зростає з навантаженням (див рисунок 3.5). Це узгоджується з тим, що повторні контольні сегменти проходять спрощене опрацювання та не потребують перевстіновлення правил, обмежуючись звірянням з таблицями

доступу та станів, що знижує середній PPT. Натомість при переході до 120·10^3…150·10^3 з'єднань і високих рівнів SYN-flood саме реактивний SDN та NetFilter демонструють різкі збільшення частки пересилань до 7,6% і 6,5% відповідно, тоді як у проактивному режимі ріст помірніший і пов'язаний насамперед із завантаженням OVS. Це прямо корелює зі стрибками середнього часу на див рисунок 3.3д і підтверджує, що втрата або затримка контольних сегментів під атакою веде до каскадного збільшення латентності на рівні встановлення сесії.

З точки зору механізмів, що стоять за спостережуваними ефектами, критичну роль відіграє місце, де виконується порівняння політики доступу. Якщо зіставлення здійснює площина пересилання у vSwitch, і правила інстальовані наперед, трафік, що не пройшов перевірку станів EFSM, відсікається ще до взаємодії з контролером і фаєрволом. У такій конфігурації навіть сильний фон SYN-flood майже не впливає на узгодженість метрик і середні часи встановлення легітимних сесій. Якщо ж зіставлення виконується в прикладній логіці до інсталяції потоку, накопичення подій Packet-In у чергах RYU та посилення тиску на єдиний інтерпретатор у Python під атакою чи при різкому збільшенні розміру таблиць правил створюють помітні додаткові затримки. У випадку NetFilter інспекція даних здійснюється ін-лайн, що робить систему особливо чутливою до зростання обсягу корисних даних і до інтенсивності атак, оскільки і контольні, і частина даних проходять через вузол перевірки, виснажуючи процесор і пам'ять.

Таким чином, експериментальна оцінка демонструє низку стійких закономірностей. По-перше, проактивна інсталяція правил у площині пересилання забезпечує мінімальні середні затримки за нормального режиму і найвищу стійкість до SYN-flood завдяки тому, що шкідливий трафік відсікається на межі vSwitch до ескалації на рівень контролера. По-друге, реактивний підхід є конкурентоздатним у базових умовах та помірних навантаженнях, але стає чутливим до росту розміру таблиць доступу і високих темпів атак за рахунок накладних витрат у фаєрволі та чергах подій. По-третє, NetFilter, будучи ін-лайн рішенням, показує прийнятні часи за малих обсягів даних і відсутності атак,

проте різко деградує як при збільшенні корисного навантаження, так і під інтенсивним SYN-flood, що проявляється одночасним зростанням середнього часу встановлення, середнього часу оброблення контрольних пакетів і частки повторних пересилань.

3.3 Необхідність централізованої оркестрації політик безпеки

Сучасні мережеві інфраструктури, що працюють у хмарних та віртуалізованих середовищах, еволюціонують у бік високої динаміки топологій, швидкого життєвого циклу сервісів і розподілу функцій між численними вузлами площини даних та площини керування. За таких умов безпека перестає бути статично заданим набором правил і перетворюється на керовану, вимірювану та постійно узгоджувану властивість системи. Локальне налаштування політик безпеки на рівні окремих мережевих елементів уже не гарантує цілісності й узгодженості, адже множиться кількість точок зміни, а вплив однієї помилки здатний каскадно поширюватися ланцюжком залежностей. Саме тому в середовищі програмно-визначених мереж критичною стає централізована оркестрація політик безпеки, що поєднує глобальне бачення контролера з формальним описом вимог і автоматичною трансляцією у правила площини пересилання. Місце, де відбувається перевірка доступу та станів з’єднань, визначає як продуктивність у нормальному режимі, так і стійкість під час атаки. Побудована тестова платформа з чітко розділеними контурами трафіку для варіантів з централізованим SDN-керуванням і з класичним NetFilter надає прозоре зіставлення шляхів проходження пакетів і вимірювання внеску кожного компонента. Загальну схему такого стенду наведено на рисунку 3.2, де окремо показані контури для SDN-фаєрвола із перевіркою стану з’єднання та для NetFilter. У цій конфігурації оркестратор, фаєрвол і контролер працюють на виділеному вузлі, мережеві елементи з Open vSwitch рознесені фізично, а клієнт, сервер і атакуюча машина реалізовані як окремі віртуальні машини з виділеними інтерфейсами. Такий поділ не випадковий. Він дозволяє не лише коректно порівняти варіанти політики доступу, а й точно виявити, на якому етапі

життєвого циклу правила виникають додаткові витрати часу - у прикладній логіці, на рівні контролера чи під час інсталяції правил у комутаторі.

Порівняння середніх часів встановлення з’єднання під різними сценаріями показує, що відмова від локальної, розрізненої конфігурації на користь централізованого, заздалегідь підготовленого набору правил зменшує затримки і нарощує запас стійкості. У початкових умовах із мінімальною таблицею доступу та малим обсягом переданих даних реактивний підхід демонструє вищі затримки, ніж проактивний і варіант NetFilter, що видно з порівняння на рисунку 3.3а. Середній час для реактивної схеми становив близько 96 мілісекунд на тридцять тисяч паралельних сесій, тоді як у проактивній - 39 мілісекунд, у NetFilter - 35 мілісекунд. Однак при масштабуванні навантаження до шістдесяти - ста двадцяти тисяч сесій усі три криві збігаються до близьких значень у межах кількох десятків мілісекунд, що вказує на ефект вирівнювання черг і більш рівномірне завантаження мережевого елемента та контролера. Центральна ідея тут полягає не в тому, щоб покращити класичний механізм у кожній точці діапазону, а в тому, що централізована підготовка й узгоджене застосування правил усувають зайві коливання та роблять поведінку системи передбачуваною.

Критичні відмінності проявляються, щойно збільшується складність політик і обсяги даних. Коли таблиця доступу розширюється до тисячі записів, проактивна інсталяція правил у площину пересилання практично не змінює середні часи, адже зіставлення виконується в Open vSwitch апаратно прискореними структурами пошуку, і це чітко видно на рисунку 3.3б. Реактивна ж логіка змушена щоразу проходити додаткові цикли в прикладному модулі фаєрвола, перш ніж встановити потік, і тим самим накопичує затримки, які різко зростають на верхніх рівнях навантаження. Коли ж у кожній сесії передаються мегабайтні обсяги даних, розбіжності стають ще разючішими. На рисунку 3.3в видно, як NetFilter через інспекцію даних ін-лайн входить у режим різкої деградації, тоді як SDN-підходи, що передають у площину керування лише умови переходів між станами, утримують затримки у прогнозованих межах. Інакше кажучи, централізація політик із перенесенням перевірок на рівень

елементів мережі фактично відділяє контроль від масивного трафіку даних і тим самим стабілізує часові характеристики.

Під час атак з насиченням переваги централізованої оркестрації стають вирішальними. На рисунках 3.3г і 3.3д наведено результати для двох інтенсивностей SYN-flood. Проактивний варіант, у якому правила відсікання невідповідних станам EFSM пакетів інстальовано наперед, демонструє практично повну інваріантність до атаки. На вході в комутатор SYN-хвиля гаситься без генерації зайвих подій для контролера. Реактивна схема витримує помірні інтенсивності, але на рівні шістнадцяти тисяч SYN за секунду починає відчувати дефіцит ресурсів у прикладній логіці та в самому Open vSwitch, що проявляється лавиноподібним зростанням середніх часів у верхній частині навантаження. NetFilter же, працюючи як ін-лайн фільтр, найбільш чутливий до обох чинників - і до обсягу даних, і до інтенсивності атаки. Узагальнюючи, централізований режим із завчасним розгортанням правил на межі комутатора мінімізує площу атаки керувальної площини та забезпечує стабільність метрик навіть у стресових сценаріях.

Детальні телеметричні показники часу оброблення контрольних пакетів доповнюють картину технічним обґрунтуванням. На рисунку 3.4а видно, що в базових умовах середні часи опрацювання SYN/ACK/FIN у реактивному та проактивному SDN-підходах і у NetFilter лежать у діапазоні сотень мікросекунд, проте поза цими умовами саме NetFilter виходить на десятки мілісекунд, а за великих обсягів даних до сотень мілісекунд (рисунки 3.4б і 3.4в). Це прямий індикатор того, що ін-лайн інспекція під навантаженням вичерпує ресурси швидше й формує черги, тоді як централізована оркестрація з правильним розподілом обов’язків переносить вузьке місце із прикладної логіки на рівень швидкого, заздалегідь оптимізованого зіставлення в мережевих пристроях. Те, що відсоток повторно пересланих контрольних сегментів у сценаріях з великими даними мінімальний для SDN-варіантів і зростає лише у верхніх режимах навантаження атаками (рисунок 3.5), додатково вказує на відсутність систематичних збоїв у керуванні при централізованій стратегії.

В програмній архітектурі оркестратора мають бути зібрані у межах одного керувального контуру функції приймання й валідації політик, їх узгодження та трансляції в низькорівневі правила, а також зворотний канал телеметрії. Центральний модуль політик взаємодіє з менеджером хмарних політик, який координує процеси оцінювання та розгортання, тоді як сервер політик оперує глобальною таблицею доступу і здійснює інтерфейс до прикладного фаєрвола та контролера. Такий поділ обов'язків важливий не лише з погляду чистоти коду чи керованості, а і з позиції часової декомпозиції. Ми можемо окремо виміряти, скільки часу витрачається на логіку оркестратора, скільки на інтерпретацію в фаєрволі, скільки на відправку команд контролером і скільки на інсталяцію в комутаторі. Саме така прозорість дозволяє знайти й підсилити вузьке місце.

Оркестратор потрібен не лише для того, щоб мати центр керування, а щоб оптимізувати сам спосіб взаємодії з інфраструктурою: пакетувати зміни, мінімізувати дублікати, розумно комбінувати операції додавання, модифікації та видалення, зменшувати конфлікти й перезаписи. Локальні, ручні чи розподілені підходи позбавлені такого глобального огляду й нездатні системно зменшити витрати на інфраструктурну фазу.

Ще один аспект необхідності централізації - конфлікти політик і їх узгодженість між численними клієнтами та провайдерами. Навіть у межах одного домену безпеки політики можуть містити неоднозначності, що виникають через різні контексти, часові обмеження, перекриття суб'єктів і об'єктів. У розподіленій конфігурації виявлення та розв'язання конфліктів відбувається постфактум, після появи аномалій у трафіку або деградації продуктивності. Централізований оркестратор, володіючи єдиною моделлю опису політик та глобальною таблицею доступу, може виявляти такі ситуації на етапі формування угод і до розгортання у мережевих елементах. Принцип залишається незмінним: централізація приносить із собою формалізацію, а формалізація - можливість перевірки властивостей політик до їх застосування. У технічному вимірі це означає менше непередбачених змін, менше аварійних оновлень і меншу кількість реконфігурацій, що торкаються великої кількості елементів мережі одночасно.

Проблематика масштабування також вирішується саме централізовано. Коли кількість політик зростає до тисяч і десятків тисяч, локальні процедури перестають бути ефективними. Відсутність глобальної видимості перешкоджає агрегації правил, перерозподілу пріоритетів і ущільненню наборів зіставлення. Централізований оркестратор, маючи повну картину топології та навантаження, здатний групувати високорівневі політики в мінімальну кількість низькорівневих операцій, синхронізувати зміни з урахуванням стану черг у контролері, а також динамічно обмежувати швидкість розгортання, щоб не перевантажувати Open vSwitch. Централізація політик безпеки у програмно-визначених мережах - це не лише про продуктивність і стійкість до атак, а й про керованість змін і відтворюваність результатів. Завдяки єдиному життєвому циклу «створення → оцінювання → узгодження → розгортання → телеметрія → корекція» оркестратор забезпечує можливість відслідкувати кожну правку від бізнес-вимоги до конкретного запису у таблиці потоків. За необхідності можна локалізувати причину відхилення метрик до конкретної версії набору правил або до окремої FlowMod-операції, що була застосована на певному інтерфейсі. У термінах практичної експлуатації це означає коротші вікна змін, менше ручних операцій у нічний час і менше регресій при розширенні периметра захисту. Нарешті, слід підкреслити взаємозв’язок централізації та фізичного розміщення логіки перевірки. Найбільший виграш досягається тоді, коли і формування, і застосування правил максимально зближені з площиною пересилання, а частка обробки в прикладній логіці та на рівні контролера мінімізована. Централізована оркестрація створює умови для саме такого розміщення, бо дозволяє планувати інсталяцію, піднімати ефективність кешів у комутаторі, уникати зайвих модифікацій правил і відсікти шкідливий трафік до взаємодії з площиною керування. Роль NetFilter у цій картині стає еталонною точкою порівняння, а не альтернативною стратегією масштабування.

Таким чином, необхідність централізованої оркестрації політик безпеки в середовищі програмно-визначених мереж випливає з трьох взаємопов’язаних спостережень. По-перше, лише централізована модель забезпечує узгодженість і формальну перевірку політик перед розгортанням, що знижує ризик конфліктів

і помилок конфігурації. По-друге, лише завчасна інсталяція правил на рівні мережевої інфраструктури гарантує низькі затримки та стабільність під час атаки. По-третє, саме централізація надає змогу оптимізувати інфраструктурну фазу застосування правил, яка домінує в загальному часі життєвого циклу політики і тим самим підвищити пропускну здатність системи безпеки без жертвування точністю.

3.4 Модель взаємодії учасників NSC–NSP у контексті централізованої оркестрації політик безпеки

В основі розподілених мережевих систем із програмно-визначеною архітектурою лежить ідея абстрагування рівнів управління, контролю та пересилання даних. Це дає змогу централізовано керувати потоками трафіку, політиками доступу та механізмами безпеки, зберігаючи при цьому масштабованість і гнучкість на рівні інфраструктури. Проте в реальному середовищі взаємодія між споживачем мережевих послуг (NC) і провайдером безпеки (SP) має складну багаторівневу природу, яка вимагає уніфікованого формалізму для опису вимог, зобов’язань, узгодження умов і розгортання результатів у вигляді конкретних OpenFlow-правил. Ця взаємодія в запропонованій архітектурі реалізується через централізований оркестратор, який виступає посередником між NC і SP, забезпечуючи семантичну узгодженість, автоматизоване оцінювання сумісності політик, формування контрактів і їх трансляцію в інфраструктурні елементи. На рисунку 3.6 наведено архітектуру реалізації моделі політик, яка складається з взаємопов’язаних компонентів - обробника політик, менеджера хмарних політик, агента ранжування, агента переговорів, видавця контрактів і сервера політик, що взаємодіє з фаєрвол-додатками та контролером RYU.

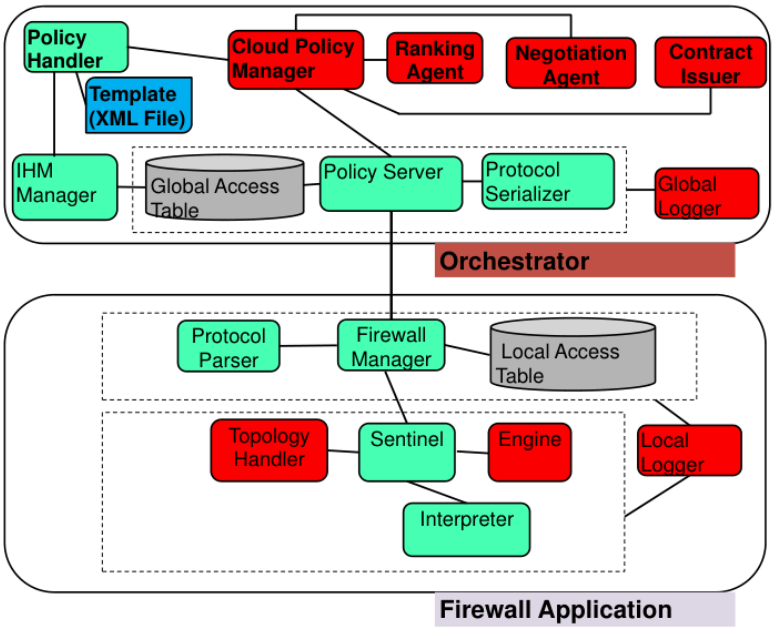


Рисунок 3.6 – Архітектура моделі політик

Побудова моделі взаємодії між NC і SP потребує єдиного лінгвістичного простору, у якому обидві сторони можуть формулювати свої вимоги та зобов'язання. Для цього запропоновано мову політик, побудовану на моделі доступу на основі атрибутів (ABAC). Вона дозволяє виражати політики безпеки у вигляді множини атомарних правил, де кожне правило складається з набору елементів - суб'єкта, дії, об'єкта і контексту. Такий підхід уніфікує семантику опису, уникає неоднозначностей між провайдерами і споживачами, а також дозволяє оркестратору автоматично співставляти вимоги NC з можливостями SP.

У межах цієї мови кожна політика представляється як логічна кон'юнкція атомарних елементів $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$, де кожен елемент має тип, домен, значення і область видимості. Тип елемента визначає його роль у політиці (наприклад, дія або контекст), домен задає категорію параметра (протокол, час, IP-адреса тощо), значення може бути як константою, так і множиною змінних, а область видимості визначає, чи є цей параметр публічною або приватною перевагою. У такий спосіб мова політик дозволяє описати як статичні, так і динамічні аспекти безпеки, не втрачаючи універсальності при переході між різними середовищами NSP.

Для досягнення сумісності між вимогами NC та зобов'язаннями SP оркестратор виконує процес оцінювання, який спирається на відношення між

елементами політик. Встановлюються такі зв’язки, як еквівалентність, порівнянність, нерівність або несумісність елементів. На їх основі визначається ступінь відповідності політик - повна відповідність (match), потенційна відповідність (potential match) або невідповідність (mismatch). Якщо політики мають лише потенційну відповідність, оркестратор ініціює переговорний процес між NC і SP для досягнення згоди.

Механізм переговорів реалізується через протокол взаємодії, який забезпечує формалізовану процедуру погодження параметрів між сторонами. Його головна мета - досягти узгодження між політиками без необхідності розкриття приватних налаштувань провайдера чи споживача. Протокол взаємодії базується на зіставленні запропонованих значень і локальних значень для кожного атомарного елемента політики. Залежно від типу значень і їх видимості визначається подальша дія: прийняття (accept), відмова (refuse) або формування контрпропозиції (propose). Якщо, наприклад, значення, отримане від SP, є константою і збігається з локальним значенням NC, оркестратор приймає його без змін. Якщо отримане значення входить до публічної множини допустимих значень, то оркестратор може запропонувати власне уточнення в межах цієї множини. Коли ж SP має приватну перевагу і не розкриває свої внутрішні обмеження, оркестратор формує компромісну пропозицію, використовуючи часткове перетинання множин значень або значення за замовчуванням, що належить локальній конфігурації NC. Така логіка дозволяє проводити переговори без порушення конфіденційності сторін.

Результатом протоколу є узгоджений контракт, у якому кожен параметр має погоджене значення.

3.5 Результати експериментів оркестрації

Експериментальна перевірка функціонування централізованої системи оркестрації політик безпеки мала на меті кількісно оцінити її ефективність, стабільність і масштабованість у межах повного життєвого циклу політики - від моменту її створення користувачем до інсталяції відповідних правил у

мережевих елементах. Цей цикл охоплює етапи створення політики, оцінювання відповідності між вимогами споживача і можливостями провайдера, переговори з уточненням параметрів, трансляцію узгодженої політики у низькорівневі OpenFlow-правила, їх відправлення через контролер RYU та застосування у віртуальних комутаторах Open vSwitch. Архітектура прототипу, що реалізує ці кроки, була наведена на рисунку 3.6, де позначено всі модулі, які беруть участь у процесі - оркестратор з обробником політик, хмарний менеджер політик, агенти ранжування та переговорів, генератор контрактів, сервер політик, інтерпретатор і модуль Sentinel, який відповідає за передачу команд у контролер.

Для вимірювання ефективності було визначено низку часових метрик, що відображають продуктивність окремих компонентів і системи в цілому. До них належать час оброблення виразу політики оркестратором (PPT), загальний час роботи оркестратора (OPT), час інтерпретації політик фаєрвол-додатком (FAPT), час роботи контролера (CPT), час інсталяції правил у комутаторі (IPT), а також сумарний час реалізації політики (PPTT). Додатково визначалися швидкості налаштування - оркестрації (OSR), фаєрвола (FSR), контролерна (CSR) та інфраструктури (ISR), що відображають кількість політик, які можуть бути оброблені, інтерпретовані чи інстальовані за одиницю часу.

Отримані результати показали, що оркестраційна система демонструє стійку масштабованість із лінійним зростанням усіх часових характеристик при збільшенні обсягу політик. На рисунку 3.7 зображено залежність часу PPT, OPT, FAPT і CPT від кількості політик.

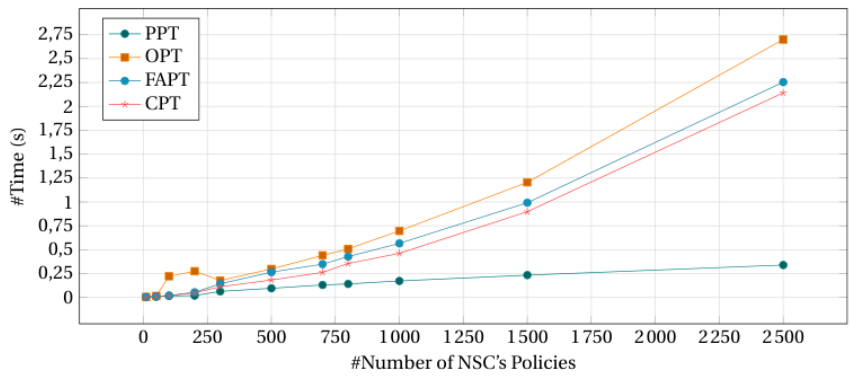


Рисунок 3.7 – Залежність часу PPT, OPT, FAPT і CPT від кількості політик

Час оброблення однієї політики оркестратором зростав від 0.00236 секунди для десяти політик до приблизно 0.6421 секунди для двох з половиною тисяч політик, що вказує на логічне зростання, пропорційне збільшенню обсягу даних. Така поведінка пояснюється тим, що оркестратор виконує серію послідовних операцій - синтаксичний і семантичний аналіз політики, перевірку на конфлікти, оцінювання відповідності, формування договору й передавання його до сервера політик. Кожна з цих операцій має сталу часову складову, тому збільшення кількості політик призводить до передбачуваного лінійного росту загального часу. Загальний час роботи оркестратора OPT та фаєрвол-додатка FAPT демонструє подібну динаміку, хоча абсолютні значення вищі, оскільки включають також час на комунікацію між модулями через сокети та час очікування відповіді від політичного сервера. Для обсягу у 2500 політик середні значення OPT, FAPT та CPT становили приблизно 2.700, 2.254 і 2.141 секунди відповідно. Це свідчить про близькість рівня навантаження на різних етапах площини керування й узгодженість між компонентами системи. Затримка на контролері CPT практично повторює динаміку FAPT, адже обидва елементи працюють у безпосередній зв’язці через модуль Sentinel: фаєрвол-додаток формує запит на встановлення або видалення правил, а контролер RYU передає їх у форматі OpenFlow до комутатора.

Найбільші часові витрати припадають на інфраструктурну фазу, що включає доставлення FlowMod-запитів до комутатора і безпосередню інсталяцію правил у таблицях потоків. Як показано на рисунку 3.8, значення IPT для 2500 політик сягає 11.025 секунди, тобто приблизно 60% усього часу PPTT.

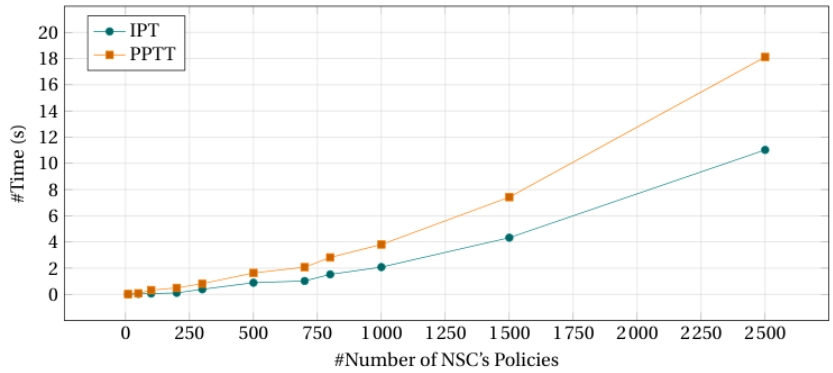


Рисунок 3.8 – Залежність часу IPT і PPP від кількості політик

Це означає, що продуктивність оркестраційної системи визначається насамперед швидкістю операцій у площині пересилання, а не складністю оброблення політик на рівні оркестратора чи фаєрвол-додатка. При цьому сам факт того, що решта 40% часу розподіляється майже рівномірно між PPT, FAPT, CPT і OPT, свідчить про ефективну реалізацію міжмодульної взаємодії без суттєвих затримок у чергах або блокуваннях потоків.

Динаміка швидкостей налаштування, наведена на рисунку 3.9, показує три характерні фази.

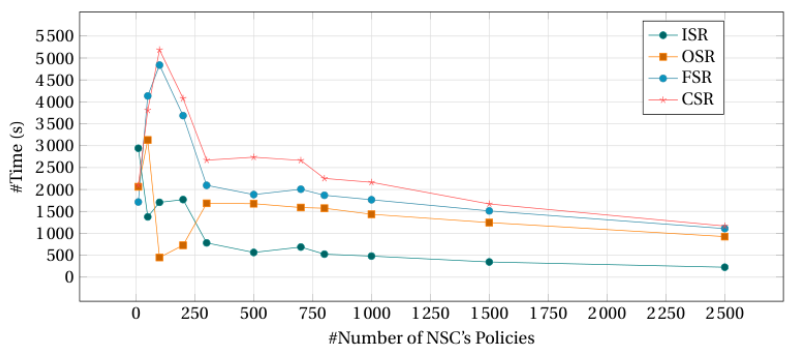


Рисунок 3.9 – Залежність часу ISR, OSR, FSR і CSR від кількості політик

На початковій фазі, коли кількість політик невелика (до 100), усі швидкості мають пікові значення. Швидкість контролера CSR становить приблизно 5186 одиниць, фаєрвола FSR - близько 4838, оркестрації OSR - 3150, а інфраструктурна ISR - близько 2941. Це пояснюється наявністю кешованих структур даних, порожніх черг у контролері та комутаторі й відсутністю конфліктів під час запису правил у таблиці потоків. У середній фазі, коли кількість політик зростає до тисячі, швидкості поступово знижуються. Це пов’язано з необхідністю дедуплікації політик, перевірки пріоритетів і синхронізації між потоками в оркестраторі. У завершальній фазі, після перевищення порогу 2000 політик, темп падіння ISR стає помітно більшим порівняно з іншими швидкостями. Це свідчить, що саме інфраструктурна складова, пов’язана з інсталяцією правил у OVS, стає головним обмежувальним чинником системи.

Наявність цього вузького місця підтверджує висновки, зроблені в розділі про SDN-фаєрвол із перевіркою стану з'єднання: продуктивність архітектури визначається тим, наскільки близько логіка перевірки розташована до площини пересилання. Якщо оброблення політик виконується централізовано й правила заздалегідь інстальовані в комутаторі, система зберігає низьку латентність і високу пропускну здатність навіть при великій кількості політик. Однак у разі масових оновлень таблиць потоків швидкість інсталяції стає домінуючим фактором. Це проявляється в нерівномірній деградації ISR, що відображає періодичні ефекти конвеєризації, блокувань і перезаписів записів у таблицях OpenFlow.

Візуальний аналіз показників підтверджує лінійний характер зростання часових витрат та обернену залежність швидкостей від кількості політик. Для невеликих обсягів системі властива мала затримка і висока динамічність розгортання, але при масштабуванні до тисяч політик спостерігається поступове насичення каналу контролера і збільшення часу на застосування FlowMod-запитів у комутаторі. Це типовий ефект для SDN-середовищ, у яких частота оновлення таблиць потоків обмежується не логічною складністю політик, а фізичними параметрами оброблення пакетів у Open vSwitch. Отримані результати вказують, що для підвищення продуктивності необхідно оптимізувати саме інфраструктурний етап: групувати правила за контекстом, застосовувати пакетну обробку FlowMod-запитів і використовувати асинхронну модель оновлення, яка дозволяє виконувати частину модифікацій у фоновому режимі без блокування основних потоків.

У процесі розгортання великої кількості політик спостерігалося поступове збільшення використання ресурсів CPU і пам'яті оркестратора, але без нелінійних стрибків. Середнє завантаження процесора при 2500 політиках не перевищувало 70%, що підтверджує ефективність реалізованої багатопотокової моделі з контролем черг. Аналогічно, використання пам'яті залишалося в межах 60–65%, а отже, система здатна до подальшого масштабування без необхідності змін архітектури. Під час експериментів із варіюванням частоти оновлення політик було помічено, що навіть при одночасній зміні сотень політик затримка

між моментом підтвердження користувачем і появою правил у таблиці потоків залишалася у межах прогнозованих лінійних залежностей. Це свідчить про те, що механізм узгодження й трансляції політик у межах оркестратора має детермінований характер і не накопичує помилок при паралельних змінах. У практичному сенсі це означає, що система придатна для використання в сценаріях динамічної безпеки - наприклад, при автоматичному оновленні політик у відповідь на зміну контексту трафіку або стану вузлів.

3.6 Висновки до розділу 3

У третьому розділі було здійснено експериментальну оцінку SDN-фаєрвола із перевіркою стану з’єднання та обґрунтовано централізовану оркестрацію політик. На стенді Mininet з RYU і Open vSwitch порівняно реактивний і проактивний підходи з класичним NetFilter. Показано, що проактивна інсталяція правил у площині пересилання забезпечує нижчі затримки за нормальних умов і найвищу стійкість до SYN-flood, оскільки невідповідний EFSM станам трафік відсікається на рівні мережевого обладнання без навантаження контролера. Реактивний режим придатний за помірного навантаження, але чутливий до збільшення обсягу політик і частоти Packet-In, тоді як NetFilter деградує при великих даних і під час атак через ін-лайн інспекцію. Оркестраційні експерименти засвідчили лінійне масштабування часових характеристик і виявили інфраструктурну фазу інсталяції правил у OVS як головне вузьке місце життєвого циклу політики. Решта складових (opchestrator, firewall-app, controller) працюють узгоджено без суттєвих черг. Централізована оркестрація забезпечує формальну узгодженість, відтворюваність змін і прогнозовану продуктивність, мінімізує площу атаки площини контролю та створює основу для масштабованої, адаптивної і стійкої мережевої безпеки у хмарних і віртуалізованих інфраструктурах.

РОЗДІЛ 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Метою роботи є підвищення мережевої безпеки в SDN за рахунок централізованого керування політиками доступу, впровадження фаєрвола із перевіркою стану з'єднання. При створенні подібних засобів потрібно дотримуватись основні правила і норми експлуатації комп'ютерів та периферійних пристроїв під час проведення дослідження.

В загальному, поняття охорона праці в комп'ютерних системах являє собою дотримання всіх вимог і нормативів, що присутні в законодавчих актах про охорону праці. Закони цієї області спрямовані на якісну і безпечну експлуатацію робочих приладів і приміщень, дотримання санітарно-гігієнічних умов праці і захист від інших небезпечних чинників на підприємстві. Ці засоби є складовими дослідження математичного і програмного забезпечення автоматизованої системи підбору команди розробників комп'ютерних систем. В основних законодавчих актах про охорону праці приділяється велика увага поліпшенню умов праці в усіх галузях господарства, впровадженню сучасних засобів техніки безпеки і забезпечення санітарно-гігієнічних умов, що запобігають виробничому травматизму і професійним захворюванням.

Охорона життя і здоров'я людини є пріоритетним напрямком соціальної політики держави. В Україні прийнято закон прямої дії «Про охорону праці», який регламентує захист конституційного права працівників на безпечні умови праці. Законодавство України про охорону праці складається із загальних законів України та спеціальних законодавчих актів. Загальними законами України, що визначають основні положення з охорони праці є Конституція України, Закон України «Про охорону праці», Кодекс законів про працю (КЗпП), Закон України «Про загальнообов'язкове державне соціальне страхування від нещасного випадку на виробництві та професійного захворювання, які спричинили втрату працездатності».

Виконання досліджень кваліфікаційної роботи передбачали використання ПК, де площа та об'єм для одного робочого місця оператора визначається згідно вимог НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», зокрема площа повинна становити не менше 6,0 квадратних метрів, об'єм - не менше 20,0 кубічних метрів.

Згідно вимог охорони праці та державних санітарних правил, стіни, стеля та підлога приміщень, в яких розміщені ЕОМ, повинні бути виготовлені з матеріалів, дозволених для оформлення приміщень органами державного санітарно-епідеміологічного нагляду.

Заземлені конструкції, що знаходяться в приміщеннях, де розміщені робочі місця операторів (батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном), повинні бути надійно захищені діелектричними щитками та сітками з метою недопущення потрапляння працівника під напругу.

Організація робочого місця оператора повинна забезпечувати відповідність усіх елементів робочого місця та їх розташування ергономічним вимогам. У приміщенні, де одночасно експлуатуються понад п'ять електронно-обчислювальних машин (ЕОМ), на помітному та доступному місці мають бути встановленні аварійні резервні вимикачі, які можуть повністю вимкнути електричне живлення приміщення, крім освітлення [27].

Дотримання правил значно знижує наслідки несприятливої дії на працівників шкідливих та небезпечних факторів, які супроводжують роботу з відео-дисплейними терміналами, зокрема можливість зорових, нервово-емоційних переживань, серцево-судинних захворювань. Виходячи з цього, роботодавець повинен забезпечити гігієнічні й ергономічні вимоги щодо організації робочих приміщень для експлуатації електронно-обчислювальних машин (ЕОМ) з ВДТ, робочого середовища, робочих місць з ЕОМ, режиму праці і відпочинку при роботі з ЕОМ тощо, які викладені у нормах НПАОП 0.00-7.15-18. Відповідно до встановлених гігієнічно-санітарних вимог роботодавець зобов'язаний забезпечити в приміщеннях з ЕОМ оптимальні параметри виробничого середовища [27].

Для захисту від прямих сонячних променів, які створюють прямі та відбиті відблиски з поверхні екранів персонального комп’ютера і клавіатури повинні бути передбачені сонцезахисні пристрої, вікна повинні мати жалюзі або штори.

Основні задачі охорони праці при використанні комп'ютерної техніки:

- аналіз впливу факторів виробничого середовища на здоров'я і працездатність користувачів персональних комп’ютерів;

- вдосконалювання методів оцінки працездатності і стану здоров'я користувачів ПК;

- розробка і впровадження організаційно-технічних, гігієнічних і соціально-економічних заходів щодо раціоналізації виробничого середовища;

- розробка і впровадження профілактичних і оздоровчих заходів, що дозволяють зберігати здоров'я людини і підвищувати її працездатність;

- вдосконалення методик навчання користувачів ПК питанням охорони праці.

Вимогам нормативних актів з охорони праці мають відповідати:

- умови праці на кожному робочому місці;

- безпека технологічних процесів, машин, механізмів, обладнання й інших засобів виробництва;

- стан засобів колективного та індивідуального захисту;

- санітарно-побутові умови.

При дослідженні методів і засобів створення програмних систем та проектуванні системи захисту від атак на сервери важливо було проаналізовати та враховати необхідні вимоги щодо охорони праці при використанні електронно-обчислювальної техніки і забезпечити умови для зручної та ефективної роботи працівників.

4.2 Шум, вібрація, ультразвук, електромагнітні випромінювання у виробничих приміщеннях для роботи з ВДТ та захист від них

Під шумом розуміють набір багаточисельних звуків, які швидко змінюються за частотою, силою і складаються з ряду гармонік [28]. З фізичної

точки зору звуки є механічними коливальними рухами частинок пружного середовища в діапазоні частот, що чує людина. Звукові гармоніки розповсюджуються у вигляді хвиль.

Шум є загально-біологічним подразником, діє не тільки на органи слуху, але може викликати порушення роботи серцево-судинної і нервової систем, зумовлювати професійні захворювання [28]. Основними характеристиками звукових коливань є інтенсивність (сила), частота і форма звукової хвилі. Інтенсивність визначається енергією, що переноситься за 1 с звуковою хвилею через поверхню площею 1 м2, яка перпендикулярна напрямку розповсюдження звукової хвилі. Діапазон тисків, що сприймає вухо людини, дуже широкий (10-12Вт/м2 – поріг больового відчуття, верхня межа) [28].

З розвитком промисловості все більший контингент людей підпадає під вплив вібрацій, які являють собою механічні коливання, що передаються тілу людини. Основні параметри вібрацій – частота та амплітуда коливань, але на відміну від шуму, при якому енергія механічних коливань передається через повітряне середовище, при дії вібрацій вона розповсюджується по тканинах і викликає їх коливання або тіла людини в цілому [28]. Найбільш небезпечна вібрація частотою 16-250 Гц, дія якої призводить до вібраційної хвороби.

Нормування шуму здійснюється згідно з “Санітарними нормами допустимих рівнів шуму на робочих місцях”. В Україні застосовується принцип нормування шуму на основі граничних спектрів (гранично допустимих рівнів звукового тиску) в октавних смугах частот та еквівалентних рівнів звуку. Гранично-допустимі рівні шумів санітарними нормами встановлені для кожного класу [28]:

– для високочастотних шумів (вище 800 Гц) – 75-85 дБ;

– для середньо частотних шумів (300-800 Гц) – 85-90 дБ;

– для низькочастотних шумів (до 300 Гц) – 90-100 дБ.

Шумові явища мають якість кумуляції, накопичуючись в організмі, вони все більше і більше пригнічують нервову систему. Відомо, що після шумової дії інтенсивністю 120 дБ протягом однієї години потрібно 5 годин, щоб гострота слуху повернулась до норми. Стабільні широкосмугові шуми, які перевищують

граничний рівень, викликають зниження темпу, ефективності й якості роботи операторів [28].

Ультразвук широко застосовують у технологічних процесах виготовлення радіоелектронної апаратури (промивка деталей, зварювання мініатюрних вузлів тощо) з частотою вище 2220 кГц. При цьому густина енергії ультразвукових коливань у мільйони разів більша густини енергії звуків, які ми чуємо. Тому під його дією відбувається нагрівання тіла, а при дії коливань через рідкі і тверді середовища відбувається розривання і руйнування тканин [28]. Захист від ультразвуку, який діє через повітряне середовище, досягається шляхом звукоізоляції установок (листова сталь, дюралюміній, що обклеєні гумою або руберойдом, гетинакс) або розміщення їх в окремій звукоізолюючій кабіні. Ультразвукові установки повинні мати блокування, яке відключає генератор ультразвукових коливань в момент відкривання кришок або кожухів [28]. Для запобігання шкідливої дії шуму і вібрації на організм працюючих проводяться технічні, організаційні і медико-профілактичні заходи. Одним з основних технічних заходів є зменшення при експлуатації та на стадії проектування, конструювання обладнання причин шуму і вібрації в самому джерелі утворення. Досягають цього завдяки використанню раціональної конструкції обладнання, заміни ударної дії деталей і машин коливальною, з'єднання елементів гнучкими зв'язками, врівноважування обертових частин механізмів, заміни металевих деталей пластмасовими, забезпечення різних власних частот коливань механізму з частотою збуджуючої сили [28]. Якщо неможливо ізолювати чи знизити шум і вібрацію самого джерела, потрібно:

– ізолювати джерело шуму або вібрації від навколишнього середовища засобами вібро- та звукоізоляції;

– раціонально планувати виробничі приміщення, що мають інтенсивні джерела шуму;

– збільшувати звукопоглинання внутрішніх поверхонь приміщення шляхом звукопоглинальних покриттів.

Якщо не вдається зменшити рівень шуму і вібрації на робочому місці до нормативних значень та необхідно використовувати засоби індивідуального

захисту: рукавиці, взуття, навушники, м’які шоломи, які зменшують рівень звукового тиску на 40-50 дБ.

У процесі виробництв, експлуатації і зберігання комп’ютерної і радіоелектронної апаратури можуть виникати механічні і динамічні дії, що характеризуються широким діапазоном частот коливань, а також амплітудою, прискоренням і часом дії [28].

При експлуатації високочастотного обладнання всередині виробничих приміщень зниження напруженості електромагнітного випромінювання досягається такими методами:

– захист часом – обмеження часу перебування людини в електромагнітному полі, що залежить від інтенсивності опромінення або напруженості ЕМП.

– захист відстанню застосовується при неможливості послабити інтенсивність опромінення в заданій зоні іншими методами: збільшують відстань між джерелом випромінювання і обслуговуючим персоналом;

– добре виконане екранування джерела і усунення нещільності у фланцевих з’єднаннях, фідерів, зазорів у обшивці корпусів, нещільних електричних контактів;

– проведення дистанційного контролю й управління роботою передавачів з екранованого приміщення;

– засобами індивідуального захисту.

В залежності від типу джерела випромінювання, його потужності, характеру технологічного процесу може застосовуватись один з вказаних методів або будь-яка їх комбінація.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи магістра було проведено повний цикл дослідження, спрямований на підвищення рівня кібербезпеки в програмно-визначених мережах (SDN) шляхом розроблення моделі централізованого керування політиками доступу та реалізації SDN-фаєрвола із перевіркою стану з'єднання. У роботі виконано аналітичне обґрунтування актуальності проблеми, побудовано архітектурні та алгоритмічні рішення, а також здійснено їхню експериментальну перевірку у тестовому середовищі.

У першому розділі було проведено аналіз сучасних підходів до забезпечення мережевої безпеки, визначено обмеження класичних фаєрволів та переваги програмно-визначених мереж. Розроблено модель активів SDN-інфраструктури та їхніх цілей безпеки, виконано формалізацію вразливостей через реверсію цілей конфіденційності, цілісності та доступності. Запропоновано модифіковану методику CVSS для SDN-середовища, яка враховує специфіку площин керування та пересилання, і розроблено підхід до визначення вагових коефіцієнтів за допомогою методу AHP для оцінювання критичності активів.

У другому розділі було розроблено архітектуру SDN-фаєрвола із перевіркою стану з'єднання, який функціонує поверх OpenFlow-інфраструктури під керуванням контролера RYU. Запропоновано формальну модель розширеного скінченного автомата (SDNEFSM) для відстеження життєвого циклу сесій та реалізовано механізм контекстно-залежної фільтрації трафіку. Описано алгоритм оркестрації політик безпеки, що забезпечує централізоване оновлення правил доступу в таблицях потоків і підтримує синхронізацію між площинами керування та пересилання. Розроблено програмні модулі для обробки подій Packet-In, ведення таблиці станів з'єднань і логування аномалій.

У третьому розділі було реалізовано експериментальний стенд для перевірки ефективності SDN-фаєрвола та оркестрації політик. Проведено серію експериментів у віртуалізованому середовищі Mininet з використанням контролера RYU і комутатора Open vSwitch. Виміряно основні показники ефективності - час реагування контролера, затримку обробки пакетів, пропускну

здатність, коефіцієнт блокування атак та навантаження на ресурси. Отримані результати підтвердили, що впровадження перевірки стану з’єднання дозволяє підвищити точність виявлення атак і зменшити затримки обробки трафіку порівняно зі статичними ACL-механізмами.

Отримані результати мають практичне значення для побудови адаптивних систем мережевої безпеки у хмарних і віртуалізованих середовищах. Запропоновані моделі й методи можуть бути використані при розробленні корпоративних SDN-сегментів, у навчальних лабораторіях з кібербезпеки, а також у дослідженнях з оркестрації політик безпеки в інтелектуальних мережах наступного покоління.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке програмно-визначені мережі (SDN) | Netwave. (n.d.). Netwave. https://netwave.ua/blog/software-defined-networking-sdn-viznachennya-j-osoblivosti-programno-viznachenih-merezh/
2. Awati, R., & Wright, G. (2025, April 2). What is a northbound interface/southbound interface? | Definition from TechTarget. WhatIs. https://www.techtarget.com/whatis/definition/northbound-interface-southbound-interface
3. 6 Basic SDN Architecture Components - Software-Defined Networking. (n.d.). IPCisco. https://ipcisco.com/lesson/sdn-architecture-components/
4. What is Open Flow Protocol Networking and How it Works? - Ruijie Networks. (n.d.). Ruijie Networks | Network Devices and Solutions Provider. https://www.ruijie.com/en-global/support/faq/open-flow-protocol
5. NetIDE: Removing vendor lock-in in SDN. (n.d.). IEEE Xplore. https://ieeexplore.ieee.org/document/7116170
6. Understanding North-South and East-West traffic and the added value of NDR in network analysis - Gatewatcher. (n.d.). Gatewatcher. https://www.gatewatcher.com/en/resource/understanding-north-south-and-east-west-traffic-and-the-added-value-of-ndr-in-network-analysis/
7. Lindemulder, G., & Kosinski, M. (n.d.). What Is a Man-in-the-Middle (MITM) Attack? | IBM. IBM. https://www.ibm.com/think/topics/man-in-the-middle
8. What is Packet Flooding? Understanding Network Traffic Overload. (n.d.). ReasonLabs Cyberpedia. https://cyberpedia.reasonlabs.com/EN/packet%20flooding.html
9. ТИМОЩУК, Д., & ЯЦКІВ, В. (2024). USING HYPERVISORS TO CREATE A CYBER POLYGON. MEASURING AND COMPUTING DEVICES IN TECHNOLOGICAL PROCESSES, (3), 52-56. https://doi.org/10.31891/2219-9365-2024-79-7
10. ТИМОЩУК, Д., ЯЦКІВ, В., ТИМОЩУК, В., & ЯЦКІВ, Н. (2024). INTERACTIVE CYBERSECURITY TRAINING SYSTEM BASED ON

SIMULATION ENVIRONMENTS. MEASURING AND COMPUTING DEVICES IN TECHNOLOGICAL PROCESSES, (4), 215-220. https://doi.org/10.31891/2219-9365-2024-80-26

11. Тимощук, В., Долінський, А., & Тимощук, Д. (2024). ЗАСТОСУВАННЯ ГІПЕРВІЗОРІВ ПЕРШОГО ТИПУ ДЛЯ СТВОРЕННЯ ЗАХИЩЕНОЇ ІТ-ІНФРАСТРУКТУРИ. Матеріали конференцій МЦНД, (24.05. 2024; Запоріжжя, Україна), 145-146. https://doi.org/10.62731/mcnd-24.05.2024.001
12. Tymoshchuk, D., Yasniy, O., Mytnyk, M., Zagorodna, N., Tymoshchuk, V., (2024). Detection and classification of DDoS flooding attacks by machine learning methods. CEUR Workshop Proceedings, 3842, pp. 184 - 195.
13. Klots, Y., Titova, V., Petliak, N., Tymoshchuk, D., Zagorodna, N. Intelligent data monitoring anomaly detection system based on statistical and machine learning approaches. CEUR Workshop Proceedings, (2025), 4042, pp. 80 – 89
14. Common Vulnerability Scoring System SIG. (n.d.). FIRST — Forum of Incident Response and Security Teams. https://www.first.org/cvss/
15. What is the Analytic Hierarchy Process (AHP)? | 1000minds. (n.d.). 1000minds. https://www.1000minds.com/decision-making/analytic-hierarchy-process-ahp
16. Types of Firewalls Defined and Explained. (n.d.). Palo Alto Networks. https://www.paloaltonetworks.com/cyberpedia/types-of-firewalls
17. What Is Security as Code (SaC)? | CrowdStrike. (n.d.). CrowdStrike: We Stop Breaches with AI-native Cybersecurity. https://www.crowdstrike.com/en-us/cybersecurity-101/cloud-security/security-as-code/
18. Stateful distributed firewall as a service in SDN -. (n.d.). AUB ScholarWorks - Home. https://scholarworks.aub.edu.lb/items/bf30bd1a-d48b-4715-baa5-187527d6d46b
19. Kanade, V. (2023, September 12). Finite State Machine Meaning, Working, and Examples | Spiceworks - Spiceworks. Spiceworks Inc. https://www.spiceworks.com/tech/tech-general/articles/what-is-fsm/

20. Демчук, В., Тимощук, В., & Тимощук, Д. (2023). ЗАСОБИ МІНІМІЗАЦІЇ ВПЛИВУ SYN FLOOD АТАК. Collection of scientific papers «SCIENTIA», (November 24, 2023; Kraków, Poland), 130-130.
21. Petliak, N., Klots, Y., Karpinski, M., Titova, V., Tymoshchuk, D. Hybrid system for detecting abnormal traffic in IoT. CEUR Workshop Proceedings, (2025), 4057, pp. 21 – 36
22. B. Lypa, I. Horyn, N. Zagorodna, D. Tymoshchuk, T. Lechachenko, Comparison of feature extraction tools for network traffic data, CEUR Workshop Proceedings, 3896, 2024, pp. 1-11.
23. Ryu SDN Controller Review: Python-Based Flexible Framework - Aptira. (n.d.). Aptira. https://aptira.com/ryu-sdn-controller-review/
24. GitHub - mininet/mininet: Emulator for rapid prototyping of Software Defined Networks. (n.d.). GitHub. https://github.com/mininet/mininet
25. hping3 | Kali Linux Tools. (n.d.). Kali Linux. https://www.kali.org/tools/hping3/
26. The netfilter.org project. (n.d.). netfilter/iptables project homepage. https://www.netfilter.org/
27. ZAGORODNA, N., STADNYK, M., LYPA, B., GAVRYLOV, M., & KOZAK, R. (2022). Network Attack Detection Using Machine Learning Methods. Challenges to national defence in contemporary geopolitical situation, 2022(1), 55-61.
28. Mishko, O., Matiuk, D., & Derkach, M. (2024). Security of remote iot system management by integrating firewall configuration into tunneled traffic. Вісник Тернопільського національного технічного університету, 115(3), 122-129.
29. Жидецький В.Ц. Охорона праці користувачів комп´ютерів. Львів: Афіша, 2011. 176 с.
30. Желібо Е.Н. Безпека життєдіяльності: Навчальний посібник. Київ: Каравела, Львів: Новий світ - 2000. 2001. 320с.

Додаток А Публікація

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

17-18 грудня 2025 року

ТЕРНОПІЛЬ
2025

УДК 004.75:004.056.5
В. Стрихарський; О. Оробчук, доктор філософії
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ЦЕНТРАЛІЗОВАНЕ УПРАВЛІННЯ ПОЛІТИКАМИ БЕЗПЕКИ У SDN ІЗ ВИКОРИСТАННЯМ МЕХАНІЗМІВ ФІЛЬТРАЦІЇ ДОСТУПУ

UDC 004.75:004.056.5
V. Strykharskyi; O. Orobchuk, Ph.D.

CENTRALIZED SECURITY POLICY MANAGEMENT IN SDN USING ACCESS FILTERING MECHANISMS

Програмно-визначені мережі (SDN) [1] є архітектурною основою сучасних хмарних платформ, віртуалізованих дата-центрів та 5G-інфраструктур, що робить їх привабливою ціллю для зловмисників. Централізоване керування трафіком, поділ на площини даних, керування та застосунків, а також активне використання API створюють розширену площину атак на контролер, канали взаємодії та мережеві функції. У таких умовах класичні фаєрволи зі статичними ACL та сигнатурні системи виявлення вторгнень виявляються недостатньо ефективними, погано масштабуються та не забезпечують потрібного рівня узгодженості політик безпеки.

У роботі запропоновано узагальнену модель активів SDN-інфраструктури, що охоплює площину даних, площину керування, площину застосунків, канали Southbound, Northbound і East–West, а також допоміжні сервіси. Для кожного класу активів систематизовано типові загрози. Запропоновано підхід до кількісної оцінки ризиків, що поєднує модифіковану метрику CVSS із методом аналізу ієрархій AHP. Такий підхід дозволив ранжувати сценарії атак і обґрунтовано визначати, які типи з'єднань мають контролюватися на рівні стану, а де достатньо статичних правил фільтрації. На основі отриманих результатів розроблено архітектуру SDN-фаєрвола із перевіркою стану з'єднання, орієнтованого на централізоване керування політиками. Фаєрвол реалізовано як набір додатків до контролера RYU [2] із зовнішнім оркестратором політик, що динамічно генерує правила OpenFlow для комутаторів Open vSwitch [3] з урахуванням поточного стану окремих сеансів. Логіку роботи фаєрвола формалізовано як скінченний автомат, адаптований до SDN-середовища. Кожен етап життєвого циклу з'єднання, включно зі встановленням, передаванням даних і завершенням, однозначно відповідає діям над таблицями потоків Open vSwitch. Ці дії охоплюють створення нового правила, його оновлення, вилучення запису та налаштування відповідних тайм-аутів. Прототип досліджено у віртуалізованому середовищі KVM [4] з використанням сценаріїв нормального трафіку та трафіку атаки. Експериментальні результати продемонстрували підвищення точності блокування небажаних з'єднань порівняно зі статичними ACL, скорочення часу реакції на інциденти та зменшення ризику для площини керування й каналів взаємодії за прийнятних накладних витрат на обробку пакетів.

Література

1. IBM. What Is Software-Defined Networking (SDN)? | IBM. URL: https://www.ibm.com/think/topics/sdn (date of access: 03.12.2025).
2. Ryu SDN Framework. URL: https://ryu-sdn.org/ (date of access: 03.12.2025).
3. Open vSwitch. URL: https://www.openvswitch.org/ (date of access: 03.12.2025).
4. Interactive cybersecurity training system based on simulation environments / D. Tymoshchuk et al. Measuring and computing devices in technological processes. 2024. No. 4. P. 215–220. URL: https://doi.org/10.31891/2219-9365-2024-80-26 (date of access: 30.11.2025).

Додаток Б Файл sdn_firewall.py

```
# sdn_firewall.py
# ----------------
# SDN-фаєрвол на Ryu (OpenFlow 1.3) з двома режимами:
#   - PROACTIVE: заздалегідь інсталює allow-правила для дозволених сервісів;
#    - REACTIVE: встановлює правила на льоту для кожного нового з'єднання (5-tuple).

import os
import time
from collections import defaultdict, deque

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER, set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet, ipv4, tcp
from ryu.lib import addrconv
from ryu.lib.packet import ether_types


def env(name, default):
    v = os.environ.get(name)
    if v is None or v == "":
        return default
    return v


class StatefulFirewall(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(StatefulFirewall, self).__init__(*args, **kwargs)
```

```
        self.mode = env("FW_MODE", "PROACTIVE").upper()  # PROACTIVE
| REACTIVE


        self.idle_timeout = int(env("IDLE_TIMEOUT_SEC", "120"))
        self.hard_timeout = int(env("HARD_TIMEOUT_SEC", "0"))


        self.syn_threshold = int(env("SYN_THRESHOLD", "100"))
        self.syn_window_sec = float(env("SYN_WINDOW_SEC", "1"))


        self.allow_policies = [
            {
                "src_ip": "10.0.0.1",      # Client
                "dst_ip": "10.0.0.2",      # Server
                "tcp_port": 80
            }
        ]


        self.syn_windows = defaultdict(lambda: deque())


        self.datapaths = {}

        self.logger.info(
            "[FW] Mode=%s, SYN_THRESHOLD=%d/%ss, IDLE_TIMEOUT=%ds",
            self.mode,   self.syn_threshold,   self.syn_window_sec,
self.idle_timeout
        )
```

```
    def _add_flow(self, datapath, priority, match, actions,
idle_timeout=0, hard_timeout=0):
        ofp = datapath.ofproto
        parser = datapath.ofproto_parser

        inst                                                    =
[parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS, actions)]
        mod = parser.OFPFlowMod(datapath=datapath,
                                priority=priority,
                                idle_timeout=idle_timeout,
                                hard_timeout=hard_timeout,
                                match=match,
                                instructions=inst)
        datapath.send_msg(mod)

    def _add_drop_flow(self, datapath, priority, match,
idle_timeout=0, hard_timeout=0):
        ofp = datapath.ofproto
        parser = datapath.ofproto_parser
        inst = []  # без дій = drop
        mod = parser.OFPFlowMod(datapath=datapath,
                                priority=priority,
                                idle_timeout=idle_timeout,
                                hard_timeout=hard_timeout,
                                match=match,
                                instructions=inst)
        datapath.send_msg(mod)

    def _send_packet_out(self, msg, actions):
        datapath = msg.datapath
        ofp = datapath.ofproto
        parser = datapath.ofproto_parser
        out = parser.OFPPacketOut(
            datapath=datapath,
            buffer_id=msg.buffer_id,
            in_port=msg.match['in_port'],
            actions=actions,
```

```
            data=None if msg.buffer_id != ofp.OFP_NO_BUFFER else
msg.data
        )
        datapath.send_msg(out)



    @set_ev_cls(ofp_event.EventOFPSwitchFeatures,
CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):

        datapath = ev.msg.datapath
        ofp = datapath.ofproto
        parser = datapath.ofproto_parser

        self.logger.info("[FW]    Switch    connected:    dpid=%s",
datapath.id)



        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofp.OFPP_CONTROLLER,
                                          ofp.OFPCML_NO_BUFFER)]
        self._add_flow(datapath,      priority=0,      match=match,
actions=actions)



        match_tcp                                                  =
parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, ip_proto=6)
        self._add_drop_flow(datapath, priority=5, match=match_tcp)

        if self.mode == "PROACTIVE":

            for pol in self.allow_policies:

                match_allow_fwd = parser.OFPMatch(
                    eth_type=ether_types.ETH_TYPE_IP,
```

```
                    ip_proto=6,                  # TCP
                    ipv4_src=pol["src_ip"],
                    ipv4_dst=pol["dst_ip"],
                    tcp_dst=pol["tcp_port"]
                )
                actions_fwd                                          =
[parser.OFPActionOutput(ofp.OFPP_NORMAL)]
                self._add_flow(datapath, priority=100,
                               match=match_allow_fwd,
actions=actions_fwd)


                match_allow_rev = parser.OFPMatch(
                    eth_type=ether_types.ETH_TYPE_IP,
                    ip_proto=6,
                    ipv4_src=pol["dst_ip"],
                    ipv4_dst=pol["src_ip"],
                    tcp_src=pol["tcp_port"]
                )
                actions_rev                                          =
[parser.OFPActionOutput(ofp.OFPP_NORMAL)]
                self._add_flow(datapath, priority=100,
                               match=match_allow_rev,
actions=actions_rev)

            self.logger.info("[FW] PROACTIVE allow-rules installed
for %d policy(ies)", len(self.allow_policies))

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
                msg = ev.msg
        datapath = msg.datapath
        ofp = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
```

```
        eth = pkt.get_protocol(ethernet.ethernet)
        if eth.ethertype != ether_types.ETH_TYPE_IP:

            actions = [parser.OFPActionOutput(ofp.OFPP_NORMAL)]
            self._send_packet_out(msg, actions)
            return

        ip4 = pkt.get_protocol(ipv4.ipv4)
        tcp_seg = pkt.get_protocol(tcp.tcp)
        if ip4 is None or tcp_seg is None:

            actions = [parser.OFPActionOutput(ofp.OFPP_NORMAL)]
            self._send_packet_out(msg, actions)
            return

        src_ip = ip4.src
        dst_ip = ip4.dst
        src_port = tcp_seg.src_port
        dst_port = tcp_seg.dst_port
        flags = tcp_seg.bits  # TCP flags


        allowed = False
        for pol in self.allow_policies:
            if src_ip == pol["src_ip"] and dst_ip == pol["dst_ip"]
and dst_port == pol["tcp_port"]:
                allowed = True
                break


        is_syn = (flags & tcp.TCP_SYN) and not (flags & tcp.TCP_ACK)

        if is_syn and allowed:
            if self._syn_flood_exceeded(src_ip, dst_ip, dst_port):

                match_drop = parser.OFPMatch(
                    eth_type=ether_types.ETH_TYPE_IP,
```

```
                    ip_proto=6,
                    ipv4_src=src_ip,
                    ipv4_dst=dst_ip,
                    tcp_dst=dst_port
                )
                self._add_drop_flow(datapath,         priority=200,
match=match_drop,

idle_timeout=int(self.syn_window_sec * 4), hard_timeout=0)
                self.logger.warning("[FW] SYN flood detected: drop
%s -> %s:%d",
                                    src_ip, dst_ip, dst_port)
                return  # не форвардимо пакет


        if self.mode == "PROACTIVE":



            actions = [parser.OFPActionOutput(ofp.OFPP_NORMAL)]
            self._send_packet_out(msg, actions)
            return


        if self.mode == "REACTIVE" and allowed and is_syn:

            match_fwd = parser.OFPMatch(
                eth_type=ether_types.ETH_TYPE_IP,
                ip_proto=6,
                ipv4_src=src_ip,
                ipv4_dst=dst_ip,
                tcp_dst=dst_port
            )
            actions_fwd = [parser.OFPActionOutput(ofp.OFPP_NORMAL)]
            self._add_flow(datapath, priority=150, match=match_fwd,
actions=actions_fwd,
```

```
                            idle_timeout=self.idle_timeout,
hard_timeout=self.hard_timeout)


            match_rev = parser.OFPMatch(
                eth_type=ether_types.ETH_TYPE_IP,
                ip_proto=6,
                ipv4_src=dst_ip,
                ipv4_dst=src_ip,
                tcp_src=dst_port
            )
            actions_rev = [parser.OFPActionOutput(ofp.OFPP_NORMAL)]
            self._add_flow(datapath, priority=150, match=match_rev,
actions=actions_rev,
                            idle_timeout=self.idle_timeout,
hard_timeout=self.hard_timeout)

            self.logger.info("[FW] REACTIVE install bi-dir flow for
%s:%d -> %s:%d (idle=%ds)",
                               src_ip,  src_port,  dst_ip,  dst_port,
self.idle_timeout)


            actions = [parser.OFPActionOutput(ofp.OFPP_NORMAL)]
            self._send_packet_out(msg, actions)
            return



        self.logger.debug("[FW]  Drop  packet:  %s:%d  ->  %s:%d  (not
allowed or not SYN/new)",
                           src_ip, src_port, dst_ip, dst_port)
        return



    def _syn_flood_exceeded(self, src_ip, dst_ip, dst_port):
```

```
now = time.time()
key = (src_ip, dst_ip, dst_port)
dq = self.syn_windows[key]


while dq and now - dq[0] > self.syn_window_sec:
    dq.popleft()

dq.append(now)
count = len(dq)

if count > self.syn_threshold:
    return True
return False
```

Додаток В Файл mininet_testbed.py

```
# mininet_testbed.py
# -------------------
# Тестовий стенд Mininet: Client(h1) – Server(h2) – Attacker(h3) через один OVS (s1)
# і віддалений Ryu-контролер (порт 6653).

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

def build_net():
    net = Mininet(controller=None, switch=OVSSwitch, link=TCLink, autoSetMacs=True, autoStaticArp=True)

    info("*** Adding controller (remote Ryu @ 127.0.0.1:6653)\n")
    c0 = net.addController('c0', controller=RemoteController, ip='127.0.0.1', port=6653)

    info("*** Adding switch\n")
    s1 = net.addSwitch('s1', protocols='OpenFlow13')

    info("*** Adding hosts\n")
    h1 = net.addHost('h1', ip='10.0.0.1/24')  # Client
    h2 = net.addHost('h2', ip='10.0.0.2/24')  # Server
    h3 = net.addHost('h3', ip='10.0.0.3/24')  # Attacker

    info("*** Creating links\n")
    net.addLink(h1, s1)
    net.addLink(h2, s1)
    net.addLink(h3, s1)

    info("*** Starting network\n")
    net.build()
```

```
    c0.start()
    s1.start([c0])


    for h in (h1, h2, h3):
        h.cmd('ip route add default dev %s-eth0' % h.name)


    inject_helpers(net, h1, h2, h3)
    return net

def inject_helpers(net, h1, h2, h3):


    def start_http():
        info("*** Starting HTTP server on h2 (port 80)\n")
        h2.cmd('pkill -f "http.server"')
        h2.cmd('python3 -m http.server 80 >/tmp/h2_http.log 2>&1 &')
        info("*** h2 HTTP started. Try from h1: curl -s
http://10.0.0.2:80/\n")

    def start_client_load():
        info("*** Generating client load from h1 -> h2:80
(background curls)\n")
        h1.cmd('pkill -f "curl"')

        h1.cmd('bash -c \'for i in $(seq 1 100); do '
               '  (while true; do curl -s http://10.0.0.2:80/
>/dev/null; sleep 0.05; done) & '
               'done\'')
        info("*** Client load started.\n")

    def start_syn_flood(rate=1000):
        info("*** Starting SYN flood from h3 -> h2:80 at ~%s SYN/s\n"
% rate)
```

```
        h3.cmd('pkill -f hping3; pkill -f nping; pkill -f
"send_syn.py"')


        cmd_hping = f'hping3 -S -p 80 --flood -i
u{int(1e6/max(rate,1))} 10.0.0.2 >/dev/null 2>&1 &'

        cmd_nping = 'nping --tcp -S -p 80 --flags syn --rate {r}
10.0.0.2 >/dev/null 2>&1 &'.format(r=rate)


        ret = h3.cmd('which hping3')
        if ret.strip():
            h3.cmd(cmd_hping)
            info("*** hping3 flood started.\n")
            return

                ret = h3.cmd('which nping')
        if ret.strip():
            h3.cmd(cmd_nping)
            info("*** nping flood started.\n")
            return


        scapy_script = r'''
from scapy.all import *
dst="10.0.0.2"
dport=80
while True:
    send(IP(dst=dst)/TCP(dport=dport, flags="S"), verbose=0)
'''
        h3.cmd('python3 - << "PY"\n' + scapy_script + '\nPY\n
>/tmp/h3_scapy.log 2>&1 &')
        info("*** scapy-based flood started (fallback).\n")

    def stop_all():
        info("*** Stopping load and attacks, HTTP server\n")
```

```
        for host in (h1, h2, h3):
            host.cmd('pkill -f hping3; pkill -f nping; pkill -f
send_syn.py; pkill -f http.server; pkill -f curl')
        info("*** Stopped.\n")


    CLI.locals.update({
        'start_http': start_http,
        'start_client_load': start_client_load,
        'start_syn_flood': start_syn_flood,
        'stop_all': stop_all
    })

def main():
    setLogLevel('info')
    net = build_net()
    info("\n=== Ready ===\n")
    info("У терміналі A має працювати Ryu фаєрволом:\n")
    info("          ryu-manager     --ofp-tcp-listen-port     6653
sdn_firewall.py\n")
    info("\nУ CLI Mininet виконайте:\n")
    info("  py start_http()\n")
    info("  py start_client_load()\n")
    info("  # py start_syn_flood(rate=16000)\n")
    info("  # зупинка всього: py stop_all()\n\n")
    CLI(net)
    net.stop()

if __name__ == '__main__':
    main()
```