МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри, канд. техн. наук,
доцент _____Є. О. Давиденко
«___»____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ЗАСТОСУНОК МОНІТОРИНГУ ТРАФІКУ ТА ВИЗНАЧЕННЯ DDoS-АТАК ЗА ДОПОМОГОЮ eBPF

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 409.21910902

Студент _____Д. Д. Бойко
«___»_____2023 р.

Керівник завідувач кафедри ІПЗ, канд. техн. наук, доцент _____Є. О. Давиденко
«___»_____2023 р.

Консультант канд. техн. наук, доцент _____А. О. Алексєєва
«___»_____2023 р.

Миколаїв 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ
Зав. кафедри
_____ Є. О. Давиденко
« __ » _____ 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи бакалавра

Видано студенту групи 409 факультету комп'ютерних наук
Бойку Данилу Дмитровичу .
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи**:**

Застосунок моніторингу трафіку та визначення DDoS-атак за допомогою eBPF

Затверджена наказом по ЧНУ ім. П. Могили від «17» березня 2023 р. № 60

2. Строк представлення кваліфікаційної роботи: «26» червня 2023 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом є застосунок моніторингу трафіку, який буде відстежувати активність в локальній інтернет-мережі, підтримувати можливість визначення DDoS-атак та надавати користувачу інфографіку для аналізу та прийняття майбутніх рішень.

4. Перелік питань, що підлягають розробці:

- аналіз предметної області та існуючих аналогів;
- виявлення основних проблем та продумування можливих рішень;
- визначення вимог до необхідного функціоналу;
- проєктування та розробка програмного забезпечення;
- проведення тестування роботи програмного забезпечення;
- здійснення аналізу результатів розробки.

5. Перелік графічних матеріалів:

Презентація.

6. Завдання до спеціальної частини

Охорона праці в робочих приміщеннях фахівців з інформаційних технологій.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алексєєва А. О.	Кафедра екології	Спеціальна частина з охорони праці

Керівник роботи канд. техн. наук, доцент Давиденко Євген Олександрович

(посада, прізвище, ім'я, по батькові)

(підпис)

Завдання прийнято до виконання

Бойко Данило Дмитрович

(прізвище, ім'я, по батькові студента)

(підпис)

Дата видачі завдання «30» січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема кваліфікаційної роботи**:**

Застосунок моніторингу трафіку та визначення DDoS-атак за допомогою eBPF__

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРБ	16.03.2023 р.	17.03.2023 р.	Виконано
2.	Огляд літератури за темою роботи	20.03.2023 р.	22.03.2023 р.	Виконано
3.	Складання календарного плану КРБ	23.03.2023 р.	24.03.2023 р.	Виконано
4.	Аналіз предметної області	27.03.2023 р.	27.03.2023 р.	Виконано
5.	Розробка проєктних рішень	28.03.2023 р.	31.03.2023 р.	Виконано
6.	Моделювання та конструювання ПЗ	03.04.2023 р.	07.04.2023 р.	Виконано
7.	Кодування, тестування розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	10.04.2023 р.	28.04.2023 р.	Виконано
8.	Розробка спеціальної частини з охорони праці	08.05.2023 р.	12.05.2023 р.	Виконано
9.	Оформлення КРБ та презентації	15.05.2023 р.	19.05.2023 р.	Виконано
10.	Відгук керівника КРБ	29.05.2023 р.	02.06.2023 р.	Виконано
11.	Попередній захист	06.06.2023 р.	07.06.2023 р.	Виконано
12.	Завершення оформлення КРБ та презентації	08.06.2023 р.	09.06.2023 р.	Виконано
13.	Рецензування	12.06.2023 р.	19.06.2023 р.	Виконано
14.	Захист кваліфікаційної роботи	26.06.2023 р.	26.06.2023 р.	Виконано

Розробив студент Бойко Данило Дмитрович _____
(прізвище, ім'я, по батькові) *(підпис)*

« ___ » _____ 202__ р.

Керівник роботи канд. техн. наук, доцент Давиденко Є. О. _____
(посада, прізвище, ім'я, по батькові) *(підпис)*

« ___ » _____ 202__ р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Застосунок моніторингу трафіку та визначення DDoS-атак за допомогою eBPF»

Студент 409 гр.: Бойко Данило Дмитрович

Керівник: канд. техн. наук, доцент Давиденко Євген Олександрович

Робота присвячена розробці програмного забезпечення для моніторингу трафіку в локальній інтернет-мережі та автоматизованого визначення DDoS-атак.

Об'єкт: процес моніторингу локальної інтернет-мережі в ОС Linux.

Предмет: технології та інструменти, які необхідні для моніторингу локальної інтернет-мережі в ОС Linux.

Мета: моніторинг трафіку в локальній інтернет-мережі за рахунок створення відповідного програмного забезпечення.

Кваліфікаційна робота бакалавра складається з вступу, чотирьох розділів, висновків та переліку джерел посилання.

У вступі визначається актуальність теми, що приймається за мету та проводиться невеликий огляд поставленого завдання, предмету та об'єкту дослідження.

У першому розділі показано порівняння існуючих альтернативних застосунків, проводиться висвітлення їх сильних та слабких сторін, що допоможе у складанні специфікації вимог до програмного забезпечення, що розробляється.

У другому розділі визначаються основні функціональні можливості та ставляться вимоги до майбутнього програмного забезпечення, щоб забезпечити виконання поставлених вимог специфікації застосунку. Детально висвітлюється процес взаємодії з ОС Linux для отримання всіх необхідних даних. Наведено алгоритм вирішення головної поставленої задачі.

У третьому розділі описуються основні модулі програмного забезпечення, надається інформація відносно технологій, мов програмування та бібліотек, розроблюються UML-діаграми.

У четвертому розділі проводиться опис користувацьких інтерфейсів,

демонструється тестування розробленого програмного забезпечення, виконується аналіз роботи, очікуваних та отриманих даних.

У висновках проводиться аналіз роботи та отриманих результатів.

КРБ викладена на 62 сторінки, вона містить 4 розділи, 53 ілюстрацій, 8 таблиць, 23 джерел в переліку посилань.

Ключові слова: *моніторинг трафіку, визначення DDoS-атак, локальна інтернет мережа, Linux Kernel, машинне навчання, розробка програмного забезпечення*.

ABSTRACT

of the Bachelor’s Thesis

"Application for traffic monitoring and DDoS-attacks detection using eBPF"

Student: Boiko Danylo

Supervisor: Candidate of Technical Sciences (Ph. D.), Associate Professor

Davydenko Yevhen

This work is devoted to the development of software for monitoring traffic in the local area network and automated detection of DDoS-attacks.

Object: the process of monitoring a local area network in the Linux operating system.

Subject: technologies and tools necessary for monitoring a local area network in the Linux operating system.

Objective: traffic monitoring in the local area network by creating appropriate software.

The bachelor’s thesis consists of an introduction, four chapters, conclusions and a list of references.

The introduction determines the relevance of the topic taken as the goal and a brief overview of the task, subject and object of research.

The first chapter describes a comparison of existing alternative applications, highlighting their strengths and weaknesses, which will help us in drawing up a specification of requirements for the software being developed.

The second chapter describes the basic functionality and requirements for the future software to ensure that the requirements of the application specification are met. The process of interacting with the Linux operating system to obtain all the necessary data is covered in detail. The algorithm for solving the main task is presented.

The third chapter describes the main software modules, provides information about technologies, programming languages and libraries, development of UML diagrams.

The fourth chapter describes the user interfaces, demonstrates the testing of the developed software, analyzes the work, expected and received data.

The conclusions analyze the work and the results obtained.

The qualification work of the bachelor is presented on 62 pages, it contains 4 sections, 53 illustrations, 8 tables, 23 sources in the list of references.

Keywords: *traffic monitoring, DDoS-attacks detection, local area network, Linux Kernel, machine learning, software development*.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних
ОС – операційна система
ПЗ – програмне забезпечення
СКБД – система керування базами даних

API – application programming interface
CLI – command line interface
CQRS – command query responsibility segmentation
DDoS – distributed denial-of-service
eBPF – extended berkeley packet filter
JIT – just-in-time
MQ – message queue
MLP – multilayer perceptron
SMTP – simple mail transfer protocol
UML – unified modeling language
UX – user experience
UI – user interface
XDP – express data path

ВСТУП

Актуальність теми кваліфікаційної роботи бакалавра зумовлена швидким розвитком інформаційних технологій та збільшенням кількості кібератак, які завдають шкоди компаніям і організаціям оскільки вони залежать від доступності мережевих ресурсів, що в кінцевому результаті може призвести до значних фінансових втрат. Моніторинг трафіку дозволяє виявляти та уникати потенційних атак і забезпечувати належний рівень безпеки. На сьогоднішній день важливо мати засоби для аналізу мережевого трафіку та виявлення аномалій які свідчать про наявність недоброзичливих користувачів, що стає більш важливим завданням для сучасних організацій різного направлення та рівня розвитку.

Подібні інструменти користуються популярністю і для поодиноких користувачів, оскільки несправжній трафік може спричинити перенавантаження домашньої мережі, що підвищить затримку при обміні інформацією з бажаними серверами. Запобігання кібератакам та забезпечення безпеки є надзвичайно важливим аспектом сучасного цифрового світу.

Об’єкт: процес моніторингу локальної інтернет-мережі в ОС Linux.

Предмет: технології та інструменти, які необхідні для моніторингу локальної інтернет-мережі в ОС Linux.

Мета: моніторинг трафіку в локальній інтернет-мережі за рахунок створення відповідного програмного забезпечення.

Для досягнення поставленої мети необхідно вирішити наступні **завдання**:

- провести аналіз аналогів;
- виявити основні проблеми та продумати можливі рішення;
- визначити необхідний функціонал застосунку;
- запропонувати алгоритм передачі інформації про активність в мережі та доцільні способи відображення кінцевої інформації користувачу;
- дослідити внутрішнє влаштування Kernel та eBPF в ОС Linux;
- розробити CLI частину застосунку на базі мов програмування Go, C;

– розробити back-end частину застосунку на базі технологій .NET, Python, gRPC, GraphQL, RabbitMQ, Redis, PostgreSQL та MongoDB;

– розробити front-end частину застосунку на базі технологій Angular та Material Design.

Практичне значення: аналізатор інтернет-мережі надає можливість проводити моніторинг трафіку для виявлення проблем та аномальної активності від недоброзичливих користувачів. Кінцевий застосунок також корисний для системних адміністраторів, оскільки може допомогти з визначенням шкідливих підключень та аналізом рівня навантаження в певний часовий період.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд існуючих аналогів

Аналіз існуючих альтернативних застосунків проводиться для висвітлення їх сильних та слабких сторін і є важливим кроком під час складання специфікації вимог до майбутнього ПЗ. Для проведення аналізу аналогів обрано наступні програмні рішення: PRTG Network Monitor (табл. 1.1), Wireshark (табл. 1.2), NetworkMiner (табл. 1.3).

PRTG Network Monitor [1]

Застосунок представляє з себе уніфікований інструмент моніторингу, який може контролювати практично будь-який об’єкт, що має IP-адресу. Він складається з основного PRTG сервера і одного або кількох зондів: основний PRTG сервер відповідає за налаштування та керування даними, зонди збирають дані та контролюють процеси на пристроях за допомогою датчиків.

Таблиця 1.1 – Опис PRTG Network Monitor

Назва	**PRTG Network Monitor**
Розробник	Компанія Paessler
Архітектура	Client-server
Мова реалізації	Delphi
Функції	1. Налаштування графіків для відображення стану пристрою. 2. Ведення тижневих журналів. 3. Надсилання сповіщень на електронну пошту та телефонний номер. 4. Підтримка простого протоколу мережевого керування (SNMP). 5. Автоматичне застосування стандартних фільтрів.
Переваги	1. Легкість використання. 2. Надання інформації в режимі реального часу. 3. Підтримка різних мережевих пристроїв.
Недоліки	1. Жорстка цінова політика. 2. Відсутність вебклієнту.
Вебсайт	paessler.com/prtg/prtg-network-monitor

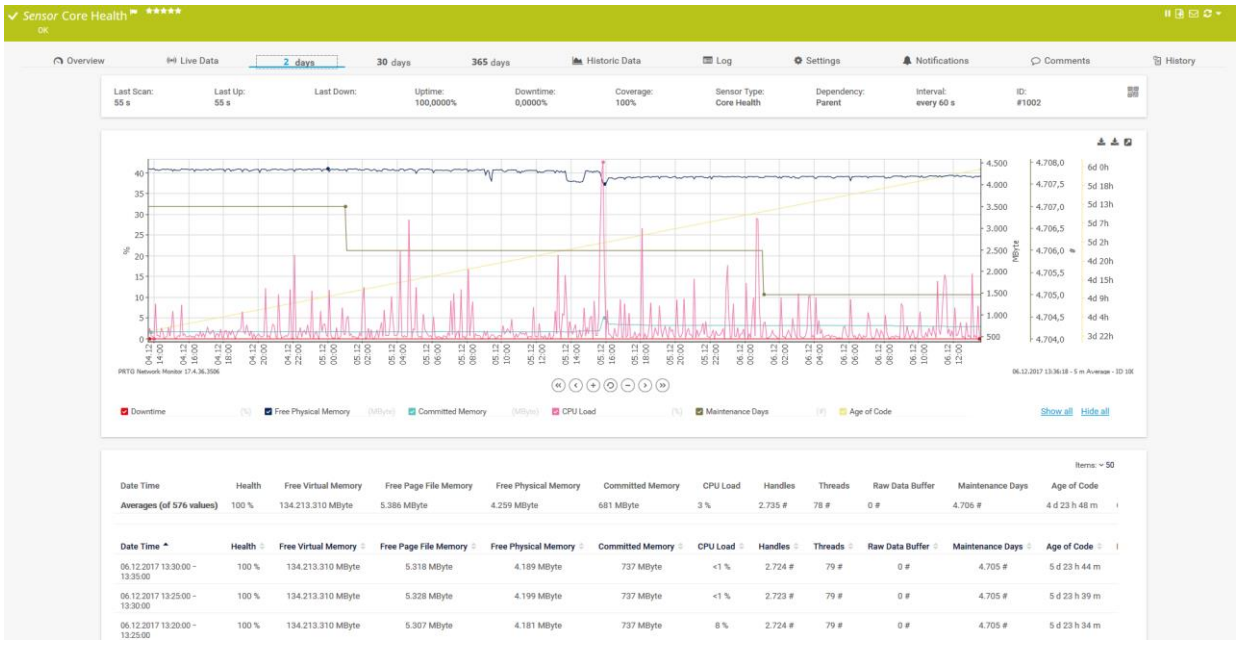

Рисунок 1.1 – Вигляд інтерфейсу застосунку «PRTG Network Monitor»

Інтерфейс користувача розроблений таким чином, щоб бути зручним та простим у взаємодії. Панель навігації знаходиться зверху та містить наступні основні пункти:

– home (домашня сторінка);
– devices (пристрої);
– alarms (сповіщення);
– reports (звіти);
– setup (налаштування).

Wireshark [2]

Застосунок є одним з найкращих у світі аналізатором мережевих протоколів та набув широкого використання. Він дозволяє побачити, що відбувається у мережі на мікроскопічному рівні та є стандартом для багатьох комерційних і некомерційних підприємств, державних установ і навчальних закладів.

Таблиця 1.2 – Опис Wireshark

Назва	**Wireshark**
Розробник	Організація Wireshark Foundation
Архітектура	Desktop application
Мова реалізації	C, Lua, C++

Кінець таблиці 1.2

Функції	1. Експорт інформації для подальшого аналізу. 2. Підтримка аналізу мережевих пакетів для різних інтерфейсів підключення та типів мереж (LAN, Wi-Fi, USB). 3. Надання користувачу широкого спектру функцій для аналізу. 4. Категоризація надходжуваного трафіку. 5. Фільтрація мережевих пакетів.
Переваги	1. Можливість моніторингу мережевої активності на мікроскопічному рівні. 2. Надання інформації про всі відвергнуті пакети. 3. Допомога у знайденні проблем безпеки та причин великої мережевої затримки. 4. Безкоштовність та доступність на різних платформах.
Недоліки	1. Для нових користувачів буде дуже складно зрозуміти, що відбувається на мікроскопічному рівні. 2. Надання занадто багато загальної інформації, через що складно знайти дійсно необхідну інформацію. 3. Складність парсингу занадто великих пакетів.
Вебсайт	wireshark.org

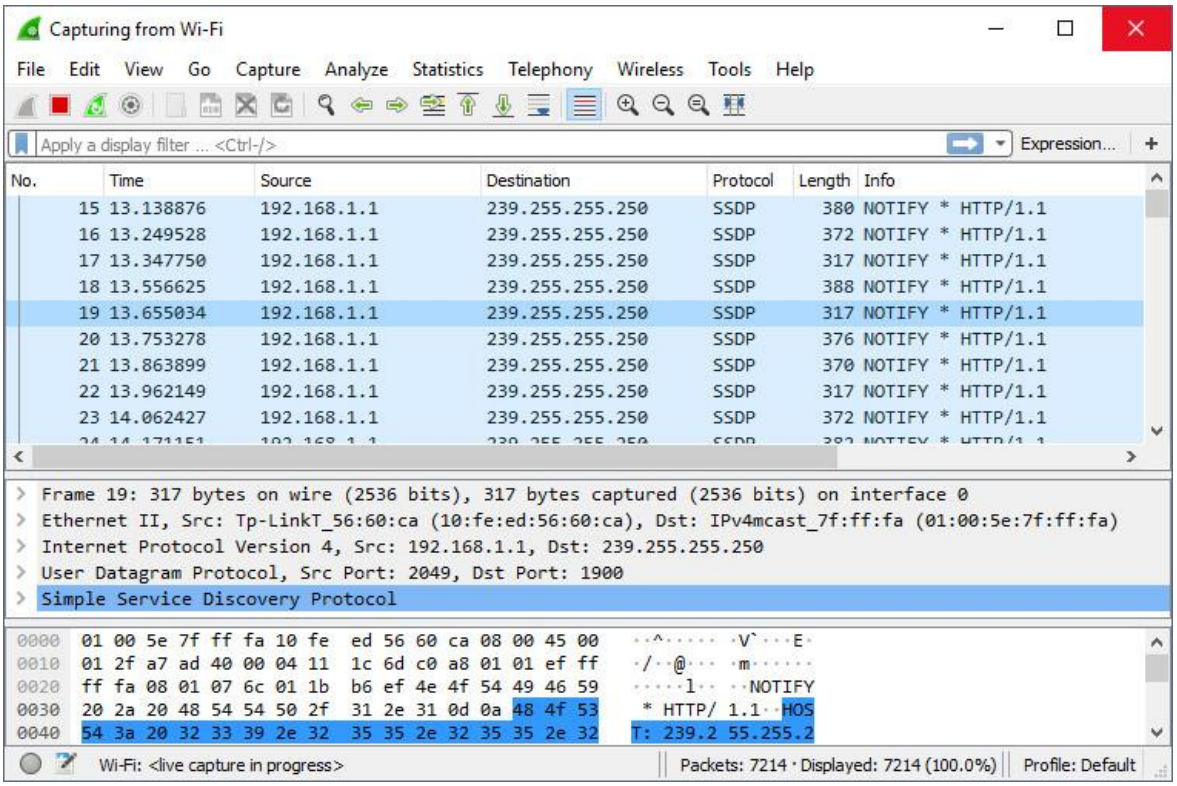

Рисунок 1.2 – Вигляд інтерфейсу застосунку «Wireshark»

Інтерфейс користувача є гнучким та налаштовуваним, він містить різноманітні інструменти та функції, які допомагають користувачу ефективно аналізувати мережевий трафік. Коли користувач обирає мережевий пакет із списку, вікно відомостей забезпечує детальний перегляд на основі доступної інформації. Виділимо наступні елементи інтерфейсу:

- список мережевих пакетів;
- вікно відомостей про пакет;
- вікно інструментів фільтру;
- панель ієрархії протоколів;
- меню статистики;
- параметри захоплення пакетів;
- параметри відображення.

NetworkMiner [3]

Застосунок є інструментом для мережевої експертизи з відкритим кодом та знаходить артефакти із захопленого мережевого трафіку у файлах PCAP.

Таблиця 1.3 – Опис NetworkMiner

Назва	**NetworkMiner**
Розробник	Організація Netresec
Архітектура	Desktop application
Мова реалізації	C#
Функції	1. Надання інформації в режимі реального часу. 2. Експорт мережевих даних до CSV, XML та JSON. 3. Визначення географічного знаходження IP-адреси. 4. Підтримка IPv6 адрес. 5. Можливість налаштування мережевих фільтрів.
Переваги	1. Ексклюзивний та інтуїтивно зрозумілий аналізатор хостів. 2. Відкритий вихідний код. 3. Доступність на різних платформах.
Недоліки	1. Можливість аналізу лише одного певного хосту. 2. Підтримка обмеженої кількості протоколів.
Вебсайт	netresec.com/?page=NetworkMiner

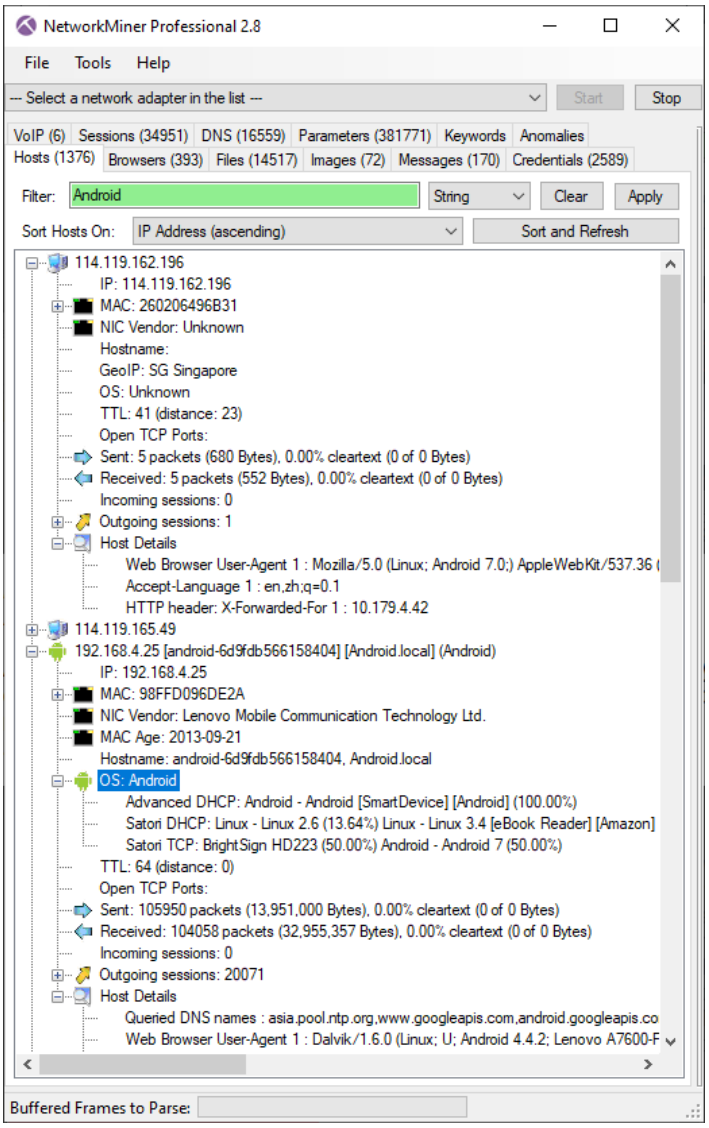

Рисунок 1.3 – Вигляд інтерфейсу застосунку «NetworkMiner»

Інтерфейс користувача є простим та інтуїтивно зрозумілим, містить різноманітні інструменти та функції, які допомагають користувачам аналізувати мережевий трафік та експортувати дані у файли.

Головне вікно відображає список захоплених мережевих пакетів у вигляді таблиці. Кожен рядок представляє один мережевий пакет, стовпці показують різні атрибути пакета.

Однією з ключових особливостей застосунку є функція експорту даних із отриманого мережевого трафіку. Користувач має можливість експортувати окремі пакети в файли різноманітних форматів, включаючи CSV, XML та JSON. Це є корисним для подальшого аналізу певних пакетів або обміну з іншими користувачами.

1.2 Аналіз системи, що розробляється

Інструменти для моніторингу інтернет-мереж мають суттєвий вплив в наші дні та забезпечують захист, продуктивність, доступності та справність багатьох ресурсів. Їх можна використовувати для швидкого і ефективного виявлення та запобігання вторгненням, а також для моніторингу мережевого трафіку. Маючи доступ до мережі, організації можуть скоротити час, необхідний для реагування на проблеми та зменшити вплив від DDoS-атак [4].

ПЗ має забезпечувати мультикористувацький режим роботи та мати гнучку систему прав доступу. Існування окремих серверів для взаємодії через CLI та вебклієнт допоможе при майбутньому масштабуванні системи.

Користувач повинен мати змогу самостійно обирати мережевий інтерфейс для аналізу через CLI, попередньо пройшовши авторизацію.

Таблиця 1.4 – Опис системи що розробляється

Основні задачі	1. реєстрація нових користувачів; 2. підтримка декількох різних пристроїв для одного користувача; 3. верифікація електронних пошт користувачів; 4. отримання, додавання, оновлення та видалення IP-фільтрів; 5. передача інформації про пакети до вебсерверу; 6. автоматичний аналіз пакетів з використанням нейронних мереж; 7. отримання інфографіки про пакети за день або тиждень; 8. вибір інтерфейсу підключення до мережі для аналізу пакетів; 9. відображення локальної статистики в CLI; 10. авторизація за основі прав доступу.
Користувачі системи	1. користувач; 2. адміністратор.
Сценарії роботи	1. користувач авторизується через CLI та починає відправляти інформацію про отримані пакети на сервер; 2. користувач авторизується через CLI з нового пристрою, система автоматично додає новий пристрій в список; 3. користувач авторизується через вебклієнт та переглядяє графіки отриманих пакетів;

Кінець таблиці 1.4

Сценарії роботи	4. користувач авторизується через вебклієнт та модифікує список IP-фільтрів.
Засоби апаратної та програмної реалізації	1. back-end: .NET, Python, gRPC, GraphQL, RabbitMQ; 2. front-end: Angular, TypeScript; 3. command line interface: Go, C, eBPF; 4. databases: PostgreSQL, MongoDB, Redis.
Вихідні дані	1. текстове представлених отриманих пакетів в CLI; 2. відображення графіку отриманих пакетів через вебклієнт; 3. отримання списку IP-фільтрів через вебклієнт.

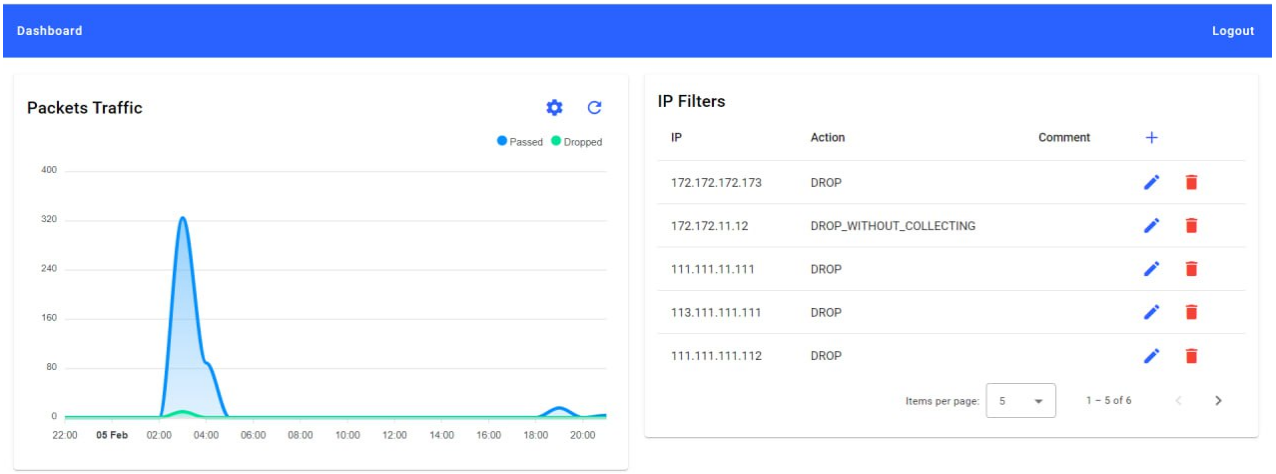

Рисунок 1.4 – Макет інтерфейсу застосунку «NetworkMonitoringTool»

Під час взаємодії через вебклієнт для користувача на головній сторінці доступні 2 основні блоки:

– packets traffic (трафік мережевих пакетів);
– IP filters (IP-фільтри).

1.3 Специфікації вимог до програмного забезпечення

ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

Призначення системи (застосунку), для якої розробляється програмне забезпечення

Призначенням застосунку є моніторинг трафіку в локальній інтернет-мережі за рахунок створення відповідного ПЗ.

Погодження, що ухвалені в програмній документації

Було погоджено, що для створення ПЗ та його злагодженої роботи будуть використовуватися допоміжні фреймворки – ASP.NET та Angular.

Межі проєкту ПЗ

Крайня дата завершення роботи над ПЗ – 04.06.2023р.

ЗАГАЛЬНИЙ ОПИС

Сфера застосування

Дане ПЗ не має обмежень у сферах його застосування, за виключенням ОС відмінних від Linux, оскільки для них необхідно використовувати інші засоби взаємодії на рівні ОС.

Характеристики користувачів

Основні характеристики користувачів: наявність ОС Linux з підтримкою Kernel та підключення до мережі Інтернет.

Загальна структура і склад системи

Основні частини для створення програмного забезпечення: gRPC сервер, GraphQL сервер, сервер для аналізу пакетів на основі машинного навчання, CLI, вебклієнт, бази даних (PostgreSQL, MongoDB та Redis).

Загальні обмеження

Обмеження для роботи з ПЗ – наявність ОС Linux з підтримкою Kernel та підключення до мережі Інтернет.

ФУНКЦІЇ СИСТЕМИ МОНІТОРИНГУ ТРАФІКУ ТА ВИЗНАЧЕННЯ DDoS-АТАК

Функція додавання нового IP-фільтру

Опис функції

Функція додавання IP-фільтру дозволяє користувачу відсікти небажаний трафік або додати IP-адресу в список безпечних (white list).

Вхідна і вихідна інформація

Вхідна інформація – IP-адреса, дія IP-фільтру;

вихідна інформація – заблоковані та дозволені мережеві пакети.

Функціональні вимоги

БД з інформацією про IP-фільтри та доступ до мережі Інтернет.

Функція отримання інфографіки про пакети з фільтрацією

Опис функції

Функція отримання інфографіки допомагає користувачу аналізувати пакети, фільтрація відповідає за відображення мережевих пакетів за певний часовий період.

Вхідна і вихідна інформація

Вхідна інформація – ім’я пристрою, часовий період;

вихідна інформація – діаграма, що задовольняє критерії пошуку.

Функціональні вимоги

БД з інформацією про пристрої і мережеві пакети користувача та доступ до мережі Інтернет.

Функція аналізу локальної мережі

Опис функції

Функція аналізу локальної мережі збирає інформацію про отримані пакети, передає інформацію на сервер та за необхідності запускає процес розпізнання DDoS-атаки.

Вхідна і вихідна інформація

Вхідна інформація – мережевий інтерфейс підключення;

вихідна інформація – отримані мережеві пакети.

Функціональні вимоги

БД з інформацією про IP-фільтри і мережеві пакети та підключення до мережі Інтернет.

ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Джерела і зміст вхідної інформації (даних)

В даному ПЗ джерелом вхідної інформації є користувач та його локальна інтернет-мережа.

Нормативно-довідкова інформація (класифікатори, довідники тощо)

Немає вимог до даного пункту.

Вимоги до способів організації, збереження та ведення інформації

Обмін даними відбувається через gRPC, GraphQL та RabbitMQ. В якості БД обрано PostgreSQL, MongoDB та Redis.

ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Жорстких вимог до технічного забезпечення немає. Користувач повинен мати комп’ютер чи ноутбук на якому встановлена ОС Linux з підтримкою Kernel та підключення до мережі Інтернет.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Архітектура програмної системи

Архітектура програмного забезпечення складається з клієнтських застосунків (CLI та Web client), серверів (gRPC, GraphQL та DDoS detection), баз даних (PostgreSQL, MongoDB та Redis) і брокеру повідомлень на основі черги (RabbitMQ).

Системне програмне забезпечення

Для написання front-end частини обрано фреймворк Angular, для CLI частини – мови програмування Go та C, для back-end частини – фреймворк ASP.NET та мову програмування Python. Обмін даними має відбуватися з використанням GraphQL, gRPC та RabbitMQ. В якості БД обрано PostgreSQL, MongoDB та Redis.

Мережне програмне забезпечення

Для створення ПЗ використовується ОС Linux з підтримкою Kernel, редактори коду: JetBrains Rider, PyCharm та WebStorm і будь-який сучасний браузер для перегляду сторінки з інфографікою.

Програмне забезпечення ведення інформаційної бази

Ведення інформаційної бази відбувається за допомогою CRUD-операцій з PostgreSQL, MongoDB та Redis.

Мова і технологія розробки ПЗ

Програмне забезпечення має бути розроблене з використанням фреймворків ASP.NET та Angular. Мови розробки – C, Go, Python, C# та TypeScript.

ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

Інтерфейс користувача

Вебклієнт має задовольняти усім вимогам UX та UI, що дозволить користувачу затратити найменше часу на розуміння роботи системи. Шаблон сторінки повинен бути розділений на дві частини, де перша частина – це інфографіка про отримані мережеві пакети, вона розміщена зліва, друга частина є список IP-фільтрів, вона розміщена справа.

Апаратний інтерфейс

Апаратний інтерфейс – пристрій користувача (ПК, ноутбук чи планшет), який він буде використовувати для взаємодії з сторінками вебклієнту.

Програмний інтерфейс

ASP.NET – фреймворк з відкритим вихідним кодом для розробки застосунків і сервісів з використанням .NET та C#. Angular – фреймворк з відкритим вихідним кодом для розробки мобільних і браузерних застосунків з використанням мови програмування TypeScript.

Комунікаційний протокол

Застосунок базується на використанні мережних протоколів WAP – протокол бездротової передачі даних та TCP/IP.

ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Доступність

Застосунок є доступним для всіх користувачів, що мають бажання ним скористатися, за умови наявності ОС Linux з підтримкою Kernel та доступу до мережі Інтернет.

Супроводжуваність

Супроводження не передбачається.

Переносимість

CLI може працювати на ОС Linux з підтримкою Kernel. Вебклієнт має сумісність з усіма сучасними браузерами та операційними системами.

Продуктивність

Продуктивність ПЗ залежить від швидкості мережі Інтернет. Час виконання

запитів не має перевищувати 2 секунди.

Надійність

Вхідні дані, які передаються користувачем повинні бути приватними. Має бути виключена можливість їх отримання будь-якими іншими способами. Користувач отримує доступ виключно до своїх даних та лише після авторизації у системі.

Безпека

Функціонал який дозволяє взаємодіяти з даними має перевіряти токен авторизація та права доступу.

Висновки до розділу 1

В першому розділі кваліфікаційної роботи бакалавра показано порівняння існуючих альтернативних застосунків, проведено висвітлення їх сильних та слабких сторін, набуто навичок у складанні специфікації вимог до програмного забезпечення. Під час розглядання аналогів звернуто увагу на їх архітектуру, мови реалізації, функціонал, користувацький інтерфейс, переваги та недоліки.

Проведено аналіз системи, що розробляється шляхом визначення основних задач, користувачів системи, сценаріїв роботи та засобів апаратної і програмної реалізації. Наведено вигляд макету інтерфейсу головної сторінки з описанням блоків розмітки. Продумано можливості майбутнього масштабування застосунку шляхом відокремлення окремий серверів під різні завдання (взаємодія через CLI, отримання даних з вебклієнту, аналіз пакетів з використанням машинного навчання).

2 МОДЕЛЮВАННЯ ЗАСТОСУНКУ МОНІТОРИНГУ ТРАФІКУ

2.1 Створення use case діаграми

Use case або сценарії використання – це перелік дій, які необхідно виконати користувачу для взаємодії з системою. Зазвичай сценарії представляють взаємодію між системою та зовнішніми акторами, показуючи, як система використовується для досягнення конкретних цілей або завдань, не вдаючись у технічні деталі програмної реалізації.

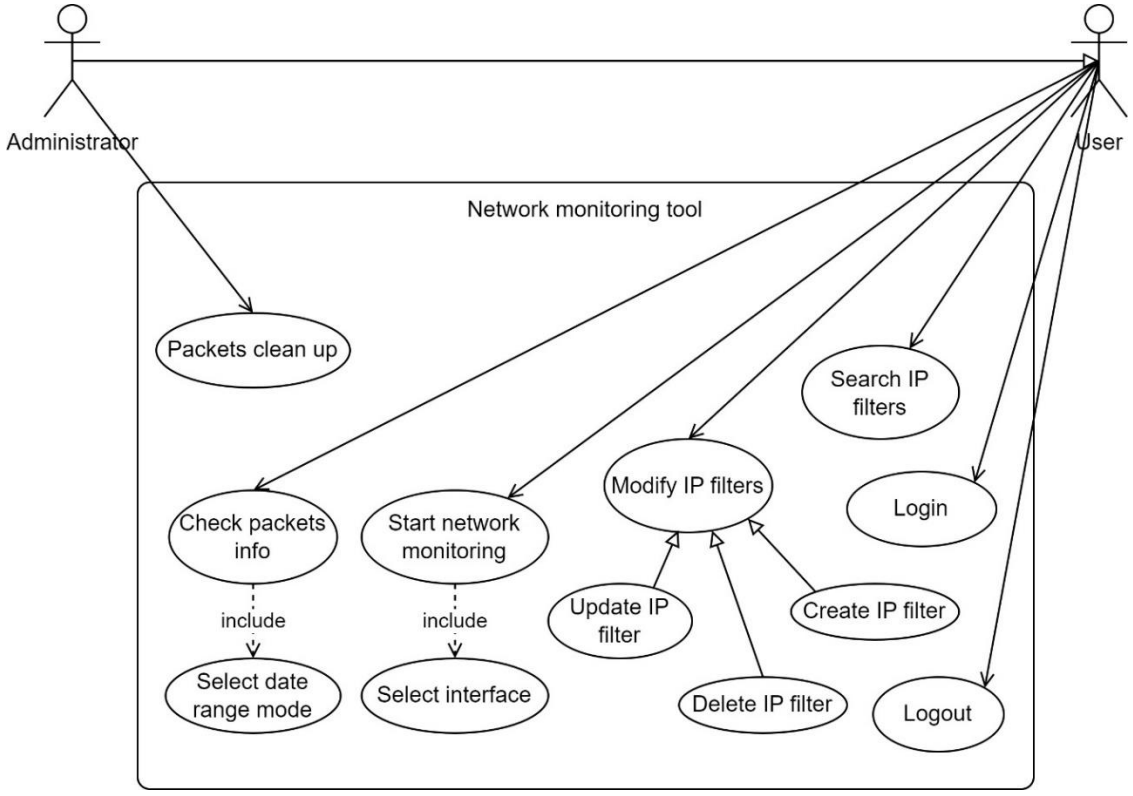

Рисунок 2.1 – Діаграма використання застосунку моніторингу трафіку

Написання сценаріїв використання дозволяє чітко визначити хто є користувачем системи та які кроки йому необхідно виконати для досягнення поставленої мети. Існує три форми написання сценаріїв використання:

1) Коротка – представляє опис в один абзац одного зі сценаріїв роботи системи (у більшості випадків успішного). Зазвичай вони необхідні під час початкового аналізу вимог до майбутньої системи.

2) Поверхнева – поверхневий опис у вільній формі усіх сценаріїв (головного та альтернативних) одного з варіантів використання системи.

3) Повна – всі кроки і дії детально описані, включають перед та пост умови виконання. Застосовується для надважливих сценаріїв використання.

Короткий use case (видалення IP-фільтру)

Користувач авторизується в системі та переходить на сторінку з IP-фільтрами. Користувач шукає необхідний IP-фільтр, натискає іконку «Видалити» та підтверджує операцію. Система відправляє запит на сервер та у разі успішного видалення з БД, оновлює UI.

Поверхневий use case (додавання IP-фільтру)

Головний сценарій (успішний):

Користувач авторизується в системі та переходить на сторінку з IP-фільтрами. Користувач натискає іконку «Додати», заповнює всю необхідну інформацію та підтверджує операцію. Система відправляє запит на сервер та у разі успішного додавання до бази даних, оновлює UI.

Альтернативні сценарії:

1) Користувач вже має фільтр для наданого IP, дублювання не допускається.

2) Токен авторизації недійсний, застосунок відправляє запит до серверу для оновлення токену та продовжує роботу.

3) Перевищується кількість можливих IP-фільтрів для певного користувача, відправляється повідомлення про помилку.

4) GraphQL сервер недоступний, взаємодія через вебсторінку неможлива, система продовжує приймати інформацію про пакети через gRPC сервер.

Таблиця 2.1 – Повний use case (відправлення пакетів на сервер)

Primary actor	Користувач
Scope	Застосунок моніторингу трафіку
Level	Мета користувача
Preconditions	Користувач успішно пройшов авторизацію в застосунку
Stakeholders and interests	Користувач зацікавлений в аналізі отриманих пакетів та визначення DDoS-атак
Main success scenario	1. Користувач проходить авторизацію через CLI.

Кінець таблиці 2.1

Main success scenario	2. Користувач проходить авторизацію через CLI. 3. Користувач вводить команду nmt_cli start та починає збирати інформацію про отримувані пакети. 4. CLI в режимі реального часу раз в 3 секунди передає інформацію на сервер (за умови, що в цей проміжок часу система отримала якісь пакети). 5. CLI в режимі реального часу виводить інформацію про пакети в консоль (за умови передачі прапору --stats під час запуску). 6. Сервер додає інформацію про отримані пакети в БД. 7. Сервер проводить поверхневий аналіз отриманих пакетів та при необхідності починає поглиблену перевірку за допомогою машинного навчання. 8. За необхідності сервер автоматично створює потрібні IP-фільтри. 9. Кроки 4-8 повторюються до моменту відміни виконання за бажанням користувача. 10. Користувач авторизується через вебклієнт та дивиться інфографіку.
Result	Користувач передав на сервер інформацію про отримані пакети
Extensions	1. Користувач не обрав інтерфейс підключення в CLI, виводиться повідомлення про помилку. 2. Користувач вже має фільтр для певного IP, дія фільтру не оновлюється. 3. Токен авторизації недійсний, застосунок відправляє запит до серверу для оновлення токену та продовжує роботу. 4. Перевищується кількість можливих IP-фільтрів для певного користувача, виводиться повідомлення про помилку. 5. Сервер не доступний, CLI припиняє свою роботу.
Special requirements	1. Адаптивний інтерфейс. 2. Встановлена ОС Linux з підтримкою Kernel. 3. Доступ до мережі Інтернет.
Frequency of occurrence	Система має змогу працювати майже безперервно

Загалом сценарії використання маю бути чіткими, короткими та легкими для розуміння, а також повинні забезпечувати високорівневе уявлення про застосунок та його взаємодію з середовищем.

2.2 Алгоритмізація логіки застосунку

Для відображення алгоритму роботи застосунку прийнято використовувати діаграми діяльності. Їх можна застосовувати для моделювання широкого діапазону систем та процесів, від простих бізнес-процесів до складних програмних систем. Вони складаються з набору символів та позначень для представлення різних дій, таких як завдання, рішення та цикли, і показують зв’язки та залежності між ними.

Для застосунку моніторингу трафіку створено три діаграми діяльності які описують наступний ключовий функціонал:

1) Створення IP-фільтру (рис. 2.2). Діаграма містить такі кроки як: авторизація користувача, відкриття головної сторінки, заповнення форми для нового IP-фільтру, перевірка введених даних та додавання запису в БД.

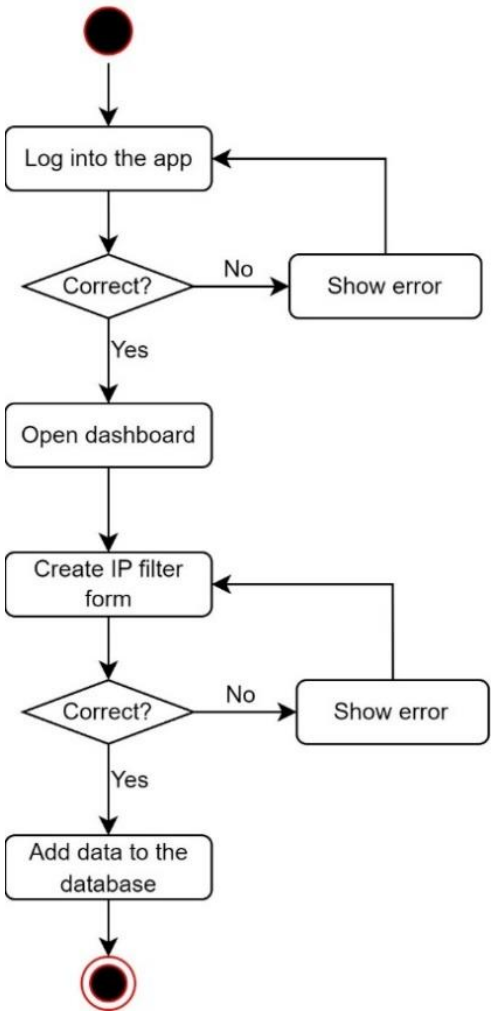

Рисунок 2.2 – Діаграма діяльності додавання IP-фільтру

2) Авторизація через CLI (рис. 2.3). Діаграма містить такі кроки як: введення імені користувача та паролю, перевірка існування пристрою та генерація токенів авторизації.

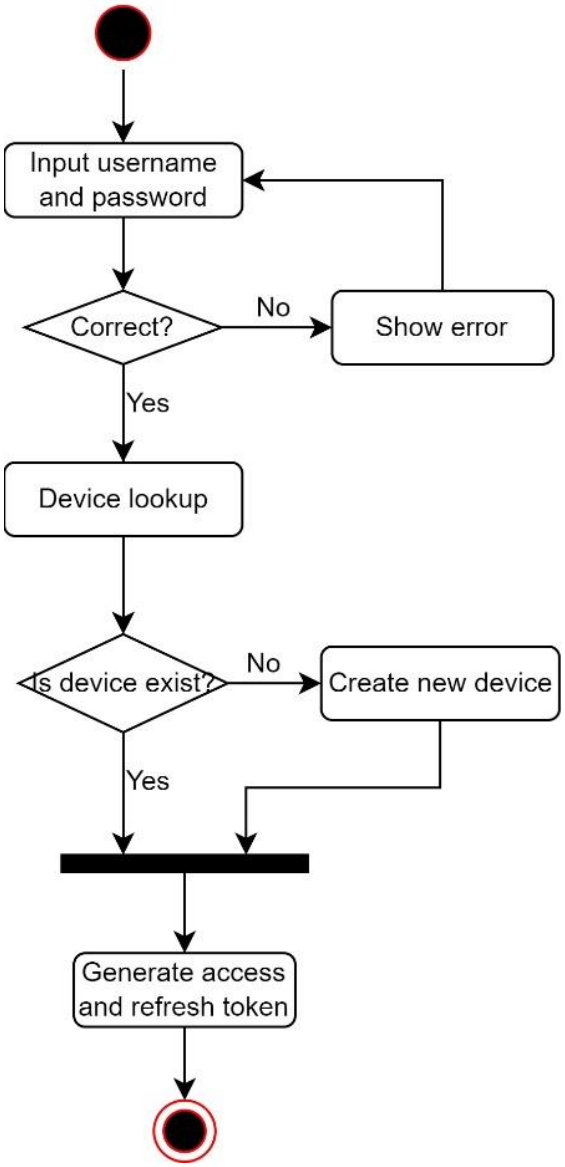

Рисунок 2.3 – Діаграма діяльності авторизації через CLI

Для уніфікації пристрою користувача можна використовувати ідентифікатор який створюється під час встановлення ОС. Для його отримання не потрібні права адміністратора та достатньо ввести команду cat /etc/machine-id [5].

3) Додавання мережевих пакетів (рис. 2.4). Діаграма містить такі кроки як: авторизація користувача, виконання команди nmt_cli start, відправка даних про мережеві пакети на gRPC сервер, розпізнання DDoS-атаки на основі машинного навчання та автоматичне створення IP-фільтрів для недоброзичливих IP-адрес.

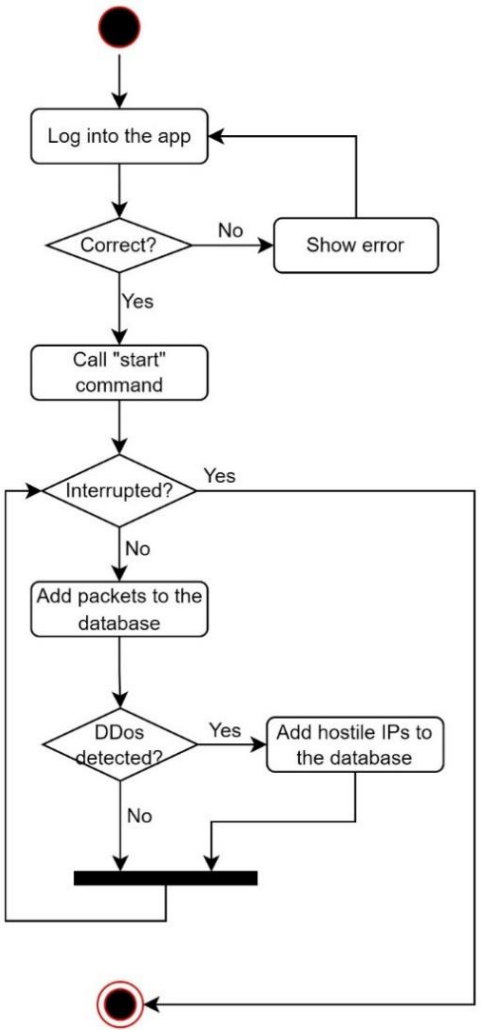

Рисунок 2.4 – Діаграма діяльності додавання мережевих пакетів

Поглянувши на діаграми діяльності, одразу приходить на думку схожість з блок-схемами, але варто зазначити ключову відмінність у тому, що діаграми діяльності дозволяють підтримувати паралельні процеси.

2.3 Конструювання та аналіз діаграми розгортання

Діаграми розгортання використовуються для моделювання фізичного розгортання компонентів ПЗ. Вони показують, як компоненти ПЗ взаємодіють на апаратних вузлах, таких як сервери, маршрутизатори та комутатори, і як вони спілкуються між собою через мережу.

Діаграми розгортання корисні для архітекторів програмного забезпечення та розробників, щоб зрозуміти інфраструктуру розгортання системи і виявити потенційні проблеми з продуктивністю та вузькі місця. Їх також можна

використовувати для планування розгортання системи у виробничому середовищі та для швидкої передачі стратегії розгортання іншим особам.

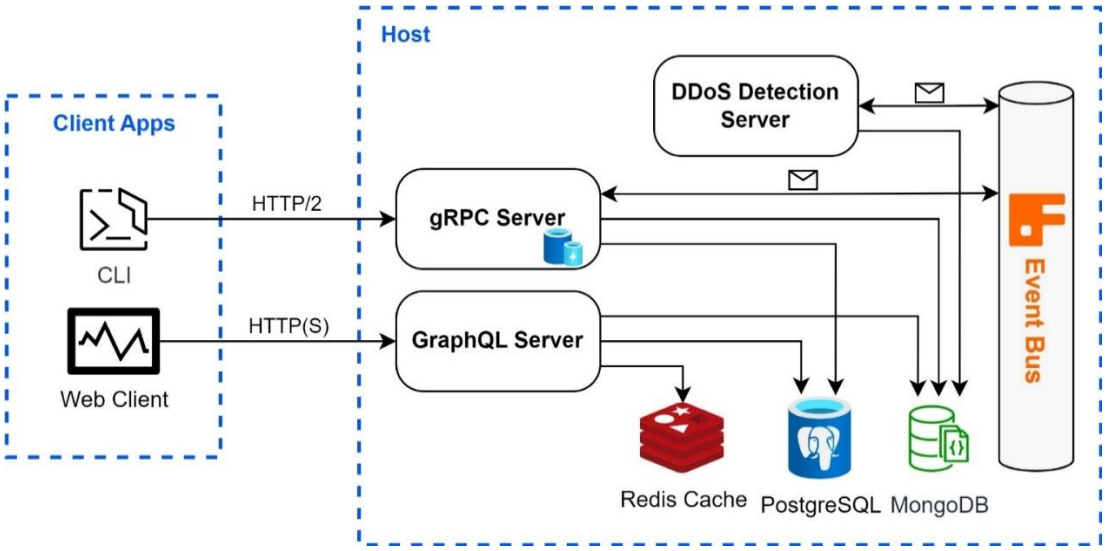

Рисунок 2.5 – Діаграма розгортання застосунку моніторингу трафіку

Таблиця 2.2 – Опис компонентів діаграми розгортання

Компонент	**Опис**
Client – CLI	Консольний застосунок який дозволяє користувачу пройти авторизацію, обрати мережевий інтерфейс підключення, за допомогою eBPF отримати інформацію про мережеві пакети та передати її на gRPC сервер.
Client – Web client	Вебзастосунок за допомогою якого можна зареєструватися в системі, підтвердити електронну пошту, переглядати та модифікувати IP-фільтри, переглядати інфографіку по отриманим мережевим пакетам за певний часовий період.
Server – gRPC	Отримує інформацію про мережеві пакети через протокол HTTP 2.0 від CLI, відслідковує стрибки в кількості мережевих пакетів на основі внутрішнього (in-memory) кешу та в разі необхідності відправляє запит на аналіз трафіку до RabbitMQ.
Server – GraphQL	Використовується як шлюз для отримання даних через вебклієнт, підтримує кешування на рівні Redis та робить запити до MongoDB для створення інфографіки.
Server – DDoS detection	Сервер основною задачею якого є аналіз мережевого трафіку та визначення DDoS-атак за допомогою машинного навчання, підтримує можливість навчання нових моделей.

Кінець таблиці 2.2

Server – Redis cache	Рівень кешування для підвищення продуктивності застосунку за рахунок зменшення кількість запитів до БД, що призведе до швидшого часу відповіді.
Server – PostgreSQL	Зберігає інформацію про користувачів, права доступу, IP-фільтри та пристрої користувача.
Server – MongoDB	Зберігає інформацію про отримані мережеві пакети.
Server – RabbitMQ event bus	Використовується як брокер повідомлень між gRPC та DDoS detection сервером, відповідає за аналіз мережевих пакетів та автоматичне створення нових IP-фільтрів.

Розподіл на окремі сервери не лише зменшить навантаження та дозволить працювати системі асинхронно завдяки черзі повідомлень, а ще додатково відкриє можливість ефективного шардингу серверів [6].

Зрозуміло, що запитів до gRPC серверу буде більше оскільки кожен користувач відправляє дані про нові мережеві пакети кожні 3 секунди та значно рідше використовує вебклієнт для отримання інфографіки. Таким чином можна зробити висновок, що коли у застосунку буде досить багато користувачів, доведеться мати декілька паралельних gRPC серверів. Це допоможе розподілити навантаження та підвищити продуктивність, безвідмовність та доступність системи.

Для досягання ефективного шардингу, дані та операції в системі необхідно розділити таким чином, щоб збалансувати навантаження та мінімізувати зв’язок між сегментами.

Одним із поширених підходів до шардингу є використання алгоритму хешування для зіставлення ключів із сегментами. У цьому підході кожному серверу призначається діапазон значень ключів, а функція зіставлення гарантує, що кожен ключ послідовно призначається одному шарду. Це має гарантувати, що кожен шард матиме приблизно однакове навантаження.

Розподілена хеш-таблиця є одним із основних компонентів, що використовуються в розподілених масштабованих системах. Для хеш-таблиць потрібні ключ, значення та хеш-функція, де хеш-функція обирає місце для

збереження значення.

Поглянувши на загальну архітектуру, можна помітити використання Redis кешування, це зменшить кількість запитів до БД, що призведе до зменшення часу очікування відповіді.

Враховуючи можливе збільшення серверів відповідальних за кешування, припустимо ми маємо n серверів, інтуїтивно зрозуміла хеш-функція для розподілу інформації між ними буде ключ % n. Це виглядає просто, але має два недоліки:

1) Таке рішення не масштабується по горизонталі. Кожного разу, коли новий хост кешу додається до системи, усі наявні хости будуть порушені. Якщо система кешування містить багато даних то стає важко запланувати час простою для оновлення всіх хостів кешування.

2) Розподілення не може бути збалансованим. Для системи кешування – це означає, що деякі кеш-таблиці стають гарячими та насиченими, а інші простоюють та майже порожні.

У таких ситуаціях послідовне хешування є хорошим способом покращити систему кешування та може бути застосованим за необхідності (рис. 2.6).

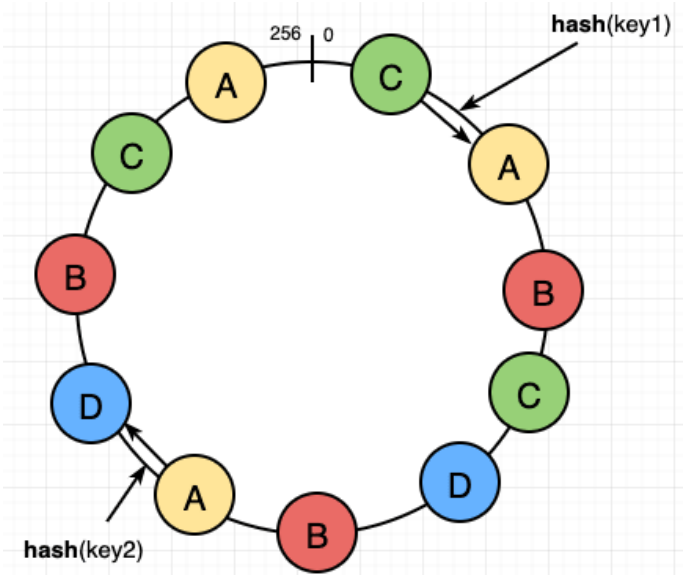

Рисунок 2.6 – Схема роботи послідовного кешування

Послідовне хешування полягає у наступному. Дано список серверів кешу та хеш-функція, яка повертає ціле число у певному діапазоні. Щоб зіставити ключ із сервером необхідно рухатися по уявному кільцю за годинниковою стрілкою, доки

не знайдеться перше найближче значення з діапазону. Поглянувши на рисунок 2.6 бачимо, що значення key1 знаходиться на сервері A, а значення key2 знаходиться на сервері D.

2.4 Розробка діаграм взаємодії

Діаграма послідовності є одним з типів діаграми взаємодії та показує взаємодію між об’єктами або компонентами в системі в упорядкованій за часом послідовності.

Таблиця 2.3 – Опис можливих компонентів діаграми взаємодії

Елемент	Опис
Актори	Актори – це сутності, які взаємодіють із системою та можуть бути представлені у верхній частині діаграми.
Об’єкти	Об’єктами є екземплярами класів у системі та представлені прямокутниками на діаграмі.
Лінії життя	Лінії життя представляють час існування об’єкта у системі та відображаються у вигляді вертикальних ліній, що тягнуться від верхньої частини вікна об’єкта.
Повідомлення	Повідомленням є взаємодія між об’єктами, які відображаються стрілками між лініями життя.
Смуги активації	Смуги активації представляють час, протягом якого об’єкт виконує метод та відображаються як горизонтальні смуги на лінії життя.
Обмеження	Обмеження використовуються для визначення умов або обмежень, які повинні бути виконані для того, щоб відбулося повідомлення або взаємодія. Їх можна використовувати для визначення передумов та післяумов під час роботи системи.

Діаграма послідовності корисна для моделювання та розуміння взаємодії між об’єктами чи компонентами в системі, а також її можна використовувати для розробки та тестування ПЗ.

Для моделювання застосунку моніторингу трафіку розроблено 3 діаграми взаємодії:

1) Перегляд головної сторінки (рис. 2.7). Діаграма представляє собою

перехід на головну сторінку вебклієнту, запит до GraphQL сервера який в свою чергу робить запит до БД та повертає інформацію користувачу.

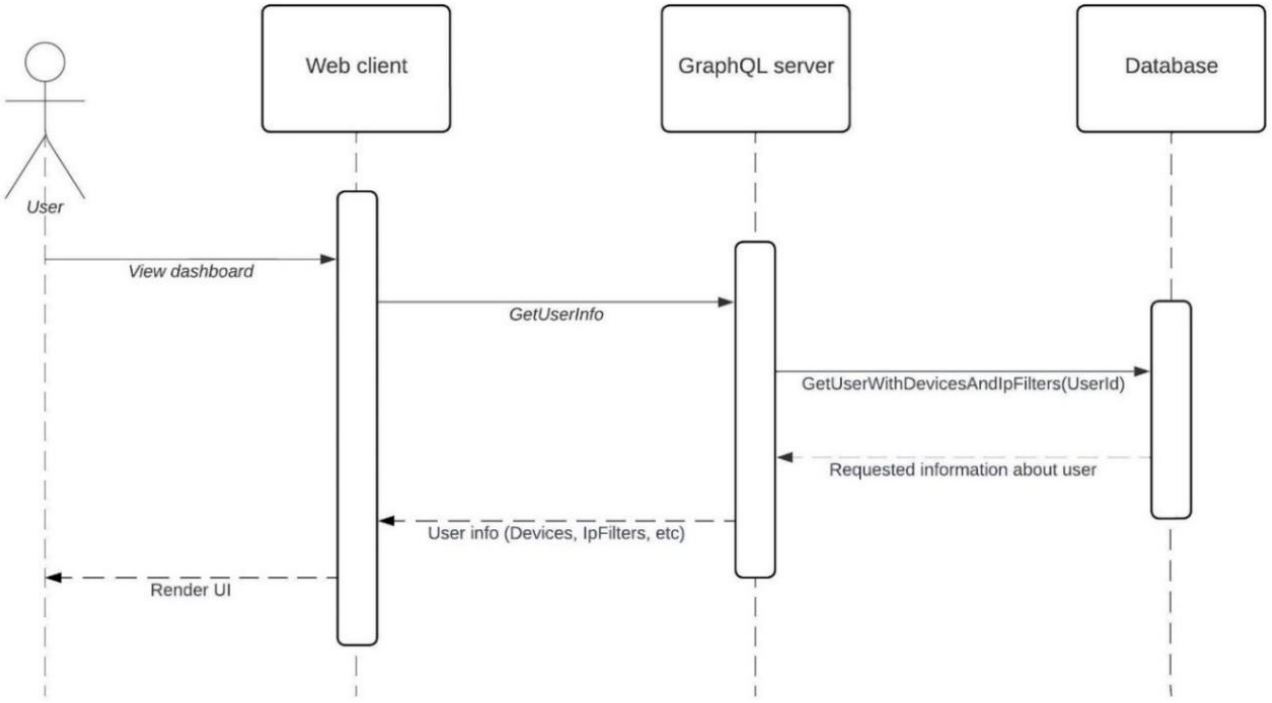

Рисунок 2.7 – Діаграма взаємодії перегляду головної сторінки

2) Додавання нового IP-фільтру (рис. 2.8). Діаграма представляє собою перехід до вікна створення IP-фільтру, запит до GraphQL сервера який додає запис про новий IP-фільтр в БД.

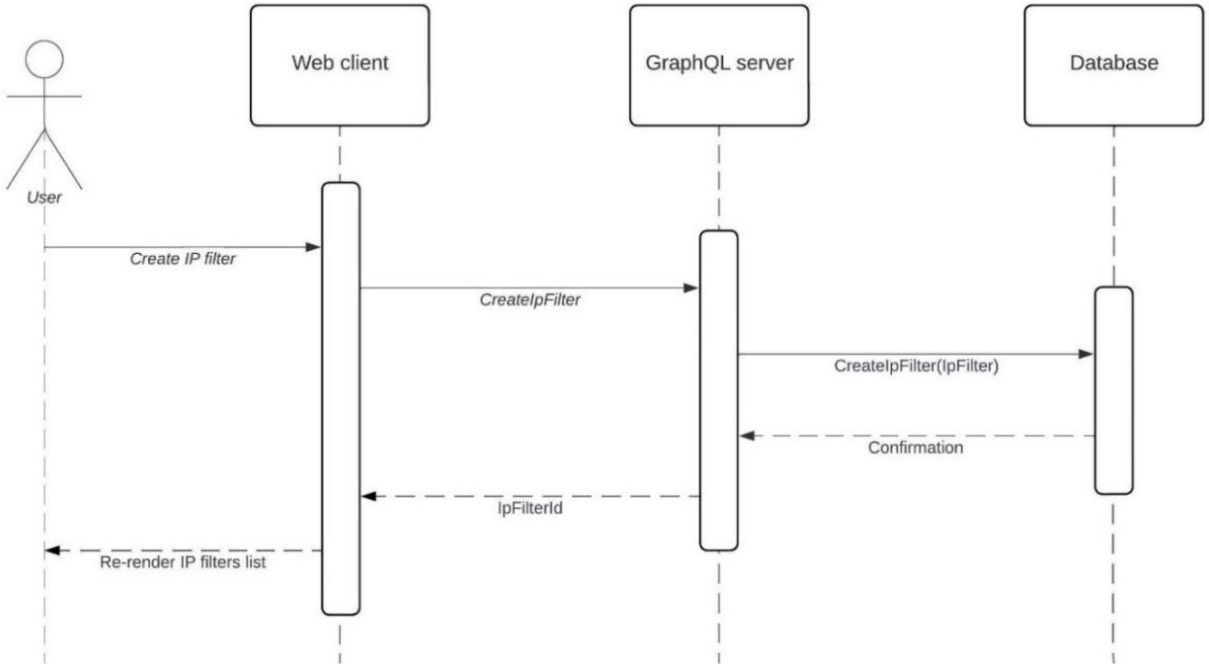

Рисунок 2.8 – Діаграма взаємодії додавання нового IP-фільтру

3) Відправлення інформації про мережеві пакети (рис. 2.9). Наступна діаграма є більш комплексною та включає в себе запит про наявні IP-фільтри, циклічне відправлення інформації про отримані мережеві пакети, умови додавання нових IP-фільтрів та виводу інформації в консоль.

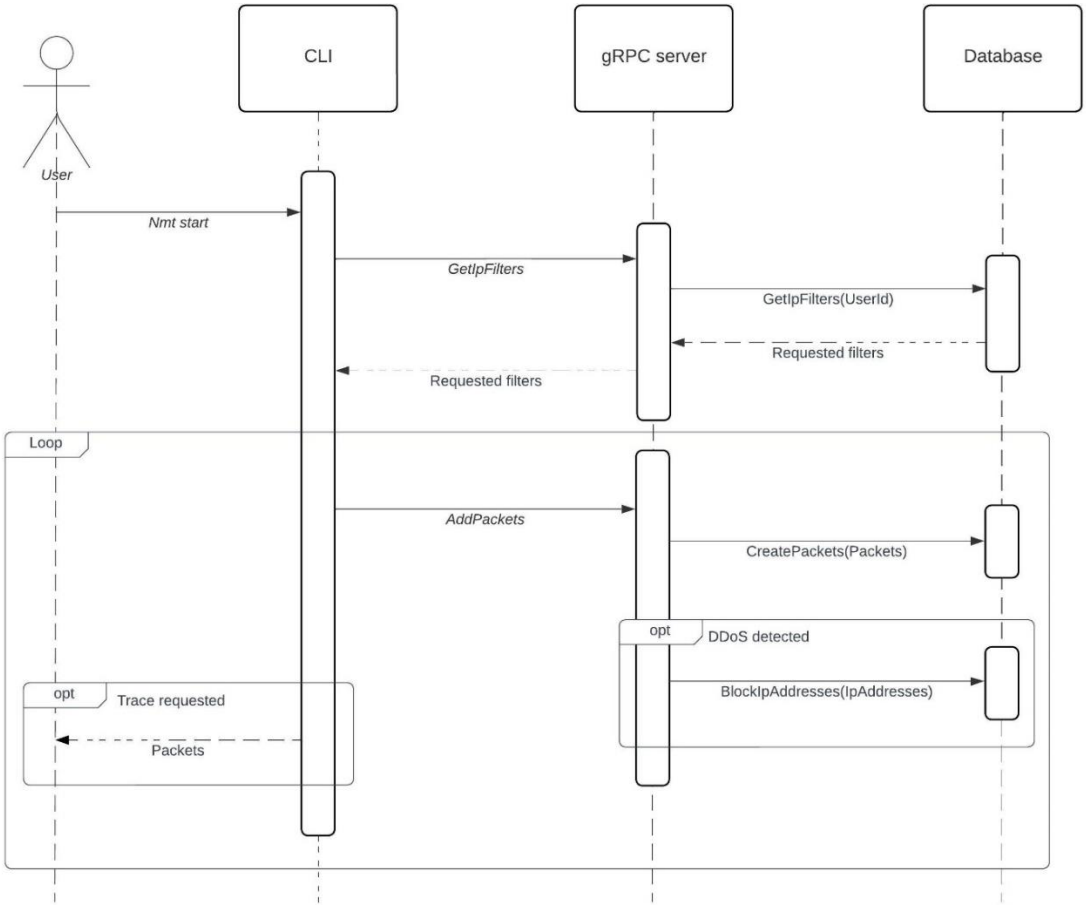

Рисунок 2.9 – Діаграма взаємодії відправлення інформації про мережеві пакети

Зазначені діаграми чітко зображують потік повідомлень та викликів методів між об’єктами і показують їх порядок взаємодії.

2.5 Архітектура взаємодії eBPF та ОС Linux

eBPF представляє собою абстрактну віртуальну машину, яка працює в ядрі Linux та може запускати програми в контрольованому середовищі. Завдяки цьому можливо виконувати визначені користувачем програми всередині пісочниці в ядрі ОС [7, 8]. Це дозволяє розроблювати низькорівневі програми моніторингу мережевих пакетів в ОС Linux та забезпечити оптимальну продуктивність. Для повного використання eBPF потрібна мінімальна версія Linux 4.4.

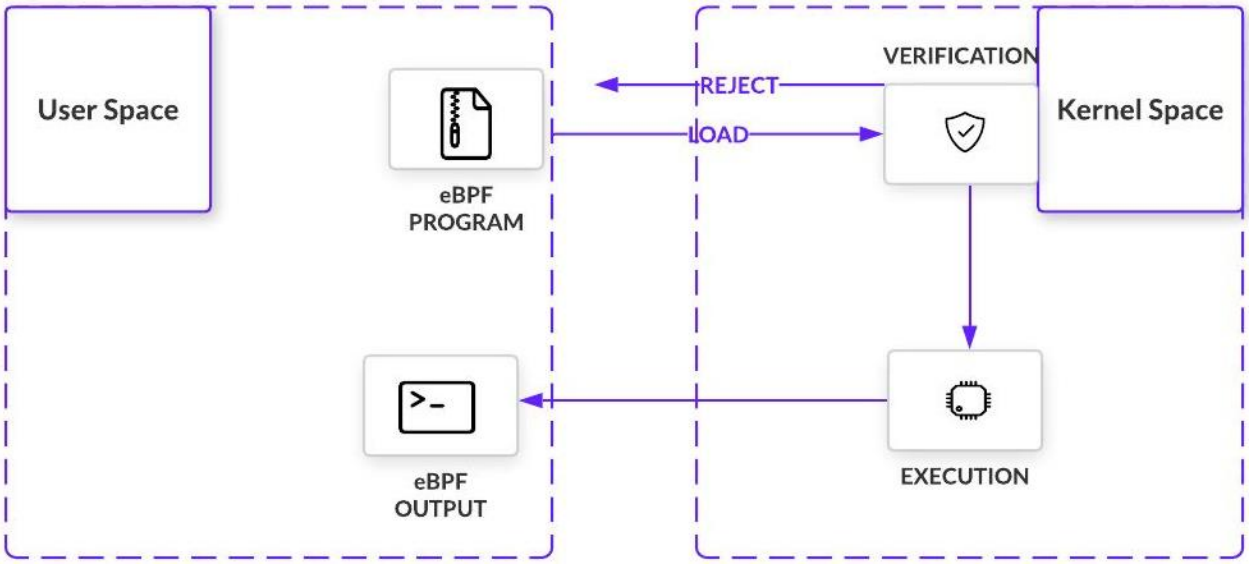

Рисунок 2.10 – Спрощена архітектура eBPF

Після перевірки, байт-код eBPF своєчасно компілюється у власний машинний код. Швидка компіляція під час виконання дає змогу eBPF залишатися продуктивним, навіть якщо він повинен спочатку пройти через віртуальну машину, як показано на рис 2.11.

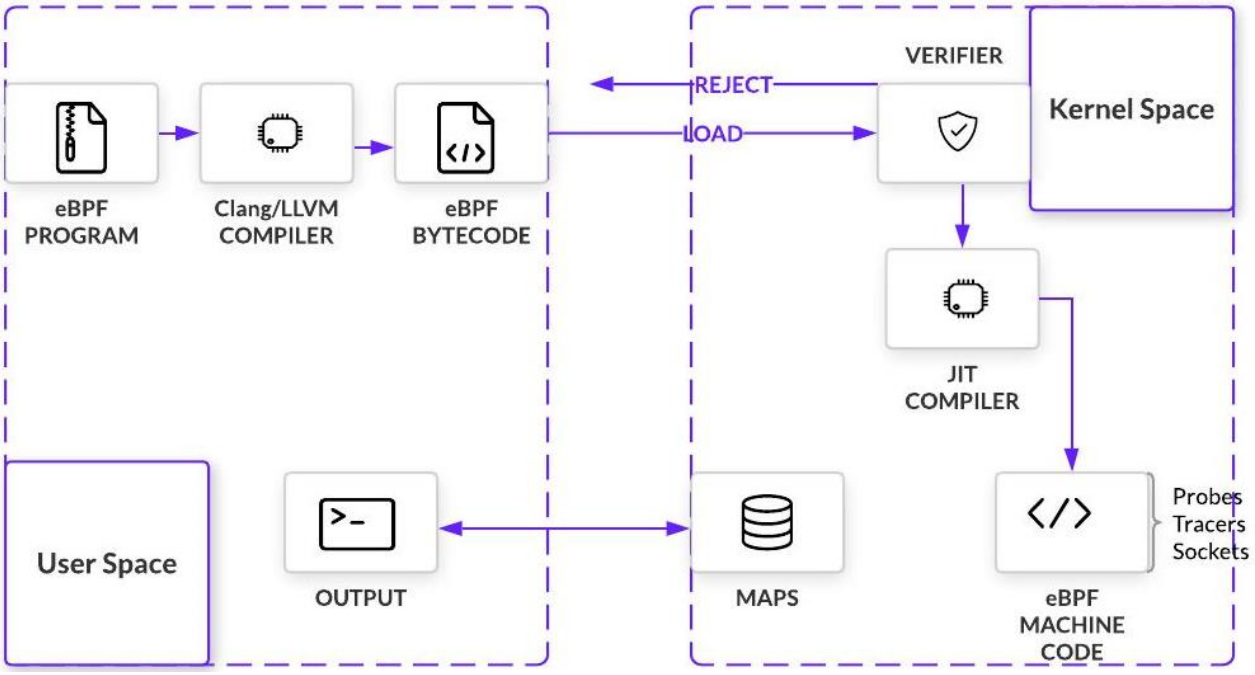

Рисунок 2.11 – Архітектура eBPF в поєднанні з JIT компіляцією

Верифікатор враховує умови того що застосунок не пошкодить і не призведе до збою системи та завжди виконуватиметься до кінця (не перебуватиме в нескінченному циклі).

eBPF має можливість зберігати свій стан і ділитися зібраними даними завдяки ряду структур даних. До них відносяться хеш-таблиці, масиви, кільцевий буфер, стек тощо. Користувачі можуть отримати доступ до цих даних за допомогою системних викликів та з самого застосунку.

Важливим є розуміння того, що eBPF відкриває доступ до подій на рівні ядра без типових обмежень, які виникають під час безпосередньої зміни коду ядра ОС.

Висновки до розділу 2

В другому розділі кваліфікаційної роботи бакалавра описано моделювання застосунку моніторингу трафіку та освітлено алгоритм вирішення головного поставлення завдання. Приділено особливу увагу побудові діаграм взаємодії, послідовності та розгортання. Для діаграми взаємодії створено сценарії використання всіх типів (короткий, поверхневий, повний).

Проведено аналіз діаграми розгортання та показано способи оптимізації навантаження системи завдяки розподілу на окремі сервери та можливості шардування. Показано приклад використання внутрішнього та розподіленого кешу, розібрано переваги та недоліки різних способів шардування серверів Redis кешу.

Описано шаблон взаємодії eBPF з ОС Linux, що включає типи даних та мінімальні вимоги до ОС. Продемонстровано архітектуру eBPF застосунку з виконанням JIT компіляції та приділено увагу процесу верифікації на рівні Kernel.

3 ПРОЄКТУВАННЯ ЗАСТОСУНКУ МОНІТОРИНГУ ТРАФІКУ ТА ОГЛЯД СТЕКУ ТЕХНОЛОГІЙ

3.1 Конструювання UML-діаграм

UML-діаграми представляють собою набір графічних нотацій [9], які використовуються для представлення різних аспектів програмних систем, таких як структура, поведінка та взаємодії. Вони зазвичай використовуються в розробці ПЗ, щоб допомогти розробникам візуалізувати і передати складні системи та їх дизайн.

Таблиця 3.1 – Опис основних UML-діаграм

Назва	Опис
Діаграма класів	Структурна діаграма, яка представляє статичну структуру системи. Вона містить класи, інтерфейси, атрибути, методи та зв’язки між ними.
Діаграма використання	Поведінкова діаграма, яка представляє функціональні вимоги системи. Вона відображає взаємодію між акторами та випадками використання.
Діаграма послідовності	Поведінкова діаграма, яка представляє взаємодію між об’єктами або компонентами в системі протягом певного часу. Вона показує порядок, у якому обмінюються повідомленнями між об’єктами.
Діаграма діяльності	Поведінкова діаграма, яка представляє потік діяльності в системі. Вона формує точки прийняття рішень, паралельні дії та шляхи розгалуження в системі.
Діаграма компонентів	Структурна діаграма, яка містить компоненти, інтерфейси та залежності між ними. Вона показує, як організовані компоненти системи та як вони взаємодіють один з одним.
Діаграма розгортання	Структурна діаграма, яка зображує фізичне розгортання компонентів та вузлів у системі. Вона показує, як компоненти розподіляються в апаратній інфраструктурі.
Діаграма пакетів	Структурна діаграма, яка представляє організацію пакетів та їхні зв’язки в системі. Вона показує, як елементи системи згруповані в пакети та як вони залежать один від одного.

Кінець таблиці 3.1

Діаграма об’єктів	Структурна діаграма, яка представляє стан системи в певний момент часу. Вона показує об’єкти та їхні зв’язки в системі.

Під час моделювання застосунку моніторингу трафіку в другому розділі кваліфікаційної роботи бакалавра показано діаграми взаємодії, діяльності та розгортання. На етапі проєктування у цьому розділі буде розроблено діаграми класів, компонентів, пакетів, станів та переходів.

3.1.1 Діаграма класів серверної частини застосунку

Діаграма класів є фундаментом для об’єктно-орієнтованого моделювання [10], оскільки саме вона визначає основні сутності та зв’язки між ними. На діаграмі класів класи прийнято зображати у вигляді прямокутників із назвою класу зверху. Створена діаграма має 6 основних сутностей та 1 допоміжну.

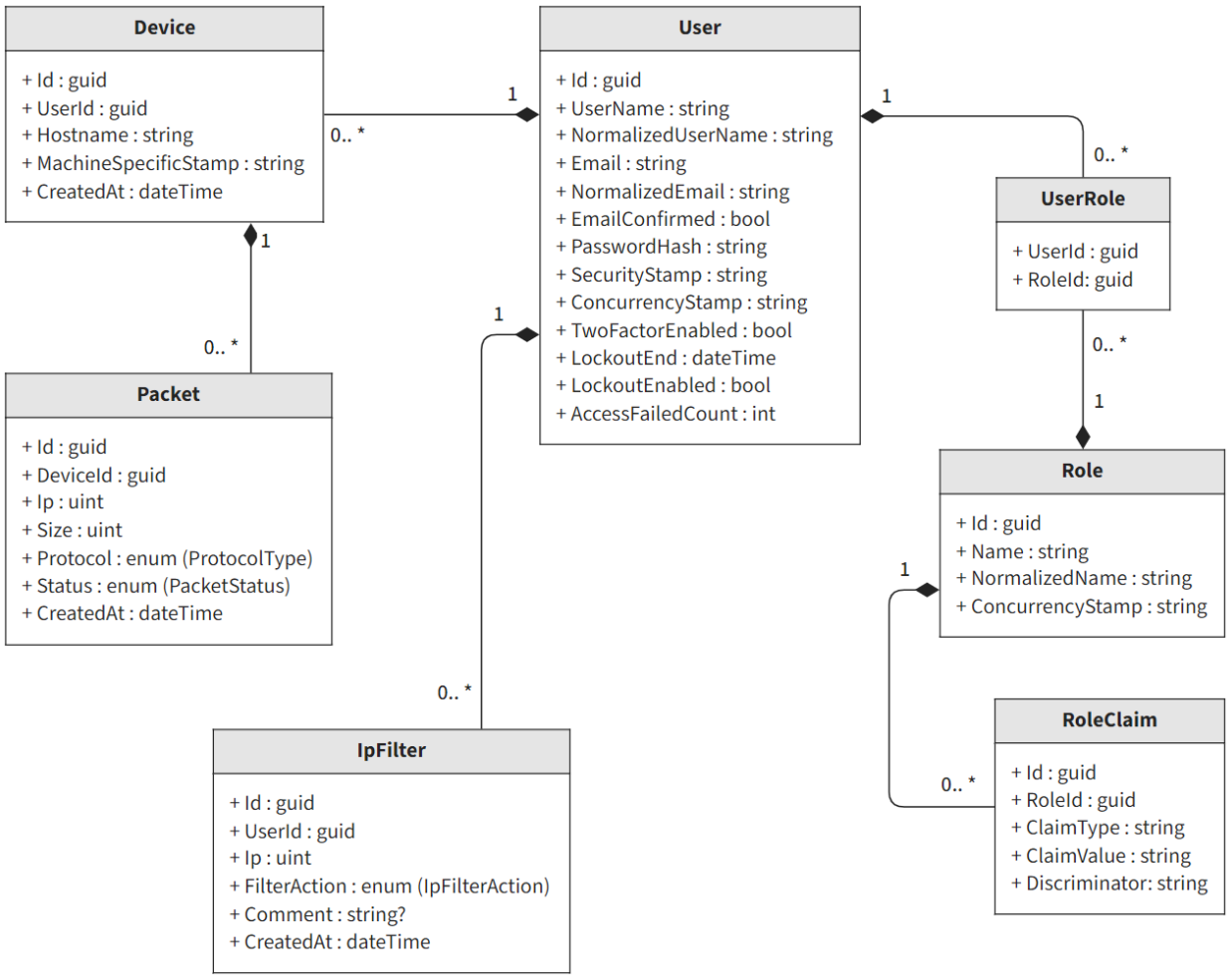

Рисунок 3.1 – Діаграма класів серверної частини

Варто зауважити, що представлені класи не містять методів та використовуються здебільшого для взаємодії з таблицями БД, що пояснюється застосуванням архітектурного підходу CQRS [11].

3.1.2 Діаграма компонентів серверної частини застосунку

Діаграми компонентів використовуються в розробці ПЗ для ілюстрації компонентів, інтерфейсів та залежностей системи. Це візуальне представлення показує, як різні частини програмної системи взаємопов’язані та як вони взаємодіють між собою для виконання функціональних можливостей застосунку.

Компонент на діаграмі компонентів представляє модульну одиницю системи, яка може бути програмним модулем, бібліотекою, підсистемою або будь-якою іншою логічною групою коду.

Діаграма компонентів серверної частини застосунку моніторингу трафіку складається з 4 частин та представлена на рис. 3.2.

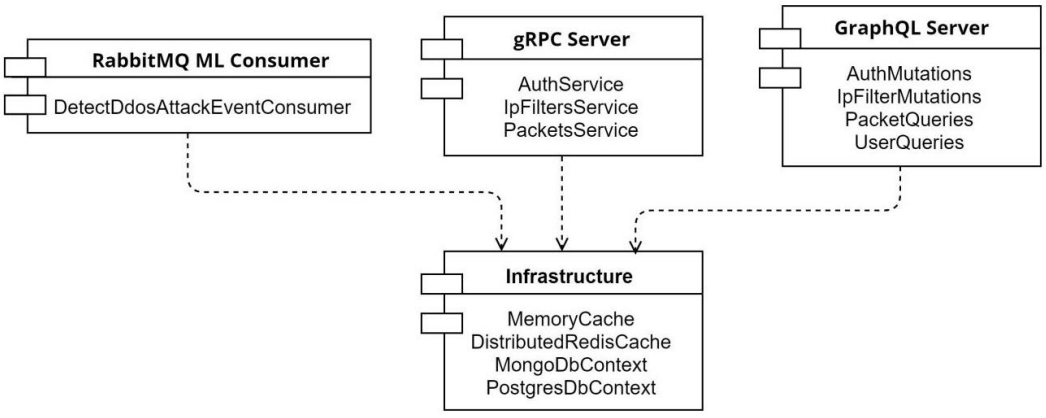

Рисунок 3.2 – Діаграма компонентів серверної частини застосунку

Слід зауважити, що компонент може складатися з інших менших компонентів та може надавати або вимагати певний інтерфейс для зв’язку.

3.1.3 Діаграма пакетів

Діаграми пакетів використовуються в розробці ПЗ для моделювання організації та зв’язків між пакетами, які є логічними групами пов’язаних класів, інтерфейсів та інших елементів у системі. Це може допомогти візуалізувати

загальну структуру системи та зрозуміти зв’язок між її частинами.

Діаграма пакетів застосунку моніторингу трафіку складається з 6 елементів та представлена на рис. 3.3.

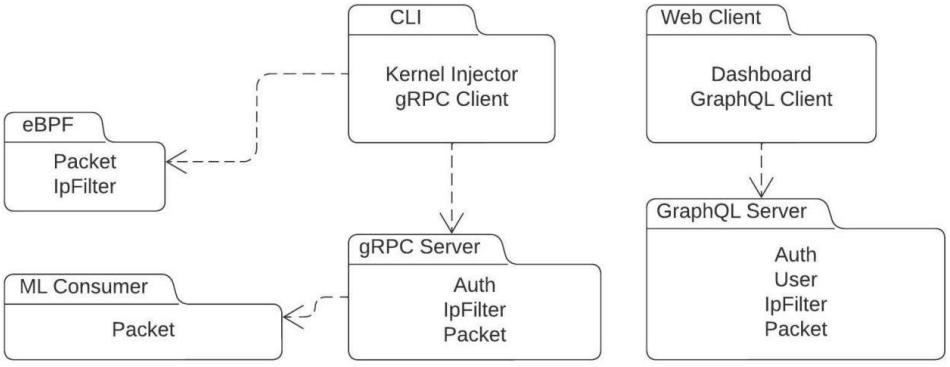

Рисунок 3.3 – Діаграма пакетів

Подібні діаграми особливо корисні для архітекторів ПЗ та розробників, щоб зрозуміти загальну структуру системи та зібрати її компоненти в логічні групи.

3.1.4 Діаграма станів та переходів

Діаграми станів та переходів показують різні стани, в яких може перебувати об’єкт або система, а також переходи між цими станами на основі різних подій або умов. Це є корисним для моделювання поведінки складних систем, таких як програмні застосунки або вбудовані системи, де поведінка системи залежить від її поточного стану та вхідних даних, які вона отримує.

Для майбутньої системи розроблено 3 діаграми станів та переходів:

1) загальна діаграма, демонструє можливості головної сторінки (рис. 3.4);

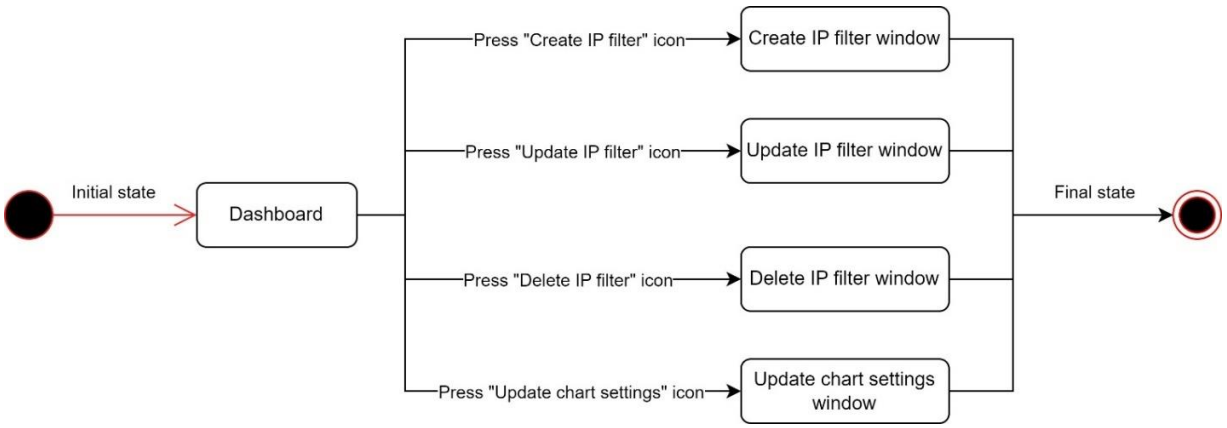

Рисунок 3.4 – Загальна діаграма станів та переходів

2) здійснення операції створення IP-фільтру (рис. 3.5);

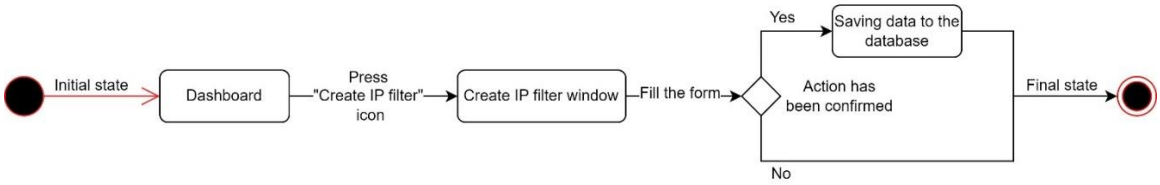

Рисунок 3.5 – Діаграма станів та переходів створення IP-фільтру

3) виконання команди nmt_cli start (рис. 3.6).

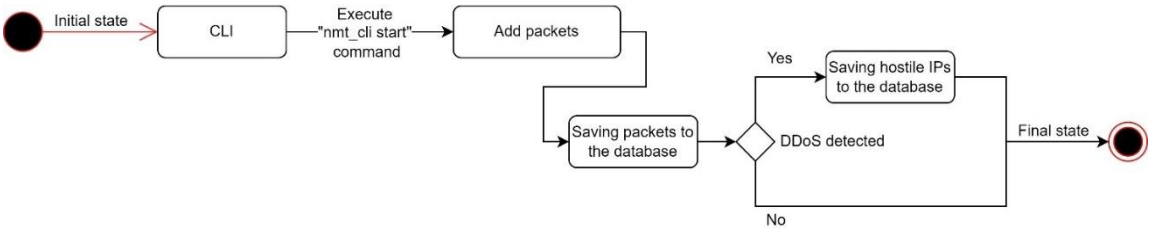

Рисунок 3.6 – Діаграма станів та переходів додавання мережевих пакетів

Діаграми станів є важливим інструментом для моделювання та аналізу і можуть суттєво допомогти покращити дизайн та продуктивність системи.

3.2 Огляд стеку технологій

Стек технологій є важливою частиною кожного застосунку, оскільки це безпосередньо впливає на продуктивність, безпеку та формування вимог до апаратного забезпечення користувача. Для розробки застосунку моніторингу використано наступні технології (рис. 3.7):

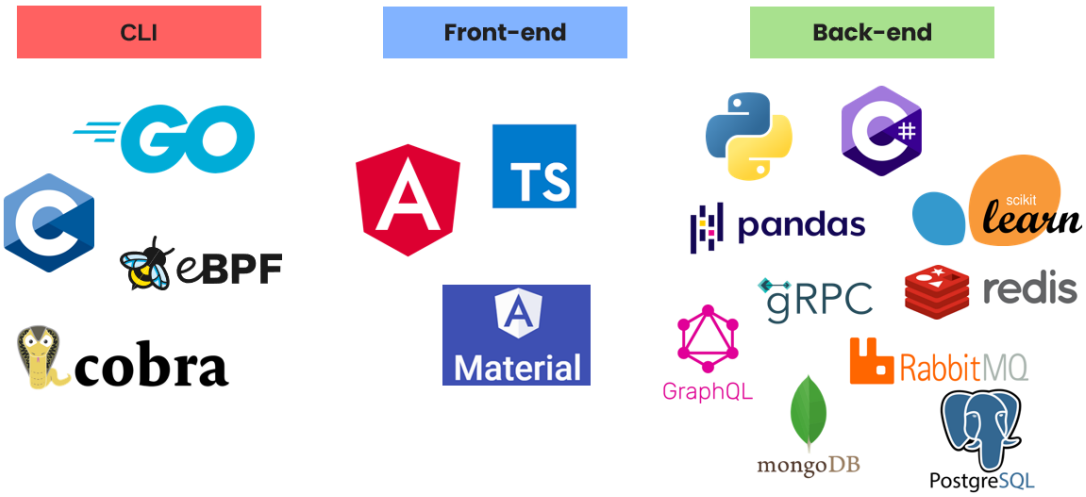

Рисунок 3.7 – Стек технологій застосунку моніторингу трафіку

Під час вибору стеку технологій важливо пам’ятати, що деякі мови програмування та фреймворки оптимізовані для конкретних завдань і можуть виконувати їх швидше та ефективніше.

3.2.1 Використані мови програмування

Мова програмування C широко використовується для системного програмування, наприклад ОС, драйверів та вбудованих систем, завдяки своїй здатності ефективно працювати з апаратними компонентами низького рівня. Вона також використовується для високопродуктивних обчислень, наукового моделювання та графічного програмування. C є структурованою мовою програмування та використовує модульний підхід, який легко читати та підтримувати. Ключовою особливістю є підтримка вказівників, які є змінними, що зберігають адреси пам’яті інших змінних, забезпечуючи більш ефективне використання пам’яті завдяки прямому доступу до неї.

Go або Golang представляє ефективну, надійну та просту у використанні мову програмування, основна увага якої зосереджена на простоті, паралелізмі та масштабованості. Підтримка паралелізму дозволяє виконувати кілька завдань одночасно та базується на концепції goroutines, які є легкими потоками, якими керує середовище виконання Go. Goroutines можуть спілкуватися між собою за допомогою каналів, які використовуються для передачі даних. Останніми роками завдяки своїм особливостям Go набув популярності у сферах хмарних обчислень та мережевого програмування.

TypeScript – це мова програмування, яка є надбудовою над JavaScript та базується на синтаксисі і функціях JavaScript з впровадженням додаткового функціоналу. Створення TypeScript спрямовано на вирішення деяких проблем, з якими стикаються розробники під час роботи з JavaScript, особливо у великих застосунках. Однією з ключових особливостей TypeScript є підтримка статичної типізації, що дозволяє розробникам визначати типи даних змінних, параметрів функцій та типів повертаючих значень під час компіляції. Це допомагає виявляти помилки на ранніх стадіях процесу розробки та полегшує читання і підтримку

коду. TypeScript компілюється в JavaScript, тобто він може працювати на будь-якій платформі, яка підтримує JavaScript. Останніми роками TypeScript набув популярності особливо у сфері вебзастосунків та здатний робити великомасштабні програми зручнішими для обслуговування.

Python представляє собою одну з найпопулярніших інтерпретованих мов програмування, яка відома завдяки своїй читабельності, простоті і універсальності та широко використовується в різноманітних програмах, включаючи веб, аналіз даних, штучний інтелект та наукові обчислення. Python підкреслює читабельність коду та дозволяє розробникам виражати концепції меншою кількістю рядків коду, ніж більшість інших мов програмування. Python підтримує як функціональні, так і об’єктно-орієнтовані парадигми програмування, надаючи розробникам гнучкість у написанні коду та має велику стандартну бібліотеку, яка включає модулі для роботи з регулярними виразами, БД та мережевим обладнанням.

C# – це сучасна об’єктно-орієнтована мова програмування, розроблена Microsoft як частина її .NET платформи. C# підтримує різноманітні парадигми програмування, включаючи імперативну, об’єктно-орієнтовану та функціональну. Вона також містить широкий спектр вбудованих функцій та бібліотек, таких як LINQ, що дозволяє розробникам робити запити до колекцій за допомогою синтаксису, подібного до SQL. Однією з ключових особливостей C# є підтримка асинхронного програмування, що дозволяє виконувати кілька завдань одночасно.

3.2.2 Технології розробки CLI

Kernel приймає eBPF програми у формі байт-коду. Можна написати цей байт-код вручну, майже так само, як можливо написати код програми мовою асемблера, але загалом для людей практичніше використовувати мову програмування вищого рівня, яка буде скомпільована в байт-код.

eBPF застосунки не можна писати довільними мовами високого рівня з кількох причин:

1) компілятор мови повинен підтримувати генерацію байт-коду eBPF, який очікує ядро ОС;

2) багато компільованих мов програмування мають функції середовища виконання (керування пам’яттю, збирання сміття тощо), це робить їх непридатними для eBPF застосунків.

На даний момент єдиними варіантами для написання eBPF застосунків є мови програмування C та Rust [12]. Переважна більшість доступного на сьогоднішній день eBPF коду написана на C, це легко пояснюється тим, що саме цю мову використовує ядро Linux.

Для реалізації CLI частини застосунку використано достатню кількість різних Go бібліотек, до основних відносяться:

1) cobra – бібліотека для створення потужних сучасних CLI застосунків, яка використовується в багатьох великих проєктах, таких як Kubernetes, Hugo та GitHub CLI;

2) gRPC-go – високопродуктивна бібліотека з відкритим вихідним кодом яка реалізує gRPC на мові програмування Go з використанням протоколу HTTP 2.0.

3) ebpf-go – бібліотека, яка надає утиліти для завантаження, компіляції та налагодження eBPF програм.

Для виконання високошвидкісної обробки пакетів у eBPF застосунках необхідно використовувати XDP обробники [13], які можна безпосередньо підключати до мережевих інтерфейсів. Щоразу, коли на мережевий інтерфейс надходить новий пакет, XDP отримує контроль над ним та може дуже швидко виконувати необхідні операції, до яких відносяться:

1) XDP_DROP – відвергає та не обробляє пакет. Завдяки цьому можна аналізувати мережевий трафік та використовувати фільтри у режимі реального часу, щоб відкидати певні типи пакетів (наприклад від недоброзичливих користувачів);

2) XDP_PASS – вказує на те, що пакет має бути направлений у звичайний мережевий стек для подальшої обробки. Програма може змінити вміст пакета, якщо це необхідно;

3) XDP_TX – пересилає пакет (який міг бути змінений) на той самий мережевий інтерфейс, який його отримав;

4) XDP_REDIRECT – обходить звичайний мережевий стек і перенаправляє пакет через інший адаптер до мережі.

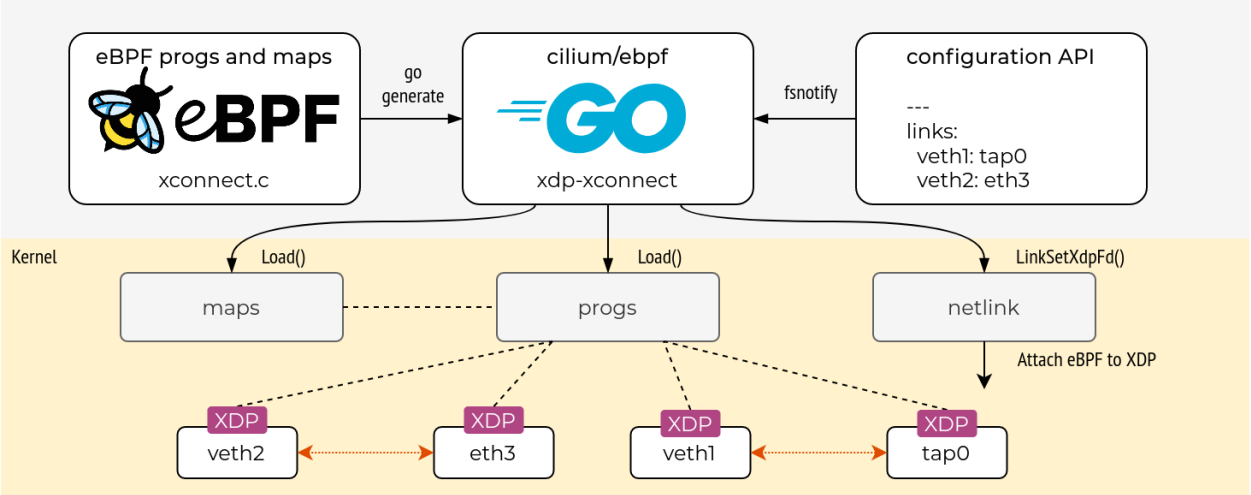

Рисунок 3.8 – Високорівнева архітектура завантаження C обробників через Go

eBPF застосунки виконуються безпосередньо в ядрі ОС, що дозволяє їм отримувати доступ та маніпулювати системними ресурсами з мінімальними витратами.

3.2.3 Технології розробки front-end

До основних засобів розробки вебклієнту можна віднести Angular, Material Design, Apollo Client та ApexCharts.

Angular є популярним фреймворком з відкритим вихідним кодом, який використовується для створення веб та мобільних застосунків. Він надає надійний набір інструментів та функцій, які дозволяють розробникам створювати динамічні, адаптивні та масштабовані програми. Деякі з ключових переваг використання Angular включають:

– двостороння прив’язка даних;
– підвищена продуктивність та масштабованість;
– ін’єкція залежностей;
– кросплатформенна розробка;
– компонентна архітектура;
– потужні можливості шаблонування.

Material Design – це бібліотека для розробки дизайну [14], яка надає набір вказівок і принципів для проектування інтерфейсів користувача та забезпечує послідовний, єдиний вигляд на різних пристроях та платформах, полегшуючи користувачам навігацію та використання програм. Вона заснована на принципах тактильного реалізму та осмисленого руху.

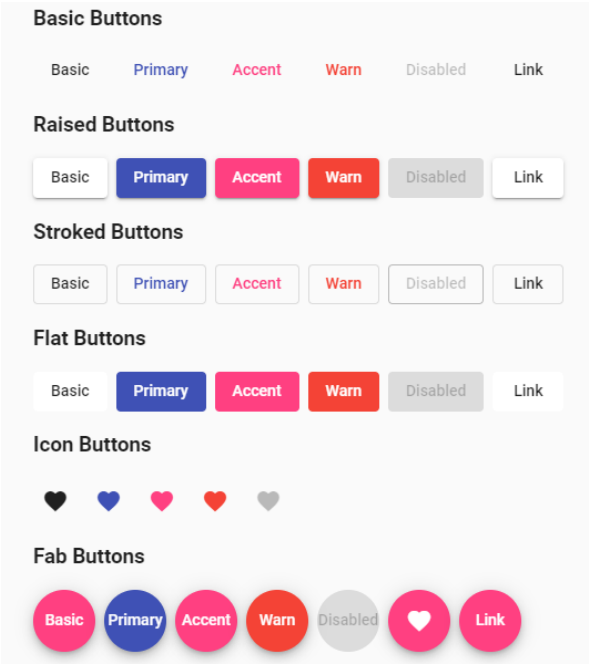

Рисунок 3.9 – Приклад дизайну кнопок Material Design

Apollo Client представляє собою JavaScript бібліотеку, яка допомагає розробникам створювати програми за допомогою GraphQL шлюзу. Вона забезпечує простий у використанні інтерфейс для зв’язку з серверами GraphQL та обробки відповідей на запити. Однією з головних переваг використання Apollo Client є його здатність керувати станом програми передбачуваним та ефективним способом.

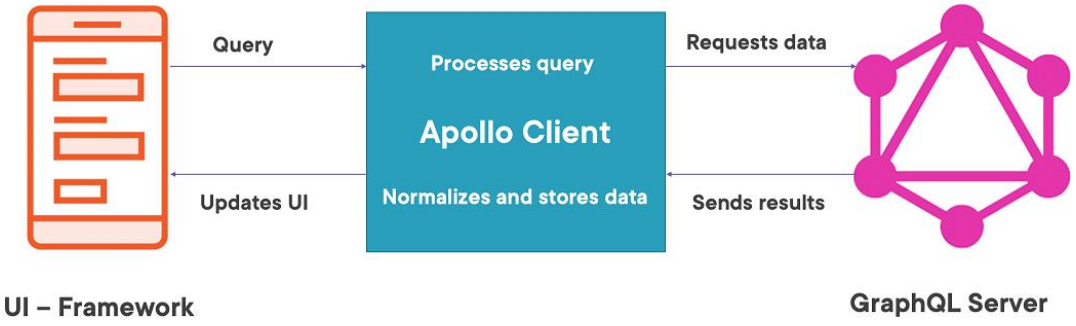

Рисунок 3.10 – Архітектура роботи Apollo Client

ApexCharts – це сучасна JavaScript бібліотека для побудови діаграм, яка дозволяє розробникам створювати красиві інтерактивні візуалізації у вебзастосунках.

Рисунок 3.11 – Приклади діаграм ApexCharts

Основою цієї бібліотеки є використання новітніх технологій для візуалізації широкого спектру різних діаграм, включаючи лінійні, стовпчасті та секторні.

3.2.4 Технології розробки back-end

Для розробки більшої back-end частини використано безкоштовний C# фреймворк з відкритим вихідним кодом під назвою ASP.NET. Він є незалежною від платформи структурою, це означає, що розробники можуть використовувати його для створення вебзастосунків, які будуть працювати на різних ОС, включаючи Windows, Linux та macOS. До основних переваг ASP.NET відносять швидкий розвиток, масштабованість та безпеку.

Рисунок 3.12 – Логотип ASP.NET Core

Для впровадження в застосунок машинного навчання використано дві популярні Python бібліотеки scikit-learn та pandas. Разом вони створюють потужний інструментарій для аналізу даних і машинного навчання та дають змогу безперебійно обробляти, аналізувати та моделювати дані, що спрощує отримання інформації та прогнозування на основі існуючих вибірок даних.

За допомогою scikit-learn можливо легко реалізувати різноманітні алгоритми машинного навчання, такі як лінійна та логістична регресія, дерева рішень, випадкові ліси та нейронні мережі.

Pandas представляє собою бібліотеку для обробки та аналізу даних. Він надає структури даних для ефективного зберігання та маніпулювання великими наборами інформації, а також інструменти для очищення, перетворення та агрегування даних.

Для зберігання, кешування та обробки даних використано наступні сховища:

1) PostgreSQL – СКБД, яка відома своєю міцністю, надійністю та високою продуктивністю. Представляє собою систему з відкритим вихідним кодом, яка підтримує широкий спектр запитів SQL та часто використовується для застосунків, які вимагають складного моделювання даних та транзакцій.

2) MongoDB – документо-орієнтована БД, яка розроблена для гнучкості та масштабованості. Найчастіше використовується для вебзастосунків та аналізу великих даних, оскільки може легко обробляти великі обсяги неструктурованої інформації.

3) Redis – популярне сховище з відкритим кодом, яке використовується для кешування, обміну повідомленнями в реальному часі та обробки даних. Він відомий своєю швидкістю, масштабованістю та застосовуваністю для різноманітних випадків використання, таких як керування сеансами, аналітика в реальному часі та планування завдань.

Для оптимізації роботи системи обрано наступні технології взаємодії між клієнтськими застосунками та серверами: GraphQL та gRPC.

GraphQL – це мова запитів до API та середовище виконання запитів із

наявними даними [15]. GraphQL надає повний і зрозумілий опис даних у вашому API та дає клієнтам можливість запитувати саме те, що їм потрібно, для заощадження інтернет-трафіку користувачів.

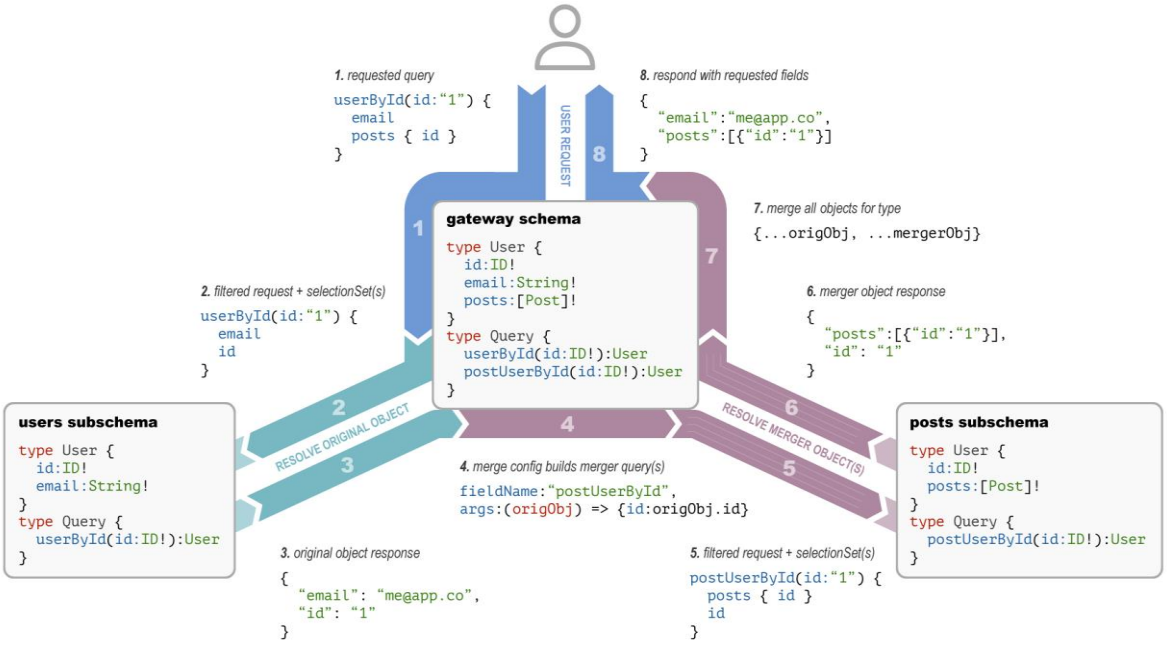

Рисунок 3.13 – Схема роботи GraphQL

gRPC побудовано на основі протоколу HTTP 2.0 який є ефективнішим, ніж попередня версія (HTTP 1.1) та надає низку функцій які роблять його добре придатним для сучасних вебзастосунків [16]. gRPC використовує буфери протоколів (Protocol Buffers) для серіалізації даних та передачі повідомлень, що є більш компактним та ефективним способом кодування даних порівняно з JSON або XML, які використовуються в REST API.

Висновки до розділу 3

В третьому розділі кваліфікаційної роботи бакалавра описано проєктування застосунку моніторингу трафіку та проведено огляд стеку використаних технологій CLI, front-end та back-end частин системи. Приділено увагу побудові діаграм класів, компонентів, пакетів, станів та переходів.

Описано використані мови програмування, фреймворки, бібліотеки, бази даних та способи обміну даними з серверами. Наведено приклад архітектури eBPF застосунку на мові програмування C та визначено кроки верифікації Kernel.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ МОНІТОРИНГУ ТРАФІКУ

4.1 Огляд команд CLI

Розроблений CLI застосунку моніторингу трафіку містить 3 основні команди (start, auth login, auth logout) та коротку довідку по кожній з них.

```
daniel@Daniel:~/git/network-monitoring-tool/cli$ ./bin/nmt_cli --help
nmt_cli - a simple CLI to inspect activity in your network

Usage:
  nmt_cli [command]

Available Commands:
  auth        Authenticate with network monitoring tool
  completion  Generate the autocompletion script for the specified shell
  help        Help about any command
  start       Start the processing of xpd packets

Flags:
  -h, --help   help for nmt_cli

Use "nmt_cli [command] --help" for more information about a command.
```

Рисунок 4.1 – Перегляд загальної довідки CLI

Перед взаємодією з іншими командами CLI користувачу спочатку необхідно авторизуватися в системі виконавши команду auth login.

```
daniel@Daniel:~/git/network-monitoring-tool/cli$ ./bin/nmt_cli auth login --help
Authenticate with a NMT host

Usage:
  nmt_cli auth login [flags]

Flags:
  -h, --help   help for login
```

Рисунок 4.2 – Перегляд довідки команди auth login

Основною командою безперечно є start, оскільки вона збирає інформацію про отримані мережеві пакети на відправляє її на gRPC сервер.

```
daniel@Daniel:~/git/network-monitoring-tool/cli$ ./bin/nmt_cli start --help
Start the processing of xpd packets

Usage:
  nmt_cli start <protocol> [flags]

Flags:
  -h, --help    help for start
      --stats   Print stats of the network to console
```

Рисунок 4.3 – Перегляд довідки команди start

Щоб переконатися в роботі застосунку запустимо команду start для інтерфейсу eth0 передавши прапорець --stats для виведення інформації на консоль.

```
daniel@Daniel:~/git/network-monitoring-tool/cli$ sudo ./bin/nmt_cli start eth0 --stats
2023/04/30 13:07:48 Attached XDP program to iface "eth0" (index 2)
2023/04/30 13:07:51 IP: 13.69.106.217, size: 54, status: 1, protocol: 6
2023/04/30 13:07:51 IP: 13.69.106.217, size: 54, status: 1, protocol: 6
2023/04/30 13:07:51 IP: 13.69.106.217, size: 54, status: 1, protocol: 6
2023/04/30 13:07:51 IP: 172.30.224.1, size: 148, status: 1, protocol: 17
2023/04/30 13:07:51 IP: 172.30.224.1, size: 148, status: 1, protocol: 17
2023/04/30 13:07:51 IP: 172.30.224.1, size: 74, status: 1, protocol: 1
```

Рисунок 4.4 – Виконання команди start

Трасування допоможе налагоджувати взаємодію з інтерфейсом підключення за рахунок опису кожного отриманого мережевого пакету.

4.2 Огляд дизайну вебклієнту

Розроблений вебклієнт застосунку моніторингу трафіку містить 4 основні сторінки (авторизація, реєстрація, підтвердження електронної пошти, головна сторінка).

Для авторизації в системі через вебклієнт користувачу необхідно ввести ім’я, пароль та натиснути кнопку «Вхід». Якщо все введено коректно то користувач перенаправляється на головну сторінку.

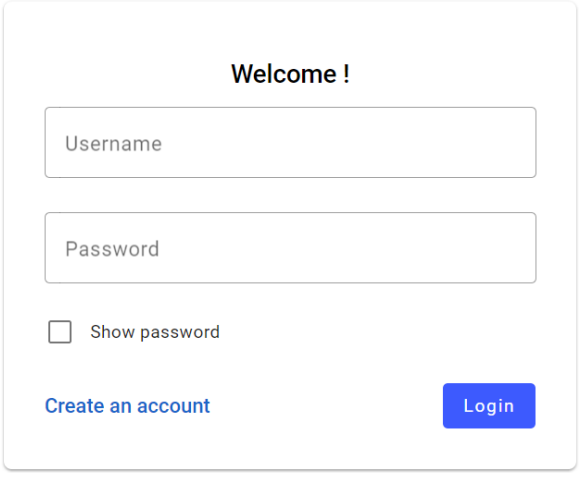

Рисунок 4.5 – Сторінка авторизації

Для створення нового облікового запису користувачу необхідно ввести ім’я, електрону пошту, пароль, повтор паролю та натиснути кнопку «Зареєструватися».

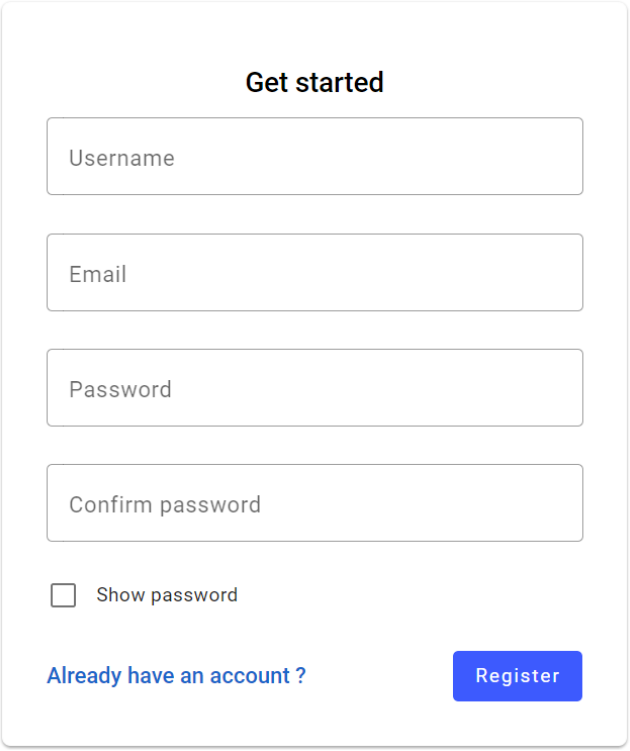

Рисунок 4.6 – Сторінка реєстрації користувача

Наступним кроком після введення даних під час реєстрації буде отримання коду підтвердження на електрону пошту та активація акаунту.

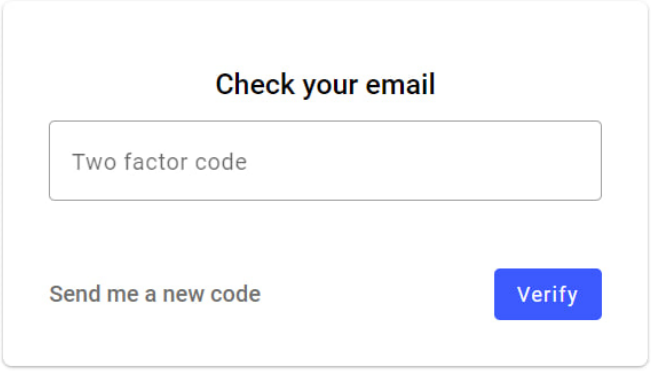

Рисунок 4.7 – Сторінка підтвердження електронної пошти

Для авторизованого користувача на головній панелі доступні наступні блоки:

1) Трафік пакетів (packets traffic) – відповідає за графічне відображення кількісного представлення пакетів за певний часовий період (доба або тиждень).

2) IP-фільтри (IP filters) – представляє собою таблицю з повним набором опцій для модифікації фільтрів (додавання, оновлення, видалення).

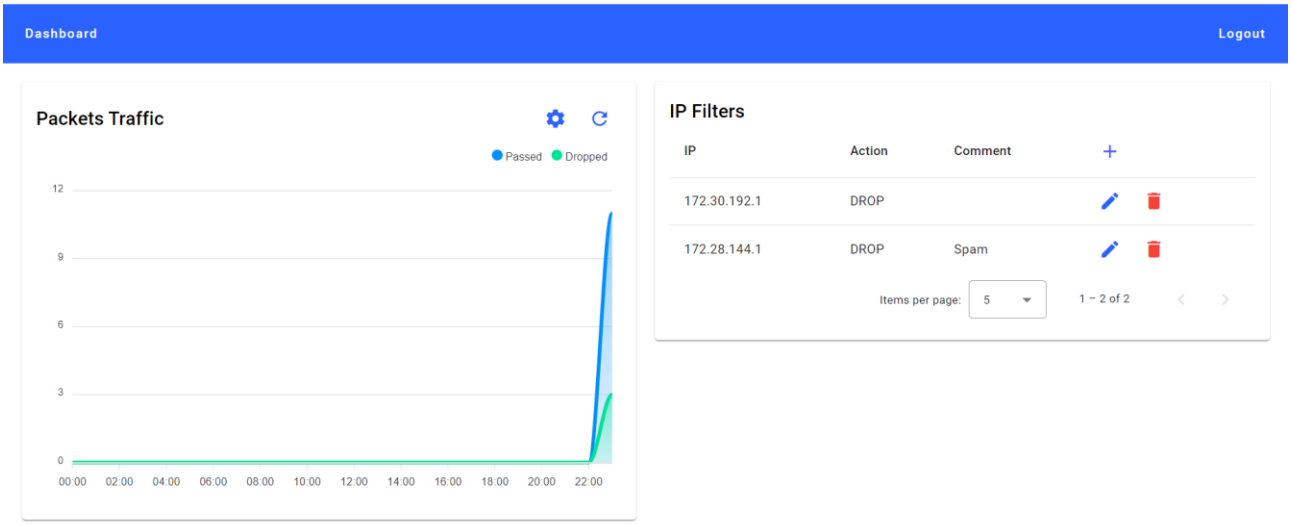

Рисунок 4.8 – Загальний вигляд головної сторінки

Під час роботи з майже кожним застосунок від користувача вимагається змінювати певні дані тим чи іншим шляхом. Застосунок моніторингу трафіку не є винятком, тому прийнято рішення показати вікна модифікації на прикладі створення IP-фільтрів та зміни часового періоду інфографіки мережевих пакетів.

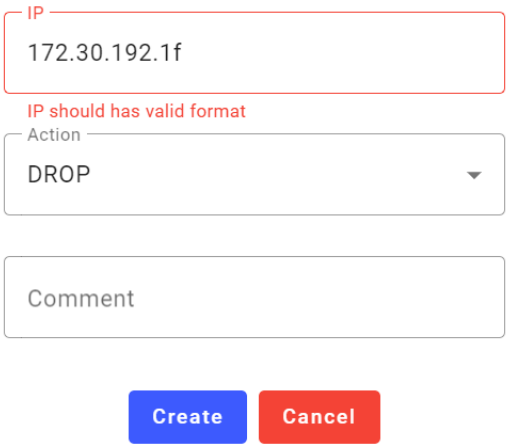

Рисунок 4.9 – Вікно створення IP-фільтра

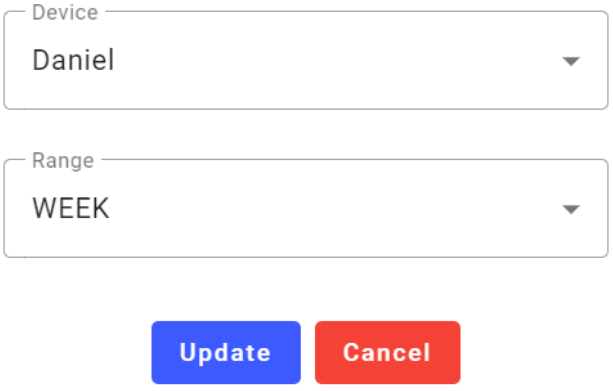

Рисунок 4.10 – Вікно вибору часового періоду інфографіки отриманих пакетів

Під час обробки дій користувача вебклієнт робить запити на GraphQL сервер та отримує лише необхідні дані з кешу.

4.3 Програмна реалізація основних частин системи

4.3.1 eBPF XDP застосунок моніторингу мережевих пакетів

Для зберігання інформації в eBPF застосунках необхідно використовувати колекції створені через SEC(“maps”), до них відносяться:

– BPF_MAP_TYPE_UNSPEC;
– BPF_MAP_TYPE_HASH;
– BPF_MAP_TYPE_ARRAY;
– BPF_MAP_TYPE_PROG_ARRAY;
– BPF_MAP_TYPE_PERF_EVENT_ARRAY;
– BPF_MAP_TYPE_PERCPU_HASH;
– BPF_MAP_TYPE_STACK_TRACE;
– BPF_MAP_TYPE_LRU_HASH.

В даному випадку для швидкого та оптимального доступу до даних використовується черга та хеш-таблиця.

```
// Value - Packet
struct bpf_map_def SEC("maps") packets_queue = {
        .type        = BPF_MAP_TYPE_QUEUE,
        .key_size    = 0,
        .value_size  = sizeof(struct Packet),
        .max_entries = queue_max_entries,
        .map_flags   = 0
};

// Key   - IPv4 address
// Value - IP Filter Action
struct bpf_map_def SEC("maps") ip_filters_map = {
        .type        = BPF_MAP_TYPE_LRU_HASH,
        .key_size    = sizeof(uint),
        .value_size  = sizeof(enum IpFilterAction),
        .max_entries = map_max_entries,
        .map_flags   = 0
};
```

Рисунок 4.11 – Створення колекцій для зберігання даних

Для доступу до даних про мережевий пакет можна використати структуру xdp_md. Завдяки перевірці наявності IP-фільтру з певним IP, можна досягти фільтрації трафіку.

```
SEC("xdp")
int bpf_xdp_handler(struct xdp_md *ctx) {
    void *data     = (void *)(long)ctx->data;
    void *data_end = (void *)(long)ctx->data_end;

    // Parse the ethernet header.
    struct ethhdr *eth = data;
    if ((void *)(eth + 1) > data_end) {...}

    // If the protocol is not IPv4.
    if (ntohs(eth->h_proto) != ETH_P_IP) {...}

    // Parse the IP header.
    struct iphdr *iph = data + sizeof(struct ethhdr);
    if ((void *)(iph + 1) > data_end) {...}

    int XDP_ACTION = XDP_PASS;

    // Check ip filters.
    uint ip = ntohl(iph->saddr);
    enum IpFilterAction *ip_filter_action = bpf_map_lookup_elem(&ip_filters_map, &ip);
    if (ip_filter_action) {
        switch (*ip_filter_action) {...}
    }
```

Рисунок 4.12 – Отримання інформації про пакет та фільтрація трафіку

Далі всю необхідну інформацію можна зібрати в загальний об’єкт та передати в чергу, яку оброблює CLI.

```
struct Packet packet;
__builtin_memset(&packet, 0, sizeof(packet));

packet.ip = ip;
packet.size = data_end - data;
packet.protocol = iph->protocol;
packet.status = XDP_ACTION == XDP_DROP ? Dropped : Passed;

// Add information about packet to queue.
int error = bpf_map_push_elem(&packets_queue, &packet, BPF_ANY);
if (error != 0) {
    bpf_printk("An error was occurred while adding to queue (code: %i)", error);
}
```

Рисунок 4.13 – Передача інформації про мережевий пакет в CLI

Черги мають ефективні операції вставки та видалення з часовою складністю O (1), що робить їх хорошим вибором для ситуацій, які потребують частого вставлення та видалення елементів.

4.3.2 Написання GraphQL запитів

GraphQL використовується для заощадження трафіку користувачів шляхом вибірки лише необхідних в даний момент даних, що може бути дуже корисним особливо в країнах де доступ в мережу Інтернет коштує досить дорого. Для написання GraphQL запитів необхідно використовувати спеціальний синтаксис.

```
query GetUserInfo {
  userInfo {
    id,
    username,
    email,
    devices {
      id,
      hostname,
      machineSpecificStamp,
      createdAt
    },
    ipFilters {
      id,
      ip,
      filterAction,
      comment,
      createdAt
    }
  }
}
```

Рисунок 4.14 – GraphQL запит для отримання інформації про користувача

Після зберігання GraphQL запиту до файлу та налаштування Apollo Client, розробники можуть виконувати запит викликавши метод fetch.

```
@Injectable({
  providedIn: 'root'
})
export class UsersService {
  no usages  Danylo Boiko
  constructor(private readonly _getUserInfo: GetUserInfoGQL) {
  }

  1 usage  Danylo Boiko
  public getUserInfo() {
    return this._getUserInfo.fetch();
  }
}
```

Рисунок 4.15 – Виклик GraphQL запиту через Apollo Client

Далі можна використати створений UsersService для обробки отриманих даних в своїх цілях.

```
private getUserInfo(): void {
  this._usersService
    .getUserInfo() …
    .pipe(
      untilDestroyed(this),
      filter( predicate: response => !response.loading),
      map( project: response => (<ApolloQueryResult<GetUserInfoQuery>>response).data!.userInfo)
    ) Observable<{…}>
    .subscribe( observer: {
      next: (user: UserDto) => {
        this.user = user;
        this.dashboardLoaded = true;
      },
      error: (error: ApolloError) => this._toasterService.showError(error.message)
    });
}
```

Рисунок 4.16 – Обробка результату GraphQL запиту

Обробник помилок GraphQL спрацьовує, коли на сервері трапляється непередбачувана ситуація, наприклад помилка БД.

4.3.3 Написання proto контрактів та їх реалізація

Файли .proto [17] описують структуру користувацьких типів за допомогою спеціальної мови, яка містить стандартні типи даних. Після створення .proto файлу можна використати спеціалізований protoc компілятор для генерації коду необхідної мови високого рівня.

```
syntax = "proto3";

option csharp_namespace = "Nmt.Grpc.Protos";
option go_package = "pkg/grpc";

package auth;

service Auth {
  rpc Login(LoginRequest) returns (AuthResponse);
  rpc RefreshToken(RefreshTokenRequest) returns (AuthResponse);
}

message LoginRequest {
  string username = 1;
  string password = 2;
  string hostname = 3;
  string machineSpecificStamp = 4;
}

message RefreshTokenRequest {
  string accessToken = 1;
  string refreshToken = 2;
}

message AuthResponse {
  string accessToken = 1;
  string refreshToken = 2;
}
```

Рисунок 4.17 – Приклад .proto файлу авторизації

Щоб реалізувати .proto файл у C# та перевизначити методи, необхідно створити новий C# клас, який успадковує згенерований клас з .proto файлу та перевизначає методи базового класу за допомогою ключового слова override.

```
public override async Task<AuthResponse> Login(LoginRequest request, ServerCallContext context)
{
    var tokens :TokenDto = await _mediator.Send( request: new LoginCommand
    {
        Username = request.Username,
        Password = request.Password,
        Hostname = request.Hostname,
        MachineSpecificStamp = request.MachineSpecificStamp
    }, context.CancellationToken); // Task<TokenDto>

    return new AuthResponse
    {
        AccessToken = tokens.AccessToken,
        RefreshToken = tokens.RefreshToken
    };
}

0+2 usages  Danylo Boiko
public override async Task<AuthResponse> RefreshToken(RefreshTokenRequest request, ServerCallContext context)
{
    var tokens :TokenDto = await _mediator.Send( request: new RefreshTokenCommand
    {
        AccessToken = request.AccessToken,
        RefreshToken = request.RefreshToken
    }, context.CancellationToken); // Task<TokenDto>

    return new AuthResponse
    {
        AccessToken = tokens.AccessToken,
        RefreshToken = tokens.RefreshToken
    };
}
```

Рисунок 4.18 – Приклад реалізації .proto файлу авторизації

Зазвичай подібний підхід використовується для визначення API та міжпроцесного зв’язку в розподілених системах.

4.3.4 Побудова діаграми мережевих пакетів

Щоб агрегувати дані з MongoDB за допомогою C#, необхідно використати спеціальну бібліотеку (MongoDB Driver [18]). Драйвер надає вільний API, який дозволяє створювати конвеєри агрегації за допомогою коду C#.

Для створення конвеєру потрібно виконати метод Aggregate до необхідної колекції. Далі необхідно зробити фільтрацію по Id пристрою і часовому діапазону

та згрупувати мережені пакети по дням або годинам.

```
private async Task<Dictionary<string, int>> GetPacketsAggregate(Guid deviceId, DateTime dateFrom, DateTime dateTo,
    DateRangeMode dateRangeMode, CancellationToken cancellationToken)
{
    var packetsAggregate :List<{Id,Count}>? = await _dbContext.Packets // IMongoCollection<Packet>
        .Aggregate()
        .Match(p :Packet => p.DeviceId == deviceId && p.CreatedAt >= dateFrom && p.CreatedAt <= dateTo) // IAggregateFluent<Packet>
        .Group( id: p :Packet => new
        {
            DateTime = dateRangeMode == DateRangeMode.Day
                ? new DateTime(p.CreatedAt.Year, p.CreatedAt.Month, p.CreatedAt.Day, p.CreatedAt.Hour, minute: 0, second: 0)
                : new DateTime(p.CreatedAt.Year, p.CreatedAt.Month, p.CreatedAt.Day, hour: 0, minute: 0, second: 0),
            Status = p.Status
        }, group: g :IGrouping<{DateTime,Status},...> => new
        {
            Id = g.Key,
            Count = g.Count()
        }) // IAggregateFluent<{Id,Count}>
        .ToListAsync(cancellationToken); // Task<List<...>>

    return packetsAggregate.ToDictionary(k :{Id,Count} => $"{k.Id.DateTime}-{k.Id.Status}", v :{Id,Count} => v.Count);
}
```

Рисунок 4.19 – Запит до БД для отримання інформації про мережеві пакети

Щоб налаштувати вигляд та поведінку ApexCharts діаграм, необхідно використовувати об’єкт спеціальних параметрів який підтримує бібліотека.

```
private getChartOptions(packetsChartData: PacketsChartDataDto, dateRangeMode: DateRangeMode) {
  const series = packetsChartData.series.map((el :… ) => ({
    name: el.key,
    data: el.value
  }));

  return {
    chart: {
      height: 380,
      type: 'area',
      toolbar: false
    },
    dataLabels: { enabled: false},
    stroke: { curve: 'smooth' },
    series: series,
    xaxis: {
      type: 'datetime',
      categories: packetsChartData.categories,
      labels: { datetimeUTC: false }
    },
    tooltip: {
      x: {
        format: dateRangeMode == DateRangeMode.Day ? 'dd/MM/yy HH:mm' : 'dd/MM/yy'
      }
    },
    legend: { position: 'top', horizontalAlign: 'right' }
  };
}
```

Рисунок 4.20 – Створення конфігурації для ApexCharts діаграми

Дані для діаграм можна передавати безпосередньо в об’єкт конфігурації або отримувати їх з зовнішнього джерела.

4.3.5 Відправка повідомлень на пошту

SMTP – це стандартний протокол, який використовується для надсилання та отримання електронних повідомлень через мережу Інтернет. Клієнт електронної пошти надсилає повідомлення на сервер електронної пошти, який потім передає повідомлення на електрону пошту користувача.

Щоб надіслати електронний лист за допомогою SMTP в мові програмування C# необхідно використовувати клас SmtpClient.

```
public async Task SendEmailAsync(string receiverEmail, string subject, string text, CancellationToken cancellationToken)
{
    var emailMessage = new MailMessage(_fromAddress, new MailAddress(receiverEmail))
    {
        Subject = subject,
        Body = text
    };

    using var smtpClient = new SmtpClient()
    {
        Host = "smtp.gmail.com",
        Port = 587,
        EnableSsl = true,
        DeliveryMethod = SmtpDeliveryMethod.Network,
        UseDefaultCredentials = false,
        Credentials = new NetworkCredential( userName: _googleSmtpConfig.Email, _googleSmtpConfig.Password)
    };

    await smtpClient.SendMailAsync(emailMessage, cancellationToken);
}
```

Рисунок 4.21 – Метод для відправлення електронного листа

В застосунку моніторингу трафіку надсилання електронних листів необхідно для підтвердження електронної пошти користувача.

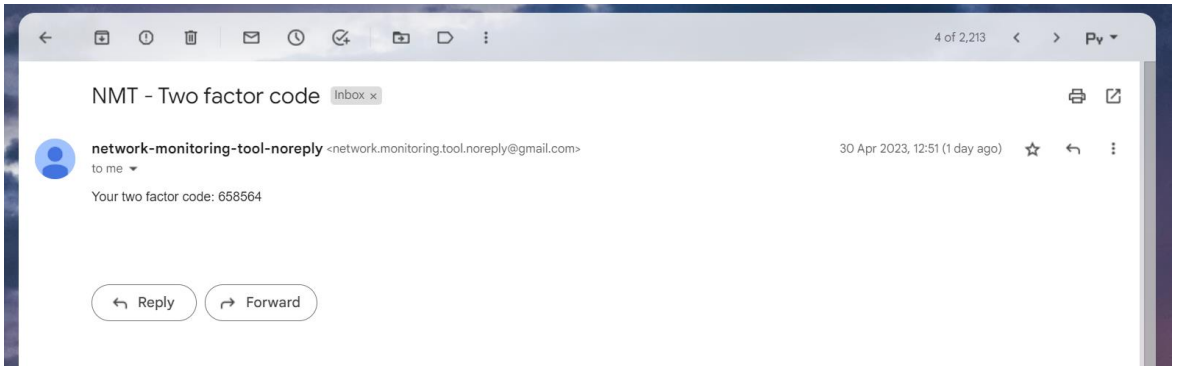

Рисунок 4.22 – Результат відправлення електронного листа

SMTP базується на моделі запит-відповідь, де клієнт ініціює розмову, надсилаючи команду, а сервер відповідає кодом стану.

4.3.6 Навчання моделей та визначення DDoS-атак

Основою навчання нових моделей є клас MLPClassifier з бібліотеки scikit-learn який представляє собою реалізацію багатошарового персептрона [19, 20] на основі нейронної мережі для класифікації представлених даних.

```
def train_model(self) -> None:
    classifier = MLPClassifier(hidden_layer_sizes=(100, 100), activation='logistic', max_iter=1000, verbose=True,
                               tol=0.00000001, early_stopping=True, shuffle=True)

    csv = pandas.read_csv(self._get_dataset_file(), delimiter=',')
    data = self._encode_labels(csv)

    x = data[self.ml_model_columns]
    y = data['Target']

    x_train, x_test, y_train, y_test = train_test_split(x.values, y)

    start_time = timer()
    classifier.fit(x_train, y_train)
    time_taken = timer() - start_time

    predictions = classifier.predict(x_test)

    hostile = safe = 0
    for prediction in predictions:
        if prediction == 1:
            hostile += 1
        else:
            safe += 1

    print(f'Number of iterations: {classifier.n_iter_}')
    print(f'Safe packets: {safe}')
    print(f'Hostile packets: {hostile}')
    print(f'Time taken: {time_taken}')
    pickle.dump(classifier, open(self._get_model_file(), 'wb'))
```

Рисунок 4.23 – Метод для тренування нових моделей

Конструктор класу MLPClassifier може приймати достатньо велику кількість параметрів для більш точної та конкретної класифікації даних, до основних з них можна виділити:

1) hidden_layer_sizes – кортеж, який визначає кількість нейронів у кожному прихованому шарі MLP. Наприклад: (5) = 1 шар з 5 вузлів, (5, 5) = 2 шари по 5 вузлів.

2) activation – функція активації (logistic є еквівалентом сигмоїдальній функції активації).

3) max_iter – максимальна кількість ітерацій для алгоритму оптимізації.

4) verbose – прапор для виведення на екран інформації про ітерацію.

5) tol – десятковий знак, якого має досягнути функція втрати.

6) shuffle – прапор який визначає, чи перемішуються дані навчання перед кожною епохою навчання.

Для асинхронного блокування недоброзичливих IP-адрес використано чергу повідомлень RabbitMQ та заздалегідь навчену модель. В даному випадку для визначення DDoS-атак використовуються поля: протокол, IP-адреса та розмір пакету [21, 22].

Особливу увагу потрібно приділити методу aggregate_pandas_all з бібліотеки pymongoarrow.monkey тому, що він є хорошим вибором для групування та агрегування даних з MongoDB, оскільки забезпечує зручний та ефективний спосіб виконання складних агрегацій за допомогою синтаксису конвеєра агрегації, а також забезпечує інтеграцію з форматом DataFrame [23].

```
def consume(self, channel, method, properties, body) -> None:
    detect_ddos_attacks_event_json = json.loads(body.decode('UTF-8'))['message']
    detect_ddos_attacks_event = DetectDdosAttacksEvent.parse_from_json(detect_ddos_attacks_event_json)

    passed_packets = self.mongo_readonly_client.packets_collection.aggregate_pandas_all([
        {'$match': {
            'DeviceId': detect_ddos_attacks_event.device_id,
            'CreatedAt': {...},
            'Status': 1  # 'Passed' status
        }},
        {'$project': {
            '_id': 0,
            'Protocol': '$Protocol',
            'IP': '$Ip',
            'Size': '$Size'
        }}
    ])

    passed_packets['Protocol'] = passed_packets['Protocol'].apply(lambda protocol: self.protocols[protocol])
    passed_packets['IP'] = passed_packets['IP'].apply(lambda ip: ip_address(ip).__str__())

    ips_to_block = self.detection_service.detect_hostile_ips(passed_packets)

    if len(ips_to_block) == 0:
        return

    block_ip_addresses_event = BlockIpAddressesEvent(detect_ddos_attacks_event.device_id, ips_to_block)
    channel.basic_publish(exchange=EventExchanger.BLOCK_IP_ADDRESSES,
                          routing_key=EventQueue.BLOCK_IP_ADDRESSES,
                          properties=BasicProperties(message_id=block_ip_addresses_event.id),
                          body=block_ip_addresses_event.to_masstransit_format(self.rabbit_mq_host))
```

Рисунок 4.24 – Обробник автоматичного блокування недоброзичливих IP-адрес

Залишається лише використати DataFrame для машинного аналізу та повернути недоброзичливі IP-адреси.

```
def detect_hostile_ips(self, packets: DataFrame) -> list[int]:
    ips = list(packets['IP'])
    data = self._encode_labels(packets)
    x = data[self.ml_model_columns]

    trained_model = pickle.load(open(self._get_model_file(), 'rb'))
    predictions = trained_model.predict(x.values)

    hostile_ips = set()
    for i in range(len(predictions)):
        if predictions[i] == 1:
            hostile_ips.add(ip_address(ips[i]).__int__())

    return list(hostile_ips)
```

Рисунок 4.25 – Метод для пошуку недоброзичливих IP-адрес

Таким чином, після отримання всіх шкідливих IP-адрес, створюється нове повідомлення для RabbitMQ, яке пізніше буде оброблено gRPC сервером.

Висновки до розділу 4

В четвертому розділі кваліфікаційної роботи бакалавра наведено програмну реалізацію ключового функціоналу застосунку моніторингу трафіку. Проведено огляд та вербальний опис команд CLI та дизайну вебклієнту.

Розглянуто eBPF XDP застосунок моніторингу мережевих пакетів на мові програмування C та детально описано способи зберігання даних в подібних системах. Закріплено навички написання запитів до реляційних і постреляційних БД та GraphQL серверів.

Описано алгоритм написання .proto контрактів та продемонстровано їх реалізацію на мові програмування C#. Розглянуто засоби машинного навчання для створення нових моделей та їх застосування на практиці.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра автоматизовано процес моніторингу трафіку в локальній інтернет-мережі за рахунок створення відповідного програмного забезпечення.

Для досягнення поставленої мети виконано визначені завдання:

– проведено аналіз аналогів;

– виявлено основні проблеми та продумано можливі рішення;

– визначено необхідний функціонал застосунку;

– запропоновано алгоритм передачі інформації про активність в мережі та доцільні способи відображення кінцевої інформації користувачу;

– досліджено внутрішнє влаштування Kernel та eBPF в ОС Linux;

– розроблено CLI частину застосунку на базі мов програмування Go, C;

– розроблено back-end частину застосунку на базі технологій .NET, Python, gRPC, GraphQL, RabbitMQ, Redis, PostgreSQL та MongoDB;

– розроблено front-end частину застосунку на базі технологій Angular та Material Design.

Виконано порівняння існуючих альтернативних застосунків, проведено висвітлення їх сильних та слабких сторін, звернуто увагу на архітектуру, мови реалізації, функціонал, користувацький інтерфейс, переваги та недоліки. Сформовано вимоги до ПЗ та проведено моделювання застосунку шляхом створення різноманітних UML-діаграм.

Описано способи оптимізації навантаження системи завдяки розподілу на окремі сервери та шардуванню, розібрано переваги та недоліки різних підходів. Продемонстровано архітектуру eBPF застосунку з виконанням JIT компіляції та приділено особливу увагу процесу верифікації на рівні ОС.

Проведено вибір стеку технологій на основі вимог до ПЗ та розроблено всі частини застосунку. Розглянуто засоби машинного навчання для автоматизації процесів шляхом створення нових моделей та застосуванням їх на практиці.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Analog application: Paessler RPTG Network Monitor. URL: paessler.com/prtg/prtg-network-monitor (Last accessed: 27.03.2023).

2. Analog application: Wireshark. URL: wireshark.org (Last accessed: 27.03.2023).

3. Analog application: NETRESEC Network Miner. URL: netresec.com/?page=NetworkMiner (Last accessed: 27.03.2023).

4. Brij B. Gupta, Amrita Dahiya. Distributed Denial of Service (DDoS) Attacks: Classification, Attacks, Challenges and Countermeasures. CRC Press, 2021. 140 p.

5. Local machine ID configuration file. URL: manpages.ubuntu.com/manpages/bionic/man5/machine-id.5 (Last accessed: 28.03.2023).

6. Martin Kleppmann. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017. 616 p.

7. Бойко Д. Д., Давиденко Є. О. Засоби обробки та аналізу мережевих пакетів в ОС Linux. Ольвійський форум – 2023 : стратегії країн Причорноморського регіону в геополітичному просторі : XVII Міжнар. наук. конф. 15–18 червня 2023 р., м. Миколаїв : тези доп. : Комп'ютерна інженерія. Інтелектуальні інформаційні системи. Моделі, методи та засоби програмної інженерії. Автоматизація та комп'ютерно-інженерні технології / Чорном. нац. ун-т ім. Петра Могили. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2023.

8. A Gentle Introduction to eBPF. URL: infoq.com/articles/gentle-linux-ebpf-introduction (Last accessed: 12.04.2023).

9. Gerardus Blokdyk. UML Tool A Complete Guide – 2020 Edition. 5STARCooks, 2021. 303 p.

10. Matthias Noback. Object Design Style Guide. Simon and Schuster, 2019. 288 p.

11. Architecture Design Patterns (CQRS). URL: learn.microsoft.com/en-us/azure/architecture/patterns/cqrs (Last accessed: 14.04.2023).

12. The art of writing eBPF programs: a primer. URL: sysdig.com/blog/the-art-of-writing-ebpf-programs-a-primer (Last accessed: 14.04.2023).

13. eBPF XDP: The Basics and a Quick Tutorial. URL: tigera.io/learn/guides/ebpf/ebpf-xdp (Last accessed: 20.04.2023).

14. What Is Material Design and How Should It Be Used. URL: elementor.com/blog/what-is-material-design (Last accessed: 24.04.2023).

15. Roy Derks, Gaetano Checinski. Fullstack GraphQL: Complete Guide to Building Servers and Clients in GraphQL. Newline, 2021. 230 p.

16. HTTP/2 vs HTTP/1.1: How do they affect web performance. URL: cloudflare.com/learning/performance/http2-vs-http1.1 (Last accessed: 27.04.2023).

17. Protobuf. What is it, why you should care, and when should you use it. URL: adaptiv.nz/protobuf-what-is-it-why-you-should-care-and-when-should-you-use-it (Last accessed: 27.04.2023).

18. Start Developing with MongoDB – MongoDB Drivers. URL: mongodb.com/docs/drivers (Last accessed: 28.04.2023).

19. Aurelien Geron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems – 2nd Edition. O'Reilly Media, 2022. 861 p.

20. Vishal Maini, Samer Sabri. Machine Learning For Humans: Introduction to Machine Learning with Python. Alanna Maldonado, 2023. 98 p.

21. Considerations when choosing a machine learning model. URL: towardsdatascience.com/considerations-when-choosing-a-machine-learning-model-aa31f52c27f3 (Last accessed: 02.05.2023).

22. How to Choose the Right Machine Learning Algorithm: A Pragmatic Approach. URL: labelyourdata.com/articles/how-to-choose-a-machine-learning-algorithm (Last accessed: 02.05.2023).

23. How to create a Pandas DataFrame in Python. URL: machinelearningplus.com/pandas/how-to-create-pandas-dataframe-python (Last accessed: 04.05.2023).