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

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»
УДК _____

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____2023 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Комп'ютерні системи та мережі»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Спосіб захисту програмно-конфігурованої мережі від атак»

Виконав:
студент VI курсу, групи ІО-11мн
Воробйов Анатолій Олександрович _____

Керівник:
доц., к.т.н., доц.,
Волокита Артем Миколайович _____

Консультант з нормоконтролю:
проф., д.т.н.,
Кулаков Юрій Олексійович _____

Рецензент:
доц. каф. ІСТ, к.т.н., доц.,
Шимкович Володимир Миколайович _____

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

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – другий (магістерський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-наукова програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Воробйов Анатолій Олександрович

1. Тема дисертації «Спосіб захисту програмно-конфігурованої мережі від атак», науковий керівник дисертації Волокита Артем Миколайович, доц. каф. ОТ, д.т.н., затверджені наказом по університету від «__» _____ 2023 р. № _____
2. Термін подання студентом дисертації _____
3. Об’єкт дослідження: процес захисту програмно-конфігурованих мереж від атак.
4. Предмет дослідження: способи, технології та алгоритми захисту програмно-конфігурованих мереж від атак, включаючи забезпечення їхньої стійкості та надійності
5. Перелік завдань, які потрібно розробити: аналіз існуючих типів мереж, огляд структури досліджуваної мережі, визначити найбільш незахищений компонент, формування списку існуючих рішень і виокремлення їх переваг і

недоліків реалізації, формування власного підходу, проектування принципу, побудова тестового середовища, відображення результатів роботи підходу

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: загальнотематичні ілюстрації по темі дисертації.

7. Орієнтовний перелік публікацій _____

8. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормконтроль	Кулаков Ю.О. д.т.н., професор		

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	Затвердження теми роботи	31.01.2023	
2	Вивчення та аналіз завдання	01.02.2023-13.02.2023	
3	Постановка та формалізація задачі	14.03.2023-27.02.2023	
4	Розробка інформаційного та програмного забезпечення	28.02.2023-27.03.2023	
5	Проведення експериментальних досліджень	28.03.2023-24.04.2023	
6	Оформлення пояснювальної записки	25.04.2023-02.05.2023	
7	Передзахист		
8	Захист		

Студент Анатолій ВОРОБЙОВ

Науковий керівник Артем ВОЛОКИТА

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Спосіб захисту програмно-конфігурованої мережі від атак

студентом: Воробйовим Анатолій Олександровичем

Актуальність теми: підхід по вирішенню проблеми безпеки в мережах. Мережі стрімко розвиваються, кількість трафіку зростає кожен день, починають з'являтись нові підходи по вирішенню проблем традиційних мереж. З появою нових підходів з'являються проблеми безпеки для цих самих підходів. Важливо визначити критичні проблеми й порівняти з наявний рішеннями. В рамках дисертації виявлено недоліки наявних рішень і запропонований новий шлях по їх вирішенню.

Мета дослідження - розробка та впровадження ефективних способів захисту програмно-конфігурованих мереж від різних видів атак з метою забезпечення безпеки та стабільності роботи мережі.

Об'єкт дослідження – процес захисту програмно-конфігурованих мереж від атак.

Наукова новизна: запропоновано модифікований спосіб захисту програмно-конфігурованих мереж від атак, який за рахунок горизонтального масштабування шляхом аналізу потокових таблиць дозволяє підвищити швидкість виявлення атак та забезпечити аудиту мережевих показників.

Завдання дослідження:

1. Аналіз сучасних мереж і виокремлення типів атак які притаманні їм, визначення мало вивчених напрямів атак і дослідження наявних переваг і недоліків певних реалізацій.
2. Дослідження підходу який базується на існуючих алгоритмах, що виконує аналіз трафіку мережі і дозволить виявляти аномалії з мінімальними часовими і ресурсними затратами.

3. Захист мережевих компонентів від існуючого типу атак, за рахунок адаптації запропонованого підходу.

Предмет дослідження – способи, технології та алгоритми захисту програмно-конфігурованих мереж від атак, включаючи забезпечення їхньої стійкості та надійності.

Методи дослідження базуються на використанні середовищ емуляції мереж, які дозволяють створити аналог реальних мереж, та застосування нейроних мереж для аналізу трафіку цієї мережі, та застосування моделювання.

Практичне значення одержаних результатів роботи полягає в покращені безпеки мережі загалом і застосування певних заборон або обмежень до мережевих компонентів які створюють загрозу. Відповідно як результат підвищення ефективності мережі загалом, що дозволить оминути затримки та втрати пакетів в мережі. Це відповідно дозволить ефективно використовувати мережу та забезпечувати відповідну її якість.

Основні задачі дослідження можна визначити за рахунок мети і тому полягають у наступному:

1. Аналіз існуючих типів мереж, визначення їх актуальності застосування і тенденції до їх залучення у сучасних мережах.
2. Огляд структури досліджуваної мережі, з метою виявити рівень важливості компонентів та аналіз типів атак на ці рівні.
3. Визначити найбільш незахищений компонент мережі базуючись на наявних дослідженнях.
4. Формування списку існуючих рішень і виокремлення їх переваг і недоліків реалізації.
5. Формування власного підходу до вирішення встановленної проблематики, визначення переваг і недоліків рішення, обгрунтування використання саме цього принципу.

6. Проектування принципу, формування детальної архітектури з описом компонентів які будуть задіяні з поясненням їх вибору.
7. Побудова тестового середовища з описом принципу тестування і задіяних компонентів .
8. Відображення результатів роботи підходу, виявлення недоліків і переваг, опис можливих модифікацій розробленого програмного продукту і шляхи його розвитку.

Особистий внесок здобувача полягає в аналізі мереж, дослідженні наявних підходів з виокремленням їх переваг і недоліків, і як наслідок формування нового підходу, його експериментальній перевірці і визначення шляхів розвитку цього підходу у майбутньому.

СПОСІБ ЗАХИСТУ ПРОГРАМНО-КОНФІГУРОВАНОЇ МЕРЕЖІ ВІД АТАК

ABSTRACT

for a master's thesis

"An approach of protecting a software-defined network from attacks"

author: Vorobiov Anatolii Oleksandrovich

Relevance of the Topic: Approaching the issue of security in networks. Networks are rapidly evolving, with the amount of traffic increasing every day, and new approaches are emerging to address the problems of traditional networks. However, with the emergence of new approaches, security issues arise for these very approaches. It is important to identify critical problems and compare them with existing solutions. Within the scope of this dissertation, the shortcomings of existing solutions have been identified, and a new approach to addressing these shortcomings has been proposed.

Research Objective: The development and implementation of effective methods for protecting software-defined networks from various types of attacks, with the aim of ensuring network security and stability.

Research Object: The process of protecting software-defined networks from attacks.

Scientific Novelty: A modified method for protecting software-defined networks from attacks has been proposed, which, through horizontal scaling by analyzing flow tables, allows for faster attack detection and provides network performance auditing.

Research Objectives:

1. Analyzing modern networks and identifying the types of attacks inherent to them, identifying underexplored attack directions, and investigating the advantages and disadvantages of specific implementations.
2. Researching an approach based on existing algorithms that analyzes network traffic and allows for the detection of anomalies with minimal time and resource costs.
3. Protecting network components from existing types of attacks through the adaptation of the proposed approach.

Research Subject: Methods, technologies, and algorithms for protecting software-defined networks from attacks, including ensuring their stability and reliability.

Research Methods: The research is based on the use of network emulation environments, which allow for the creation of analogs of real networks, as well as the application of neural networks for traffic analysis within these networks and the use of modeling techniques.

The practical significance of the obtained results lies in enhancing overall network security and applying specific restrictions or limitations to network components that pose threats. Consequently, this will improve network efficiency, enabling the avoidance of delays and packet losses within the network. This, in turn, allows for effective network utilization and ensures the desired quality of service.

The main research tasks, based on the objective, are as follows**:**

1. Analyzing existing types of networks, determining their relevance for implementation, and identifying trends in their deployment in modern networks.
2. Examining the structure of the network under investigation to determine the importance level of its components and analyzing attack types targeting these levels.
3. Identifying the most vulnerable network component based on existing research.
4. Compiling a list of existing solutions and highlighting the advantages and disadvantages of their implementations.
5. Formulating a novel approach to addressing the identified issues, determining the advantages and disadvantages of the solution, and justifying the use of this specific principle.
6. Designing the approach, forming a detailed architecture with descriptions of the involved components and explanations for their selection.
7. Building a testing environment with a description of the testing principle and the involved components.

8. Presenting the results of the approach, identifying its strengths and weaknesses, describing possible modifications of the developed software, and outlining future directions for its development.

The researcher's personal contribution involves analyzing networks, investigating existing approaches, highlighting their advantages and disadvantages, and subsequently forming a new approach. This is followed by experimental verification and identifying paths for the approach's future development.

AN APPROACH OF PROTECTING A SOFTWARE-DEFINED NETWORK FROM ATTAC

ЗМІСТ

ВСТУП

Захист мереж є однією з найважливіших задач у сфері інформаційної безпеки. Мережі забезпечують передачу інформації між пристроями та користувачами, включаючи конфіденційну та особисту інформацію. Недостатній захист мереж може призвести до витоку цієї інформації, що може мати негативний вплив на людей та організації, які її стосуються. Крадіжка даних є найбільш очевидною причиною захисту мереж є захист конфіденційної та особистої інформації, яка може бути перехоплена з недостатньо захищених мереж. Це може включати конфіденційну інформацію про користувачів, фінансові дані та інші цінні дані. Втрата продуктивності: атаки на мережі можуть призвести до втрати продуктивності та доступності ресурсів мережі. Це може стати причиною затримок у роботі та зниження ефективності користувачів та пристроїв, які залежать від цих ресурсів. Втрата репутації: якщо мережа не є достатньо захищеною, це може вплинути на репутацію організації. Втрата конфіденційної інформації або втрата продуктивності можуть призвести до недовіри та втрати довіри клієнтів та інших зацікавлених сторін. Через стрімкий ріст кількості трафіку у мережах, масштаб мережі виріс на порядок за останні роки, отримали проблему з традиційними мережами які не спроможні ефективно масштабуватись, мають прив’язку до виробника обладнання, мають велику складність у налаштуванні, ускладнюють, часом унеможливлюють реалізацію нових ідей в мережі [1]. Для того, щоб уникнути цих проблем розроблені нові архітектури. Дані архітектура починає активно впроваджуватись великими IT гігантами такими як Cisco, Microsoft, IBM, VMware [2]. Як відомо, нові технології не позбавлені проблем, тому нові підходи мають власні недоліки безпеки, які включають вже широко розповсюджені проблеми аутентифікації, людини по середині, атаки на відмову та інші [3], [4]. Розробляється доволі велика кількість рішень щоб уникнути наявних вразливостей, але тим не менш проблеми залишаються оскільки ключовим моментом є швидкість виявлення вторгнень.

В даній дисертації зосереджено увагу на вирішенні проблеми безпеки мережі враховуючи обмеження майже реального часу, тобто вимагається визначити часові межі за які буде знайдена вразливість, враховуючи затримки на зчитування і пересилку даних. Важливо визначити критичні елементи і проаналізувати готові рішення і визначити їх недоліки або неосвітлені моменти.

Тіж SDN (Software-Defined Networking) і NFV (Network Functions Virtualization) - це нові технології, які з'явилися в останні роки. Вони були створені для того, щоб допомогти організаціям знизити витрати на управління мережею і забезпечити більшу гнучкість і швидкість в розгортанні нових мережевих послуг.

Актуальність теми пов’язана із актуальність впровадження нових архітектурних рішень в мережах. Більшість впроваджених рішень не є достатньою для покриття всіх можливих атак, та і ефективність існуючих потребує покращення. Реалізація повинна бути уніфікованою і не залежати від поставщика програмного забезпечення.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ВРАЗЛИВОСТЕЙ МЕРЕЖ

1.1 Аналіз мереж

Важливо визначити мережі для яких є актуально розробляти захист, оскільки за останні роки існує велика тенденція на хмарні обчислення, соціальні мережі, обробки великих даних [1]. Всі ці фактори вимагають великої пропускної спроможності, доступність, динамічності переналаштування. Традиційні мережі все ще становлять і будуть становити значний відсоток в мережі, але в той же час нові підходи такі як SDN та NFV починають займати роль в мережі за рахунок своїх переваг.

1.1.1 Традиційні мережі

Мережі в яких рівень контролю і рівень даних знаходяться в одному місці, на перемикачі. Такі мережі мають перемикачі від різних вендорів тому конфігурація обладнання відрізняється. Традиційні мережі зазвичай були побудовані на обладнанні, яке було спеціально розроблене для мережевих цілей, та мають обмежену гнучкість. У таких мережах важко забезпечити масштабованість і гнучкість, оскільки вони зазвичай потребують значного зусилля на розгалуження та підтримку. Традиційні мережі - це мережі, які використовують традиційну архітектуру мережі, таку як використання статичних мережевих пристроїв, таких як комутатори і маршрутизатори, для передачі даних. Ці мережі зазвичай мають централізоване управління і використовують протоколи, такі як OSPF або BGP, для керування маршрутизацією даних. Встановлення потокових правил в таких мережах є децентралізоване, тобто треба обмінюватись повідомленнями з сусідніми перемикачами щоб встановити відповідні правила на перемикачі. При додаванні нового обладнання в мережу системі потрібен доволі великий час щоб сформувати маршрути в яких цей перемикач був би задіяний [5]. Гострим питанням в таких мережах є відмовостійкість, оскільки в разі відмови одного з перемикачів відбувається голосування за новий маршрут що випливає в великі затримки при

передачі даних або навіть їх втрату. Розробка захисту для традиційних мереж є складним завданням оскільки для моніторингу тих же перемикачів потрібна інтеграція з програмним забезпеченням різних виробників, маємо проблему з опитуванням і застосуванням дій по протидії атакам. Як бачимо на рис.1 архітектура традиційних мереж повністю зосереджена на тому що рівень контролю і рівень даних знаходяться в одному місці, тобто в момент часу не має централізованого елемента який би знав про стан всіх перемикачів. Для отримання даних з перемикача потрібно звертатися напряму до нього. Підтвердити чи допустимий стан має перемикач ніяк не можливо, оскільки ті ж потокові правила могли бути вставлені зловмисником і перевірити це крім аналізу логів неможливо. При оптимальній конфігурації перемикачів з відключеним повним запису лог-файлів навіть інформацію про те яким чином були змінені правила в таблицях дізнатись буде неможливо.

Зараз традиційні мережі широко використовуються в більшості організацій, оскільки вони забезпечують достатню швидкість передачі даних, якість обслуговування і безпеку, а також не вимагають великих витрат на інвестиції. Однак, з поширенням хмарних технологій і IoT, з'являється потреба у мережах, які можуть бути більш гнучкими, масштабованими та ефективними.

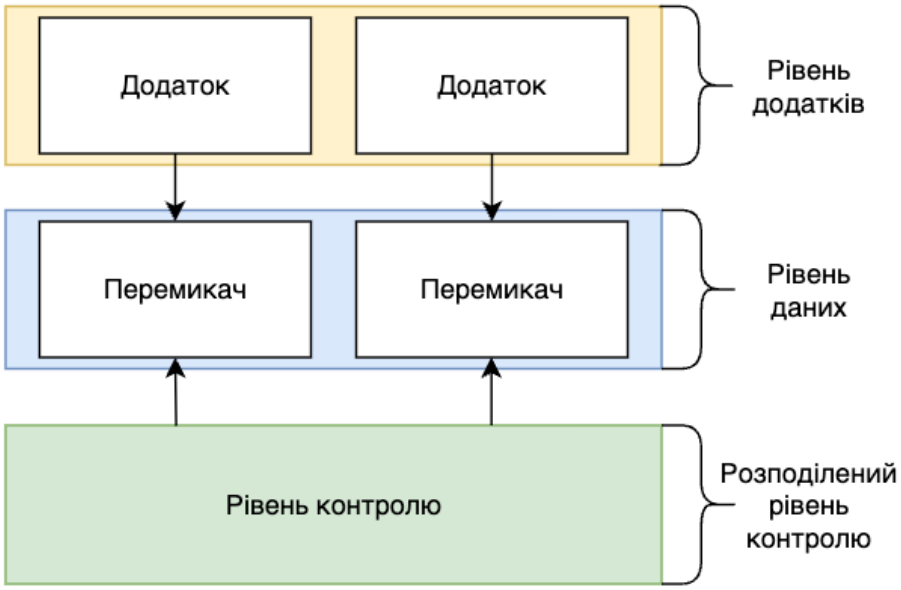

Рис. 1.1 - Архітектура традиційних мереж

Тож особливістю архітектури традиційних мереж є наявність розподіленого рівня контролю, тобто де налаштування обладнання відбувається за рахунок внутрішньої комунікації перемикачів між собою.

1.1.2 NFV підхід

Концепція віртуалізації мережевих функцій - дозволяє відмовитись від пропрієтарних перемикачів, тим самим зменшити складність конфігурації. Тобто функції елементів мережі можуть виконуватись програмним забезпеченням яке можливо запустити на звичайних серверах. Такі мережеві компоненти взаємодіють між собою з метою надати послугу зв'язку. За рахунок віртуалізації знижуються витрати на розвертання пропріетарного обладнання, кожен перемикач в мережі починає мати один і той самий інтерфейс до якого можна звернутись [6]. Такий підхід дає основу для розробки компонентів по аналізу мережі, щоб протидіяти вторгненням. Є можливість опитувати перемикачі мережі не зважаючи увагу на програмне забезпечення як було у пропріетарного обладнання. Недоліками такого підходу є те що підвищується ризик нових атак, в порівнянні з фізичним обладнанням яке розташоване за зачиненими дата центрами. Не дивлячись на можливі наслідки від недоліків, оператори починають поступово впроваджувати у свої системи віртуалізацію, оскільки цей підхід не потребує виведення обладнання яке вже працює з експлуатації. А за рахунок заміни частини мережі віртуальними функціями оператори зв'язку починають зменшувати витрати на обслуговування пропріетарного обладнання.

1.1.3 SDN підхід

Програмно-конфігуруванні мережі - архітектура яка має найбільші зміни в конструюванні мережі. Цей підхід базується на виділенні рівня контролю і рівня даних окремо. Це дозволяє за рахунок контролера який є серцем SDN мережі виконувати вставку потокових правил на основі раніше встановлених. Аналізуючи навантаження на окремих ділянках мережі, можна реконструювати маршрутизацію для зменшення навантаження на канал зв'язку. Головним протоколом взаємодії

перемикачів і контролера є OpenFlow протокол. Через те що пропріетарне обладнання повинно мати підтримку цього протоколу щоб конфігурація перемикачів відбувається за рахунок контролера - це обмежує сучасні мережі, оскільки досить великий відсоток обладнання не має підтримки OpenFlow протоколу. З появою стандарту виробники обладнання почали інтегрувати підтримку нового протоколу, тому через декілька років буде можливість переводити більшість мереж на SDN архітектуру. Головною перевагою SDN є те що контролер знає всю інформацію про стан мережі, він відповідає за встановлення потокових правил на перемикачах. Враховуючи навантаження ділянок мережі, визначає оптимальні маршрути для пакетів встановлює їх на перемикачах. Таким чином отримуємо велику продуктивність, оскільки в тих же традиційних мережах для встановлення маршрутів потрібно було обмінюватись інформацією з сусіднім обладнанням що могло займати доволі великі проміжки часу. За рахунок можливості виділити контролер окремо, це дозволяє реалізовувати різні алгоритми по маршрутизації, варіанти налаштування перемикачів. Додавання нового обладнання не потребує складних операцій, за рахунок протоколу OpenFlow Discovery, контролер знаходить учасників мережі [7].

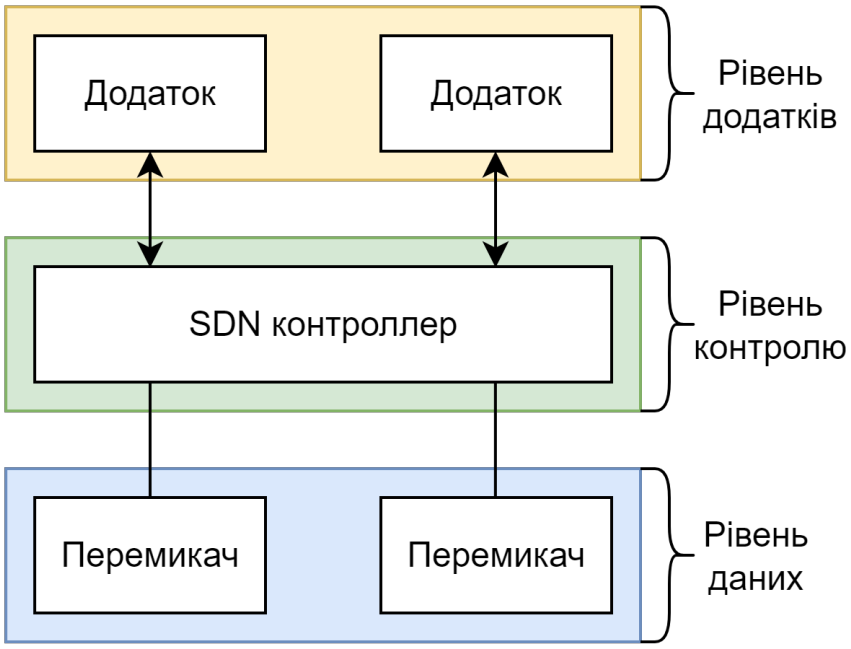

Рис. 1.2 - Архітектура SDN мереж

OpenFlow є головним при взаємодії перемикачів і контролера. Він забезпечує взаємодію рівню даних з рівнем контролю. Перемикач збирає дані статистики о стані, дані передачі даних, створює мітки в таблицях переадресації і надає за рахунок протоколу цю інформацію до контролера. Як тільки пакет з'являються на перемикачі, вибирається його заголовок з полями порівняння і при знаходженні співпадаючого запису до пакету застосовуються зазначені дії. Якщо ж відповідності не знайдено то заголовок відправляється на контролер який і визначає дії необхідні до застосування.

Для того щоб передача команд між контролером і перемикачами по OpenFlow протоколу відбувалося безпечно, використовується SSL протокол який інкапсулює пакети команд криптографічно [9].

1.1.4 Висновок по аналізу мереж

Відповідно до аналізу можна підсумувати що традиційні мережі поступово почнуть зникати за рахунок віртуалізації та більшої ефективності використання перемикачів. Потреби в імплементації способів безпеки для традиційних мереж є, але складність за рахунок пропрієтарного обладнання є дуже високою. Оскільки на заміну традиційних мереж активно виходить SDN і NFV технології то раціонально провести аналіз безпеки саме цих архітектур. NFV в свою чергу за рахунок віртуалізації зможе надати єдиний інтерфейс керування мережевим обладнанням, але все таки SDN є головною концепцією, яка в свою чергу може задіяти переваги NFV і додати свої власні. Тому раціонально розглянути проблеми безпеки саме SDN підходу.

1.2 Рівні вразливостей SDN

За рахунок того що SDN розділився на 3 окремі рівні, тим самим це стало причиною виникнення проблем безпеки на кожному з них. Кожен рівень відповідає за свої завдання в мережі SDN і співпрацює з іншими рівнями, щоб забезпечити ефективну роботу мережі. Тому важливо провести аналіз можливих типів атак на кожному з рівнів.

1.2.1 Рівень контролю

Це серце мережі SDN. Він відповідає за збір інформації від комутаторів, прийняття рішень щодо керування мережею та надсилання необхідних інструкцій комутаторам. Рівень контролю SDN відповідає за взаємодію з перемикачами, він отримує інформацію про пакети для яких відсутні правила, генерую відповідні і передає команду на вставку відповідних правил на перемикач, враховуючи статистичні дані відомі йому про стан мережі в момент часу t.

1) Атака на відмову в обслуговуванні - одна з найважливіших проблем безпеки для SDN контролера. Ціллю цією атаки є вивести контролер з дії тим самим змусити його перестати відповідати на запити перемикачів, що може спричинити повну не роботоспособність мережі. Оскільки для кожного нового пакету перемикачі відправляють запит на контролера щоб отримати відповідні потокові правила, то в разі відмови перемикачі почнуть ігнорувати пакети [10]. Тобто атака складається з 3 кроків: присилати пакет на перемикач для якого невідоме потокове правилом в таблиці маршрутизації, перемикач відправляє заголовок пакету на перемикач, контролер отримує запит і обробляє його і відправляє результат назад. При такій атаці відбувається навантаження каналу зв’язку і на ресурси контролера. Враховуючі що контролер в середньому може обробляти 30 тисяч запитів в секунду, для великих мереж навантаження на контролер може бути значно більшим.

2) Загрози від додатків - додатки які реалізовані поверх рівня контролю, оскільки додатки зазвичай мають повний доступ до ресурсів контролера, вони можуть як запитувати так і відправляти інформацію. Виникають проблеми з аутентифікацією цих додатків та авторизацією їх до доступу к певним ресурсам оскільки не має сформованих стандартів. Важливим в цьому напрямі є ізоляція і аудит дій додатків. З’являється проблема в тому що безпека контролера залежить напряму від захисту встановленого на самих додатках [11].

3) Вразливості через масштабування - за рахунок того що всі перемикачі при отриманні пакетів для яких відсутні записи в таблиці потоків відправляють запит до контролера, контролер стає “вузьким місцем”, відповідно до дослідів [12] говориться що контролер не здатний обробляти над великі об’єми даних. Також через те що контролер у великих мережах може знаходитись на великій кількості хопів, це може спричиняти великі затримки по конфігурації перемикачів. В той же час компанії починають реалізовувати кластерні варіанти реалізації контролерів щоб можна було балансувати навантаження.

4) Загрози неправильної конфігурації - через те що мережею можу керувати декілька контролерів є проблема в конфлікті конфігурацій цих контролерів, за рахунок цього можуть виникати ситуації коли контролери змагаються за те щоб встановити свої правила на перемикачах тим самим порушувати роботу мережі.

5) Загрози розподіленого рівня контролю - через велике навантаження на контролер використовують кластерний режим його роботи. Кожен контролер відповідальний за певну підмережу і це може мати проблеми з автентифікацією і авторизацією [13].

1.2.2 Рівень даних

Цей рівень відповідає за маршрутизацію пакетів, отриманні потокові правила наповнюються в залежності від режиму контролера. Можливі 3 варіанти роботи: реактивний, про-активний, гібридний. Суть реактивного полягає що таблиці правил можуть змінюватись під час роботи мережі. Суть про-активного полягає в тому що таблиці наповнюються ще до початку роботи самої мережі. Гібридний же режим комбінує реактивний і про-активний режим роботи. Тому для цього рівня маємо наступні види атак:

1) Атака на режим контролера - оскільки контролер працює зазвичай в реактивному і про-активному режимах, то це може бути експлуатовано. Режим роботи конкретного контролера доволі легко визначити за рахунок затримки

першого пакета для якого перемикач зробить запит на контролер. Є можливість встановити на перемикачах несанкціоновано потокові правила в реактивному режимі контролера [13].

2) Атака “людина посередині” - через відсутність заголовку автентифікації в OpenFlow пакеті відбувається підміна правил, через що трафік починає проходити через не аутентифікований вузол в мережі. Також є можливість arp-спуфінгу що також дозволить зловмиснику отримувати весь трафік іншого користувача мережі. Звісно що для шифрованого трафіку ризики значно менші, але можуть уволюватись затримки для кінцевого користувача і довіра до такої мережі буде значно менша [14].

1.2.3 Рівень додатків

Рівень інтерактивно взаємодіє і змінює стан мережевих елементів за рахунок взаємодії з контролером. Додатки знають весь стан мережі можуть доступатись до ресурсів її компонентів. Додатки реалізуються мережеві функції, але через повний доступ до мережі вони становлять загрозу для неї самої. Можна виділити наступні проблеми безпеки для цього рівня:

1) Автентифікація і авторизація - оскільки додатки представляють публічний інтерфейс доступним адміністраторам мережі, є проблема авторизації і автентифікації. Оскільки немає стандартів по запровадженню обмежень, зловмисник може отримати доступ до додатків тим самим виконувати команди на контролері.

2) Вставка шкідливих правил - оскільки моніторинг додатків не відбувається, вони можуть генерувати потокові правила які б дозволяли виконувати атаки “людина посередині” в разі компрометації самого додатка.

Таблиця видів атак в залежності від рівня SDN

Таблиця 1.1

Загрози рівнів SDN мереж

SDN рівень	Тип загрози	Опис
Рівень контролю	Атака на відмову в обслуговуванні	Відмова мережевих ресурсів, неспроможність відповідати на запити
	Загрози від додатків	Несанкціоноване керування контролером через додатки
	Вразливості через масштабування	Неправильність конфігурацій мульти-контролерних мереж
Рівень контролю	Загрози неправильної конфігурації	Конфлікти контролерів при вставці потокових правил
	Загрози розподіленого рівня контролю	Проблеми авторизації і автентифікації
Рівень даних	Атака на режим контролера	Встановлення фальшивих потокових правил
	Атака “людина посередині”	Перехват трафіку який відправляється користувачами
Рівень додатків	Автентифікація і авторизації	Режим доступу до інтерфейсу користувача
	Вставка шкідливих правил	Відсутність форми авторизації при взаємодії з контролером

Тож таблиця наводить список актуальних атак на кожному з рівнів SDN, описує їх особливості, таблиця включає тільки найбільш небезпечні види атак. Це ті види атак які потребують ретельного аналізу.

1.2.4 Висновки по вразливостям SDN

Хоч і SDN є новим підходом, він також має доволі значну частину проблем безпеки. Для рівня додатків можна побачити відсутність стандарту авторизації і автентифікації, можливість використовувати повний доступ до мережі тим самим ставити під загрозу дії мережі. Оскільки рівень даних більше залежить від рівня захищеності встановлений самим адміністратором мережі і додатків, не є раціонально розробляти механізми протидії виявленим атак. Дивлячись на рівень даних можна побачити доволі значущі атаки, які б слідувало дослідити, в роботах [15]. Тому доцільно буде розглянути проблеми рівня контролю, а саме атака на відмову в обслуговуванні оскільки вона має найбільший ефект на працездатність мережі.

1.3 Огляд видів атак на відмову та дослідження існуючих рішень

1.3.1 Види атак на відмову

Атаки на відмову відбуваються шляхом коли злочинець відправляє надзвичайно великий об’єм фейкових пакетів в одну мережу щоб атакований сервіс перестав відповідати на запити звичайних користувачів. Або завантажити канал зв’язку великим об’ємом даних що не дозволить іншим користувачам отримати доступ до працюючого сервісу. Також є розподілені атаки, суть яких також як у звичайної атаки на відмову за винятком те що пакети відправляються не з одного вузла який доволі просто можна заблокувати. Пакети відправляються зразу великою кількістю заражених вузлів, при грамотній конфігурації виявити чи реальний користувач робить запит є надзвичайно важкою задачею. В SDN можливі наступні види атак на відмову:

1) Атака за рахунок протоколу - відправляються Echo запити на довільні порти жертви, жертва має певні сервіси які працюють на портах, при отриманні

великої кількості пакетів порт на якому працює сервіс починає обробляти велику кількість даних, що спричиняє відмову в обслуговуванні для звичайних користувачів [16]. Дані атаки можливі для сервісів працюючих на протоколі UDP, ICMP, TCP, і для протоколів вищого рівня.

2) Підробка IP адреси пакету - злочинець замінює IP отримувача пакета на іншого вузла чи навіть на неіснуючого. Сервер починає в нескінченному циклі обробляти запит оскільки не може відправити відповідь.

3) Фрагментація IP - при якій пакет розбивається на частини, заголовок фрагменту IP пакету містить заголовок оригінального пакету, але кінцевий пакет не відправляється, стек TCP/IP почне їх збирати IP пакет разом, але при невдалій спробі буде повторювати цю дії в циклі.

4) Мертвий ICMP запит - злочинець відправляє пакет з розміром більшим за ліміт IP протоколу. Машина жертви не може обробити пакет більшого розміру тому отримає помилку і починає рестарт системи.

5) Атака на потокові правила - ця атака більш характерна для SDN мереж оскільки вона спрямована на контролер мережі, що може спричинити непрацездатність всієї мережі. Злочинцю достатньо відправляти пакети з невідомим хостом отримувачем. Перемикачі будуть переправляти заголовки до контролера, що змусить його опрацьовувати велику кількість фейкових повідомлень. При великій частоті таких запитів контролер перестане відповідати на запити.

1.3.2 Актуальність проблеми DoS

Відповідно до проаналізованих даних можна побачити що більшість атак спрямованих саме на те що вивести сервіс з ладу, щоб він перестав обробляти запити звичайних клієнтів. В рамках SDN таким атакам можуть бути підвернуті всі користувачі мережі. Звичайним рішенням по забезпеченню сервісів від DoS/DDoS атак є встановлення IDS [17], в якості реальної реалізації беруть Snort. Ця система не пропускає трафік злодія до кінцевого сервісу. За рахунок таких систем досягається швидкодія кінцевих сервісів, вони є захищенні від великої кількості вже

відомих атак. Але в рамках SDN якщо застосувати такий підхід то по-перше це б заблокувало трафік від перемикачів, щоб унеможливити їх налаштування, по друге це зроби майже не реально, оскільки в рамках OpenFlow протоколу відправляється тільки заголовки, які зазвичай не несуть достатньої кількості інформації.

Тому в даній дисертації увага буде зосереджена саме на забезпечені безпеки контролера в мережі SDN, оскільки він є ключовим елементом даної архітектури.

1.3.3 Дослідження існуючих рішень

Оскільки проблема атаки на відмову є дуже актуальною існує велика кількість пропозицій які намагаються вирішити дану проблему.

Для більшості сервісів для того щоб захистити їх в атаки на відмову використовують IDS [17]. В рамках SDN є також можливість встановити [18] IDS. IDS дозволяє проводити аналіз пакетів на аномалії відомі в його базі даних. В статтях [18] пропонується використання IDS який аналізує кожен пакет, в разі виявлення відправляє запит на контролер встановити відповідні правила на перемикач. Перевагами такого підходу є те що більшість методів аналізу вже реалізовані, проста конфігурація, обширний список аномалій які аналізуються. Недоліком же підходу є те що велика кількість працюючих ids, майже для кожного перемикача, якщо програмне забезпечення самого перемикача не може підтримувати IDS додаток то доведеться ставити додаткове обладнання яке буде виконувати цю роль. Також недоліком реалізації є відносно великі затримки для кожного пакету, що є дуже критичним під час роботи мережі.

Розглядається кластерна організація контролерів з метою балансувати мережу при великих навантаженнях [19]. Як варіант мати гарячі резерви кластерів, щоб в рамках відмови обладнання система продовжувала працювати. Перевагою підходу є що система продовжує працювати, тобто наслідки атаки не впливають на користувачів мережі. Недоліком є необхідність тримати резервні контролери працюючими що може призвести до великих витрат. Також недоліком є те що OpenFlow протокол не надає механізму координації головного контролеру з його

репліками. Тому існує можливість що ряд конфігурацій при переключенні на інший контролер будуть втрачені і може призвести до не найкращих наслідків. Також в ряд недоліків реалізації можна відвести зосередженість на конкретній реалізації NOX контролера, такий підхід може бути проблемний у впровадженні для інших контролерів.

Також описують спосіб протидії атакам на відмову за рахунок легкого аналізу на основі збирання потокових правил у перемикачів і їх аналізу [20]. Метод не основується на аналізі пакетів, він використовує потокові таблиці перемикачів. Розробники використовують різні методи для виявлення DDoS-атак на мережах. Одним з таких методів є моніторинг потоків, що протікають через мережеві комутатори. В залежності від способу реалізації, моніторинг може бути реалізований за допомогою програмного забезпечення на окремих серверах або безпосередньо на комутаторах, що є частиною мережі. Використовується нейронна мережа без учителя яка дозволяє класифікувати трафік. Метод протестований на штучно створеній мережі і виконує аналіз на основі власне сформованого дата сету. Перевагою підходу є мінімізація навантаження при аналізі трафіку, до недоліків можна віднести обмеженість дослідження для великих систем, де потрібно масштабування та відсутність можливості системи працювати в разі відмови сусідніх вузлів, що унеможливить встановлення потокових правил за рахунок контролера.

Таблиця 1.2

Існуючі рішення

Існуючий підхід	Опис
Кластерна організація контролерів	Гарячі резерви кластерів, щоб в рамках відмови обладнання система продовжувала працювати. Перевага полягає в безперервній роботі рівню контролю. Недолік полягає у відсутності способу синхронізації даних між контролерами

Продовження таблиці 1.2

Існуючий підхід	Опис
IDS	IDS дозволяє проводити аналіз пакетів на аномалії відомі в його базі даних. Перевага полягає в легкій інтеграції цього способу і висока якість перевірки атак. Недоліками є затримки які збільшиться через цей аналіз для кожного пакету і обмежена можливість інтеграції у перемикачі.
Використання нейронної мережі	Аналіз потокових правил нейронною мережею. Перевагою є значно менші ресурсні витрати для виконання аналізу наявності вторгнень. До недоліків можна віднести відсутність досліджень по масштабуванню і аудити аналізу цієї мережі, та у неможливості реагувати на атаки у випадку перевантаження вузлів.

Тож виконавши аналіз можна побачити переваги та недоліки існуючих підходів, вони спроможні вирішувати проблему атаки на контролер, але дані підходи або мають значні ресурсні атаки для виявлення атаки, або не покривають теми масштабування та застосування в мульти-контролерних середовищах.

Висновки до розділу 1

Проаналізовано види мереж актуальних на сьогоднішній день, розглянуті традиційні мережі, SDN та NFV підходи. Виявлено що велику актуальність мають SDN мережі які в свою чергу мають проблеми безпеки на 3 рівнях.

Хоча традиційні мережі можуть бути менш гнучкими та складними у керуванні, вони все ще мають значний потенціал для застосування в різних сферах. Однак, SDN - це нова технологія, яка має значні переваги в порівнянні з традиційними мережами, особливо з точки зору гнучкості та простоти управління мережею. Тому, обрано дослідження SDN це перспективна технологія, яка може забезпечити більш ефективне управління мережами в майбутньому.

Провівши аналіз виявлено головні проблеми безпеки та відокремлені ті які мають найбільший вплив на SDN мережі. У ході дослідження SDN мережі виявлено, що основні проблеми безпеки пов'язані з управлінням, зберіганням та обробкою даних на кожному рівні SDN.

Наприклад, на рівні прикладного програмного забезпечення можуть бути проблеми з безпекою, пов'язані з недостатньою перевіркою даних, відправлених до контролера та комутаторів, що може привести до небажаних наслідків, таких як втрата даних чи порушення приватності.

На рівні керування мережею можуть виникати проблеми з безпекою, пов'язані зі зберіганням конфігураційних даних контролера та доступом до них з боку зловмисників. Однак найменш дослідженою проблемою є DDoS/DoS атаки на SDN контролер. При цьому, однією з причин може бути те, що комутатори відправляють запит до контролера, коли не можуть знайти відповідні потокові правила, що може створювати значне навантаження на контролер та створювати уразливість для DDoS/DoS атак.

Тож на основі проведеного аналізу виявлені напрями в SDN мережах які потребують захисту, одним з важливих є забезпечення безпеки контролера в мережі SDN від DDoS атак.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ПІДХОДУ

2.1 Опис підходу на основі блокчейн

Оскільки постає проблема захисту контролера від атак на відмову, так як зловмисники, починають відправляти запити до точок доступу для яких відсутні встановленні правила. Встановлення правил на перемикачах які діють з контролером за OpenFlow протоколом може відбуватись в трьох режимах, реактивний, про-активний і гібридний. Особливістю реактивного з гібридним є те що вони дозволяють встановлювати нові правила під час роботи мережі, тобто при отриманні пакету для якого відсутній запис в TCAM, перемикач робить запит до контролера для отримання потокових правил. Тоді як особливістю про-активного режиму полягає в тому що він встановлює правила на перемикачі заздалегідь і всі невідомі запити він ігнорує. Для визначення режиму роботи контролера достатньо визначити затримку першого пакета нового трафіку. Очевидно що виконати атаку на відмову на контролер в про-активному режимі неможливо, тому такий варіант і не буде розглядатись. Оскільки про-активний режим має певну кількість недоліків, тому даний режим не використовується у великих мережах.

Запропонований підхід базується на твердженні що використовується реактивний або гібридний режим контролера при якому користувачі мережі можуть відправляти пакети до будь-яких вузлів. Підхід говорить про те що кожен користувач мережі грає роль у забезпеченні безпеки своєї мережі. Тому необхідними елементами є:

- встановлення публічного режиму читання таблиць маршрутизації у всіх перемикачів
- встановлення сервісу координатору-брандмауеру
- встановлення інтервалу збору даних
- тренування SOM моделі на дата сеті
- встановлення блокчейну мережі

Публічний доступ користувач на читання потрібен для отримання інформації про потокові таблиці які і задіяні при класифікації SOM моделлю.

Роль сервісу координатора-брандмауера буде полягати в тому що він буде координувати користувачів мережі блокчейн, надаючи їм інформації про перелік перемикачів за які користувачі будуть нести відповідальність. Також сервіс буде читачем блокчейну і аналізуючи отриману інформацію з нього буде встановлювати відповідні обмеження на перемикачах для вузлів які будуть здаватись йому підозрілими.

Інтервал потрібен кожному сервісу для того щоб зібрати дані для аналізу, але важливо мінімізувати кількість звернень до перемикачів, щоб зменшити навантаження. Тому важливо правильно вибрати алгоритм публічного консенсуса для блокчейн мережі.

SOM - нейронна мережа без учителя, яка підходить для визначення залежності для не розмічених даних. За рахунок дискретизації отримаємо 1- або 2-вимірну карту або сітку для вхідних n-вимірних шаблонів. Відповідно до топологічного упорядкування збираються схожі особливості разом у решітку. При цьому нейрони змагаються один з одним за місце у вихідному шарі нейронної мережі.

Для того щоб зберегти результати аналізу нейронної мережі потрібен децентралізований незмінне сховище, технологія блокчейн відповідає таким вимогам. Децентралізація дозволить проводить аналіз мережі навіть при умові що деякі підмережі недоступні. Наприклад при використанні централізованої бази даних вона може бути схильна до атак на відмову що у неможливість запис і читання її. Також до блокчейну не може бути несанкціонованого доступу що гарантує незмінність і коректність зберігаємих даних.

Блокчейн - технологія яка контролюється учасниками мережі, без централізованого елемента. Всі зміни які вносяться в блокчейн доступні кожному учаснику мережі, кожен може перевірити їх валідність. Блокчейн представляє

собою централізоване сховище блоків із транзакцій, блоки посилаються як у одно зв'язному списку. Блоки додаються до блокчейну за рахунок роботи алгоритму консенсуса. Алгоритм вирішує хто і за яких умов має змогу додати блок в ланцюг.

2.2 Проектування блокчейну

Блокчейн є ключовим елементом реалізації, тому важливо розробити його правильно. Суть блокчейну полягає в децентралізації, це означає що для атаки на систему треба керувати більш ніж 50% учасників мережі блокчейн. Структура даних блокчейну є дуже простою, представляє однозв'язний список з блоків які містять транзакції у собі в середині. Блок складається з заголовка і самих даних які є незмінними в подальшому. Заголовок в свою чергу містить ідентифікатор блока, час створення, корінь Меркла та хеш значення попереднього блока. Початковий блок є винятковий він не містить хеш значення попереднього блоку.

Перевагами блокчейну є

1) Децентралізація - за рахунок того що відсутній центральний елемент який може бути скомпрометований, робить блокчейн мережею з великим рівнем довіри [21]. Додавання блоків даних відбувається за рахунок алгоритму консенсусу який виконується для учасників мережі.

2) Довіра - оскільки дані до блокчейну додаються шляхом алгоритму консенсусу, то в момент часу T до блокчейну є певний рівень довіри, в залежності від кількості реальних децентралізованих клієнтів [21].

3) Незмінність - за рахунок того що блокчейн не приймає зміни до свого ланцюга це дозволяє жорстко фіксувати всі зміни [21]. Змінить можуть дані якщо відбудеться атака 51%, коли більшість учасників мережі мають вже змінені дані.

4) Великий рівень доступності - за рахунок P2P основи блокчейну, це дозволяє будь-якому учаснику мережі стати вузлом який синхронізує ланцюг і приймає участь у голосуванні [21].

5) Великий рівень безпеки - за рахунок того що всі транзакції мають свої хеш-значення і використовуючи дерево Меркла отримаємо в результаті хеш-

значення блоку, це дозволяє всім учасникам блокчейн мережі верифікувати отриманні дані [21].

6) Відмовостійкість - є ключовим фактором, оскільки дозволяє системі працювати при виходу з ладу великої кількості користувачів [21].

2.2.1 Проектування алгоритму консенсусу

Алгоритм консенсуса є важливим елементом блокчейн мережі, за його рахунок визначається який учасник і при якій умові може додати блок до ланцюга. Вибір алгоритму є доволі важливим оскільки це на пряму вплине на швидкість додавання нових блоків. Зазвичай розрізняють приватні і публічні алгоритми. Приватні використовується в організаціях, де користувачами мережі блокчейн є довірені вузли. Публічні же алгоритми призначені для використання будь-ким, тому вони часто є витратним за ресурсами оскільки вимагають доволі складних обчислень для досягання консенсуса. Зазвичай використовуються наступні алгоритми:

1) Proof of Authority - алгоритм консенсусу який базується на певний інтервал і лідер вузла в конкретний момент часу T. Алгоритм припускає що кількість учасників мережі відома, всім учасникам відомо інтервал додавання нового блоку. В момент часу T, лідер формує блок і розсилає ближнім учасникам мережі підписаний блок, учасники розсилають цей блок далі наперед перевіривши його підпис. Оскільки кожен учасник мережі знає публічний ключ кожного учасника мережі це дозволяє перевіряти підпис блоку. Алгоритм дозволяє за фіксовані проміжки часу формувати нові блоки. Але такий алгоритм не є зручним для організації публічно, оскільки додавання нового учасника в мережу потребує інформування всіх учасників в цій мережі і зміни їх номерів [22].

2) Proof of Elapsed Time - алгоритм полягає в тому що в системами генерується випадковий час за рахунок криптографічних розширень в процесорних елементах. Клієнт чекай згенерований час і тому кому дістався менший час буде першим на відправку згенерованого блока. За рахунок технологій таких як SGX які

присутні на сучасних процесорах, учасники мережі блокчейн можуть перевірити відправлений блок, чи дійсно той хто відправляв чекав потрібний час. Такий алгоритм є значно ефективнішим в порівнянні з PoW, оскільки система не повинна витрачати ресурси на складні розрахунки, більшу частину часу клієнт буде тільки аналізувати таблиці перемикачів, а не виконувати складні розрахунки щоб досягти консенсусу. Перевагою є чітко визначенні максимальні затримки на формування нового блоку, додавання нових учасників мережі є доволі простим в порівнянні з Proof of Authority і не вимагає значних витрат на обрахунки для досягання консенсуса. Недоліком є потреба мати спеціалізоване обладнання для генерації випадкового часу [23].

3) Proof of Work - є однією з перших реалізацій, базується на формуванні хеш-значення з відповідною кількістю нулів у значенні. Вузли мережі підбирають “сіль” до сформованого блоку таку щоб на виході отримати хеш-значення з певною кількістю нулів. Перевагою є те що формування блоку є дійсно випадковим, у такого алгоритму консенсуса великий рівень довіри, також динамічне пристосування до швидкості формування системою хеш-функцій. Недоліком є над великі ресурси які витрачаються на виконання роботи які сама по собі не несе користі. Такий алгоритм використовує дуже багато електроенергії. Через цей недолік блокчейн мережі які базуються на цьому підході починають заборонятися країнами [24].

4) Proof of Importance і Proof of Stake - алгоритми консенсусу базуються на тому що кожен учасник мережі має певний рівень довіри який він здобув за рахунок взаємодії в мережі блокчейн. Кожен вузол має певний рівень довіри, під час формування блоку рівень довіри береться до уваги і використовуючи Proof of Work механізм визначається вузол який додає блок до ланцюгу. Перевагою алгоритму є не великі витрати потужносте для досягання консенсуса, але при великій кількості верифікантів, проблема залишається [25].

Проаналізувавши головні алгоритми консенсусів [26], можна зробити вивід що алгоритми Proof of Work, Proof of Importance і Proof of Stake є доволі широко розповсюджений, вимагають великих витрат на розрахунки для досягання консенсуса. В той час як алгоритми Proof of Authority і Proof of Elapsed Time є енергоефективними, тому що не потребують важких розрахунків. Proof of Authority в свою чергу потребує налаштування кожного вузла окремо, а Proof of Elapsed Time вимагає спеціалізованого обладнання для генерації часу випадково. Як конкретна в дисертації буде використовуватись Proof of Elapsed Time яка дозволить масштабувати мережу нескінченно, дозволить встановити часові рамки для генерації блоків.

2.2.2 Визначення обмежень обладнання

Оскільки буде використовуватись Proof of Elapsed Time алгоритм консенсуса, важливо визначити які розширення повинні бути у обладнання щоб вузол міг приймати m у додаванні блоків. Для роботи цього виду алгоритму потрібно хоч одне з наступних розширень: ARM Trustzone, AMD SEV, Intel SGX [23]. Ці розширення надають безпечну середу, дозволять роботу віддалених обчислень. Такі розширення генерують час очікування який має чекати вузол блокчейну. За формулою (2.1) визначається час очікування вузлом.

$$w = minW - LocalMean.log(d), d \in (0,1) \quad (2.1)$$

де, w - час очікування

minW - мінімальний час очікування

d - рівномірний розподіл

Відповідно до сайту виробника обладнання Intel [27], можна побачити широкий список процесорів з вбудованим розширенням SGX. Більшість процесорів починаючи з 2015 почали бути обладнаними цим розширенням. Доцільно використовувати SGX оскільки інші розширення такі як ARM Trustzone, AMD SEV хоч і відповідають вимогам, але все ще не мають реальної імплементації в алгоритмі консенсуса Proof of Elapsed Time, тому їх вразливості все ще не встановлені.

2.2.3 Формат обміну повідомленнями

Формат передачі і зберігання даних в мережі блокчейн є важливим питанням, оскільки це вплине на швидкість відправки, на розмір даних і швидкість пошуку кінцевих даних. Важливо визначити формат правильно щоб мережа могла розвиватись в подальшому, оскільки додавання нових даних в повідомленнях може викликати проблеми обробки їх на кінцевих вузлах. Доцільно зробити аналіз відомих форматів даних: JSON, Avro, Protobuf, CSV.

JSON - текстовий формат який надає дані у структурованому вигляді, при дисереліазації таких даних отримується структура хеш-таблиць, що дозволяє запитувати дані з швидкістю O(1), надає зручний формат для сприйняття людиною, дозволяє розширювати схему даних динамічно, недоліком є не детермінованість структури даних що ускладнює розробку додатків, також формат має надлишок додаткових знаків згідно його синтаксису.

Avro - формат даних який серіалізується відповідно до встановленої схеми даних. Має всі ті ж переваги JSON, але додатково надає схему даних яку можна змінювати згідно з планом еволюції, формат зменшує вагу даних після серіалізації. Має 2 режими серіалізації: schemaless - повідомлення без самої схеми і базовий - коли повідомлення включає схему десиріалізації в собі, але таке повідомлення має більшу вагу. Згідно з планами еволюції дозволяється 7 типів які дозволяють додавати і видаляти поля без впливу на не оновлені системи.

Protobuf - формат даних який серіалізується згідно з встановленої схеми даних. Також має всі ті ж переваги Avro але швидкість серіалізації згідно більшості тестів в нього більша, але в той самий час розмір даних теж більший.

CSV - формат даних, застосовується для відображення у форматі таблиць, надлишкових мета даних у цьому форматі немає, є зручним для прочитання людиною, є проблеми з кінцевою структурою даних, потребує формування хеш-таблиці вручну. Відсутня схема даних, що ускладнює розробку додатків.

Проаналізувавши формати даних для зберігання і відправки даних, можна підсумувати що оптимальними рішеннями, які дозволять розвиток блокчейн мережі є Avro і Protobuf. За рахунок того що Avro може надавати централізовані репозиторії схем при оновленнях даних, надає фіксовану схему даних і пропонує еволюцію без впливу на старих клієнтів мережі. Тому в даній дисертації Avro формат буде використовуватись для серіалізації даних з метою зменшити розмір даних і пришвидшити передачу в мережі.

2.2.4 Визначення хеш-функції блоків

Кожен блок в мережі блокчейн містить список транзакцій, при додаванні блоку в блокчейн він стає незмінним. Щоб досягнути верифікації незмінності даних у блоці використовують дерево Меркла [28]. Тобто транзакції утворюють бінарне хеш-дерево.

$$Хеш\ транзації\ 0 = Хеш(Tp[0]) = SHA(SHA(TP[0])) \quad (2.2)$$

При умові, що хеш алгоритмом є SHA

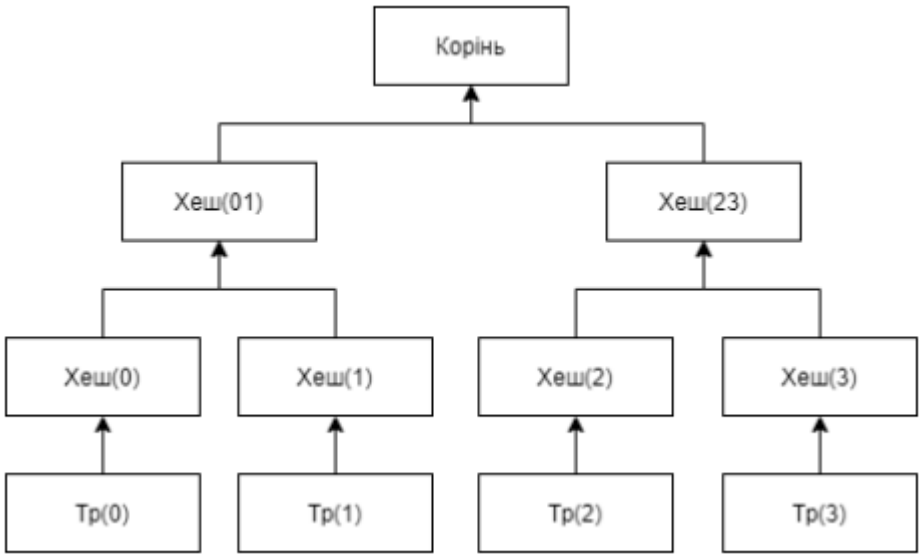

Рис. 2.1 - Дерево Меркла

Щоб уникнути підміни даних у транзакціях, важливо визначити правильний алгоритм хешування який не дозволить зловмисникам перерахувати хеш значення щоб скомпрометувати дані. Хеш-функція - це функція яка перетворює довільні

вхідні дані у строку з фіксованою довжиною. При виклику цієї функції на тих самих даних результат буде одним і тим самим. Є ряд вимог до хеш-функцій: забезпечення захисту від колізій, відповідає лавинному ефекту, тобто при змінні хоча б одного біту даних результат змінює половину вихідного значення.

Проблему для хеш-функцій становлять ASIC прилади, оскільки вони є спеціалізованим обладнанням для виконання певного ряду задач. Більшість таких приладів розробляються для вирішення конкретних алгоритмів по хешуванню. Ці пристрої мають потужність на порядок більшу звичайних процесорів і відеокарт. За рахунок їхньої швидкості є можливість підмінити дані таким чином щоб хеш-значення дорівнювало початковому. Це дозволить зловмиснику змінювати дані таким чином, що вузли які будуть перевіряти це значення будуть йому довіряти. Важливо вибрати алгоритм який наддасть достатній рівень безпеки і унеможливить створення розгалужень в ланцюгу блокчейну. Розглянемо наступні хеш-функції: MD5, SHA256, X11, SHA-3.

MD5 - хеш-функція результат роботи якої є 128 бітне хеш-значення. Даний алгоритм широко використовувався певний період часу, але було виявлено вразливість цього алгоритму [29]. Вразливість дозволяє знаходити колізії за лічені секунди. Хоч алгоритм все ще застосовується, але для систем з великим рівнем безпеки він вже не підходить.

SHA256 - криптографічна хеш-функція результатом роботи якої є 256 бітне хеш-значення. Функція широко використовується в сучасних блокчейн мережах. Доволі великий проміжок часу функція була не вразлива до підбору колізій, але з часом було знайдено колізії для 22 та 31 ітерацій. Є доволі велика кількість ASIC приладів які побудовані на роботу з цим алгоритмом.

X11 - криптографічна хеш-функція яка використовує 11 алгоритмів хешування. Алгоритм забезпечував хорошу криптографічну безпеку і був енергоефективним, але нажаль 2016 року були розроблені ASIC прилади які знівелювали його переваги.

SHA-3 - криптографічна хеш-функція, яка в порівнянні з sha256 позбавлена її недоліків через кардинально іншу структуру. Оскільки алгоритм є доволі новим, для нього відсутнє спеціалізоване обладнання для підбору значень. Результат хеш-функції є вижимки криптографічної губки. Хеш-функція може генерувати хеш-значення 224, 256, 384, 512 бітні.

З метою надати максимальний криптографічний захист для блоків в мережі блокчейн в дані статті використаємо алгоритм SHA3 з 256 бітним хешем.

2.2.5 Визначення криптографічного алгоритми підпису

Цифровий підпис в мережі блокчейн також є невід'ємним компонентом, він дозволяє ідентифікувати вузол який надав транзакцію, що повинно зменшити кількість фальшивих транзакцій. Цифровий підпис базується на асиметричних криптографічних алгоритмах. Кожен учасник мережі генерує два ключі, публічний і приватний. За рахунок приватного здійснюється підпис транзакцій, а публічний дозволяє іншим користувачам мережі впевнитись у тому хто дійсно підписав транзакцію. Доцільно розглянути RSA і ECC. Оскільки дані алгоритми широко використовувались і використовуються сучасними системами.

RSA - криптографічний алгоритм який базується на факторизації цілих чисел, тому вимагає доволі великі потужності для розрахунку. Недоліком алгоритму є схильність атаці Padding Oracle, яка використовує вирівнювання повідомлення вразливість. Алгоритм широко використовується і по сьогодення.

ECC - криптографічний алгоритм який базується на еліптичних кривих. Розмір генерованих ключів є значно меншим ніж в RSA, принцип використання такий же самий як і в RSA. Має значно складніший алгоритм підпису, надійність на пряму залежить від конкретної формули кривої яка буде використана під час підписання. Головною переваго є стійкість до атаки Padding Oracle.

Хоч RSA все ще має хорошу криптографічну стійкість, але швидкість підписання даних і наявність проблеми з можливою атакою Oracle Padding робить цей алгоритм не підходящим для блокчейн мережі яка націлена на швидкодію. Тому

в даній дисертації буде використовуватись ECC асиметричні криптографічні алгоритми, що дозволить підписувати малі транзакції значно швидше.

2.2.6 Проектування структури транзакцій

Транзакція є одиницею інформації, вона зберігається в блоці, який в свою чергу є елементом ланцюгу блокчейну. Визначення розміру даних однієї транзакції дозволить розрахувати необхідні обсяги як оперативної так і файлового простору.

Оскільки планується зберігати інформацію вже готових результатів перевірки перемикачів в момент часу T, то доцільно мати інформацію про IP адресу перемикача, беручи до уваги що IPv6 має розмірність в 128 бітів. Також важливо мати результат перевірки потокового запису 8 бітів. Щоб брандмауер міг застосувати певні дії до перемикача, важливо включити ip адреси відправника і отримувача, тому ще 2 поля по 128 бітів. Важливим є включення хеш значення транзакції і цифровий підпис з публічним ключем. Оскільки в роботі використовується цифровий підпис ECC, тому розмір публічного ключа буде становити 256 бітів, а розмір самого підпису 512 бітів. Тож сумарно розмір однієї транзакції включає обов'язкові 1160 бітів, додаткова інформація може варіюватись від розмірності мережі.

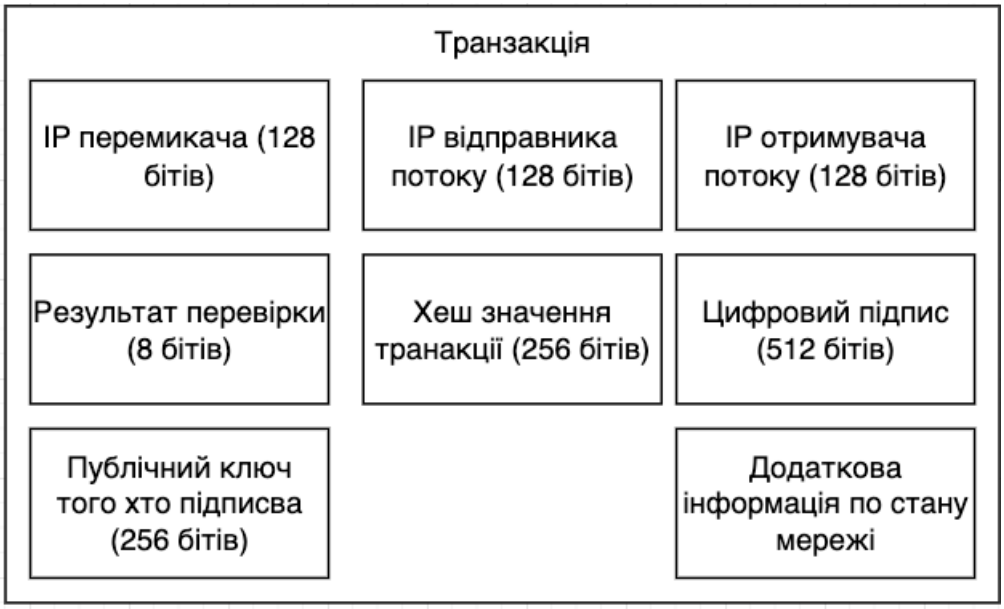

Рис. 2.2 - Структура транзакції

Структура транзакцію включає обов'язкові поля які дозволять визначити стан мережі та забезпечити її аудит для подальших досліджень.

2.2.7 Проектування структури блоків

Блок - елемент мережі блокчейн, він зберігає список транзакцій сформований за період T, корінь Меркла розрахований на основі транзакцій, хеш-значення минулого блоку(за виключенням першого блоку), час створення блоку. Корінь Меркла буде займати 256 біт за рахунок використання хеш-функції SHA-3 на 256 біт. Список транзакцій не має обмежень на кількість транзакцій, значення часу займатиме 32 біти, хеш минулого блок 256 біт. Також блок міститиме сертифікат SGX(2048 біт) яким блок і буде підписаний. Цифровий підпис становитиме 256 бітів. Загальний розмір блоку з обов’язковою інформацією буде становити 2848 біти.

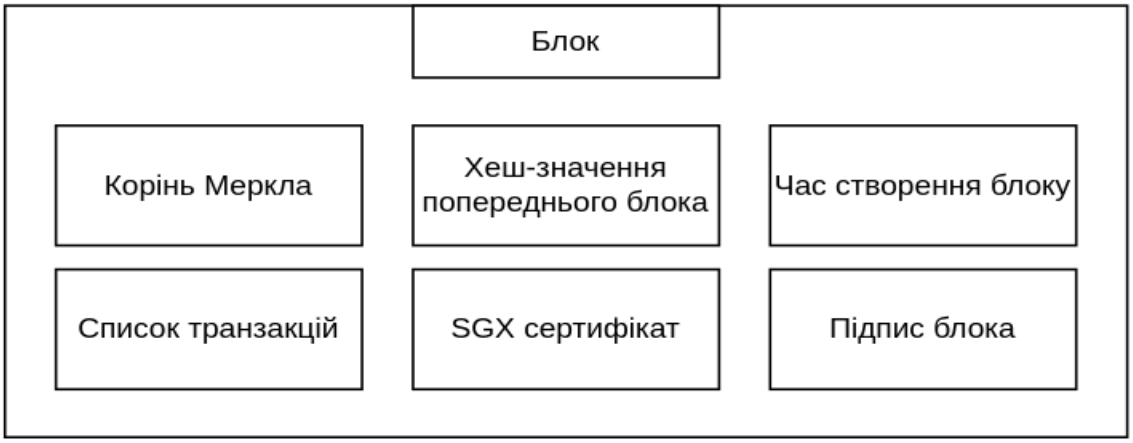

Рис. 2.3 - Структура блоку

Тож структура блоку це репрезентація списку транзакцій які включають інформацію про стан мережі проводячи тим самим її аудит, також містить всю необхідну інформацію про блокчейн клієнта що б інші учасники мережі могли аутентифікувати клієнта, його право на додавання блоку в певний момент часу.

В результаті отримаємо ланцюг блоків які зберігають в середині список транзакцій. Ланцюг буде починатись з блоку в якому відсутні транзакції і відсутнє

хеш-значення попереднього блоку, тому корінь Меркла для першого блока буде розраховуватись без використання значення попереднього блока.

Рис. 2.4 - Структура ланцюгу блокчейн

Структурна схема відображає повну складову ланцюгу який і буде зберігатись для реалізації блокчейн технології. Саме така структура, без альтернативних гілок буде гарантувати цілісність і повний аудит системи. Звісно індексація таких даних може стати проблемою, потрібно використовувати буде додаткові інструменти або організовувати власні алгоритми для швидкого пошуку.

2.2.8 Проектування нейронної мережі

В роботі буде використовуватись нейронна мережа з навчанням без учителя. Особливістю такого навчання є що нейронна мережа навчається на нерозмічених даних, тобто система на знає правильних відповідей на вхідні дані. За рахунок цього нейронна мережа починає напрацьовувати ваги схожості даних під час тренування. Така нейрона мережа не спроможна класифікувати дані, але вона здатна показати наскільки вхідні дані відрізняються від попередніх. При визначені атаки на відмову це є дуже важливо, оскільки під час атаки мережа значно змінює свої параметри. SOM(самоорганізуюча нейрона мережа) - представляє реалізацію неконтрольованого навчання нейронної мережі. Робота базується на тому що вхідні дані мають багато атрибутів, і за рахунок роботи нейронної мережі зменшуються розмірність і дискретизують вихідні дані. SOM базується на змагальному навчанні, тобто нейрони змагаються за право бути активованими на наборі вхідних. SOM реалізується за рахунок латерального гальмування, тобто нейрон-переможець знижує активність сусідніх нейронів, що дозволяє їм взаємодіяти на коротких підключеннях.

Нейронна мережа потребує навчання. Для того щоб навчити треба сформувати список особливостей і підготувати тестові дані. Мережа класифікує при умові що вона навчена на доволі великому наборі даних. Мережа дозволяє за рахунок навчених ваг класифікувати дані отримані з потокових таблиць OpenFlow перемикачів. Для класифікації використаємо 5 модифікованих ознак [20]: APfUsN, PPfUsN, GSfUsN, ABfUsN і ADfUsN. Дані ознаки будуть збиратись на основі потокових потокових таблиць перемикачів у відповідних підмережах.

1) APfUsN(середня кількість пакетів на потокове правило в рамках підмережі) - оскільки головною особливістю атаки на відмову є підробка IP адреси відправника, це ускладнює відстеження джерела таки. Це спричиняє генерацію великої кількості потокових записів з невеликою кількістю пакетів, приблизно 3 пакети на потік. Припускаючи що звичайний трафік реальних клієнтів включає має

велику кількість пакетів, тому враховується середнє значення. Визначається за формулою (2.1)

$$\text{APfUsN} = X((n\ +\ 1)\ /\ 2), n\ \epsilon\ 2Z\ \cup \frac{X(n/2) + X((n+1)/2)}{2}, x\ \epsilon\ 2Z + 1 \qquad (2.3)$$

де X-кількість пакетів; n - кількість потокових записів;

2) ABfUsN (середня кількість байтів на потокове правило в рамках заданої підмережі) - також особливістю атак на відмову є розмір корисного навантаження пакетів з метою підвищити ефективність атаки. При тому же TCP flooding відправляються пакети лише з 120 байтним розміром. Тому важливо використати середнє значення байтів на потік.

$$\text{ABfUsN}\ =\ X((n\ +\ 1)\ /\ 2), n\ \epsilon\ 2Z\ \cup \frac{X(n/2) + X((n+1)/2)}{2}, x\ \epsilon\ 2Z + 1 \qquad (2.4)$$

де X-кількість байтів; n - кількість потокових записів;

3) ADfUsN (середній час на потік в рамках підмережі) - середнє значення витраченого часу в таблиці потоків. Ця особливість зменшує кількість помилкових визначень, коли додатки обмінюються невеликою кількістю пакетів.

$$ADf\text{UsN} = X((n\ +\ 1)\ /\ 2), n\ \epsilon\ 2Z\ \cup \frac{X(n/2) + X((n+1)/2)}{2}, x\ \epsilon\ 2Z + 1 \quad (2.5)$$

де X- час потоку; n - кількість потокових записів;

4) PPfUsN (відсоток парних потоків в рамках підмережі) - особливість дозволяє перевіряти як багато парних потоків існує протягом певного інтервалу. Перевіряється умова чи IP відправника в одному потоці є отримувачем в іншому потоці в рамках одного і того самого протоколу обміну. При атаці на відмову відправка пакетів не супроводжується відповіддю на них, оскільки запити йдуть до не існуючих IP адрес.

$$PPf\text{UsN} = \frac{2 * PF}{n} \qquad (2.6)$$

де PF-кількість парних потоків; n - кількість потоків загалом;

5) GSfUsN (Показник росту одиничних потоків в рамках підмережі) - на початку атаки, кількість одиничних потоків починає різко зростати, для обрахунку

цієї особливості береться загальна кількість потоків і віднімається подвійна кількість парних потоків, ділиться отримане значення на інтервал часу протягом якого проводиться аналіз.

$$GSF\text{UsN} = \frac{n - (2*PF)}{in} \quad (2.7)$$

де PF-кількість парних потоків; n - кількість потоків загалом; in - значення інтервалу;

2.3 Проектування брандмауеру та архітектури підходу

2.3.1 Проектування брандмауеру

Брандмауер буде грати роль автоматизованого адміністратора мережі, оскільки його роль буде полягати в тому щоб читати дані с мережі блокчейн, аналізувати результат отриманих даних і застосовувати певний ряд заходів по протидії атакам. Він повинен надавати інформацію отриману від блокчейн мережі, надавати її адміністратору і приймати власні рішення які б унеможливили б атаку, тим самим така автоматизація блокування зменшить ризики вторгнень в систему.

Відповідно до офіційної документації по адмініструванню перемикачів які працюють на OpenFlow протоколі(30), можна виділити наступні команди:

1) add-flow - додавання вручну нових потокових правил, які в свою чергу мають ряд дій можливих до застосування. Можливо використовувати відразу декілька дій до пакетів. Маємо наступний список дій які можна застосувати до пакету: drop, continue, controller, evict, noevict, vacancy, novacancy.

- drop - дія яка при отриманні пакету видаляє його.
- evict - видаляє потокові правила відповідно до алгоритму
- continue - пакет переходить під відповідальність наступної таблиці потоків
- controller - відправляється пакет на контролер для встановлення потокових правил
- noevict - забороняється встановлення нових потокових правил

- vacancy:low - дія яка відправляє до контролера відсоток завантаженості таблиці потоків
- novacancy - дія яка відключає повідомлення про завантаженність таблиці потоків

2) mod-flows - змінна існуючих потокових правил на перемикачах

3) del-flows - видалення існуючих потокових правил на перемикачах

Також задача брандмауера полягає в тому щоб координувати користувачів мережі, надавати користувачам списки перемикачів які потрібно відслідковувати, надавати метаінформацію щодо блокчейн мережі.

Важливою складовою брандмауера є його гнучка конфігурація відповідно до незмінних даних отримуваних від блокчейн мережі. Такий підхід дозволить мати різні варіації по протидії атакам без значної зміни складових. Також за рахунок того що брандмауер має детальну інформацію про всі перемикачі і користувачів, він може координувати завантаженість певних вузлів мережі.

Брандмауер має всю інформацію о системі за рахунок комунікації з контролером. Контролер має змогу передавати топологію мережі з маршрутизацією перемикачів, це великий розмір даних, але кешування цих даних дозволить доволі швидко застосовувати алгоритми по протидії. Всі учасники мережі повинні реєструватись в брандмауері, це ще одна перевага цього сервісу, оскільки сервіс отримує змогу мати повну картину ситуації в мережі і направляти ресурси більш рівномірно в мережі.

Ключовим цього сервісу є те, що доступ до нього повинен бути дуже захищенний, конкретно до середовища в якому працює цей сервіс. При не авторизованому доступі до такого сервісу виникає велика ймовірність того що мережевий трафік залишиться не захищеним. Виявленням такого неавторизованого доступу до сервісу повинен займатись адміністратор мережі, який повинен заздалегідь прийняти всі міри по обмеженню публічного доступу до цього сервісу.

2.3.2 Архітектура підходу

В результаті компонування всіх компоненіт підходу має 3 головні сервіси: блокчейн, брандмауер, SDN-мережа. В описі цієї архітектури представлена загальна структура підходу до захисту, враховуючи специфіку роботи з SDN та потреби щодо безпеки мережі. Опис включає візуальне представлення основних компонентів архітектури та їхніх взаємодій, деталізуючи різні рівні захисту та контролю.

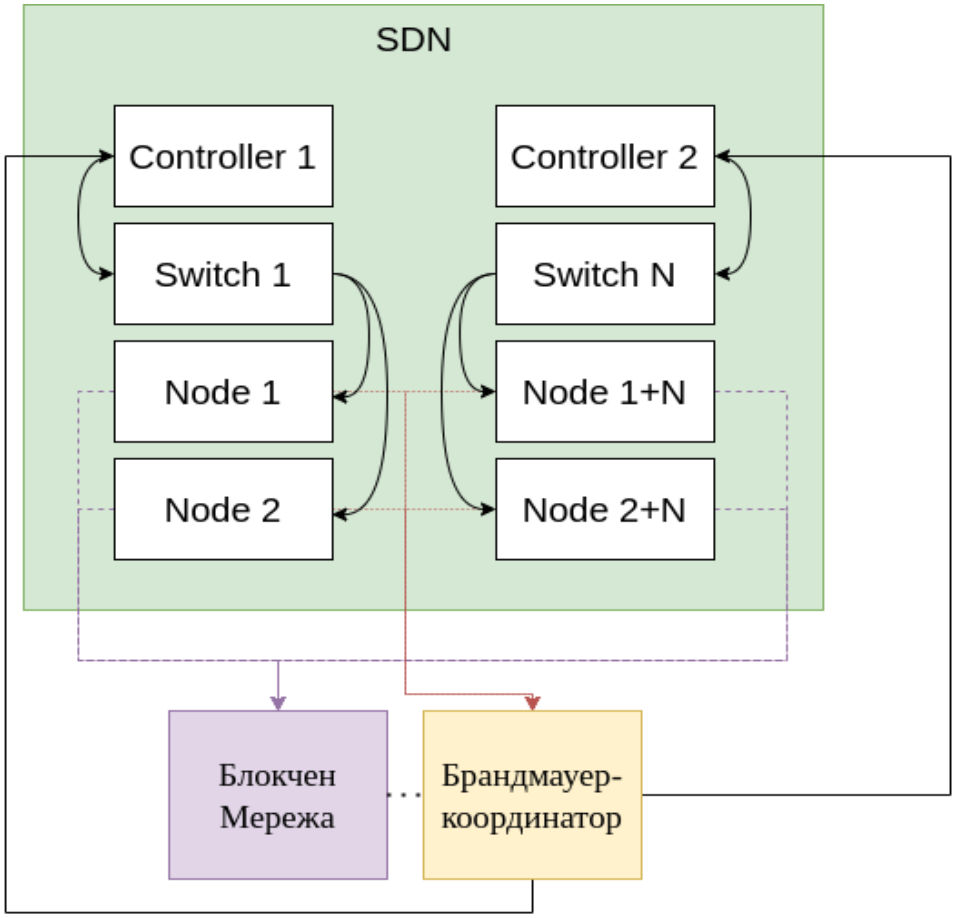

Рис. 2.5 - Архітектура підходу

Відповідно контролери взаємодіють з перемикачами, учасники мережі SDN захищають саму мережу шляхом аналізу потокових таблиць цих перемикачів, після аналізу записавши інформацію в блокчейн який збереже всю інформацію по аудиту системи і дозволить брандмауеру виконати сам захист мережі.

Висновки до розділу 2

У дослідженні проаналізовано переваги та недоліки блокчейн технології в контексті застосування її для зберігання результатів аналізу DDoS/DoS атак. Виявилось, що блокчейн може бути корисним для забезпечення надійності та безпеки зберігання результатів аналізу, оскільки він гарантує цілісність даних та може забезпечити довіреність результатів.

Зокрема, досліджено алгоритм Proof of Elapsed Time як спосіб досягнення консенсусу в блокчейні. Цей алгоритм відзначається своєю високою швидкодією, що робить його привабливим для використання в контексті зберігання результатів аналізу DDoS/DoS атак. Крім того, Proof of Elapsed Time має децентралізовану структуру та може забезпечити високий рівень безпеки та захисту від зловмисних атак.

У порівнянні з іншими алгоритмами консенсусу, такими як Proof of Work і Proof of Stake, Proof of Elapsed Time має певні переваги, такі як економію енергії та високу швидкодію. Однак, Proof of Elapsed Time має свої власні недоліки, такі як необхідність довіряти процес вибору переможця лотереї віддаленим комп'ютерам.

Отже, на основі дослідження, зроблено висновок, що блокчейн з алгоритмом Proof of Elapsed Time може бути корисним для зберігання результатів аналізу DDoS/DoS атак з метою забезпечення надійності та безпеки.

У дослідженні, визначено, що буде використано SGX (Software Guard Extensions) enclave для реалізації Proof of Elapsed Time консенсусу в контексті зберігання результатів аналізу DDoS/DoS атак. Оскільки SGX є технологією апаратного забезпечення, вона забезпечує високий рівень безпеки та надійності від зловмисних атак.

Однак, для використання SGX enclave необхідне спеціальне обладнання. Зокрема, має бути наявна процесорна підтримка SGX, така як Intel Xeon E3, E5 або E7 серії. Крім того, також потрібна материнська плата з підтримкою SGX, яка має мати спеціальну фірмову програму для обробки ключів і конфігурації SGX enclave.

У дослідженні були проаналізовані різні формати обміну повідомленнями для використання у блокчейн мережі, зокрема JSON, Avro, Protocol Buffers, CSV та інші. Для кожного з форматів були вивчені переваги та недоліки з точки зору їх використання в блокчейн мережі. Після аналізу було вирішено використовувати формат Avro для обміну повідомленнями в блокчейн мережі. Одним з головних факторів, що вплинули на це рішення, є те, що Avro має досить легкий синтаксис та є досить зрозумілим для людей, що спрощує роботу з повідомленнями для програмістів та користувачів. Крім того, Avro має бінарний формат, що дозволяє передавати повідомлення більш ефективно та швидко порівняно з іншими.

У дослідженні було проаналізовано різні алгоритми хешування, оскільки це важливий аспект для безпеки блокчейн мережі. Були розглянуті такі алгоритми, як MD5, SHA256, X11 та SHA-3. Кожен з алгоритмів має свої переваги та недоліки з точки зору швидкодії, безпеки та ефективності. Після аналізу було вирішено використовувати алгоритм SHA-3 для хешування в блокчейн мережі. Одним з головних факторів, що вплинули на це рішення, є те, що SHA-3 є новітнім стандартом NIST для хешування та має високий рівень безпеки. Крім того, SHA-3 є досить швидким та ефективним алгоритмом порівняно з іншими алгоритмами, такими як SHA256.

У дослідженні блокчейн мережі розроблена структуру блоків та транзакцій, що відповідає основним принципам блокчейн технології. Структура блокчейну містить блоки та транзакції. Кожен блок містить унікальний хеш, який ідентифікує його в мережі, та список транзакцій. Кожна транзакція містить унікальний ідентифікатор, адреси відправника та отримувача, а також кількість коштів, що перераховуються. Також в транзакції можуть бути вказані додаткові дані, які відповідають вимогам додатку.

Описаний алгоритм SOM моделі, який використовується для тренування нейронної мережі з метою виявлення аномалій в мережі SDN. SOM модель дозволяє кластеризувати дані трафіку, що дозволяє виявляти незвичайну поведінку в мережі.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ

3.1 Визначення моделі мережі

З метою протестувати розроблене програмне забезпечення є потреба у визначенні середовища тестування і його компонентів. Мета середовища симуляції полягає у наданні таких умов, які були б близьким до реальних. Середовище повинно надати всі мережеві компоненти необхідні для побудови SDN мережі. Вибір середовища емуляції знизить час конфігурації, знизить кількість обладнання необхідного для дослідів, а також наддасть зручний інтерфейс який дозволить наочно побачити стан системи в будь-який час T. Також важливо є вибрати SDN контролер який буде виконувати оркестрацію мережі, важливо проаналізувати принцип і можливість конкретної реалізації контролера працювати в кластерному режимі. В ході розробки системи важливо є розробити сталу архітектуру мережі на якій і буде відбуватись тестування реалізованого підходу.

3.1.1 Визначення середовища моделювання мережі

Важливою складовою моделювання тестового стенду мережі є саме середовище емуляції, яке буде складатись з простих вершин цієї мережі, комутаторі і контролера. Важливим є можливість побудови мережі програмним шляхом і можливість довільної конфігурації кожного компонента цієї мережі. Оскільки є потреба в змінні мережевої конфігурації з доступом до терміналу цих компонентів. Для зменшення ресурсних витрат на симуляцію мережі важливо щоб середовище могло працювати з механізмом ізоляції, замість віртуалізації. Широко розповсюдженими є середовища симуляції такій як: Mininet, Cisco Packet Tracer, GNS3, ns3, omnet++, maxiNet. Кожне з цих середовище має широку спільноту яка може допомогти у вирішенні проблем з якими може зустрітись користувач.

Таблиця 3.1

Середовища емуляції мережі

Середовище емуляції	Дослідження
Cisco Packet Tracer	Програмне забезпечення яке надає можливість виконувати симуляцію мережі. Має перемикачі, комутатори, контролери і багато іншого в своєму розпорядження. Дозволяє успішно моделювати мережу з великою кількістю компонентів, має готові реалізації серверів. Інструмент розробляється компанією яка спеціалізується на мережевих складових. Головним же недоліком є те що функціональність пристроїв обмежена і не дозволяє задіяти всі можливості реального обладнання, тобто це унеможливлює підключення зовнішніх компонентів які б могли виконувати аналіз мережі, тим самим запропонований підхід не зможе виконати аналіз мережі у цій системі симуляції. Також для користування цим програмним забезпеченням потребується наявність ліцензії.
ns3	Графічний симулятор мереж, метою якого є надати відкриту платформу вільну до розширення, через що проект з відкритим кодом. Інструмент дозволяє виконувати тільки симуляції, без емуляції. Має широкий список бібліотек з різним функціоналом. Носить виключно навчальний характер.

Продовження таблиці 3.1

Середовище емуляції	Дослідження
omnet++	Модульний компонентно-орієнтований інструмент для дискретно-подійного моделювання мережею. Перевагою інструменту є його низькі потреби до ресурсів, також переваго є його модульність, що дозволяє підключати бібліотеки з різним функціоналом без необхідності реалізовувати мережеві компоненти власноруч. Як конкретна реалізація цього середовища існує фреймворк INET. Недоліком ж є неможливість інтеграції компонентів цього середовища з іншими додатками оскільки відбувається повна симуляція роботи замість часткової емуляції.
mininet	Середовище яке дозволяє створювати вершини, для створення вершин не використовується віртуалізація, натомість використовують контейнеризацію. Такий підхід значно зменшує ресурсні витрати. Для ефективності, всі контейнери mininet поширюють одну і ту саму файлову систему. Конфігурація топології мережі дозволяється програмним чином, що дозволяє генерувати різні топології за дуже короткі періоди часу. Тож mininet базується на лекговісній емуляції замість симуляції і повної емуляції, це дозволяє робити системи великих розмірів без впливу на ефективність системи на якій відбуваються тести.

Продовження таблиці 3.1

Середовище емуляції	Дослідження
maxiNet	Емулятор є розширенням mininet який дозволяє розширити емуляцію на декілька фізичних машин. Маємо всі переваги малих за розміром контейнерів в поєднанні з декількома фізичними, це дозволяє робити мережі великих розмірів і проводити їх аналіз.
GNS3	Графічний емулятор який дозволяє будувати мережеві топології, за рахунок того що він має можливість запускати операційні системи маршрутизаторів. Функція емуляції дозволяє мати повний функціонал обладнання яке емулюється. Також за рахунок емуляції з'являється можливість побудови гетерогенних мереж, тобто з обладнанням від різних виробників що є дуже близьким до реальних умов, оскільки дуже рідко коли обладнання є від одного і того самого виробника в організації. До недоліків можна віднести неможливість емуляції комутаторі, оскільки вони побудовані на ASIC мікросхемах, і звичайні комп'ютери не спроможні на емуляцію їх. Також GNS3 потребує великі ресурси, оскільки відбувається повна емуляція кожного обладнання. І головним недоліком є нестабільність програмного забезпечення згідно з відгуками спільноти.

Раціональним вибором серед середовищ є mininet, оскільки він дозволяє по будувати мережу яка не потребує багато ресурсів, з усіма перевагами емуляції над симуляцією, надає зручний інтерфейс для побудови цієї мережі, також дає змогу задавати топологію мережі програмним шляхом. Інструмент з відкритим кодом і

активно підтримується спільнотою. Має всі елементи мережі, тобто просто достатньо їх використати.

3.1.2 Вибір реалізації контролера

Важливо вибрати контролер який на сьогодні є широковживаним і який підтримує кластерний режим роботи, щоб провести повноцінне тестування реалізації підходу. На сьогодні маємо наступних вендорів SDN контролерів: Floodlight, faucet, Beacon, POX.

- Floodlight - контролер є одним з найрозповсюдженіших у SDN мережах, він надає зручний UI і API для керування ним. Має такі переваги як навчання перемикачів і балансування навантаження. Має підтримку кластерного моду роботи. При такому режимі floodlight визначає майстер вершину, яка і синхронізує данні між нодами контролерів [32]. Контролер має контейнеризований образ, що спрощує налаштування його.
- Faucet - OpenFlow контролер з підтримкою протоколу OpenFlow 1.3 перемикачів, підтримує конфігурацію фаєвому за рахунок вставки відповідних правил на перемикачах [32]. Має моніторинг інтеграції для стеження статусу самого контролера. Контролер є доволі молодим, але вже сформував свою спільноту. Про підтримку кластерної роботи в документації нічого не згадано.
- Beacon - SDN контролер розробляємий з 2010 року тому і пропонує високу стабільність роботи, підтримує понад 100 перемикачів [33]. Контролер дозволяє динамічно налаштовувати мережу, надаючи зручний інтерфейс. Контролер має високу ефективність, тести показують що він спроможний обробляти 250 000 L2 вхідних запитів на секунду за рахунок одного потока на процесорі з частотою 2.4ghz. Про підтримку кластерного режиму роботи в документації нічого не вказано. Існує образ для запуску контролера у вигляді контейнера.

Хоч і всі перелічені контролери відповідають вимогам для роботи з мережею SDN, для емуляції мережі використаємо Floodlight контролер який підтримує кластерний режим для якого якраз і відбувається реалізація захисту. Floodlight має API яке і дозволить автоматизації процесу встановлення правил на перемикачах.

3.1.3 Налаштування симуляції

Використаємо систему контейнеризації Docker на яку не будуть впливати зовнішні процеси. Дана система створить міст, тобто створить підмережу яка буде доступною на цьому же комп'ютері. Також назначить список IP адресів, які будуть доступні в цій мережі.

Для симуляції мережі з вказаною топологією буде використано граф, у якому кожна вершина має степінь 6 (тобто кількість ребер, що йдуть з неї, дорівнює 6) і загальна кількість вершин буде обрана з урахуванням діаметру 4. Діаметр мережі означає максимальну кількість кроків, необхідних для передачі повідомлення від однієї вершини до найбільш віддаленої вершини мережі.

Оскільки візуалізація графу є доволі складною для mininet та SDN контролера, тому на рис 3.1 наведений просто граф топології. Оскільки візуалізація яку надає SDN контролер є складним для сприйняття, оскільки контролер намагається стиснути граф до мінімальних розмірів, тому зв'язки накладаються один на один, що і лишає зміст додавати дану візуалізацію.

У випадку з mininet схожа ситуація, оскільки топологія включає клієнти підключенні до перемикачів, також контролер відображає зв'язки з всіма перемикачами, тому кількість інформації надто висока і багато речей перекривають один одного.

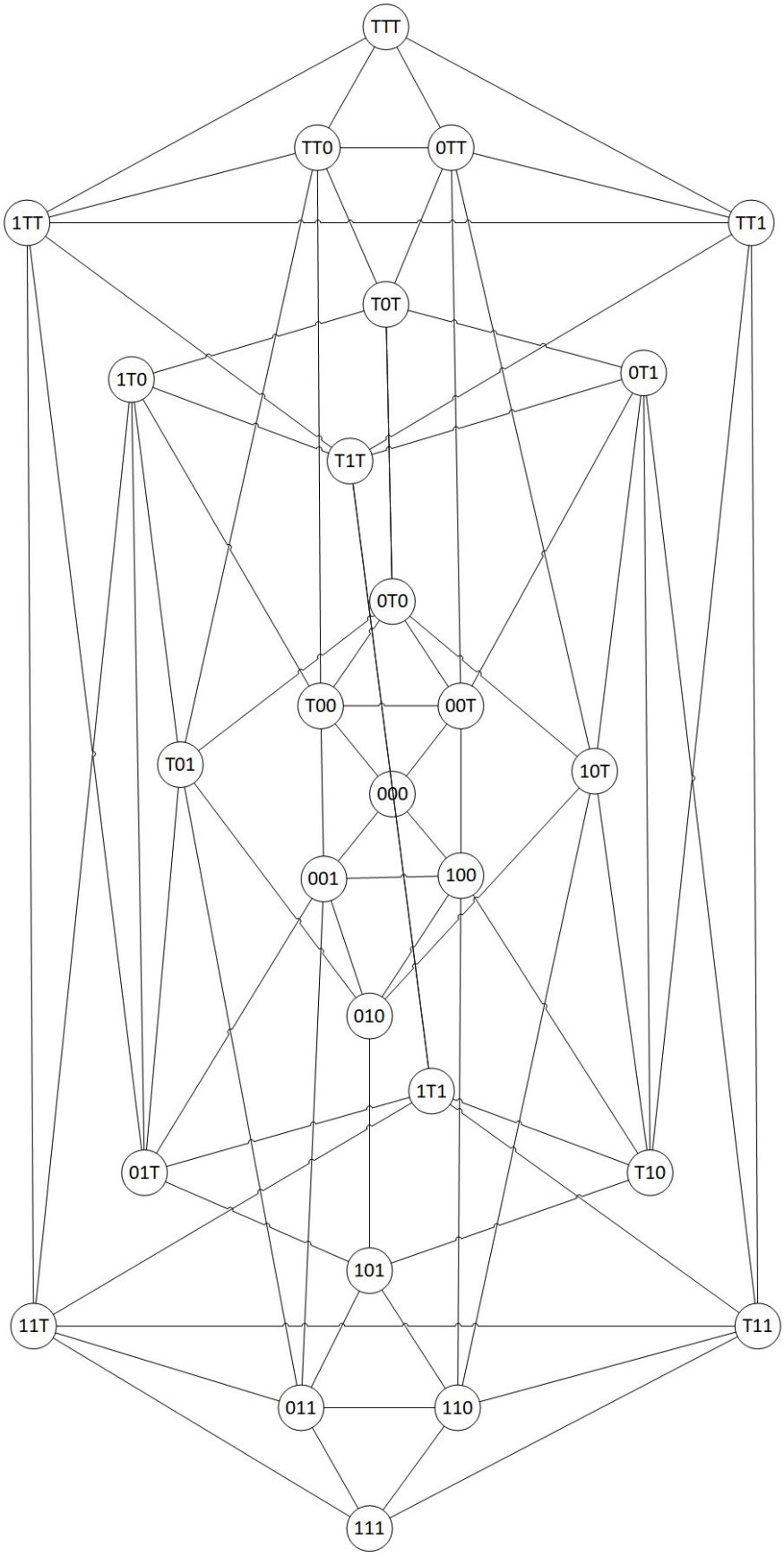

Рис. 3.1 - Зображення топології для проводження дослідження

Для того щоб отримувати лог-файли в яких міститься інформація про потокові правила які надав контролер, зробимо загально доступні директорії, до яких будуть мати доступ всі додатки. Після запуску контейнера з контролером в середні, зробимо пере направлення портів. Пере направимо порт 8080 и 6653, так як на порту 8080 запуститься інтерфейс який дозволить відстежувати стан системи, а на порту 6653 запуститься сам контролер. Цю інформації занесемо до стенду емуляції Mininet.

3.2 Визначення інструментарію симуляції DDoS атак

Для збирання тестових даних необхідний інструментарій який буде виконувати саму атаку, прогнавши його на тестовій системі, можна буде зібрати данні і навчити нейронну мережу розпізнавати різкі зміни у системі.

Розділяють наступні методи DDoS атаки:

- UDP flood - метод включає відправку великої кількості udp пакетів до жертви, що спричиняє блокування жерти, тобто унеможливлює відправку іншого трафіку та вершина перестає визначати відправника тому фільтрація брандмауера перестає працювати.
- ICMP flood - метод полягає у використання ICMP протоколу за рахунок відправки великої кількості ICMP запитів до жертви. Дана атака спричиняє значну деградація жертви.
- HTTP flood - атака на прикладний рівень, за рахунок великої кількості HTTP запитів до жертви також спричиняє її деградацію.

В цій роботі розглядається же DDoS атака на контролер, за рахунок того що отримувач повідомлення не є відомий. Той як метод атаки може бути вибраний будь-який з перелічених. Але важливо визначити метод який відправляє малий розмір корисного навантаження, щоб збільшити кількість шкідливих повідомлень і вибрати той протокол за яким було б не так очевидно що відбувається атака. Бо той же ICMP запит в публічній мережі не є повсякденно таким який щоденно

використовується. HTTP в свою чергу є розповсюдженим протоколом, мінімальний розмір його корисного навантаження становить 26 байтів, що є прийнятним.

Тож раціонально вибрати інструмент для реалізації DDoS атак, але важливо щоб ір адрес кожного разу змінювався на кожному запиті, оскільки якщо буде відправка великої кількості пакетів на невідомий адрес то перемикач просто встановить правило відкидати такі запити самостійно. Наприклад маємо наступний список інструментів [34]:

- GoldenEye - інструмент з відкритим кодом, принцип роботи полягає у створенні TCP-з’єднання, далі починає відправляти сотні повідомлень через регулярні проміжки часу. Дія застосовується виключно до одної жертви. Він здатен генерувати великий об'єм трафіку з використанням різних протоколів, таких як HTTP, TCP та UDP, з метою перевантаження цільового сервера, мережі або додатку
- Slowloris - один з розповсюджених інструментів для виконання атаки на відмову, базується на створені певної кількості підключень до жертви і підтримки таких підключень якомога довший проміжок часу. Інструмент ініціює відправку HTTP запитів але ніколи їх не завершує. Інструмент використовує дуже малі корисні навантаження.
- Scapy - це програма для створення та надсилання мережевих пакетів у Python. Вона надає можливості для маніпулювання та конструювання пакетів на рівні бітів, байтів та окремих полів. Scapy дозволяє користувачам генерувати та надсилати власні пакети, перехоплювати та аналізувати мережевий трафік, а також виконувати сканування мережі та тестувати безпеку. Scapy є потужним інструментом для мережевого тестування та дослідження. Його можна використовувати для створення тестових сценаріїв, перевірки безпеки мережі, розробки та тестування мережевих додатків, а також для аналізу мережевого трафіку

Тож, більшість інструментів для виконання атаки на відмову працюють за принципом одна жертва. В рамках цієї роботи існують інші потреби, оскільки потрібно відправляти запити завжди на різні IP адреси. Тож для реалізації цього підходу буде використовуватись scapy який підходить для звичайної генерації трафіку.

3.3 Навчання нейронної мережі

Нейронна мережа буде аналізувати таблиці перемикачів, щоб навчити дану мережу розрізнити зміни в потокових таблицях, перед тим як мережа зможе почати розрізняти, вона повинна пройти навчання на тестових даних.

3.3.1 Формування навчальних і тестових даних

Для формування тестових даних буде використовуватись та сама модель системи яка і буде аналізуватись в майбутньому, використаємо написаний раніше інструмент для виконання DDoS атак, встановимо інтервал збору інформації з перемикачів.

Кожен перемикач може надати інформації про потокові правила так як всі перемикачі мають ovs-ofctl інтерфейс підключення. Тим самим для збору інформації з моделі достатньо встановити базовий трафік між усіма вузлами цієї мережі, наприклад ICMP, встановити інтервал збору інформації з перемикачів для інструменту агрегації тестових даних. В результаті матимемо необхідні особливості для навчання таблиці Кохонена.

Команда dump-flows у Mininet дозволяє переглядати поточний стан правил OpenFlow комутаторів, які наявні у мережі. Використовуючи Mininet для моделювання нейронних мереж, вивід mininet dump може бути дуже корисним для аналізу цих мереж. Вивід mininet dump включає широкий список параметрів, які дозволяють отримати детальну інформацію про кожен елемент мережі, такий як хост, комутатор і контролер. Наприклад, параметр "flows" дозволяє отримати інформацію про потоки даних, які проходять через комутатор, а параметр "hosts" надає інформацію про кожен хост в мережі.

Крім того, вивід mininet dump дозволяє створити 5 особливостей, які потрібні нейронній мережі для виконання аналізу. Потокові записи є особливим типом даних, які мають час життя. Це означає, що потоковий запис існує протягом певного періоду часу і з часом автоматично пропадає. Крім того, при проектуванні системи з потоковими записами, варто враховувати, що ці записи можуть мати різні терміни життя в залежності від їх призначення і використання.

Таблиця 3.2

Стан потокових таблиць перемикача при передачі трафіку

Перемикач	Дамп запис
Перемикач-1	s1 cookie=0x0, duration=188.89s, table=0, n_packets=60, n_bytes=4800, idle_age=45, priority=65535,ip,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:2 cookie=0x0, duration=186.77s, table=0, n_packets=61, n_bytes=4880, idle_age=43, priority=65535,ip,in_port=2,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:1 cookie=0x0, duration=188.88s, table=0, n_packets=60, n_bytes=4800, idle_age=44, priority=65535,ip,in_port=2,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:1 cookie=0x0, duration=186.77s, table=0, n_packets=61, n_bytes=4880, idle_age=43, priority=65535,ip,in_port=1,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:2

Параметри n_packets та n_bytes можуть бути використані для розрахунку різних характеристик мережі, що можуть бути корисними при навчанні нейронної мережі. Наприклад, на основі кількості пакетів та байтів, що передаються через мережу, можна розрахувати параметри, такі як APfUsN та ABfUsN. Крім того, час,

вказаний в параметрі duration, може бути використаний для розрахунку параметра ADfUsN (average duration per flow). Цей параметр відображає середній час життя потоку даних в мережі та може бути корисним для визначення оптимальної тривалості життя потокових правил. Також, параметри nw_src та nw_dst можуть бути використані для розрахунку параметра PPfUsN. Кількість записів у потоковій таблиці може бути використана для розрахунку параметра GSfUsN.

Таблиця 3.3

Призначення параметрів в потокових правилах

Параметр	Зміст
cookie	Унікальний ідентифікатор, який асоціюється з потоковим правилом в дампі потокової таблиці, розмірність сягає 64-біти. Має значення за замовчуванням, яке дорівнює нулю
duration	Визначає тривалість того скільки запис в таблиці потоків буде зберігатись. Встановити є можливість з точністю до мілі секунд
n_packets	Кількість пакетів переданих за конкретним потоковим правилом
n_bytes	Кількість байт з отриманих пакетів які були передані конкретним потоковим правилом за весь час
idle_timeout	Визначає тривалість часу протягом якого запис буде існувати в потоковій таблиці
idle_age	Тривалість часу який пройшов без отримання жодного пакету за конкретним потоковим правилом
priority	Числове значення між 0 і 65535, в залежності від значення буде залежить порядок перевірки.
in_port	Вхідний порт

Наведена вище таблиця містить опис параметрів потокових правил, які можуть бути корисними при навчанні нейронної мережі. Зокрема, параметри такі як cookie, duration, n_packets, n_bytes, idle_timeout, idle_age, priority та in_port можуть бути використані для аналізу роботи мережі та побудови ефективної SOM (Self-Organizing Map) нейронної мережі.

Крім того, параметри duration та idle_timeout можуть допомогти визначити, як довго потокові правила будуть зберігатись у таблиці потоків та коли вони повинні бути видалені. Це може бути корисним для оптимізації ресурсів мережі та побудови ефективної SOM-мережі.

3.3.2 Побудова карти Кохонена

Карта Кохонена - це метод навчання без вчителя, який використовується для зменшення вимірності даних та виявлення взаємозв'язків між ними. Карта Кохонена може бути використана для кластеризації даних, побудови низько розмірної репрезентації великих наборів даних, а також для виявлення аномалій у даних.

При побудові карти Кохонена для 6 особливостей необхідно спочатку виконати нормалізацію даних. Після нормалізації особливостей можна використати для створення векторів даних. Кожен вектор містить значення всіх 6 особливостей для одного прикладу даних.

Карта Кохонена складається з сітки нейронів, кожен з яких є вектором з випадковими початковими вагами. Під час навчання, кожен вхідний вектор даних використовується для визначення найближчого до нього нейрону на карті. Потім ваги цього нейрону та нейронів в його околі оновлюються, щоб вони краще відповідали вхідному вектору даних.

Отже, для побудови карти Кохонена для 6 особливостей необхідно виконати наступні кроки:

1. Нормалізувати дані, щоб кожна особливість мала середнє значення 0 та стандартне відхилення 1.
2. Створити карту Кохонена, яка містить випадкові початкові ваги для кожного нейрону.
3. Навчити карту Кохонена на вхідних даних, оновлюючи ваги нейронів на основі найближчого до них вхідного вектора.
4. Визначити кластери даних на основі найближчого до них нейрону на карті Кохонена.

Як конкретна реалізація карти Кохонена можна використати готову бібліотеку ml-som. Бібліотека ml-som (Machine Learning Self Organizing Maps) - це бібліотека машинного навчання на JavaScript, яка надає інструменти для реалізації самоорганізуючих карт (Self-Organizing Maps, SOM). Основна функціональність бібліотеки включає в себе наступні можливості:

1. Створення SOM мережі з заданими параметрами: розміром карти, кількістю полів, швидкістю навчання, кількістю ітерацій та іншими параметрами;
2. Тренування SOM мережі на вхідних даних;
3. Візуалізація SOM мережі та результатів її роботи.

Бібліотека ml-som має простий інтерфейс та хорошу документацію, що робить її зручним інструментом для реалізації SOM нейронних мереж у JavaScript.

В цій бібліотеці відсутній функціонал класифікації вхідних параметрів. Тому потрібно самостійно додати цей функціонал. Ця бібліотека надає метод predict який повертає індекс найближчого вузла (BMU) на SOM (самоорганізуюча карта), що відповідає вхідному вектору. BMU - це вузол на SOM карті, який має найбільш близьку відстань до вхідного вектора, а індекс BMU - це числове значення, яке ідентифікує вузол на SOM карті. У метод predict передається вхідний вектор, який має такий же розмір, як і особливості SOM мережі. Наприклад, якщо SOM мережа має 6 особливостей, то вхідний вектор має бути масивом довжиною 6. Після виклику методу predict він повертає індекс BMU на SOM карті, який відповідає цьому вхідному вектору. Під час тренування SOM мережі, відстань між вхідним вектором і кожним вузлом SOM обчислюється і зберігається. Після тренування, коли новий вектор подається на вхід SOM мережі, вона повертає індекс BMU, який вказує на те, який вузол SOM найбільше відповідає новому вектору. Індекс BMU можна використовувати для подальшої обробки даних, наприклад, для кластеризації або класифікації нових даних.

Щоб сформувати класи на основі отриманих даних застосуємо KMeans алгоритм кластеризації, який використовується для розділення вхідних даних на кілька кластерів на основі їх схожості. Це означає, що KMeans може бути використаний для додаткової обробки результатів SOM мережі для подальшої класифікації вхідних даних. Щоб використати KMeans для класифікації результатів SOM мережі, потрібно спочатку отримати індекси BMU для всіх вхідних даних. Після цього, можна застосувати алгоритм KMeans до отриманих індексів BMU, щоб розділити їх на кластери. Кількість кластерів, на яку будуть розділені індекси BMU, може бути визначена заздалегідь або визначена динамічно залежно від кількості класів, на які мають бути розділені дані. Крім того, можна використовувати різні метрики відстані, такі як евклідова відстань або косинуса відстань, для визначення схожості між індексами BMU.

3.4 Опис розробленої системи

Зважаючи на те, що захист мережі від кібератак стає все більш актуальною проблемою, розробка систем, які можуть відстежувати потенційні загрози і запобігати їм, стає все важливішою. В цьому контексті можна виділити вашу систему, яка використовує ряд інноваційних технологій, таких як блокчейн, SOM нейронні мережі, та потокові правила, для захисту SDN мережі від DDoS атак та інших форм кібератак.

Блокчейн мережа, що працює на proof of elapsed time консенсусі, дозволяє забезпечити високу стійкість до змін та підробок. Брандмауер-координатор, що відображає інформацію отриману з блокчейн мережі, може допомогти адміністратору оцінити стан мережі та попередити можливі загрози. SOM нейронна мережа відповідає за аналіз даних з блокчейн мережі, що дозволяє виявити можливі атаки на контролер та ідентифікувати джерело атаки. Додавання нових потокових правил для блокування хостів може запобігти подальшим атакам.

Однак, перед використанням такої системи варто пам'ятати про те, що вона може бути витратною. Також важливо забезпечити надійність та захищеність

системи, оскільки вона містить конфіденційну інформацію про мережу та потенційно може бути піддається кібератакам. Вивчення особливостей та визначення можливих ризиків впровадження такої системи у конкретному середовищі є ключовим фактором для її успішного застосування.

Розробка такої системи включає в себе використання ряду бібліотек та технологій. Для навчання SOM нейронної мережі на 5 особливостях буде використовувати ml-som бібліотеку, а для класифікації результатів - алгоритм KMeans.

Для формування трафіку та DDoS атак будуе використовувати scapy бібліотеку, яка дозволяє створювати, відправляти та отримувати пакети на різних рівнях мережі.

Для встановлення потокових даних та заборони трафіку від певних вершин буде використовувати API контролера FloodLight. Перед цим SOM нейронну мережу буде навчено на mininet середовищі, щоб вона могла правильно класифікувати дані.

Також для розробки блокчейн мережі використано SGX enclave. Enclave - це ізольований обчислювальний контейнер, який забезпечує стійкий рівень безпеки. Клієнти мережі можуть реєструватись через Брандмауер-координатор, який видаватиме їм enclave та дозволить стати учасниками блокчейн мережі.

Ці технології та бібліотеки, які використовуються для розробки такої системи, можуть забезпечити її ефективну роботу та захищеність.

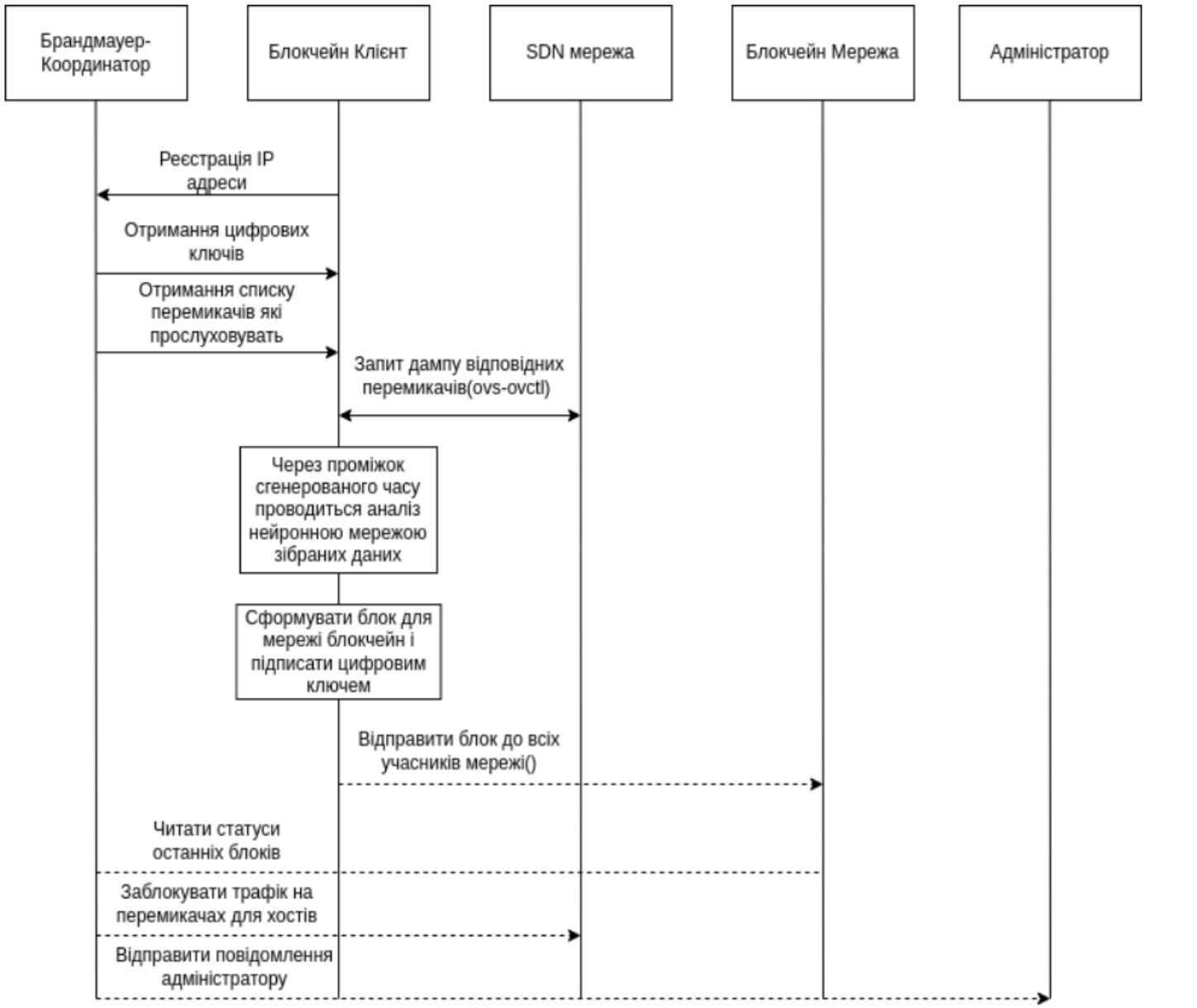

Рис. 3.2 - Зображення послідовності роботи підходу

Наведена вище рис. 3.2 відображає діаграму взаємодії компонентів підходу, який має забезпечити безпеку мережі. Схема включає всі ключові компоненти які приймають участь у захисті мережі. Описана послідовність дій кожного компонента для того щоб в результаті блокчей зафіксувавши вторгнення повідомив про це адміністратора мережі. Блокчейн мережа в цій діаграмі абстракція інших учасників мережі які повинні досягнути консенсус.

Висновки до розділу 3

У сучасному світі кібербезпека стає все більш актуальною проблемою, і розробка систем, які можуть відстежувати та запобігати кібератакам, стає все важливішою. Розглянута вами система, яка використовує різноманітні технології, такі як блокчейн, SOM нейронні мережі та потокові правила, може допомогти забезпечити високий рівень безпеки та захисту SDN мережі від DDoS атак та інших форм кібератак.

Зокрема, для навчання SOM нейронної мережі обрано бібліотеку ml-som, оскільки вона є легкою до налаштування та має високу точність класифікації результатів. Для класифікації результатів SOM нейронної мережі використано алгоритм KMeans, який також має високу точність та швидкодію.

Для формування трафіку та DDoS атак обрано бібліотеку scapy, оскільки вона дозволяє створювати, відправляти та отримувати пакети на різних рівнях мережі. Для встановлення потокових даних та заборони трафіку від певних вершин використоно API контролера FloodLight, оскільки він є досить простим у використанні та має гнучкі налаштування правил.

Для розробки блокчейн мережі обрано SGX enclave, який забезпечує високий рівень безпеки та ізольованості. SGX enclave був обраний для розробки блокчейн мережі з урахуванням використання алгоритму консенсуса proof of elapsed time. Цей алгоритм базується на використанні технології Intel SGX для забезпечення високого рівня безпеки. Використання технології SGX дозволяє зменшити витрати на обробку транзакцій та створення блоків у мережі, що робить цей алгоритм особливо привабливим для використання в децентралізованих системах.

Усі ці компоненти були обрані з урахуванням їх ефективності, точності та швидкодії, а також можливості їх інтеграції зі збіркою загальної системи. В результаті була розроблена система, яка може ефективно захищати мережу від кібератак та запобігати їх поширенню.

РОЗДІЛ 4
МОДЕЛЮВАННЯ

В розділі тестування мережі блокчейну на детекцію атак, будуть представлені результати тестування системи на ефективність детекції атак та недоліки, що були виявлені в процесі тестування. Також будуть розглянуті можливі шляхи покращення результатів тестування та усунення недоліків.

Буде наданий графічний інтерфейс на брандмауері дозволяє візуалізувати роботу блокчейну. Користувач може побачити список блоків блокчейну і позначення чи є виявлені атаки в кожному блоку. Для кожного блоку відображається його хеш-значення, момент часу створення, попередній хеш-значення та кількість транзакцій у блоку. Якщо були виявлені атаки, це буде відображено відповідним чином. Для кожної атаки буде вказана її кількість. Також слід мати на увазі, що недоліком підходу є те, що він базується на детекції аномалій, тому можливі помилкові спрацювання в разі наявності нестандартних відправників пакетів.

Також важливо створити SGX enclave, наприклад, можна використовувати Intel SGX SDK, що надається безкоштовно на офіційному сайті Intel. SDK містить необхідні інструменти та документацію для розробки та тестування SGX enclave. Всі вершини використовуватимуть один і той самий SGX enclave для генерації випадкових чисел. В цьому випадку, інші вершини можуть перевірити час, який був згенерований, з використанням спільного ключа.

Такий чином для забезпечення спрощеної реєстрації блокчейн клієнтів, вибрано використати цей варіант з одним SGX enclave. В цьому випадку всі блокчейн вершини будуть використовувати один і той самий enclave для генерації випадкового часу. Це спростить процес верифікації згенерованого часу, оскільки всі вершини будуть мати доступ до одного публічного ключа для перевірки підпису.

Щоб відрізняти, яка саме блокчейн вершина згенерувала блок, ми будемо додавати публічний ключ та підписаний хеш блоку приватним ключем у сам блок.

Це дозволить ідентифікувати, який саме клієнт згенерував блок, і відслідковувати історію транзакцій у мережі.

4.1 Розгортання середовища

Для розгортання середовища використаємо інструмент Tilt, який дозволить автоматизувати процес розгортання і конфігурування різних компонентів системи.

У складі середовища будуть наступні компоненти:

Mininet - програмне забезпечення для емуляції мережі та створення тестових середовищ для SDN.

Блокчейн ноди - децентралізовані вузли, які забезпечують роботу блокчейн мережі та зберігають копії блокчейну.

2 SDN контролери на основі Floodlight - програмні контролери для управління SDN мережею.

Брандмауер-координатор - централізований компонент, який відповідає за збір та аналіз даних про потік пакетів в мережі та виявлення атак.

Tilt буде використовуватись для автоматичного розгортання та налаштування кожного компонента в окремому контейнері, який буде запущений на Docker. Кожен контейнер буде мати власну ізольовану систему та свій конфігураційний файл.

Після розгортання, буде створено SDN мережу, що скрадатиметься з OpenFlow комутаторів, яка буде керуватись Floodlight контролерами. Мережа буде використовуватись для експериментів з різними типами атак, які будуть виявлятисьі оброблятись брандмауер-координатором. Графічний інтерфейс на Брандмауері буде використовуватись для візуалізації стану мережі, списку блоків блокчейну та позначення чи є виявлені атаки і їх кількість. Таким чином, можна буде зручно відслідковувати та аналізувати стан мережі та блокчейну в реальному часі.

Використовую власний для навчання та тестування SOM мережі на заданих особливостях. Даний дата сет - це набір даних, що використовується для тестування

алгоритмів виявленнь вторгнень. Він складається зі зразків мережевого трафіку, які містять різні види атак та нормальний трафік. Ці дані можуть бути використані для навчання та тестування алгоритмів виявлення вторгнень. Використання цього дата сету дозволяє перевірити ефективність алгоритмів виявлення вторгнень.

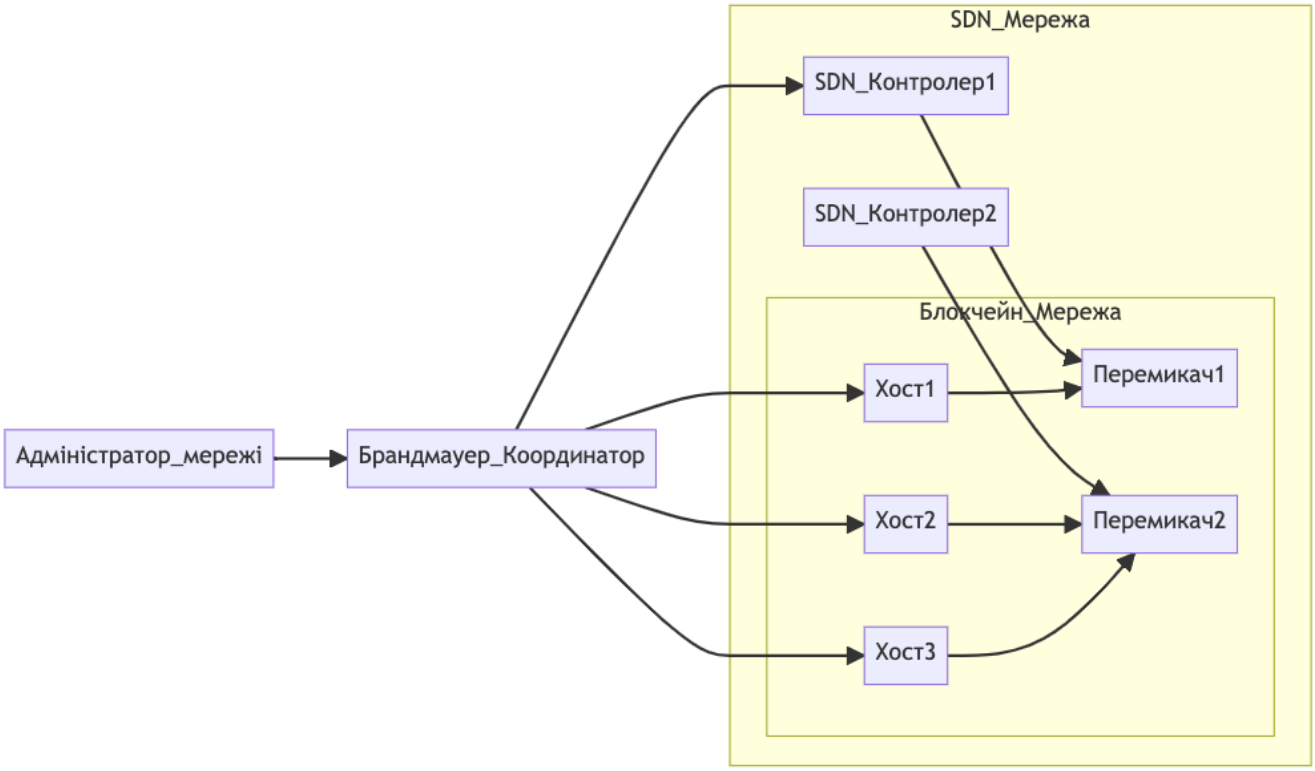

Рис. 4.1 - Спрощена структура мережі

Також важливо створити доступ на читання перемикачів через dpctl, dpctl є інструментом управління OpenFlow-сумісними перемикачами, які можуть використовуватись у SDN-мережах. Цей інструмент дозволяє адміністраторам мережі виконувати різні операції з перемикачами, такі як перевірка статусу портів, встановлення правил маршрутизації та інше.

Для створення read-only користувача для dpctl, можна використовувати механізм авторизації на базі ролей. Це означає, що потрібно створити користувача з певною роллю, яка дозволяє лише читати інформацію з перемикачів. Зазвичай ця роль називається "viewer" або "monitor".

Щоб створити read-only користувача для dpctl, потрібно:

1. Створити нового користувача з роллю "viewer" або "monitor" в системі авторизації, якщо вона використовується.
2. Додати користувача до групи, яка має доступ до виконання команд dpctl, але з обмеженням прав доступу.
3. Налаштувати права доступу для групи, яка дозволяє читати інформацію з перемикачів, але не дозволяє вносити зміни.

Тип авторизації для dpctl може залежати від конфігурації системи. Зазвичай, для авторизації використовується база даних користувачів та паролів, а також ролей, які визначають дозволені дії користувачів. Для безпеки, може бути використано шифрування або інші механізми захисту даних.

Взаємодія блокчейн хостів відбувається в рамках одного SGX enclave, що дозволяє зібрати хостам інформацію з перемикачів, і як результат прогнати вхідні данні через SOM модель, після цього досягти косенсусу в мережі блокчейну що спричинить додавання блоку у ланцюг.

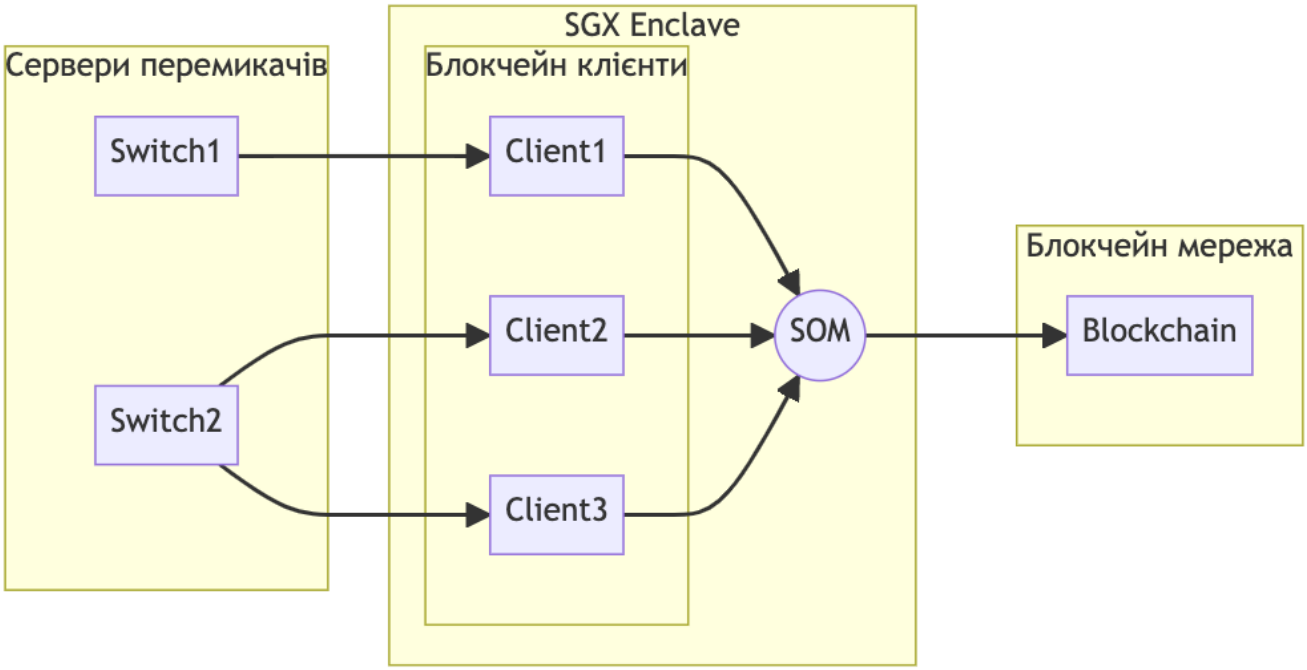

Рис. 4.2 - Спрощена архітектура підходу

Дана архітектура включає компоненти підходу що і описує принцип роботи при якому клієнти в рамках одного enclave використовують SOM нейронну мережу в яку подають данні з перемикачів і результат записують в сам блокчейн.

4.2 Результати тестування

У ході моделювання було проведено аналіз зміни часових витрат при горизонтальному масштабуванні мережі. Однак, врахування реальних значень витрат часу не є раціональним підходом, оскільки час є варіативним і залежить від багатьох факторів. Для дослідження ефективності горизонтального масштабування було запропоновано декілька топологій, включаючи звичайні трикутник і квадрат, а також випадково згенеровані топології з заданими діаметрами і ступенями. Одиницею часу вважалося 1T, з урахуванням витрат часу на передачу даних через 1 хоп, який дорівнює 1T.

$$\text{Час виконання} = \sum_{i}^{N} \frac{\sum_{j}^{N-1} hopes(i,j)}{scale} \tag{4.1}$$

де N – кількість перемикачів, hopes – відстань в хопах між перемикачами, scale - масштабування

Формула 4.1 відображає принцип розрахунку при якому використовується найкоротша відстань між перемикачами які треба опитати блокчейн клієнтам, розрахунок відбувається з врахуванням блокчейн мережі, що дозволяє знизити витрати на мережевий трафік, оскільки блокчейн клієнти знаходяться при масштабуванні ближче до кінцевих перемикачів. Якщо таке масштабування виконувати в рамках одного і того самого перемикача, то значних переваг це не наддасть, по-перше це навантажить канали зв’язку, по-друге відстань в хопах залишиться все тією ж. Тому масштабування в рамках одного перемикача не є надійним і створює нові ризики для мережі при умовах її непрацездатності.

Таблиця 4.1

Вплив горизонтального масштабування на швидкість аналізу мережі за заданою топологією

Назва топології	Масштаб-ування	Час виконання	D сер	C (ціна)	T (Трафік)	Кількість вершин
Трикутник(D=1, S=2)	1	5	0.50	6	0.50	3
Трикутник(D=1, S=2)	2	4	0.50	6	0.50	3
Трикутник(D=1, S=2)	3	3	0.50	6	0.50	3
Квадрат(D=2, S=2)	1	8	0.67	16	0.67	4
Квадрат(D=2, S=2)	2	6	0.67	16	0.67	4
Квадрат(D=2, S=2)	3	5	0.67	16	0.67	4
Квадрат(D=2, S=2)	4	4	0.67	16	0.67	4
Довільна(D=4, S=5)	1	36	1.16	220	0.47	11
Довільна(D=4, S=5)	4	22	1.16	220	0.47	11
Довільна(D=4, S=5)	8	15	1.16	220	0.47	11
Довільна(D=4, S=5)	11	11	1.16	220	0.47	11

Продовження таблиці 4.1

Назва топології	Масштаб-ування	Час виконання	D сер	C (ціна)	T (Трафік)	Кількість вершин
Топологія-27 вершин(D=4, S=6)	1	90	1.05	672	0.35	27
Топологія-27 вершин (D=4, S=6)	2	72	1.05	672	0.35	27
Топологія-27 вершин (D=4, S=6)	3	65	1.05	672	0.35	27
Топологія-27 вершин (D=4, S=6)	4	63	1.05	672	0.35	27
Топологія-27 вершин (D=4, S=6)	9	48	1.05	672	0.35	27
Топологія-27 вершин (D=4, S=6)	10	47	1.05	672	0.35	27
Топологія-27 вершин (D=4, S=6)	11	45	1.05	672	0.35	27
Топологія-27 вершин (D=4, S=6)	15	41	1.05	672	0.35	27
Топологія-27 вершин (D=4, S=6)	18	38	1.05	672	0.35	27
Топологія-27 вершин (D=4, S=6)	26	30	1.05	672	0.35	27
Топологія-27 вершин (D=4, S=6)	27	27	1.05	672	0.35	27

У таблиці масштабування наведено діаметр, діаметр системи визначається як мінімальна відстань між двома найвіддаленішими вершинами. Також вказаний середній діаметр мережі, де Середній діаметр мережі - це середнє значення діаметрів усіх можливих пар вершин в мережі. Чим менше середній діаметр, тим більш ефективна мережа, оскільки даний параметр вказує на те, наскільки швидко можливо передавати повідомлення між вузлами мережі. Вказаний також ступінь мережі, ступінь (S) - це параметр топології системи, який визначається як максимальне число з'єднань, що приходять до вершини (процесора) у графі топології системи. Також моделювання включає вартість С мережі, де вартості сумарна кількості зв'язків у системі. Значення Т відображає топологічний трафік, який визначає міру використання зв'язків у системі, тобто кількість даних, які передаються через зв'язки між вузлами мережі. І головним значення таблиці є час виконання який дозволяє визначити швидкість аналізу мережі при певному масштабуванні.

Для повноцінного тестування і розуміння впливу блокчейн підходу на швидкодію при горизонтальному масштабуванні необхідно розрахувати часові затримки на відправку транзакцій між всіма учасниками мережі, тобто що б мати інформацію про всю систему загалом потрібно щоб кожен учасник мережі обмінявся даними з іншим учасником мережі блокчейн. Для проводження цього моделювання, візьмемо часову затримку 1 хопа, або затримки на відправку між Перемикачем-1 та Перемикачем-2 як 1Т. Як умову візьмемо те що кожен хост може в одиницю часу або читати дані які надсилає один хост, або відправляти. Тобто виключаємо ситуації коли вершини може працювати паралельно над декількома задачами для правильності розрахунків.

$$\text{Час на відправку} = \sum_{j}^{scale} hopes(i,j) + delay(j) \qquad (4.2)$$

де N – кількість перемикачів, hopes – відстань в хопах між перемикачами, scale – масштабування, delay – затримка яка виникає на те що хост повинен читати і не може відправляти в цей час.

Таблиця 4.2

Затримки відправки транзакцій при масштабуванні в залежності від топології

Назва топології	Масштабування	Час на відправку транзакції
Трикутник(D=1, S=2)	1	0
Трикутник(D=1, S=2)	2	2
Трикутник(D=1, S=2)	3	6
Квадрат(D=2, S=2)	1	0
Квадрат(D=2, S=2)	2	4
Квадрат(D=2, S=2)	3	8
Квадрат(D=2, S=2)	4	9
Топологія-27 вершин (D=4, S=6)	1	0
Топологія-27 вершин (D=4, S=6)	2	6
Топологія-27 вершин (D=4, S=6)	3	14
Топологія-27 вершин (D=4, S=6)	4	17
Топологія-27 вершин (D=4, S=6)	5	27
Топологія-27 вершин (D=4, S=6)	6	30
Топологія-27 вершин (D=4, S=6)	8	46
Топологія-27 вершин (D=4, S=6)	9	52
Топологія-27 вершин (D=4, S=6)	10	58
Топологія-27 вершин (D=4, S=6)	15	97
Топологія-27 вершин (D=4, S=6)	17	108
Топологія-27 вершин (D=4, S=6)	18	113
Топологія-27 вершин (D=4, S=6)	20	128
Топологія-27 вершин (D=4, S=6)	24	151
Топологія-27 вершин (D=4, S=6)	26	160
Топологія-27 вершин (D=4, S=6)	27	169

Як видно з таблиці 4.2, при масштабуванні значно зростають витрати трафіку по обміну транзакціями які містять інформацію про атаки в мережі. Тому важливо при розробці враховувати не тільки час при масштабуванні на виконання аналізу, а і час витрачений на відправку, важливо зберігати баланс кількості вершин і тоді система наддасть максимальну швидкість виявлення вторгнень. Але важливо при цьому зважати на топологію самої мережі.

У підсумку, для того щоб зрозуміти вплив блокчейн підходу на швидкодію при горизонтальному масштабуванні, необхідно розрахувати часові затримки на відправку транзакцій між усіма учасниками мережі. Для проведення такого моделювання необхідно визначити часову затримку 1 хопа в заданій системі і визначити топологію для того щоб розробити планування по масштабуванню системи.

Швидкість виявлення загрози в блокчейні залежить від різних факторів, включаючи конфігурацію інтервалу збору даних від перемикачів та натренованість SOM-мережі. Якщо інтервал збору даних від перемикачів встановлений дуже коротким, може виникнути перевантаження мережі та збільшення часу обробки даних. З іншого боку, якщо інтервал збору даних від перемикачів встановлений дуже довгим, це може призвести до втрати потоків, які можуть бути видалені з часом самим контролером.

Під час тестування програма виводить кількість сформованих транзакцій і позначає блок успішним, тобто зеленого кольору кольору, тим самим ідентифікуючі чи все проходить успішно, програма відображає номер порядковий блоку, що дозволяє знайти його і провести необхідний аудит системи в певний проміжок часу. Під час впливу атаки на мережу можна побачити що програма фіксує понаднормову кількість транзакцій і формує список тих транзакцій які не пройшли успішно класифікацію нейронною мережею. Можна побачити що детальна інформація містить список транзакцій де відправником фігурує один і той самий IP адрес з якого і відбувалась атака на контролер. Також ця інформація

включає публічний ключ того хто сформував цей аналіз, тим самим можна переконатись що до висновку про те що відбувається атака прийшли декілька клієнтів блокчейн, а не просто один почав хибно обробляти дані.

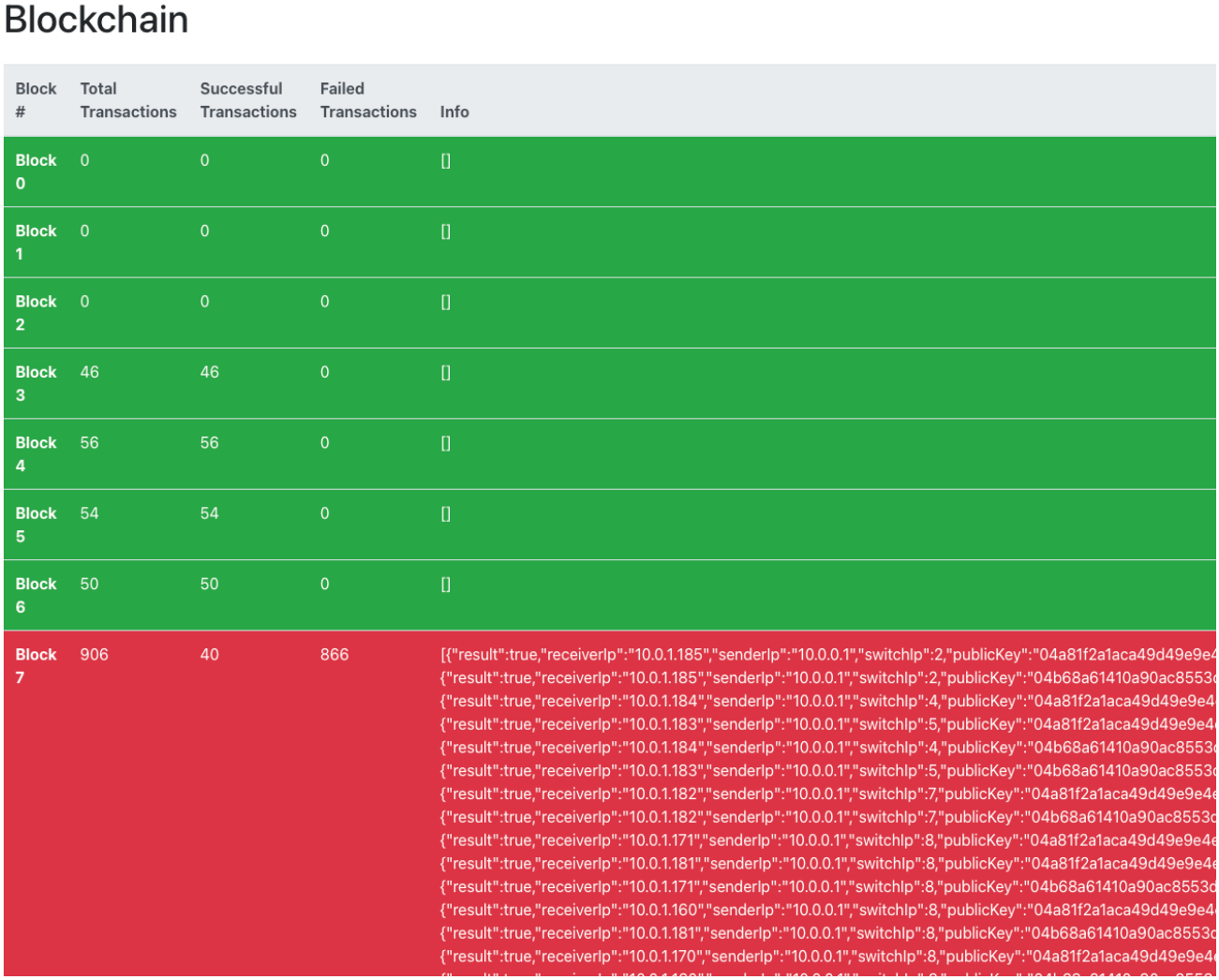

Рис. 4.3 - Результат роботи підходу

В результаті тестування було виявлено, що точність системи детектування атак залежить від заданих проміжків збору даних від перемикачів. При занадто малому інтервалі збору даних точність може бути низькою, оскільки можуть бути втрачені парні потоки, які могли б вплинути на кінцеву детекцію атаки. Тому, необхідно балансувати між точністю і часом відгуку системи, обираючи оптимальні інтервали збору даних для кожної конкретної мережі.

Додатково до впливу на точність детекції, при частому зборі інформації від перемикачів може виникати проблема з завантаженням мережевих шляхів, що може

погіршувати швидкодію всієї мережі. Це може стати особливо проблематичним, якщо мережа досить велика або зайнята великою кількістю трафіку. Тому для збирання інформації від перемикачів важливо збалансувати між точністю детекції та впливом на швидкодію мережі.

Швидкість виявлення загрози в блокчейні залежить від різних факторів, включаючи конфігурацію інтервалу збору даних від перемикачів та натренованість SOM-мережі. Якщо інтервал збору даних від перемикачів встановлений дуже коротким, може виникнути перевантаження мережі та збільшення часу обробки даних. З іншого боку, якщо інтервал збору даних від перемикачів встановлений дуже довгим, це може призвести до втрати потоків, які можуть бути видалені з часом самим контролером.

Таблиця 4.3

Стан особливостей для нейронної мережі під час роботи підходу

APfUsN	ABfUsN	ADfUsN	PPfUsN	GSfUsN	IP відправника	Є атака
2363	170275	521.111	1.1304	0	10.0.0.4	ні
1112	204330	523.682	0.956	0	10.0.0.3	ні
1668	163464	501.234	1.043	0	10.0.0.2	ні
1737.5	170275	500.608	1.101	0	10.0.0.4	ні
1598.5	156653	463.704	1.014	0	10.0.0.5	ні
417	40866	113.493	0.869	0.016	10.0.0.8	ні
139	13622	109.87	0.898	0	10.0.0.6	ні
12	2341	52341.4	0.25	7.65	10.0.0.1	так
67	4091	47150.6	0.255	21.35	10.0.0.1	так
43	917	1967.61	0.89	16.3	10.0.0.1	так
101	851	49174.0	0.255	12.41	10.0.0.1	так
0	0	44327.1	0.25	13.7	10.0.0.1	так

Як можна побачити показники різко змінюються під час початку атаки, вони саме змінюються у відправника який і виконує атаку, відправка пакетів і відповідно байтів стоїть на нулі оскільки відправки самих даних до пункту призначення не відбувається. Також можна помітити стрімкий ріст одиночних потокових правил. Оскільки відбуваються такі різкі зміни в системі нейронній мережі легше зрозуміти чи відбувається атака, оскільки числа дійсно радикально відрізняються. Якщо розглядати варіант з поступовим нарощуванням то система перший час не буде помічати вплив на систему поки не буде перейдена гранична поділка, яку доволі важко детермінувати у на тренованій мережі, оскільки розмірність даних була значно зменшена у SOM.

4.3 Опис недоліків реалізації та пропозиції по вдосконаленню

4.3.1 Недоліки реалізації

Одним з можливих недоліків такого підходу є те, що використання блокчейну може вимагати додаткового часу для формування результатів внаслідок проходження процесу консенсусу. Це може призвести до затримок у відповідях на запити та ускладнити роботу системи в режимі реального часу. Також, використання блокчейну може призвести до збільшення обсягу зберіганої інформації, що може вимагати додаткового обладнання та збільшення витрат на її зберігання. Звичайно, хоча існують недоліки використання блокчейну для зберігання даних та формування результатів, але переваги цієї технології перекривають їх.

Тренування SOM може вимагати значних обчислювальних ресурсів, зокрема обсягу оперативної пам'яті та часу обчислень. Наприклад, при тренуванні SOM на дата сеті сформованому дата сеті, який містить понад 4 мільйони записів та 41 атрибут, може знадобитись значна кількість ресурсів. Для підвищення ефективності тренування SOM на великих дата сетах можуть використовуватись різні оптимізаційні підходи, такі як зменшення розмірності даних, зменшення кількості вузлів у мережі та зменшення кількості епох тренування. Крім того, можуть

використовуватись спеціальні алгоритми, такі як batch training, які дозволяють розбити великий датасет на менші пакети та тренувати мережу на них послідовно.

4.3.2 Пропозиції по вдосконаленню

Можлива пропозиція по вдосконаленню полягає в розбитті хостів на групи для аналізу їх не окремо кожного хоста, а разом. Це дозволить збільшити ефективність виявлення DDoS атак, оскільки такий підхід дозволить виявити шаблони атак на групу хостів. При цьому можна використовувати методи машинного навчання, такі як кластеризація, для автоматичного визначення груп хостів на основі характеристик їх мережевого трафіку, забезпечуючи більш точне виявлення DDoS атак. Крім того, такий підхід дозволить знизити кількість фальшиво-позитивних виявлень DDoS атак, що може виникати при аналізі окремих хостів. Такий підхід до аналізу хостів в групах потребує більш складної нейронної мережі, яка зможе аналізувати дані в більш широкому контексті. Також потрібно мати достатню кількість тестових даних для навчання мережі, які включатимуть і дані для аналізу хостів у групах. При цьому потрібно буде забезпечити достатню швидкість обробки даних, оскільки реальний час реакції на атаку є важливим фактором для успішного захисту мережі. Таким чином, покращення аналізу хостів у групах потребує глибокого вивчення методів машинного навчання та розробки потужних і швидких систем обробки даних.

Одним із можливих напрямів удосконалення системи захисту SDN контролера від DDoS/DoS атак є використання алгоритму розподілення відповідальності між клієнтами блокчейну. Такий алгоритм може забезпечити більш ефективний та швидкий аналіз даних, зменшити навантаження на окремих клієнтів та забезпечити більш високий рівень безпеки системи.

Для досягнення цієї мети можна використовувати алгоритми розподілення завдань між учасниками блокчейну. Наприклад, можна розділити мережу на окремі зони, кожна з яких відповідає за аналіз певних перемикачів SDN мережі. Після цього, учасники блокчейну можуть зареєструватися в тій зоні, за яку вони

відповідають, та виконувати аналіз даних з цієї зони. Інший можливий алгоритм може базуватись на розподіленні завдань за принципом round-robin. У цьому випадку, кожен учасник блокчейну буде виконувати аналіз даних з перемикачів SDN мережі по черзі.

Враховуючи специфіку розробленої системи захисту SDN контролера від DDoS/DoS атак, використання алгоритму розподілення відповідальності між клієнтами блокчейну може значно покращити ефективність системи та забезпечити більш високий рівень безпеки. Однак, перед впровадженням такого алгоритму необхідно детально проаналізувати характеристики системи та врахувати можливі труднощі в процесі розподілення завдань. Застосування алгоритму розподілення відповідальності між клієнтами блокчейну, як засіб для покращення захисту SDN контролера від DDoS/DoS атак, також може зменшити кількість трафіку в мережі. Це пояснюється тим, що не всі блокчейн клієнти будуть опитувати всі перемикачі.

Наприклад, якщо використовувати алгоритм розподілення за принципом round-robin, то кожен учасник блокчейну буде виконувати аналіз даних з перемикачів SDN мережі по черзі. Це означає, що не всі учасники будуть опитувати всі перемикачі одночасно, а будуть опитувати їх по черзі. Це зменшить навантаження на окремих клієнтів, зменшить кількість трафіку в мережі та забезпечить більш ефективний та швидкий аналіз даних.

Вибір оптимального алгоритму розподілення відповідальності між клієнтами блокчейну є важливим фактором для ефективної роботи системи захисту SDN контролера від DDoS/DoS атак. Одним з головних критеріїв вибору такого алгоритму є мінімізація кількості хопів для отримання інформації, тобто зменшення кількості переходів між вузлами мережі для передачі даних.

Висновок до розділу 4

Як результат розроблений тестовий проект на детекцію атак на відмову є успішним. Було сформоване тестове середовище mininet, в якому була здійснена симуляція трафіку та DDoS/DoS атак на контролер. За різних конфігурацій були отримані різні результати. В кращому випадку відбувається визначення атаки без помилок за короткий проміжок часу, відправляється нотифікація адміністратору та додається потокове правило в SDN, що забороняє трафік від хоста, що атакує. В проекті використовується власний дата сет та натренована som мережа, що дозволяє ефективно виявляти атаки на відмову в SDN мережах. Підхід з використанням блокчейн технологій дозволяє підвищити рівень безпеки в мережі та забезпечити надійну систему виявлення атак на відмову.

За допомогою графічного інтерфейсу на брандмауері можна зручно переглядати всі зміни в мережі, зокрема зміни в блоках блокчейну, результати детекції атак, статуси вузлів мережі та інші важливі події. Інтерфейс дозволяє швидко і зручно відстежувати поточний стан мережі та здійснювати необхідні корективи. Зокрема, можна відслідковувати виявлення атак, переглядати історію атак, вивчати динаміку змін у мережі, а також робити різноманітні запити до блокчейну. Це допомагає адміністраторам мережі швидко реагувати на зміни в мережі та виявляти аномальну активність.

Описане програмне забезпечення забезпечує високий рівень безпеки та можливість захисту від різноманітних DDoS/DoS атак. Щоб брати участь у мережі, необхідно пройти реєстрацію в брандмауері, що дозволяє отримати sgx enclave, приватний та публічний ключі. Далі, після підключення до мережі, клієнт отримує список перемикачів для моніторингу. Зібрані дані аналізуються за допомогою нейронної мережі і результат зберігається в блокчейні, що забезпечує стійкість та надійність збереження інформації. Завдяки графічному інтерфейсу, користувачі можуть зручно переглядати всі зміни мережі і результати аналізу. Це дозволяє забезпечити високий рівень безпеки мережі та швидко реагувати на можливі атаки.

ВИСНОВКИ

В даній магістерській дисертації був проведений аналіз видів мереж та їх актуальності та поширеності. В результаті аналізу було з'ясовано, що мережі SDN є однією з найбільш перспективних та швидко розвиваючих технологій у галузі мережевих технологій.

Далі була проведена робота з визначення видів атак на SDN мережі та їхнього впливу на систему. Було досліджено різноманітні атаки, які можуть бути спрямовані на контролер, перемикачі та трафік в мережі. Особлива увага була приділена захисту SDN мереж від DDoS/DoS атак. Для цього було запропоновано використання блокчейн технології та SOM нейронної мережі. Було обґрунтовано використання цих технологій для підвищення рівня безпеки системи та забезпечення більш ефективного та швидкого аналізу даних.

Було проведено аналіз структури блокчейну, його переваг та недоліків, визначений алгоритм консенсусу, визначені криптографічні вимоги та досліджені обмеження систем, які спроможні працювати з SGX enclave, який є важливим компонентом блокчейну для забезпечення приватності та безпеки даних. Структура блокчейну була проаналізована з точки зору його основних складових: блоків, транзакцій, хешів та підписів. Основні переваги блокчейну, такі як децентралізація, безпека, надійність та прозорість були проаналізовані з точки зору їх впливу на систему.

Був проведений аналіз алгоритмів консенсусу в блокчейні з метою вибору найбільш підходящого для застосування в системах, що працюють з виявленням атак. В результаті аналізу було визначено, що алгоритм Proof of Elapsed Time (PoET) є найбільш підходящим для використання у системах з виявленням атак через його високу швидкодію та низький рівень використання ресурсів. Це особливо важливо для систем, що працюють з виявленням атак, оскільки вони повинні бути здатні реагувати на атаки якнайшвидше.

Отже, результати дослідження свідчать про те, що застосування SDN мереж для побудови мережевих інфраструктур є актуальним та перспективним напрямком розвитку технологій мережевого забезпечення. Запропоновані методи захисту SDN мереж від DDoS/DoS атак з використанням блокчейн технології та SOM нейронної мережі є ефективними та можуть забезпечити більш високий рівень безпеки та ефективності системи. Продовженням дослідження може бути розробка і впровадження в практику таких методів захисту SDN мереж.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lali, M.I., Mustafa, R.U., Ahsan, F., Nawaz, M.S. and Aslam, W., 2017. Performance evaluation of software-defined networking vs. traditional networks. The Nucleus, 54(1), pp.6-22.
2. Top Providers of SDN Services - https://www.cambridgewireless.co.uk/news/2020/jan/28/top-providers-sdn-services/
3. Pattaranantakul, M., He, R., Song, Q., Zhang, Z. and Meddahi, A., 2018. NFV security survey: From use case-driven threat analysis to state-of-the-art countermeasures. IEEE Communications Surveys & Tutorials, 20(4), pp.3330-3368
4. Scott-Hayward, S., O'Callaghan, G. and Sezer, S., 2013, November. SDN security: A survey. In 2013 IEEE SDN For Future Networks and Services (SDN4FNS) (pp. 1-7). IEEE.
5. Li, D., Wang, S., Zhu, K. and Xia, S., 2017. A survey of network update in SDN. Frontiers of computer science, 11(1), pp.4-12.
6. Herrera, J.G. and Botero, J.F., 2016. Resource allocation in NFV: A comprehensive survey. IEEE Transactions on Network and Service Management, 13(3), pp.518-532.
7. Е.В. Дуравкин, Е.Б. Ткачева, Иссам Саад. АРХИТЕКТУРА SDN. АНАЛИЗ ОСНОВНЫХ ПРОБЛЕМ НА ПУТИ РАЗВИТИЯ
8. Lara, A., Kolasani, A. and Ramamurthy, B., 2013. Network innovation using OpenFlow: A survey. IEEE communications surveys & tutorials, 16(1), pp.493-512.
9. Namal, S., Ahmad, I., Gurtov, A. and Ylianttila, M., 2013, November. Enabling secure mobility with OpenFlow. In 2013 IEEE SDN for Future Networks and Services (SDN4FNS) (pp. 1-5). IEEE.

10. Coughlin, M., 2014. A survey of SDN security research. University of Colorado Boulder.
11. Software-driven networks statement [Електронний ресурс] / S. Hartman, M. Wasserman, D. Zhang - Режим доступу: https://tools.ietf.org/html/drafthartman-sdnsec-requirements-00]
12. Shin, S., Yegneswaran, V., Porras, P. and Gu, G., 2013, November. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (pp. 413-424).
13. Ahmad, I., Namal, S., Ylianttila, M. and Gurtov, A., 2015. Security in software defined networks: A survey. IEEE Communications Surveys & Tutorials, 17(4), pp.2317-2346.
14. Fatima, K., Zahoor, K. and Zakaria Bawany, N., 2021, April. SDN Control Plane Security: Attacks and Mitigation Techniques. In Proceedings of the 4th International Conference on Networking, Information Systems & Security (pp. 1-6).
15. Alhaj, A.N. and Dutta, N., 2022. Analysis of security attacks in SDN network: A comprehensive survey. Contemporary Issues in Communication, Cloud and Big Data Analytics: Proceedings of CCB 2020, pp.27-37.
16. Merouane, M., 2017. An approach for detecting and preventing DDoS attacks in campus. Automatic Control and Computer Sciences, 51(1), pp.13-23.
17. Li, H., Wei, F. and Hu, H., 2019, March. Enabling dynamic network access control with anomaly-based IDS and SDN. In Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization (pp. 13-16).
18. Fonseca, P., Bennesby, R., Mota, E. and Passito, A., 2012, April. A replication component for resilient OpenFlow-based networking. In 2012 IEEE Network operations and management symposium (pp. 933-939). IEEE.

19. Xu, Y. and Liu, Y., 2016, April. DDoS attack detection under SDN context. In IEEE INFOCOM 2016-the 35th annual IEEE international conference on computer communications (pp. 1-9). IEEE.
20. Singhal, B., Dhameja, G. and Panda, P.S., 2018. Beginning Blockchain: A Beginner's guide to building Blockchain solutions. Apress.
21. De Angelis, S., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A. and Sassone, V., 2018. PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain.
22. Pal, A. and Kant, K., 2021, June. DC-PoET: Proof-of-Elapsed-Time Consensus with Distributed Coordination for Blockchain Networks. In 2021 IFIP Networking Conference (IFIP Networking) (pp. 1-9). IEEE.
23. Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdorf, H. and Capkun, S., 2016, October. On the security and performance of proof of work blockchains. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security (pp. 3-16).
24. Nguyen, C.T., Hoang, D.T., Nguyen, D.N., Niyato, D., Nguyen, H.T. and Dutkiewicz, E., 2019. Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities. IEEE Access, 7, pp.85727-85745.
25. Oyinloye, D.P., Teh, J.S., Jamil, N. and Alawida, M., 2021. Blockchain Consensus: An Overview of Alternative Protocols. Symmetry, 13(8), p.1363.
26. Intel SGX supportable [Електронний ресурс] / Intel Corporation - Режим доступу: - https://ark.intel.com/content/www/us/en/ark/search/featurefilter.html?productType=873&2_SoftwareGuardExtensions=Yes%20with%20Intel%C2%AE%20ME
27. Gupta, S.S., 2017. Blockchain. IBM Onlone (http://www. IBM. COM).

28. Wang, X. and Yu, H., 2005, May. How to break MD5 and other hash functions. In Annual international conference on the theory and applications of cryptographic techniques (pp. 19-35). Springer, Berlin, Heidelberg.
29. E., 2013. OpenStack: OVS Deep Dive. [Електронний ресурс] / Pettit, J. and Lopez - Режим доступу: https://man7.org/linux/man-pages/man8/ovs-ofctl.8.html
30. Morales, L.V., Murillo, A.F. and Rueda, S.J., 2015, September. Extending the floodlight controller. In 2015 IEEE 14th International Symposium on Network Computing and Applications (pp. 126-133). IEEE.
31. Bailey, J. and Stuart, S., 2016. Faucet: Deploying SDN in the enterprise. Communications of the ACM, 60(1), pp.45-49.
32. Erickson, D., 2013, August. The beacon openflow controller. In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (pp. 13-18).
33. Nagpal, B., Sharma, P., Chauhan, N. and Panesar, A., 2015, March. DDoS tools: Classification, analysis and comparison. In 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 342-346). IEEE.

ДОДАТОК А

Лістинг програми

Спосіб захисту програмно-конфігурованої мережі від атак

```
import { Dump, MininetDumpReceiver } from '../common/mininet.dump.receiver';
import { BlockchainService } from '../common/blockchain';
import { Controller, Inject, Logger, OnModuleInit } from '@nestjs/common';
import { SomNetwork } from '../common/som.network';
import { Cron } from '@nestjs/schedule';

export type Detection = {
  attackDetected: boolean;
  receiverIp: string;
  senderIp: string;
  switchId: string;
};

@Controller()
export class JobController implements OnModuleInit {
  private readonly logger = new Logger(JobController.name);

  constructor(
    private readonly blockChainService: BlockchainService,
    private readonly logReceiver: MininetDumpReceiver,
    private readonly somNetwork: SomNetwork,
  ) {}

  async onModuleInit() {
    await this.blockChainService.blockChainRegisterFirstBlock();
  }

  @Cron('*/10 * * * * *')
  async handleCron() {
    await this.blockChainService.registerAsNextCommittee();
  }

  // every 30 seconds
  @Cron('30 * * * * *')
  async generateTransaction() {
    const dump = await this.logReceiver.dump();
    const results = this.somNetwork.findDDoS(dump);
    for (const result of results) {
      await this.blockChainService.addTransaction({
        result: result.attackDetected,
        receiverIp: result.receiverIp,
        senderIp: result.senderIp,
        switchIp: result.switchId,
      });
    }
    this.logger.debug('Generated transactions: ' + results.length);
  }

  // Each minute
  @Cron('* * * * *')
  async addBlock() {
    if (!(await this.blockChainService.isCommittee())) {
      return this.logger.log('Not committee');
```

```
    }
    const block = await this.blockChainService.revealNextBlock();
    await this.blockChainService.addBlock(block);
    this.logger.debug('Block is added');
  }
}

import { Logger, Module } from '@nestjs/common';
import { ScheduleModule } from '@nestjs/schedule';
import { JobController } from './job.controller';
import { BlockchainService } from '../common/blockchain';
import { MininetDumpReceiver } from '../common/mininet.dump.receiver';
import { NodeSSH } from 'node-ssh';
import { som } from './test-neuro';
import { SomNetwork } from '../common/som.network';

const logger = new Logger();

@Module({
  imports: [ScheduleModule.forRoot()],
  controllers: [JobController],
  providers: [
    BlockchainService,
    MininetDumpReceiver,
    {
      provide: NodeSSH,
      useFactory: async () => {
        const nodeSSH = new NodeSSH();
        const MININET = '192.168.0.2';
        logger.debug('Connecting to ' + MININET);

        await nodeSSH
          .connect({
            host: MININET,
            username: 'test',
            password: 'test',
          })
          .catch((e) => {
            logger.error(e, 'Failed to connect to ' + MININET);
          });
        return nodeSSH;
      },
    },
    {
      provide: SomNetwork,
      useFactory: async () => {
        const SOM = require('ml-som');
        return new SomNetwork(SOM.load(som));
      },
    },
  ],
})
```

```
export class JobModule {}

import { NodeSSH } from 'node-ssh';
import { Injectable, Logger } from '@nestjs/common';

export type DumpItem = {
  n_packets: number;
  n_bytes: number;
  duration: number;
  src: string;
  dst: string;
  actions: string;
  switchId: string;
};
export type Dump = DumpItem[];

/// cookie=0x20000001000000, duration=100.287s, table=0, n_packets=97, n_bytes=9506,
idle_timeout=5, priority=1,ip,in_port="s1-
eth1",dl_src=9a:d6:39:b4:e8:2f,dl_dst=9a:1c:70:24:34:dc,nw_src=10.0.0.1,nw_dst=10.0.0.3
actions=output:"s1-eth3"
//  cookie=0x20000002000000, duration=100.285s, table=0, n_packets=98, n_bytes=9604,
idle_timeout=5, priority=1,ip,in_port="s1-
eth3",dl_src=9a:1c:70:24:34:dc,dl_dst=9a:d6:39:b4:e8:2f,nw_src=10.0.0.3,nw_dst=10.0.0.1
actions=output:"s1-eth1"

@Injectable()
export class MininetDumpReceiver {
  private readonly logger = new Logger();
  private readonly dpctlPort = 6634;
  private readonly switchAmount = 8;

  constructor(private readonly sshClient: NodeSSH) {}

  private async parseDump(
    results: {
      switchId: string;
      result: string;
    }[],
  ): Promise<Dump> {
    const dump: Dump = [];
    for (const data of results) {
      const switchRegexp =
        /\s+cookie.*?duration=([0-9]+\.[0-9]+).*?n_packets=([0-9]+).*?n_bytes=([0-
9]+).*?nw_src=(.+?),nw_dst=(.+?)\sactions=(.+?)(\n|$)/g;

      let parsedResult;
      while ((parsedResult = switchRegexp.exec(data.result))) {
        dump.push({
          switchId: data.switchId,
          duration: parseFloat(parsedResult[1]),
          n_packets: parseInt(parsedResult[2], 10),
          n_bytes: parseInt(parsedResult[3], 10),
          src: parsedResult[4],
```

```
        dst: parsedResult[5],
        actions: parsedResult[6],
      });
    }
  }
  return dump;
 }

 async dump(): Promise<Dump> {
  const resultArray = [];

  let switchAmount = this.switchAmount;
  do {
   const port = this.dpctlPort + switchAmount;
   const result = await this.sshClient.execCommand(
    `ovs-ofctl dump-flows -O OpenFlow13 tcp:127.0.0.1:${port}`,
   );
   this.logger.debug({ result }, 'Result from switch');
   if (result.code === 0) {
    resultArray.push({ switchId: switchAmount, result: result.stdout });
   }
  } while (--switchAmount > 0);

  const parsedDump = await this.parseDump(resultArray);
  // this.logger.debug({ parsedDump }, 'Received dump from mininet');
  return parsedDump;
 }
}

import { InjectRedis } from '@liaoliaots/nestjs-redis';
import { ConfigService } from '@nestjs/config';
import * as jsonwebtoken from 'jsonwebtoken';
import { Logger } from '@nestjs/common';
import * as crypto from 'node:crypto';
import * as eccrypto from 'eccrypto';
import { Redis } from 'ioredis';

const pr = eccrypto.generatePrivate();
console.log(pr.toString('hex'));
console.log(eccrypto.getPublic(pr).toString('hex'));

export class Transaction {
 switchIp: string;
 senderIp: string;
 receiverIp: string;
 result: boolean;
 hash: string;
 digitalSignature: string;
 publicKey: string;
}

export class Block {
 merkleHash: string;
```

```
  previousHash: string;
  timestamp: number;
  sgxCertificate: string;
  blockSignature: string;

  constructor(readonly transactions: Transaction[] = []) {}

  async finalize(privateKey: string, previousHash: string): Promise<Block> {
    const payload: Partial<Block> = {
      timestamp: Date.now(),
      previousHash,
      sgxCertificate: jsonwebtoken.sign({ previousHash }, privateKey),
      transactions: this.transactions,
    };
    const messageBuffer = crypto
      .createHash('sha256')
      .update(JSON.stringify(payload))
      .digest();
    payload.merkleHash = messageBuffer.toString('base64');
    payload.blockSignature = (
      await eccrypto.sign(Buffer.from(privateKey, 'hex'), messageBuffer)
    ).toString('base64');

    return payload as Block;
  }
}

export const BLOCKCHAIN_KEY = 'blockchain_key';
const TRANSACTION_KEY = 'transaction_channel';
const NEXT_COMMITTEE = 'next_committee';
const ROOT_BLOCK_LOCK = 'root_block_lock';

export class BlockchainService {
  private readonly logger = new Logger(BlockchainService.name);
  private readonly publicKey: string;
  private readonly privateKey: string;

  constructor(
    @InjectRedis() private readonly redis: Redis,
    private readonly configService: ConfigService,
  ) {
    this.publicKey = this.configService.getOrThrow('PUBLIC_KEY');
    this.privateKey = this.configService.getOrThrow('PRIVATE_KEY');
  }

  async registerAsNextCommittee() {
    const isSet = await this.redis.setnx(NEXT_COMMITTEE, this.publicKey);
    if (isSet) {
      await this.redis.expire(NEXT_COMMITTEE, 60);
    }
  }

  async addBlock(block: Block) {
```

```
  return this.redis.rpush(BLOCKCHAIN_KEY, JSON.stringify(block));
}

async addTransaction(
  transaction: Omit<Transaction, 'digitalSignature' | 'hash' | 'publicKey'>,
) {
  const payload = {
    ...transaction,
    publicKey: this.publicKey,
  };
  const hash = crypto
    .createHash('sha256')
    .update(JSON.stringify(payload))
    .digest();

  const payloadWithHash: Partial<Transaction> = {
    ...payload,
    hash: hash.toString('base64'),
  };
  payloadWithHash.digitalSignature = (
    await eccrypto.sign(Buffer.from(this.privateKey, 'hex'), hash)
  ).toString('base64');

  return this.redis.lpush(
    TRANSACTION_KEY,
    JSON.stringify(payloadWithHash as Transaction),
  );
}

async revealNextBlock() {
  let transactionsLen = await this.redis.llen(TRANSACTION_KEY);
  let transactions = await this.redis.lrange(
    TRANSACTION_KEY,
    0,
    Number.MAX_SAFE_INTEGER,
  );

  while (transactionsLen !== transactions.length) {
    this.logger.warn('Waiting for transactions');
    transactionsLen = await this.redis.llen(TRANSACTION_KEY);
    transactions = await this.redis.lrange(
      TRANSACTION_KEY,
      0,
      Number.MAX_SAFE_INTEGER,
    );
  }
  this.redis.del(TRANSACTION_KEY);
  const previousBlock: Block = JSON.parse(
    (await this.redis.lrange(BLOCKCHAIN_KEY, 0, 1))[0],
  );
  return new Block(transactions.map((val) => JSON.parse(val))).finalize(
    this.privateKey,
    previousBlock.merkleHash,
```

```
    );
  }

  async isCommittee() {
    return (await this.redis.get(NEXT_COMMITTEE)) === this.publicKey;
  }

  async blockChainRegisterFirstBlock() {
    const isSet = await this.redis.setnx(ROOT_BLOCK_LOCK, 'true');
    if (!isSet) return this.logger.warn('Root block already registered');

    await this.addBlock(await new Block([]).finalize(this.privateKey, 'none'));
  }
}

#include <napi.h>
#include <sgx_urts.h>
#include <sgx_tcrypto.h>

Napi::Value sgxReadRand(const Napi::CallbackInfo& info) {
  Napi::Env env = info.Env();

  uint8_t random_number[4];
  sgx_status_t status = sgx_read_rand(random_number, 4);

  if (status != SGX_SUCCESS) {
    Napi::TypeError::New(env, "Failed to generate random number").ThrowAsJavaScriptException();
    return env.Null();
  }

  uint32_t result = (random_number[0] << 24) + (random_number[1] << 16) + (random_number[2]
<< 8) + random_number[3];
  return Napi::Number::New(env, result);
}

Napi::Object Init(Napi::Env env, Napi::Object exports) {
  exports.Set(Napi::String::New(env, "sgxReadRand"), Napi::Function::New(env, sgxReadRand));
  return exports;
}

NODE_API_MODULE(sgx_random, Init)


import createGraph, { Graph } from 'ngraph.graph';
import path from 'ngraph.path';
import {
  Document,
  Packer,
  Paragraph,
  Table,
  TableCell,
  TableRow,
  WidthType,
```

```
} from 'docx';
import * as fs from 'node:fs';

export type Topology = {
  [key: string]: string[];
};
export interface TopologyStats {
  topologyName: string;
  scale: number;
  time: number;
  nodes: number;
  topologyMetrics: TopologyMetrics;
}

interface TopologyMetrics {
  D: number;
  S: number;
  D_avg: number;
  C: number;
  T: number;
}

function topologyToGraph(topology: Topology): Graph {
  const graph = createGraph();

  for (const [node, links] of Object.entries(topology)) {
    graph.addNode(node);
    for (const link of links) {
      if (!graph.hasLink(node, link)) {
        graph.addLink(node, link);
      }
    }
  }

  return graph;
}

function calculateTopologyMetrics(graph: Graph): TopologyMetrics {
  const pathFinder = path.aStar(graph);
  const nodeCount = graph.getNodesCount();
  const nodeIds = [];
  graph.forEachNode((node) => {
    nodeIds.push(node.id);
  });

  let maxDistance = 0;
  let sumDistances = 0;

  for (let i = 0; i < nodeCount; i++) {
    for (let j = i + 1; j < nodeCount; j++) {
      const distance = pathFinder.find(nodeIds[i], nodeIds[j]).length - 1;
      sumDistances += distance;
```

```
      if (distance > maxDistance) {
        maxDistance = distance;
      }
    }
  }

  const D = maxDistance;
  const S = Math.max(
    ...nodeIds.map((nodeId) => {
      const links = graph.getLinks(nodeId) || [];
      const filteredLinks = [...links].filter((val) => val.fromId === nodeId);
      return filteredLinks.length || 0;
    }),
  );
  const D_avg = sumDistances / (nodeCount * (nodeCount - 1));
  const C = D * nodeCount * S;
  const T = (2 * D_avg) / S;

  return {
    D,
    S,
    D_avg,
    C,
    T,
  };
}

function generateTopology(D: number, S: number): Graph {
  if (D <= 0 || S <= 0) {
    throw new Error('D and S must be positive integers.');
  }

  const graph = createGraph();

  // Initialize node counter and add the first node
  let nodeCounter = 1;
  graph.addNode(`Switch${nodeCounter}`);

  // Initialize maxDistance and create a path finder instance
  let maxDistance = 0;
  const pathFinder = path.aStar(graph);

  // Loop until the maximum distance between any two nodes is equal to D
  while (maxDistance < D) {
    // Increment the node counter and add a new node
    nodeCounter++;
    const currentNode = `Switch${nodeCounter}`;
    graph.addNode(currentNode);

    // Try to connect the new node to an existing node
    let addedLink = false;

    for (
```

```
      let parentNodeIndex = 1;
      parentNodeIndex < nodeCounter;
      parentNodeIndex++
    ) {
      const parentNode = `Switch${parentNodeIndex}`;

      // Check if the existing node has room for more connections
      if ((graph.getLinks(parentNode)?.size || 0) < S) {
        // Calculate the distance between the existing node and the new node
        const currentDistance =
          pathFinder.find(parentNode, currentNode).length - 1;

        // Add a link between the nodes if the distance is less than D
        if (currentDistance < D) {
          graph.addLink(parentNode, currentNode);
          addedLink = true;
          break;
        }
      }
    }

    // If the new node could not be connected, remove it and stop adding nodes
    if (!addedLink) {
      graph.removeNode(currentNode);
      break;
    }

    // Update the maximum distance between any two nodes in the graph
    maxDistance = 0;

    for (let i = 1; i <= nodeCounter; i++) {
      for (let j = i + 1; j <= nodeCounter; j++) {
        const distance = pathFinder.find(`Switch${i}`, `Switch${j}`).length - 1;
        maxDistance = Math.max(maxDistance, distance);
      }
    }
  }

  return graph;
}

export function graphToTopology(graph: Graph): Topology {
  const topology: Topology = {};

  graph.forEachNode((node) => {
    topology[node.id] = [];
    graph.forEachLinkedNode(
      node.id,
      (linkedNode) => {
        topology[node.id].push(linkedNode.id.toString());
      },
      false,
    );
```

```
  });

  return topology;
}

function analyzeTimeBasedOnScaling(topo: Graph, scale: number): number {
  const pathFinder = path.aStar(topo);
  const switches = [];
  topo.forEachNode((node) => {
    switches.push(node.id);
  });
  const n = switches.length;

  if (scale > n) {
    throw new Error(
      'Scale must be less than or equal to the number of switches.',
    );
  }

  // Assign hosts to unique switches randomly
  const hosts: string[] = [];
  const unassignedSwitches = [...switches];

  for (let i = 0; i < scale; i++) {
    const randomIndex = Math.floor(Math.random() * unassignedSwitches.length);
    hosts.push(unassignedSwitches[randomIndex]);
    unassignedSwitches.splice(randomIndex, 1);
  }

  let totalTime = 0;

  // Process requests for each switch
  for (const targetSwitch of switches) {
    let minTimeForSwitch = Infinity;

    // Calculate the time needed for each host to process the request
    for (const host of hosts) {
      const shortestPath = pathFinder.find(host, targetSwitch);
      const delay = shortestPath.length - 1; // Number of hops - 1
      const processingTime = 1; // 1T for processing

      const timeForHost = delay + processingTime;
      minTimeForSwitch = Math.min(minTimeForSwitch, timeForHost);
    }

    totalTime += minTimeForSwitch;
  }

  return totalTime;
}

const createWordTable = async (data: string[][]) => {
  const tableRows = data.map((row) => {
```

```
    const tableCells = row.map((cell) => {
      return new TableCell({
        children: [new Paragraph(cell)],
        columnSpan: 1,
      });
    });
    return new TableRow({ children: tableCells });
  });

  const table = new Table({
    rows: tableRows,
    width: {
      size: 100,
      type: WidthType.AUTO,
    },
  });

  const buffer = await Packer.toBuffer(
    new Document({
      sections: [
        {
          children: [table],
        },
      ],
    }),
  );
  fs.writeFileSync('Table.docx', buffer);
};

export function generateStatisticsTable(topologies: {
  [key: string]: {
    topology: Topology;
    topologyGraph: Graph;
    topologyMetrics: TopologyMetrics;
  };
}): void {
  const stats: TopologyStats[] = [];

  // Iterate through the topologies and scales to calculate the time for each combination
  for (const [
    topologyName,
    { topology, topologyMetrics, topologyGraph },
  ] of Object.entries(topologies)) {
    for (let scale = 1; scale <= Object.keys(topology).length; scale++) {
      const time = analyzeTimeBasedOnScaling(topologyGraph, scale);
      stats.push({
        topologyName,
        scale,
        time,
        topologyMetrics,
        nodes: Object.keys(topology).length,
      });
    }
```

```
  }

  // Output the statistics table
  console.log(
    'Назва топології       | Масштабування | Час виконання |   Dсер  | C(ціна) | T(Трафік) | Кількість вершин',
  );
  console.log(
    '----------------      |   -------     | -----         |  -----  |  ----   | --------- | ----------------|',
  );
  for (const stat of stats) {
    const metr = stat.topologyMetrics;
    const scale = stat.scale.toString().padEnd(13);
    const name = stat.topologyName.padEnd(21);
    const time = stat.time.toString().padEnd(13);
    const dAvg = metr.D_avg.toFixed(2).toString().padEnd(7);
    const c = metr.C.toString().padEnd(5).padEnd(7);
    const t = metr.T.toFixed(2).padEnd(9);
    const nodes = stat.nodes.toString();

    console.log(
      `${name} | ${scale} | ${time} | ${dAvg} | ${c} | ${t} | ${nodes}`,
    );
  }
  createWordTable([
    [
      'Назва топології',
      'Масштабування',
      'Час виконання',
      'Dсер',
      'C(ціна)',
      'T(Трафік)',
      'Кількість вершин',
    ],
    ...stats.map((stat) => {
      const metr = stat.topologyMetrics;
      return [
        stat.topologyName,
        stat.scale.toString(),
        stat.time.toString(),
        metr.D_avg.toFixed(2),
        metr.C.toString(),
        metr.T.toFixed(2),
        stat.nodes.toString(),
      ];
    }),
  ]);
}

const topoTriangle = {
  Switch1: ['Switch2', 'Switch3'],
  Switch2: ['Switch1', 'Switch3'],
  Switch3: ['Switch1', 'Switch2'],
```

```
};

const topoSquare = {
  Switch1: ['Switch2', 'Switch4'],
  Switch2: ['Switch1', 'Switch3'],
  Switch3: ['Switch2', 'Switch4'],
  Switch4: ['Switch1', 'Switch3'],
};

const triangleMetrics = calculateTopologyMetrics(topologyToGraph(topoTriangle));
const squareMetrics = calculateTopologyMetrics(topologyToGraph(topoSquare));
const topologies: Record<
  string,
  { topology: Topology; topologyMetrics: TopologyMetrics; topologyGraph: Graph }
> = {
  [`Трикутник(D=${triangleMetrics.D}, S=${triangleMetrics.S})`]: {
    topology: topoTriangle,
    topologyMetrics: triangleMetrics,
    topologyGraph: topologyToGraph(topoTriangle),
  },
  [`Квадрат(D=${squareMetrics.D}, S=${squareMetrics.S})`]: {
    topology: topoSquare,
    topologyMetrics: squareMetrics,
    topologyGraph: topologyToGraph(topoSquare),
  },
};

for (let D = 4; D < 7; D++) {
  for (let S = 5; S < 8; S++) {
    const graph = generateTopology(D, S);
    topologies[`Довільна(D=${D}, S=${S})`] = {
      topology: graphToTopology(graph),
      topologyMetrics: calculateTopologyMetrics(graph),
      topologyGraph: graph,
    };
  }
}

generateStatisticsTable(topologies);

#!/usr/bin/env python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():
```

```
net = Mininet( topo=None,
            build=False,
            ipBase='10.0.0.0/8')

info( '*** Adding controller\n' )
c0=net.addController(name='c0',
                  controller=RemoteController,
                  ip='192.168.0.3',
                  protocol='tcp',
                  port=6653)

c1=net.addController(name='c0',
                  controller=RemoteController,
                  ip='192.168.0.8',
                  protocol='tcp',
                  port=6653)

info( '*** Add switches\n')
s1 = net.addSwitch('s1', cls=OVSKernelSwitch, listenPort=6635)
s2 = net.addSwitch('s2', cls=OVSKernelSwitch, listenPort=6636)
s3 = net.addSwitch('s3', cls=OVSKernelSwitch, listenPort=6637)
s4 = net.addSwitch('s4', cls=OVSKernelSwitch, listenPort=6638)
s5 = net.addSwitch('s5', cls=OVSKernelSwitch, listenPort=6639)
s6 = net.addSwitch('s6', cls=OVSKernelSwitch, listenPort=6640)
s7 = net.addSwitch('s7', cls=OVSKernelSwitch, listenPort=6641)
s8 = net.addSwitch('s8', cls=OVSKernelSwitch, listenPort=6642)
s9 = net.addSwitch('s9', cls=OVSKernelSwitch, listenPort=6643)
s10 = net.addSwitch('s10', cls=OVSKernelSwitch, listenPort=6644)
s11 = net.addSwitch('s11', cls=OVSKernelSwitch, listenPort=6645)

info( '*** Add hosts\n')
h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
h5 = net.addHost('h5', cls=Host, ip='10.0.0.5', defaultRoute=None)
h6 = net.addHost('h6', cls=Host, ip='10.0.0.6', defaultRoute=None)
h7 = net.addHost('h7', cls=Host, ip='10.0.0.7', defaultRoute=None)
h8 = net.addHost('h8', cls=Host, ip='10.0.0.8', defaultRoute=None)

info( '*** Add links\n')
net.addLink(s11, s2)
net.addLink(s2, s1)
net.addLink(s6, s5)
net.addLink(s2, s3)
net.addLink(s1, h4)
net.addLink(s1, s3)
net.addLink(h3, s3)
net.addLink(s3, s4)
net.addLink(h1, s4)
net.addLink(h2, s4)
net.addLink(h5, s9)
net.addLink(h6, s8)
```

```
    net.addLink(s9, s7)
    net.addLink(s8, s7)
    net.addLink(h7, s7)
    net.addLink(s10, s7)
    net.addLink(s10, s3)
    net.addLink(s5, s7)
    net.addLink(s6, s7)
    net.addLink(s5, s11)
    net.addLink(h8, s6)

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s1').start([c0,c1])
    net.get('s2').start([c0,c1])
    net.get('s3').start([c0,c1])
    net.get('s4').start([c0,c1])
    net.get('s5').start([c1,c0])
    net.get('s6').start([c1,c0])
    net.get('s7').start([c1,c0])
    net.get('s8').start([c1,c0])
    net.get('s9').start([c1,c0])
    net.get('s10').start([c1,c0])
    net.get('s11').start([c1,c0])

    info( '*** Post configure switches and hosts\n')
    runTraffic(net)
    CLI(net)

    net.stop()


def runTraffic(net):
    # Get the IP addresses of the hosts in the network
    ip_addresses = [host.IP() for host in net.hosts]

    # Save the Bash script to a variable
    key = ' '.join('"{}"'.format(ip) for ip in ip_addresses)
    script = '''#!/bin/bash
    ip_addresses=({0})

    while true; do
        for ip in "${{ip_addresses[@]}}"; do
            ping -c 1 -p "61626364" "$ip" > /dev/null 2>&1
        done
        sleep 5
    done
    '''.format(key)
```

```
    print('Saving files')
    # Save the script to a file on each host
    for host in net.hosts:
        command = 'echo "{0}" > ping_ips.sh'.format(script)
        host.cmd(command)

    print('Permitting files')
    # Give the script execute permission on each host
    for host in net.hosts:
        host.cmd('chmod +x ping_ips.sh')

    print('Executing files')
    #Run the script on each host in the background without displaying the output

    #net.hosts[0].cmd('noecho sh ./ping_ips.sh')
    print('Started traffic')

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

```
package net.floodlightcontroller.custom;

import java.util.*;

import net.floodlightcontroller.core.FloodlightContext;
import net.floodlightcontroller.core.IFloodlightProviderService;
import net.floodlightcontroller.core.IOFMessageListener;
import net.floodlightcontroller.core.IOFSwitch;
import net.floodlightcontroller.core.module.FloodlightModuleContext;
import net.floodlightcontroller.core.module.FloodlightModuleException;
import net.floodlightcontroller.core.module.IFloodlightModule;
import net.floodlightcontroller.core.module.IFloodlightService;
import net.floodlightcontroller.devicemanager.IDevice;
import net.floodlightcontroller.devicemanager.IDeviceService;
import net.floodlightcontroller.packet.Ethernet;
import org.projectfloodlight.openflow.protocol.*;
import net.floodlightcontroller.packet.IPv4;
import net.floodlightcontroller.packet.ARP;

import org.projectfloodlight.openflow.protocol.instruction.OFInstruction;
import org.projectfloodlight.openflow.protocol.action.OFAction;
import org.projectfloodlight.openflow.protocol.match.Match;
import org.projectfloodlight.openflow.protocol.match.MatchField;
import org.projectfloodlight.openflow.types.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


public class CustomController implements IOFMessageListener, IFloodlightModule {
    protected IFloodlightProviderService floodlightProvider;
    protected IDeviceService deviceService;
    protected static Logger logger;
```

```
    @Override
    public String getName() {
        return CustomController.class.getSimpleName();
    }

    @Override
    public boolean isCallbackOrderingPrereq(OFType type, String name) {
        return false;
    }

    @Override
    public boolean isCallbackOrderingPostreq(OFType type, String name) {
        return false;
    }


    @Override
    public Command receive(IOFSwitch sw, OFMessage msg, FloodlightContext cntx) {
        if (msg.getType() == OFType.PACKET_IN) {
            OFPacketIn pi = (OFPacketIn) msg;
            Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
IFloodlightProviderService.CONTEXT_PI_PAYLOAD);

            if (eth.getEtherType() == EthType.ARP && ((OFPacketIn) msg).getReason() ==
OFPacketInReason.NO_MATCH) {
                ARP arp = (ARP) eth.getPayload();
                IPv4Address dstIP = arp.getTargetProtocolAddress();
                IPv4Address srcIP = arp.getSenderProtocolAddress();

                // Implement your logic here to check if the destination IP exists.
                boolean ipExists = checkIfIPExists(dstIP);

                if (!ipExists) {
                    this.addDropFlow(sw, dstIP, srcIP);
                    return Command.STOP;
                }
            }
        }
        return Command.CONTINUE;
    }

    private boolean checkIfIPExists(IPv4Address dstIP) {
        return knownIPs.contains(dstIP);
    }

    private void addDropFlow(IOFSwitch sw, IPv4Address dstIP, IPv4Address srcIP) {
        OFFactory factory = sw.getOFFactory();
        OFVersion ofVersion = factory.getVersion();

        // Create a match for the destination IP address
        Match match = factory.buildMatch()
                .setExact(MatchField.ETH_TYPE, EthType.IPv4)
```

```
            .setExact(MatchField.IPV4_DST, dstIP)
            .setExact(MatchField.IPV4_SRC, srcIP)
            .build();

    // Create an empty list of actions to drop the packet
    List<OFAction> actions = new ArrayList<>();

    // Create an Apply-Actions instruction with the empty actions list
    OFInstruction applyActions = factory.instructions().applyActions(actions);

    // Create a list of instructions and add the Apply-Actions instruction
    List<OFInstruction> instructions = new ArrayList<>();
    instructions.add(applyActions);

    // Create a flow modification message and set its properties
    OFFlowAdd flowAdd = factory.buildFlowAdd()
            .setPriority(32768) // Set an appropriate priority for the flow entry
            .setMatch(match)
            .setInstructions(instructions)
            .setBufferId(OFBufferId.NO_BUFFER)
            .setIdleTimeout(60) // Set an idle timeout, e.g., 60 seconds
            .setHardTimeout(0) // Set no hard timeout
            .setTableId(TableId.of(0)) // Set the appropriate table ID, usually 0 for the first table
            .setOutPort(OFPort.ANY)
            .setOutGroup(OFGroup.ANY)
            .setFlags(EnumSet.of(OFFlowModFlags.SEND_FLOW_REM))
            .build();

    // Write the flow modification message to the switch
    sw.write(flowAdd);
}

@Override
public Collection<Class<? extends IFloodlightService>> getModuleServices() {
    return null;
}

@Override
public Map<Class<? extends IFloodlightService>, IFloodlightService> getServiceImpls() {
    return null;
}

@Override
public Collection<Class<? extends IFloodlightService>> getModuleDependencies() {
    Collection<Class<? extends IFloodlightService>> l =
            new ArrayList<Class<? extends IFloodlightService>>();
    l.add(IFloodlightProviderService.class);
    return l;
}

@Override
public void init(FloodlightModuleContext context) throws FloodlightModuleException {
    floodlightProvider = context.getServiceImpl(IFloodlightProviderService.class);
```

```
        deviceService = context.getServiceImpl(IDeviceService.class);
        logger = LoggerFactory.getLogger(CustomController.class);
    }

    @Override
    public void startUp(FloodlightModuleContext context) throws FloodlightModuleException {
        floodlightProvider.addOFMessageListener(OFType.PACKET_IN, this);
    }
}
```