НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«МЕРЕЖЕВИЙ АНАЛІЗАТОР ТРАФІКУ І СТРУКТУРИ МЕРЕЖІ НА RASPBERRY»

Виконав: здобувачка 4 курсу
Рівень бакалавр
Галузь знань 12 «Інформаційні технології»,
спеціальність 125 «Кібербезпека»
Дехтярук Анна Євгенівна

Керівник: к.т.н., доцент
Кухаренко Сергій Вікторович

Рецензент: к.ф.-м.н., доцент
Черевко Євген Володимирович

Одеса – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»

Факультет **кібербезпеки та інформаційних технологій**
Кафедра **кібербезпеки**
Галузь знань: **12 Інформаційні технології**
Спеціальність: **125 Кібербезпека**
Назва освітньої програми: **Кібербезпека**

ЗАТВЕРДЖУЮ
Завідувач кафедри кібербезпеки
доцент С.А. Горбаченко
_____ «____»_____ 20__ року

ЗАВДАННЯ
на кваліфікаційну (бакалаврську) роботу
здобувачці Дехтярук Анна Євгенівна

1. Тема роботи: «Мережевий аналізатор трафіку і структури мережі на Raspberry»
Керівник роботи: к.т.н, доцент Сергій Вікторович Кухаренко
затверджені наказом НУ ОЮА від «18» березня 2025 року № 548-6
2. Строк подання здобувачем роботи «19» травня 2025 року
3. Дата видачі завдання «16» вересня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему: «Мережевий аналізатор трафіку і структури мережі на Raspberry» на здобуття першого (бакалаврського) рівня вищої освіти за спеціальністю 125 «Кібербезпека та захист інформації», освітньою програмою: Кібербезпека, містить 15 рисунків, 7 таблиці, 5 додатків, 31 використаних джерела за переліком посилань. Робота виконана на 59 сторінках загального тексту, з яких 40 сторінок становить основний текст.

Метою роботи є розробка легковагової, доступної системи аналізу трафіку та топології мережі на базі Raspberry Pi з використанням мови програмування C. У процесі дослідження реалізовано модулі для захоплення, обробки, зберігання та візуалізації трафіку в реальному часі з використанням бібліотек libpcap, SQLite, ncurses. Здійснено експериментальне дослідження функціональності системи, оцінено продуктивність та ефективність у порівнянні з існуючими рішеннями. Виявлено переваги використання Raspberry Pi як бюджетного інструменту для мережевого моніторингу в умовах обмежених ресурсів.

Результати роботи можуть бути використані в освітніх, дослідницьких та практичних цілях для побудови системи базового аналізу мережевого трафіку в малих офісах або домашніх мережах.

Можливими напрямами подальших досліджень є інтеграція модулів машинного навчання для автоматичного виявлення аномалій та розширення функціоналу для роботи з хмарними базами даних і віддаленим управлінням.

МЕРЕЖЕВИЙ АНАЛІЗАТОР, ТРАФІК, ТОПОЛОГІЯ, RASPBERRY PI, LIBPCAP, NCURSES, C.

ABSTRACT

The bachelor's qualification thesis titled "Network Traffic and Topology Analyzer Based on Raspberry", submitted for the first (bachelor's) level of higher education in specialty 125 Cybersecurity and information protection, educational program: Cybersecurity, includes 15 figures, 7 tables, 5 appendices, and 31 references. The work consists of 59 pages in total, including 40 pages of main content.

The aim of the thesis is to develop a lightweight, affordable system for traffic and network structure analysis based on Raspberry Pi using the C programming language. The project implements modules for real-time packet capture, data processing, storage, and visualization using libpcap, SQLite, and ncurses libraries. An experimental evaluation of system functionality was carried out, and performance was assessed in comparison with existing solutions. The advantages of using Raspberry Pi as a low-cost network monitoring tool in limited-resource environments were demonstrated.

The results can be applied for educational, research, and practical purposes to build a basic network traffic analysis system in small office or home networks.

Future research directions include integrating machine learning modules for automatic anomaly detection and expanding functionality to support cloud databases and remote access.

NETWORK ANALYZER, TRAFFIC, TOPOLOGY, RASPBERRY PI, LIBPCAP, NCURSES, C.

ЗМІСТ

ВСТУП 6
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ 8
1.1 Предметна область та теоретичні основи аналізу мережевого трафіку 8
1.2 Огляд сучасних інструментальних засобів аналізу мережі 13
1.3 Визначення задачі дослідження 19
2 ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ СИСТЕМИ АНАЛІЗУ ТРАФІКУ НА БАЗІ RASPBERRY PI 21
2.1 Математичні моделі аналізу мережевого трафіку 21
2.2 Алгоритми виявлення активності у мережі та визначення її структури 24
2.3 Архітектура програмного рішення: основні модулі та їх взаємодія 28
2.4 Обґрунтування вибору інструментів, бібліотек та середовища розробки 30
3 РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ АНАЛІЗАТОРА 33
3.1 Реалізація програмного продукту на Raspberry Pi 33
3.2 Інтерфейс користувача та запуск системи моніторингу 37
3.3 Проведення експериментів: дослідження роботи алгоритмів та оцінка ефективності 39
3.4 Порівняльний аналіз із існуючими рішеннями та висновки щодо досягнутих результатів 41
ВИСНОВКИ 44
ПЕРЕЛІК ПОСИЛАНЬ 46
Додаток А: Фрагмент коду модуля захоплення трафіку 49
Додаток Б: Код модуля обробки даних мережевого трафіку 50
Додаток В: Модуль аналізу трафіку IP-адрес 54
Додаток Г: Збереження та обробка мережевого трафіку в базі даних SQLite 56
Додаток Д: Графічне відображення мережевих з’єднань у терміналі з використанням ncurses 59

ВСТУП

У сучасному цифровому суспільстві забезпечення безпеки мережевої інфраструктури набуває критичного значення. Щоденне зростання обсягів мережевого трафіку, розвиток інтернету речей (IoT), а також збільшення кількості кіберзагроз створюють потребу у постійному моніторингу та аналізі мережевої активності. Традиційні системи моніторингу, хоча й ефективні, часто є дорогими, складними в налаштуванні та потребують потужного апаратного забезпечення. У зв’язку з цим особливо актуальним стає використання бюджетних і енергоефективних рішень, таких як Raspberry Pi.

Одноплатний комп’ютер Raspberry Pi відкриває можливості для створення портативних, недорогих, але функціональних систем для аналізу мереж. Зважаючи на обмежені ресурси цього пристрою, особливої уваги потребує адаптація існуючих методів та інструментів моніторингу для стабільної та ефективної роботи. Аналіз існуючих програмно-апаратних рішень, таких як Wireshark, ntopng, Security Onion тощо, показав, що їх використання в середовищі Raspberry Pi або надто ресурсомістке, або потребує значної оптимізації. Це підкреслює актуальність розроблення власного інструменту аналізу, що поєднує доступність, простоту та практичну користь.

Метою даної роботи є розробка та дослідження програмного рішення для аналізу мережевого трафіку і структури мережі на базі Raspberry Pi, що дозволяє в режимі реального часу здійснювати моніторинг активності та візуалізувати топологію мережі з урахуванням обмежених обчислювальних ресурсів.

Для досягнення цієї мети були поставлені наступні завдання:

- провести аналіз існуючих методів та інструментів моніторингу мереж;
- дослідити можливості Raspberry pi як платформи для аналізу мережевого трафіку;
- розробити програмний продукт з модульною архітектурою для збору, обробки, зберігання та візуалізації мережевого трафіку;
- реалізувати механізм побудови структури мережі на основі перехоплених даних;

- провести тестування системи та оцінити її ефективність у реальному середовищі.

Об’єктом дослідження є процес аналізу мережевого трафіку у комп’ютерних мережах.

Предметом дослідження є програмна реалізація аналізатора трафіку і структури мережі на основі Raspberry Pi з урахуванням обмежених ресурсів пристрою.

Результати цієї роботи можуть бути використані для створення портативних систем моніторингу у малих офісах, навчальних лабораторіях, а також у домашніх умовах. Розроблене рішення дозволяє швидко виявляти нові підключення до мережі, виявляти аномалії в трафіку та будувати топологію мережі. За рахунок використання недорогого обладнання, система може бути впроваджена без значних фінансових витрат, що робить її привабливою для широкого кола користувачів – від студентів до фахівців з інформаційної безпеки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Предметна область та теоретичні основи аналізу мережевого трафіку

Комп’ютерна мережа – це система зв’язку комп’ютерів або іншого обладнання, що спільно використовують ресурси, розташовані на вузлах мережі або надані ними. Комп’ютери використовують загальні протоколи зв’язку через цифрові канали зв’язку для взаємодії один з одним. Ці взаємозв’язки складаються з технологій телекомунікаційних мереж, заснованих на провідних, оптичних і безпровідних радіочастотних методах, які можуть бути організовані в різноманітні мережеві топології [1].

Структура комп’ютерної мережі визначається її топологією. Під топологією звичайно розуміють взаємне розташування один щодо одного вузлів мережі. Топологія – це конфігурація фізичних зв'язків між вузлами мережі. Характеристики мережі залежать від типу встановлюваної топології. Зокрема, вибір тієї чи іншої топології впливає:

- На склад необхідного мережевого обладнання;
- На можливості мережевого обладнання;
- На можливості розширення мережі;
- На спосіб управління мережею [2].

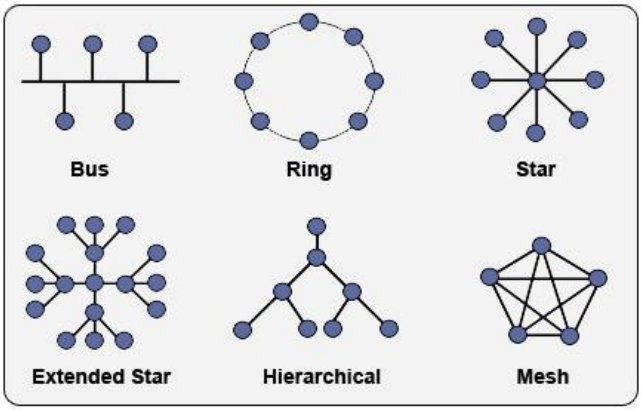

Рисунок 1 – Основні типи топологій комп’ютерних мереж

Наведемо приклади технологій з апаратним забезпеченням.

На рисунку 2 зображено типологію «Шина», замість шини зображено хаб (тому що хаб, відтворює логіку спільної шини).

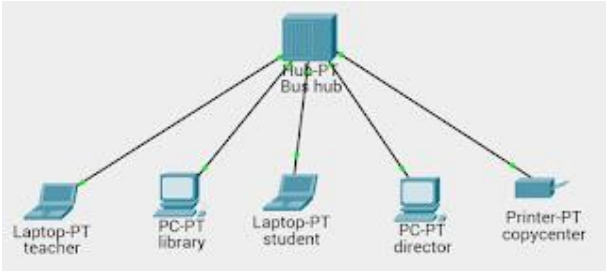

Рисунок 2 – Топологія «Шина»

На рисунку 3 зображено типологію «Зірка», на якому в якості центрального вузла виступає світч (або мережевий комутатор).

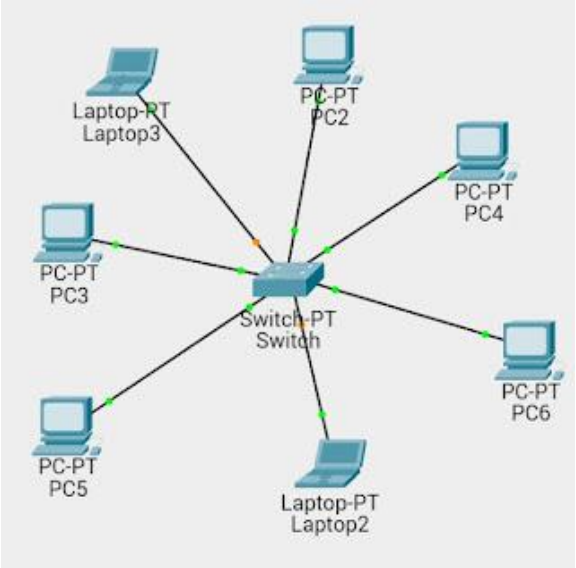

Рисунок 3 – Топологія «Зірка»

На рисунку 4 зображено коміркову типологію, де видно що мережа поділяється на так звані «комірки».

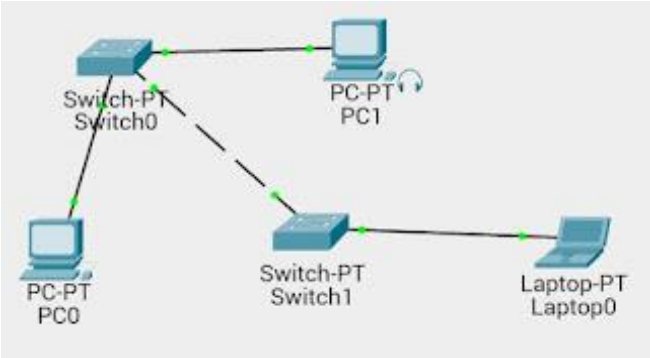

Рисунок 4 – Коміркова типологія

Мережевий трафік відноситься до обміну даними, що відбувається між пристроями, підключеними до мережі. Це охоплює різні форми комунікації, такі як надсилання електронних листів, перегляд вебсайтів або доступ до файлів на спільному диску. Розуміння мережевого трафіку є важливим для виявлення потенційних загроз безпеці та забезпечення ефективної роботи мережі.

Таблиця 1 – Основні типи топологій

Тип топології	Опис	Переваги	Недоліки
Шина	Усі пристрої підключені до одного спільного кабелю (шини).	Простота реалізації, низька вартість.	Висока ймовірність колізій, складність виявлення несправностей.
Кільце	Кожен пристрій з’єднаний з двома іншими, утворюючи замкнуте коло.	Відсутність колізій, рівномірний розподіл навантаження.	Вихід з ладу одного вузла може зупинити всю мережу.
Зірка	Усі пристрої підключені до центрального вузла (комутатора або концентратора).	Легкість управління, ізоляція несправностей.	Залежність від центрального вузла.

Продовження таблиці 1

Сітка (Mesh)	Кожен пристрій з’єднаний з усіма іншими.	Висока надійність, відмовостійкість.	Висока вартість реалізації, складність управління.
Дерево	Ієрархічна структура з підмережами, що підключені до центрального вузла.	Масштабованість, структурованість.	Складність конфігурації, залежність від центральних вузлів.

Моніторинг та аналіз мережевого трафіку є важливими для аналітиків безпеки та IT-фахівців для виявлення аномальних шаблонів або потенційних загроз безпеці. Ось деякі ключові аспекти аналізу трафіку:

− Виявлення аномалій: Аналізуючи мережевий трафік, фахівці з безпеки можуть виявити аномальні шаблони або поведінку, які можуть свідчити про несанкціонований доступ або зловмисні дії. Наприклад, раптовий сплеск мережевого трафіку з невідомого джерела може вказувати на атаку типу «відмова в обслуговуванні» (DDoS).

− Виявлення загроз безпеці: Аналіз мережевого трафіку дозволяє виявити потенційні загрози безпеці, такі як зараження шкідливим програмним забезпеченням або спроби зловживати вразливостями мережевих пристроїв. Досліджуючи мережевий трафік, фахівці з безпеки можуть вжити проактивних заходів для запобігання інцидентам безпеки.

− Оптимізація продуктивності мережі: Аналіз даних мережевого трафіку дозволяє IT-фахівцям контролювати використання та продуктивність мережі. Вони можуть виявити програми, що використовують велику ширину смуги, або вузькі місця в інфраструктурі мережі і вжити необхідних заходів для оптимізації продуктивності мережі [3].

Існують різні методи аналізу мережевого трафіку: Пасивний аналіз та активний аналіз.

Активний моніторинг включає генерацію тестового трафіку для виявлення проблем у мережі. Він сповіщає про збої, здійснюючи збір вимірювань між двома кінцевими вузлами. Активні системи вимірювання оперують показниками, як-от: маршрути, затримка пакетів, втрати, повторне надсилання, нестабільність синхронізації, пропускна здатність.

Основними активними інструментами моніторингу є ping та traceroute. Ping вимірює затримку та втрату пакетів, тоді як traceroute дозволяє визначити топологію мережі. Обидва ці інструменти відправляють пробні пакети ICMP до пункту призначення та очікують відповіді.

На рисунку 5 показано приклад команди *ping*, яка використовує активний метод вимірювання: надсилає ICMP Echo-запит від джерела через мережу до заданої точки. Далі отримувач пересилає Echo-відповідь назад до джерела, звідки вона надійшла.

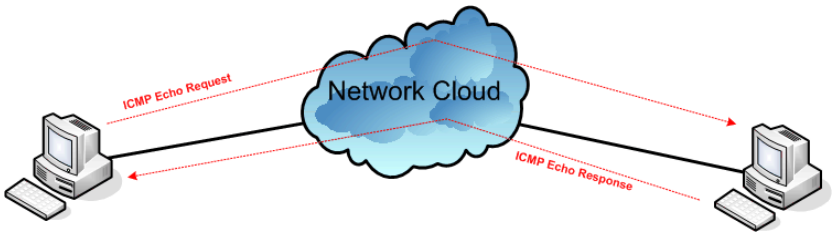

Рисунок 5 – Команда ICMP ping (активний вимір)

Активні вимірювання дозволяють виявити динамічні зміни в мережі, недоступні при пасивному аналізі, але потребують втручання у трафік та можуть впливати на продуктивність мережі.

Пасивний моніторинг передбачає збір даних без втручання в мережевий трафік. Такий моніторинг не додає нового трафіку до мережі та не змінює існуючий. Він дозволяє фіксувати мережеві пакети у певній точці мережі й аналізувати їх після збору. На відміну від активного моніторингу, пасивний збір даних відбувається лише з однієї точки, що дає точніші вимірювання, але не дозволяє оцінити взаємодію між кількома вузлами.

Інструменти, як-от Wireshark та tcpdump, дозволяють здійснювати пасивний аналіз мережі. Пасивні вимірювання зосереджуються на таких параметрах, як обсяг

трафіку, співвідношення протоколів, швидкість передачі (кількість бітів), синхронізація пакетів і час між їх надходженням. Отримані дані аналізуються постфактум, ізольовано, не як складові одного цілого. Це призводить до труднощів при обробці великих обсягів даних, накопичених у процесі вимірювань.

На рисунку 6 показано розташування системи пасивного моніторингу, де монітор розміщений на одному каналі між двома кінцевими точками й відстежує трафік в міру його проходження через канал [4].

Рисунок 6 – Установка пасивного моніторингу

1.2 Огляд сучасних інструментальних засобів аналізу мережі

Є декілька інструментів для аналізу мережевого трафіку, які умовно поділяються на локальні та мережеві моніторингові системи, залежно від їх призначення, способу розгортання та масштабу використання.

Локальні інструменти - це програми, які працюють безпосередньо на пристрої, підключеному до мережі. Вони використовуються для аналізу трафіку на одному інтерфейсі, що дозволяє отримувати точну інформацію про вхідні та вихідні пакети. Розглянемо один з них це Wireshark.

Wireshark – це потужний аналізатор трафіку, що дозволяє захоплювати й аналізувати мережеві пакети, підтримує фільтрацію, розшифрування зашифрованих протоколів (SSL/TLS, SSH тощо) та дає змогу вивчати протоколи на різних рівнях OSI-моделі. Загалом, Wireshark є потужним інструментом для аналізу мережевого трафіку, який допомагає зрозуміти, контролювати і вдосконалювати роботу мережевих систем [5].

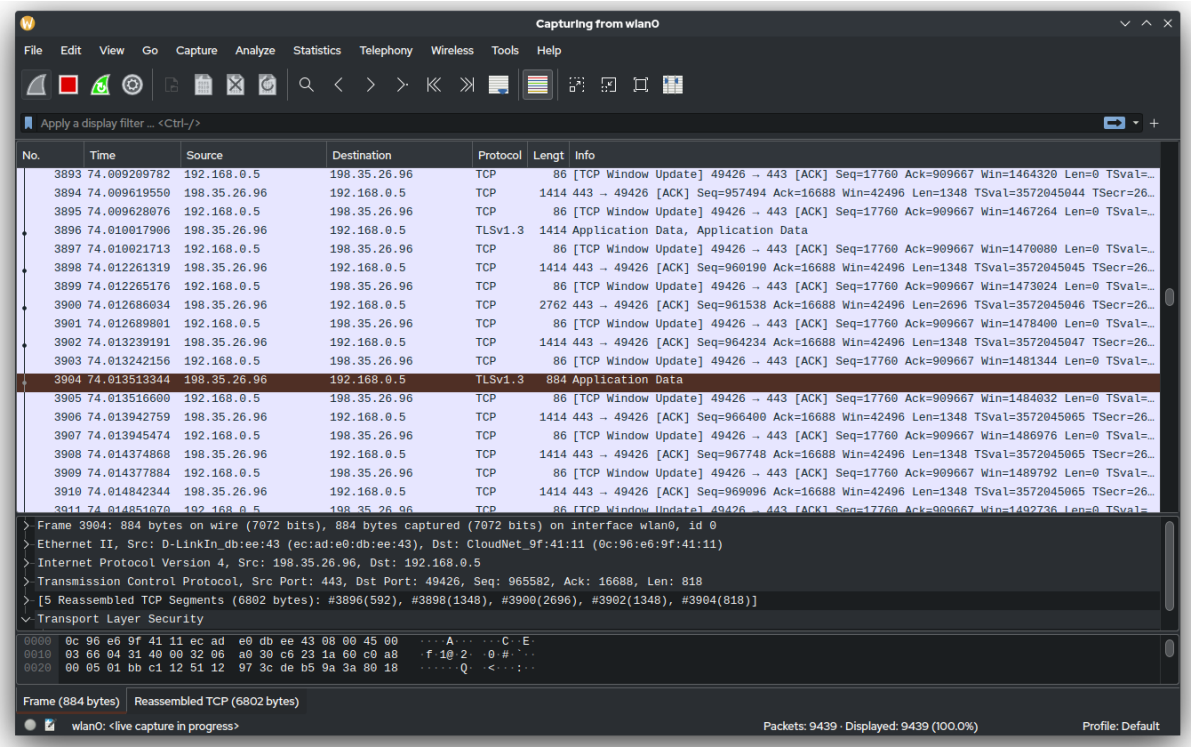

Рисунок 7 – Основне вікно Wireshark

Ntopng – потужна web-орієнтована система для моніторингу та аналізу мережевого трафіку, яка підтримує Deep Packet Inspection і інтеграцію з NetFlow/sFlow. Ntopng в основному є зонд мережевого трафіку, який контролюватиме використання мережі. Він заснований на libpcap, бібліотеці, написаній як частина великої програми, що називається TCP Dump Ntopng заснований на сервері значень ключів Redis, а не на традиційній базі даних, використовує nDPI для виявлення протоколу, підтримує геолокацію хостів і може відображати аналіз потоку в реальному часі для підключених хостів [6].

Ntopng є прикладом більш ресурсоємних, але дають змогу централізовано відстежувати стан великої мережі з гнучкими механізмами оповіщення, історії та автоматичного аналізу.

З метою вибору найкращого програмного засобу аналізу мережевого трафіку треба провести аналіз переваги та недоліки означених програмних засобів.

За допомогою таблиці 2 можемо зробити наступні висновки що Wireshark краще підходить для глибокого аналізу мережевих пакетів, особливо у навчальних цілях, де потрібна деталізація та простота, а ntopng є кращим вибором для візуального

моніторингу мережевої структури та інтеграції в реальну інфраструктуру, але може потребувати потужнішого обладнання або платних модулів.

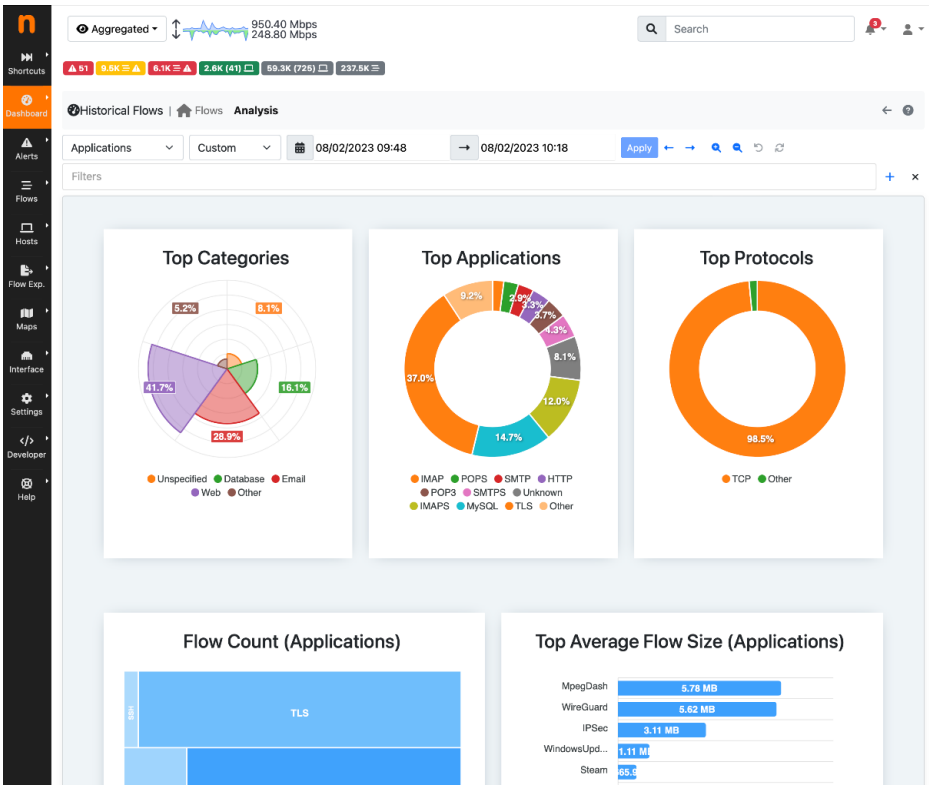

Рисунок 8 – Вигляд системи ntopng

Таблиця 2 – порівняння переваг та недоліків Wireshark та ntopng

Критерій	Wireshark	Ntopng
Тип інтерфейсу	Графічний інтерфейс на базі GTK+	Веб-інтерфейс у браузері
Підтримка протоколів	Переважна більшість відомих протоколів	NetFlow, sFlow, IPFIX та інші
Глибина аналітики	Висока деталізація пакетів, потужна система фільтрів	Висока аналітика по хостах, сервісах, портах, геолокації
Використання на Raspberry Pi	Можливе, але потребує оптимізації інтерфейсу	Можливе з обмеженнями (потрібна оптимізація параметрів збору даних)

Продовження таблиці 2

Вартість	Повністю безкоштовний	Базова версія безкоштовна, деякі функції (через nProbe) – платні
Інтеграція з іншими системами	Обмежена	Підтримка інтеграцій (Nagios, Zabbix)
Системні вимоги	Середні–високі	Середні, потребує оптимізації для Raspberry Pi

Апаратні засоби аналізу мережевого трафіку – це спеціалізовані пристрої або системи, призначені для перехоплення, обробки та аналізу даних, що передаються в комп'ютерних мережах. Вони забезпечують глибоку інспекцію пакетів, моніторинг у реальному часі та виявлення аномалій, що є критично важливими для забезпечення безпеки та ефективності мережі.

Таблиця 3 – Приклади апаратних засобів аналізу мережі

Назва	Тип	Основні функції	Особливості
SolarWinds Network Bandwidth Analyzer	Аналізатор пропускної здатності	Моніторинг трафіку, виявлення вузьких місць	Інтеграція з іншими продуктами SolarWinds
Fluke Networks OptiView	Аналізатор мережі	Повний аналіз мережевої інфраструктури	Портативний пристрій з вбудованим дисплеєм
Raspberry Pi з Pi-hole	Блокування реклами, моніторинг DNS	Фільтрація DNS-запитів, статистика трафіку	Доступне рішення на базі одноплатного комп'ютера

Одним із провідних рішень у цій сфері є SolarWinds Network Performance Monitor (NPM), який забезпечує автоматичне виявлення мережевих пристроїв та їх моніторинг. Цей інструмент надає візуалізацію топології мережі, дозволяє відстежувати продуктивність маршрутизаторів, комутаторів, брандмауерів та інших пристроїв з підтримкою SNMP. Він також включає розширені можливості сповіщення, що дозволяють адміністраторам оперативно реагувати на потенційні проблеми [7].

Іншим важливим апаратним інструментом є Paessler PRTG Network Monitor, який використовує концепцію «датчиків» для моніторингу різних аспектів мережі. Цей інструмент підтримує протоколи NetFlow, sFlow, jFlow та IPFIX для збору детальної інформації про трафік. PRTG дозволяє відстежувати IP-пакети, фільтрувати трафік за протоколами UDP та TCP, а також моніторити трафік на маршрутизаторах, комутаторах, серверах та віртуальних машинах [8].

Для аналізу бездротових мереж ефективним рішенням є Kismet – інструмент з відкритим вихідним кодом, який дозволяє виявляти та аналізувати бездротові мережі, включаючи ті, що приховують свій SSID. Kismet підтримує моніторинг стандартів 802.11b/g/n та може бути використаний для виявлення несанкціонованих точок доступу та аналізу безпеки бездротових мереж.

Крім того, Security Onion є комплексним дистрибутивом Linux з відкритим вихідним кодом, призначеним для моніторингу мережевої безпеки. Він об'єднує кілька інструментів, таких як Snort, Suricata та Zeek, в єдину платформу, що спрощує налаштування та управління комплексним рішенням для моніторингу мережевої безпеки. Security Onion дозволяє централізувати журнали, проводити аналіз у режимі реального часу та створювати панелі моніторингу мережевої безпеки.

Ці апаратні рішення забезпечують глибоку видимість мережевого трафіку, дозволяють виявляти аномалії та потенційні загрози, а також оптимізувати використання мережевих ресурсів. Їх впровадження сприяє підвищенню надійності та безпеки мережевої інфраструктури.

Raspberry Pi є малопотужним одноплатним комп’ютером, який має обмежені апаратні ресурси порівняно з традиційними ПК або серверними рішеннями. Наприклад, процесор (CPU) моделі Raspberry Pi 4 оснащена 64-бітним

чотириядерним процесором Cortex-A72 з тактовою частотою 1,5 ГГц. Це забезпечує достатню продуктивність для виконання помірно вимогливих завдань, однак при обробці великих обсягів трафіку можливі обмеження [9].

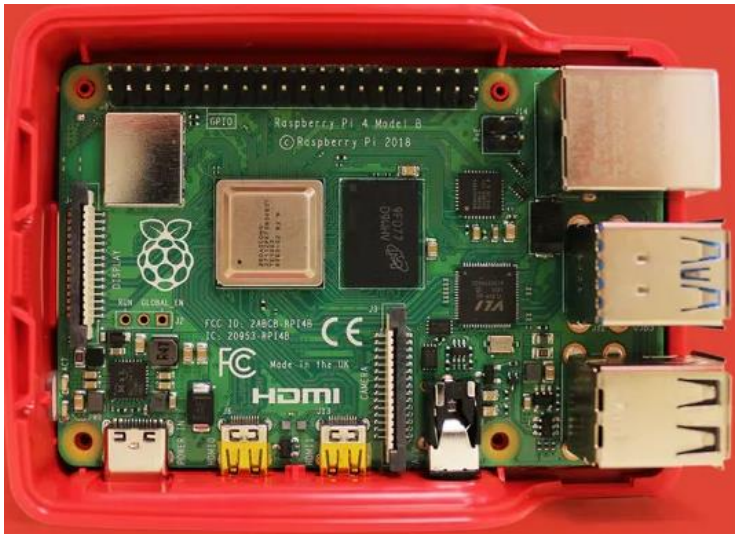

Рисунок 9 – Мікрокомп’ютер Raspberry Pi 4

Однією з ключових переваг Raspberry Pi є його здатність працювати з різноманітними операційними системами, включаючи спеціалізовані дистрибутиви для мережевого аналізу, такі як Kali Linux. Це дозволяє використовувати потужні інструменти, як-от Wireshark, tcpdump, Nmap, Kismet, Suricata та Zeek, без необхідності в дорогому обладнанні.

Для аналізу бездротових мереж, зокрема Wi-Fi, Raspberry Pi може бути оснащений спеціальними драйверами, які дозволяють використовувати режим моніторингу. Наприклад, встановлення Re4son-Kernel або використання драйвера Nexmon дозволяє активувати режим монітора на вбудованому Wi-Fi-модулі Raspberry Pi, що є необхідним для пасивного захоплення трафіку та аналізу бездротових мереж [10].

Крім того, Raspberry Pi може бути використаний для створення портативних лабораторій з кібербезпеки. Зокрема, встановлення Kali Linux на Raspberry Pi Zero 2 W дозволяє створити компактний пристрій для тестування безпеки мереж, який можна контролювати зі смартфона через SSH або VNC [11].

Завдяки своїй енергоефективності та компактним розмірам, Raspberry Pi може працювати цілодобово, забезпечуючи постійний моніторинг мережі. Це особливо корисно для виявлення нових пристроїв у мережі, аналізу трафіку та виявлення потенційних загроз [12].

Таким чином, Raspberry Pi є потужним та доступним інструментом для аналізу мережевого трафіку та структури мережі, який може бути адаптований до різноманітних завдань, від моніторингу домашньої мережі до проведення тестів на проникнення в корпоративних середовищах.

1.3 Визначення задачі дослідження

У попередніх підрозділах було детально розглянуто предметну область аналізу мережевого трафіку, різновиди мережевих топологій, а також сучасні інструменти моніторингу. Були виділені два основних підходи до аналізу трафіку: пасивний (Wireshark, tcpdump) та активний (ping, traceroute), кожен із яких має власні переваги й обмеження.

Значна увага була приділена аналізу можливостей Raspberry Pi як платформи для реалізації системи мережевого моніторингу. Було встановлено, що цей одноплатний комп’ютер є дешевою, мобільною та енергоефективною альтернативою традиційним системам моніторингу.

Таблиця 4 – недоліки інструментів для аналізу мереж.

№	Категорія	Недолік	Приклад
1	Сигнатурні системи	Не виявляють нові загрози	Snort, Suricata
2	Поведінковий аналіз	Велика кількість хибних спрацювань	Security Onion
3	Машинне навчання	Високі обчислювальні ресурси	Zeek, TensorFlow
4	Програмні інструменти	Складність у використанні, потреба в навичках	Wireshark
5	Апаратні системи	Висока вартість	SolarWinds, Fluke OptiView

Продовження таблиці 4

6	Використання на Raspberry Pi	Обмеження потужності для важких обчислень	ntopng, Wireshark

Таким чином, критичний огляд існуючих рішень показав, що хоча інструментів мережевого моніторингу існує чимало, їх повноцінне застосування у доступних системах, як Raspberry Pi, потребує адаптації. У дипломній роботі буде розроблено власну систему, що поєднує простоту, функціональність та оптимальність.

2 ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ СИСТЕМИ АНАЛІЗУ ТРАФІКУ НА БАЗІ RASPBERRY PI

2.1 Математичні моделі аналізу мережевого трафіку

Моделювання мережевого трафіку є ключовим етапом у розробці ефективних систем моніторингу та аналізу мереж. Існують різні підходи до моделювання, серед яких виділяють:

- Модель пакетного потоку: його розглядають трафік як послідовність пакетів, що передаються мережею. Ці моделі дозволяють аналізувати характеристики трафіку, такі як інтенсивність, розподіл розмірів пакетів та міжпакетні інтервали. Вони широко використовуються для вивчення поведінки мережі та оптимізації її продуктивності.
- Моделі черг: описують процес обробки пакетів у мережевих пристроях, таких як маршрутизатори та комутатори. Наприклад, модель M/M/1 передбачає, що пакети надходять у чергу за експоненціальним розподілом часу між надходженнями та обслуговуються одним сервером. Ці моделі допомагають оцінити затримки, ймовірність втрати пакетів та інші показники якості обслуговування.
- Стохастичні моделі: враховують випадковий характер мережевого трафіку, що дозволяє моделювати непередбачувані зміни в навантаженні мережі. Наприклад, процеси Пуассона використовуються для моделювання випадкових надходжень пакетів. Ці моделі є основою для аналізу продуктивності мереж та планування їхньої пропускної здатності [13].
- Графові моделі: представляють мережу у вигляді графа, де вузли відповідають мережевим пристроям, а ребра - з'єднанням між ними. Ці моделі дозволяють аналізувати топологію мережі, виявляти критичні вузли та шляхи, а також оптимізувати маршрутизацію. Сучасні підходи, такі як використання графових нейронних мереж, дозволяють ефективно моделювати складні мережеві структури та передбачати ключові показники продуктивності, такі як затримка та втрата пакетів [14].
- У локальних мережах (LAN) трафік характеризується високою швидкістю передачі даних, низькою затримкою та відносно стабільним навантаженням.

Основними джерелами трафіку є обмін файлами, доступ до спільних ресурсів та внутрішні сервіси.

У мережах Інтернету речей (IoT) трафік має інші характеристики:

1. Низький обсяг даних з високою частотою передачі: Багато IoT-пристроїв передають невеликі обсяги даних, але з високою частотою, що створює постійне навантаження на мережу.

2. Обмежені ресурси пристроїв: IoT-пристрої часто мають обмежену обчислювальну потужність та енергоспоживання, що накладає обмеження на обробку та передачу даних.

3. Висока частота передачі: Деякі сенсори можуть передавати дані кілька разів на секунду, що створює постійне навантаження на мережу [15].

4. Використання нестандартних протоколів: IoT-пристрої можуть використовувати різноманітні протоколи передачі даних, що ускладнює інтеграцію та аналіз трафіку.

5. Підвищені вимоги до безпеки: Через обмежені ресурси та відсутність стандартів безпеки, IoT-пристрої часто є вразливими до атак, що вимагає додаткових заходів захисту [16].

Ці особливості вимагають спеціальних підходів до аналізу та обробки трафіку в мережах IoT, особливо при використанні обмежених пристроїв, таких як Raspberry Pi.

Для ефективного аналізу мережевого трафіку необхідно враховувати наступні параметри:

- Швидкість передачі даних: вимірюється в бітах за секунду (bps) або пакетах за секунду (pps) і визначає обсяг даних, що передаються через мережу за одиницю часу.
- Затримка передачі: час, необхідний для передачі пакету від джерела до призначення. Затримка може бути викликана обробкою пакетів, маршрутизацією та іншими факторами [17].
- Частота передачі: кількість пакетів або повідомлень, що передаються за певний проміжок часу.

- Кількість пакетів: загальна кількість пакетів, переданих або отриманих за певний період.
- Втрати пакетів: кількість пакетів, які не досягли призначення через помилки або перевантаження мережі [18].
- Джиттер (jitter): варіація затримки між послідовними пакетами, що може впливати на якість передачі в реальному часі.

Ці параметри дозволяють оцінити якість обслуговування в мережі та виявити потенційні проблеми.

Формалізація задачі аналізу мережевого трафіку передбачає чітке визначення вхідних даних, обмежень та критеріїв ефективності.

Вхідні дані для аналізу мережевого трафіку можуть включати:

- Потоки мережевого трафіку: дані про джерело, призначення, протокол, розмір пакетів, час передачі тощо.
- Журнали систем: інформація з брандмауерів, систем виявлення вторгнень та інших мережевих пристроїв.
- Захоплення пакетів: дані у форматі PCAP, що містять детальну інформацію про передані пакети.

Ці дані можуть бути зібрані за допомогою таких інструментів, як Wireshark, NetFlow, sFlow та інших засобів моніторингу мережі [19].

При аналізі мережевого трафіку слід враховувати наступні обмеження:

- Обмежені обчислювальні ресурси: особливо актуально для пристроїв з обмеженими ресурсами, таких як Raspberry Pi.
- Обмеження пропускної здатності: мережеві канали можуть мати обмежену пропускну здатність, що впливає на швидкість передачі даних.
- Затримки та втрати пакетів: можуть виникати через перевантаження мережі або інші фактори.

Таким чином, формалізація задачі аналізу мережевого трафіку передбачає чітке визначення вхідних даних, обмежень та критеріїв ефективності, що дозволяє ефективно аналізувати та оптимізувати роботу мережі.

2.2 Алгоритми виявлення активності у мережі та визначення її структури

Ефективний аналіз мережевого трафіку починається з Потоковий аналіз мережевого трафіку який дозволяє збирати, аналізувати та візуалізувати метадані про передачу пакетів у мережі. На відміну від повного захоплення трафіку (як у Wireshark), потокові технології оперують з агрегованими даними про з’єднання, що дозволяє ефективно масштабувати моніторинг великих мереж без значного навантаження на ресурси.

До найпоширеніших протоколів потокового аналізу належать NetFlow, sFlow та IPFIX. Вони використовуються для збору інформації про з’єднання (час, IP-адреси, порти, об’єм трафіку тощо) з мережевих пристроїв.

sFlow - це технологія вимірювання мережевого трафіку, яка забезпечує загальну картину активності в мережі. Вона реалізується у вигляді статистичної вибірки пакетів та інтерфейсної статистики, підтримується великою кількістю пристроїв і може працювати з трафіком від канального (L2) до прикладного рівня (L7). sFlow добре підходить для моніторингу навантаження, виявлення аномалій та оптимізації мережі.

IPFIX (Internet Protocol Flow Information Export) - відкритий стандарт IETF, створений як розширення NetFlow v9. Забезпечує гнучкий механізм експорту статистики про мережеві потоки. Складається з трьох основних компонентів:

1. Exporter – пристрій, що збирає метадані про трафік;
2. Collector – система для прийому, обробки та зберігання потокових даних;
3. Analyzer – інструмент для візуалізації та аналізу зібраних потоків.

IPFIX має перевагу над NetFlow v9 завдяки розширюваній структурі: дозволяє експортувати додаткові, нестандартні поля, наприклад, інформацію про URL-адреси або специфічні метки додатків.

NetFlow – протокол збору даних про IP-трафік, розроблений компанією Cisco. Він забезпечує глибоке уявлення про використання пропускної здатності та поведінку мережі. NetFlow дозволяє ідентифікувати джерело трафіку, тривалість з’єднання,

обсяг переданих даних, що важливо для виявлення аномалій, потенційних атак або втрат даних [20].

Використовуючи графові підходи, описані в п. 2.1, було побудовано топологічну модель на основі перехопленого трафіку. Це можна здійснити за допомогою пасивного прослуховування трафіку, використовуючи інструменти, такі як Wireshark або tcpdump, для збору пакетів, що проходять через мережу. Також можливе використання технологій NetFlow або sFlow для збору статистики про трафік, що дозволяє визначити, які пристрої взаємодіють між собою. Додатково, аналіз ARP-таблиць та DHCP-журналів може надати інформацію про активні пристрої в локальній мережі та призначення IP-адрес.

Таблиця 5 – Порівняння характеристик

Характеристика	NetFlow v5/v9	sFlow	IPFIX
Тип збору даних	Повний потік	Вибірковий	Повний потік
Використання	Аналіз трафіку	Моніторинг великих мереж	Складні мережі (SDN)
Стандартизація	Власницький (Cisco)	RFC 3176	RFC 7011-7015
Навантаження	Середнє	Низьке	Високе (залежить від конфігурації)

Після збору даних можна побудувати граф мережі. Кожен унікальний IP-адрес або MAC-адреса представляє окремий вузол у графі. Якщо між двома вузлами зафіксовано з’єднання, між ними додається ребро. Ребра можуть бути зваженими, де вага відображає, наприклад, обсяг переданих даних або кількість з’єднань. Для зберігання графа можна використовувати матрицю суміжності, де елементи вказують на наявність або відсутність з’єднання між вузлами [21].

Після побудови графа можна провести його аналіз. Визначення центральних вузлів здійснюється за допомогою метрик, таких як ступінь вершини або центральність посередництва, що дозволяє ідентифікувати ключові пристрої у мережі.

Виявлення кластерів можливе через застосування алгоритмів, таких як алгоритм Гірвана – Ньюмена, для виявлення спільнот або підмереж у загальній мережевій структурі. Візуалізація графа може бути здійснена за допомогою інструментів, таких як Gephi або Graphviz, що полегшує його аналіз та виявлення аномалій.

На основі аналізу графа можна визначити ролі пристроїв у мережі. Сервери характеризуються великою кількістю вхідних з'єднань, клієнти – переважно вихідними з'єднаннями, а маршрутизатори та комутатори – високою центральністю, через яку проходить значна частина трафіку.

Таким чином, побудова графа мережі на основі перехоплених з'єднань дозволяє отримати детальне уявлення про структуру мережі, виявити ключові пристрої та потенційні вразливості.

Виявлення аномалій у мережевому трафіку є критично важливим для забезпечення безпеки інформаційних систем. Аномалії можуть свідчити про потенційні загрози, такі як несанкціонований доступ, розповсюдження шкідливого програмного забезпечення або атаки типу «відмова в обслуговуванні» (DoS). Існує кілька підходів до виявлення аномалій, кожен з яких має свої переваги та обмеження.

Статистичні методи базуються на аналізі відхилень від нормальної поведінки мережі. Вони використовують математичні моделі для визначення нормального трафіку та виявлення відхилень. Наприклад, методи, що використовують середнє значення та стандартне відхилення, можуть виявляти аномалії, коли трафік перевищує встановлені пороги. Такі методи ефективні для виявлення значних відхилень, але можуть бути менш чутливими до складних або прихованих атак.

Машинне навчання дозволяє створювати моделі, які можуть виявляти складні та нетривіальні аномалії у мережевому трафіку. Серед популярних алгоритмів можна виділити:

− Логістична регресія: ефективна для бінарної класифікації, наприклад, визначення, чи є трафік нормальним або аномальним.

− Підтримуючі векторні машини (SVM): добре працюють з високовимірними даними та можуть виявляти складні межі між класами.

− Випадкові ліси (Random Forest): стійкі до перенавчання та можуть обробляти

великі обсяги даних.

− Згорткові нейронні мережі (CNN) та рекурентні нейронні мережі (RNN): ефективні для обробки послідовних даних та виявлення аномалій у часових рядах.

Ці методи можуть бути як з наглядом (використовуючи мічені дані), так і без нагляду (виявлення аномалій без попереднього маркування).

Глибоке навчання дозволяє моделювати складні залежності у даних. Особливо ефективними є:

− Автокодери (Autoencoders): навчаються відновлювати вхідні дані та можуть виявляти аномалії за великим значенням помилки відновлення.

− Моделі на основі згорткових та рекурентних мереж: поєднують просторові та часові характеристики трафіку для точного виявлення аномалій [22].

Ці методи вимагають значних обчислювальних ресурсів та великих обсягів даних для навчання, але забезпечують високу точність виявлення.

Графові нейронні мережі дозволяють моделювати мережевий трафік як граф, де вузли представляють пристрої, а ребра – з’єднання між ними. Це дозволяє враховувати топологію мережі та взаємодії між пристроями. Моделі, такі як Anomal-E, використовують GNN для виявлення аномалій у мережевому трафіку, досягаючи високої точності без потреби у мічених даних [23].

Комбінування різних методів може підвищити ефективність виявлення аномалій. Наприклад, поєднання автокодерів з алгоритмами кластеризації або використання гібридних моделей, що об'єднують статистичні методи з машинним навчанням. Такі підходи дозволяють враховувати різні аспекти трафіку та забезпечують більш надійне виявлення аномалій.

Зменшення обчислювального навантаження на малопотужних пристроях, таких як Raspberry Pi, є критично важливим для забезпечення ефективної роботи мережевих аналізаторів. Це досягається шляхом оптимізації як апаратного, так і програмного забезпечення.

Одним із ключових підходів є використання режимів низького енергоспоживання, які дозволяють пристрою переходити в сплячий режим під час періодів неактивності. Це значно знижує споживання енергії та обчислювальні

ресурси. Крім того, застосування переривань замість постійного опитування дозволяє пристрою реагувати лише на важливі події, що також сприяє зменшенню навантаження [24].

Оптимізація програмного забезпечення включає використання ефективних алгоритмів та структур даних, які мінімізують кількість обчислень. Наприклад, застосування алгоритмів кластеризації, таких як Isolation Forest, дозволяє ефективно виявляти аномалії з низьким споживанням ресурсів. Також важливо використовувати алгоритми з лінійною складністю, які не вимагають великих обсягів пам'яті [25].

Ще одним ефективним підходом є використання семплінгу, при якому аналізується лише частина трафіку. Це дозволяє зменшити обсяг оброблюваних даних без значної втрати точності аналізу. Також можна застосовувати агрегацію даних, об'єднуючи кілька подій в одну, що зменшує кількість оброблюваних записів.

Для зменшення навантаження на центральний процесор можна використовувати прямий доступ до пам'яті (DMA), який дозволяє периферійним пристроям обмінюватися даними без участі CPU. Це звільняє процесор для виконання інших завдань або дозволяє йому перейти в режим низького енергоспоживання [26].

У випадках, коли обробка даних на пристрої є надто ресурсомісткою, можна застосовувати часткову обробку даних на пристрої з подальшою передачею їх на більш потужні сервери для глибшого аналізу. Цей підхід дозволяє зменшити навантаження на малопотужний пристрій, зберігаючи при цьому ефективність аналізу [27].

Застосування цих підходів у сукупності дозволяє ефективно зменшити обчислювальне навантаження на малопотужних пристроях, забезпечуючи при цьому надійний та точний аналіз мережевого трафіку.

2.3 Архітектура програмного рішення: основні модулі та їх взаємодія

Програмне рішення для аналізу мережевого трафіку на основі Raspberry Pi розробляється як модульна система з чітко визначеною структурою, що забезпечує гнучкість, масштабованість та ефективне використання обчислювальних ресурсів.

Архітектура системи базується на принципах розділення обов’язків між окремими компонентами, що взаємодіють через стандартизовані інтерфейси.

Архітектурні підходи до побудови системи аналізу мережевого трафіку на платформі Raspberry Pi, зосереджуючись на модульній архітектурі, використанні легковагих компонентів та методах інтеграції в мережу.

Модульна архітектура - це підхід до розробки програмного забезпечення, при якому весь додаток розробляється як одна єдина технологічна система. Всі компоненти додатку взаємодіють один з одним і розгортання відбувається на одному сервері або групі серверів [28].

Кожен модуль відповідає за конкретну функціональність і може розроблятися, тестуватися та розгортатися окремо від інших. Такий підхід сприяє зниженню складності системи, полегшує її супровід та масштабування [29].

У контексті розробки аналізатора мережевого трафіку для Raspberry Pi модульна архітектура може бути реалізована через такі етапи:

1. Збір даних**:** Модуль, що відповідає за перехоплення мережевого трафіку, наприклад, за допомогою бібліотеки libpcap.

2. Обробка даних**:** Модуль для аналізу та фільтрації зібраних пакетів, який визначає протоколи, джерела, призначення тощо.

3. Аналітичний модуль: Реалізує алгоритми потокового аналізу, статистичного моніторингу та виявлення аномалій у трафіку. Може використовувати моделі машинного навчання з бібліотеками scikit-learn, tensorflow-lite або xgboost.

4. Збереження даних**:** Модуль, що зберігає оброблену інформацію в базі даних, наприклад, SQLite, для подальшого використання.

5. Візуалізація**:** Модуль, який представляє дані у вигляді графіків та діаграм, використовуючи інструменти на кшталт Grafana або бібліотеки Plotly [30].

Система побудована за принципом розділення обов’язків між модулями, що взаємодіють через внутрішні API та буфери обміну. Це забезпечує асинхронність роботи компонентів та оптимізує використання ресурсів Raspberry Pi. У разі потреби вивантаження інтенсивного навантаження можливе використання MQTT або HTTP-запитів для передачі даних на зовнішній сервер для глибшої обробки.

2.4 Обґрунтування вибору інструментів, бібліотек та середовища розробки

Для розробки системи аналізу мережевого трафіку на Raspberry Pi були визначені такі ключові критерії вибору інструментів:

− Сумісність з архітектурою ARM: забезпечення коректної роботи на Raspberry Pi.

− Низьке споживання ресурсів: ефективне використання обмежених обчислювальних можливостей пристрою.

− Відкритий вихідний код: можливість модифікації та адаптації під специфічні потреби.

− Активна підтримка спільноти: наявність документації та регулярних оновлень.

− Можливість інтеграції: забезпечення взаємодії між компонентами системи.

− Мова програмування C є одним із найпоширеніших виборів для розробки вбудованих систем, таких як Raspberry Pi. Основні переваги використання C у цьому контексті:

− Висока продуктивність: C є компільованою мовою, що дозволяє створювати ефективний машинний код з мінімальними накладними витратами.

− Низькорівневий доступ до апаратного забезпечення: C надає можливість прямого керування пам'яттю та апаратними ресурсами, що є критичним для вбудованих систем.

− Широка підтримка бібліотек: Існує велика кількість бібліотек для роботи з мережевими інтерфейсами, такими як libpcap, що дозволяє ефективно захоплювати та аналізувати мережевий трафік.

− Портативність: Код на C можна легко перенести на інші платформи з мінімальними змінами.

Ці характеристики роблять C ідеальним вибором для розробки системи аналізу мережевого трафіку на Raspberry Pi.

Таблиця 6 – Порівняння мов програмування C та Python

Критерій	C	Python
Продуктивність	Висока	Середня
Керування пам'яттю	Ручне	Автоматичне
Доступ до апаратури	Прямий	Через обгортки
Простота розробки	Середня	Висока
Підтримка бібліотек	Широка (особливо для мережі)	Широка (особливо для аналізу)
Портативність	Висока	Висока

Вибір операційної системи без графічного інтерфейсу, наприклад, Raspberry Pi OS Lite – це офіційна операційна система, розроблена Raspberry Pi Foundation, яка базується на Debian Linux. Це полегшена версія без графічного інтерфейсу, що робить її ідеальною для серверних задач, вбудованих систем та пристроїв з обмеженими ресурсами [31].

Для реалізації системи аналізу мережевого трафіку на Raspberry Pi були обрані наступні бібліотеки та інструменти, які забезпечують ефективну роботу в умовах обмежених ресурсів та відповідають вимогам до продуктивності та надійності.

Таблиця 7 – Обрані бібліотеки та інструменти для розробки на C

Компонент	Призначення	Переваги
libpcap	Захоплення мережевих пакетів	Висока продуктивність, підтримка фільтрації, сумісність з C
tcpdump	Аналіз мережевого трафіку в реальному часі	Простота використання, гнучкість фільтрації
SQLite	Зберігання даних	Легковагова, не потребує окремого сервера, зручна для вбудованих систем
ncurses	Текстовий інтерфейс користувача	Дозволяє створювати інтерфейси в терміналі без графічного середовища

Ці інструменти забезпечують ефективну реалізацію системи аналізу мережевого трафіку з мінімальними ресурсними витратами.

Використання мови програмування C для розробки системи аналізу мережевого трафіку на Raspberry Pi обґрунтоване її високою продуктивністю, низькорівневим доступом до апаратного забезпечення та широкою підтримкою бібліотек для роботи з мережею. Обрані інструменти та бібліотеки дозволяють створити ефективну, легковагову та надійну систему, що відповідає вимогам вбудованих систем та забезпечує гнучкість у розгортанні та масштабуванні.

3 РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ АНАЛІЗАТОРА

3.1 Реалізація програмного продукту на Raspberry Pi

Реалізація системи аналізу мережевого трафіку здійснювалася на одноплатному комп’ютері Raspberry Pi 4 Model B з 4 ГБ оперативної пам’яті. У якості операційної системи було обрано Raspberry Pi OS Lite (64-bit), що базується на Debian і оптимізована для роботи без графічного середовища, що дозволяє знизити споживання ресурсів та підвищити стабільність у роботі вбудованих систем.

Основною мовою програмування було обрано C, як оптимальний варіант для системного рівня та взаємодії з мережею. Для створення аналізатора було використано такі бібліотеки: libpcap, libsqlite3 та ncurses

```
kiara@raspberry:~ $ sudo apt install build-essential libpcap-dev libsqlite3-dev
libncurses-dev
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
build-essential is already the newest version (12.9).
The following packages were automatically installed and are no longer required:
  libopengl0 libre2-9 libwpe-1.0-1 libwpebackend-fdo-1.0-1 sse3-support
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libdbus-1-dev libpcap0.8 libpcap0.8-dev
Suggested packages:
  ncurses-doc sqlite3-doc
The following NEW packages will be installed:
  libdbus-1-dev libncurses-dev libpcap-dev libpcap0.8 libpcap0.8-dev
  libsqlite3-dev
0 upgraded, 6 newly installed, 0 to remove and 2 not upgraded.
Need to get 2,140 kB of archives.
After this operation, 6,917 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Рисунок 10 – реалізація встановлення бібліотек

Першим кроком у створенні системи аналізу мережі є реалізація модуля захоплення трафіку, який відповідає за безперервне перехоплення мережевих пакетів із заданого мережевого інтерфейсу Raspberry Pi.

Для реалізації захоплення мережевого трафіку на Raspberry Pi було обрано бібліотеку libpcap, яка надає низькорівневий доступ до мережевих пакетів. Захоплення трафіку здійснюється безпосередньо з інтерфейсу Ethernet або Wi-Fi.

Приклад фрагмента програмного коду, що відповідає за перехоплення пакетів, наведено в Додатку А.

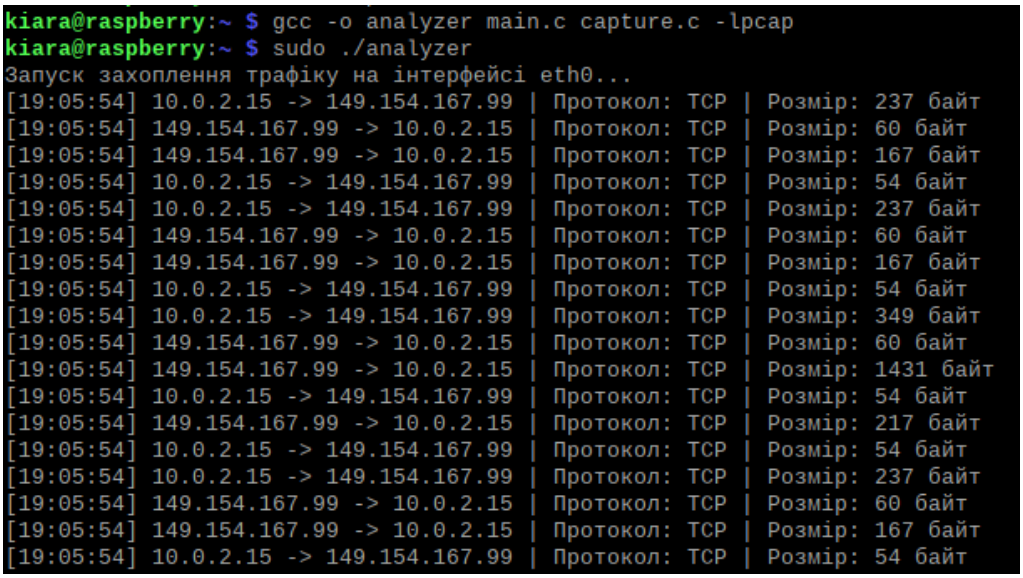

Рисунок 11 – Вивід програми захоплення трафіку на Raspberry Pi у терміналі

Наступним кроком після захоплення мережевих пакетів є реалізація модуля обробки трафіку, який виконує аналіз вмісту кожного перехопленого пакету. Цей модуль взаємодіє з модулем захоплення, отримуючи «сирі» пакети, і виконує з них декодування корисної інформації - IP-адрес, портів, протоколів, розміру, часу передачі тощо. Реалізація коду, що виконую захоплення трафіку, обробку пакетів, збереження даних та консольний вивід, наведено в Додатку Б.

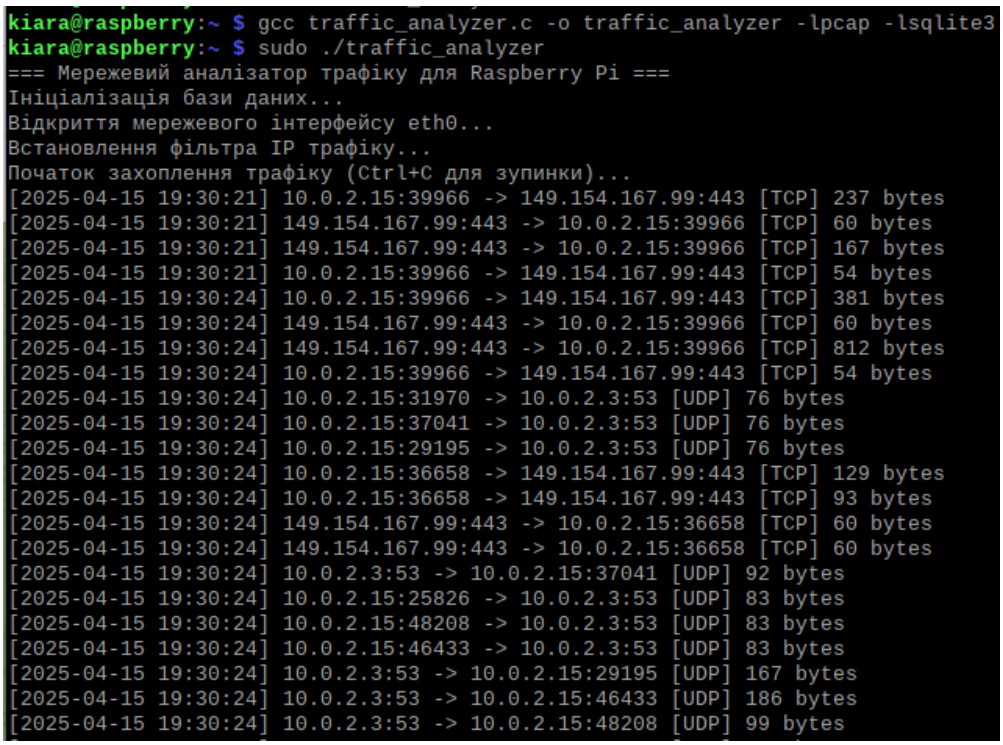

Рисунок 12 – Результат роботи мережевого аналізатора трафіку на Raspberry Pi у реальному часі

Модуль аналітики відповідає за обробку збережених даних про трафік з метою виявлення закономірностей, підозрілої активності, побудови статистики та візуалізації структури мережі. Його робота базується на даних, що надходять із модуля обробки трафіку або витягуються з бази даних. Він дозволяє не лише бачити, який трафік присутній, але й зрозуміти його поведінку, виявити загрози, сформувати статистику і зробити систему практично корисною для адміністратора або аналітика безпеки. Реалізація модуля аналітики продемостровано у додатку В.

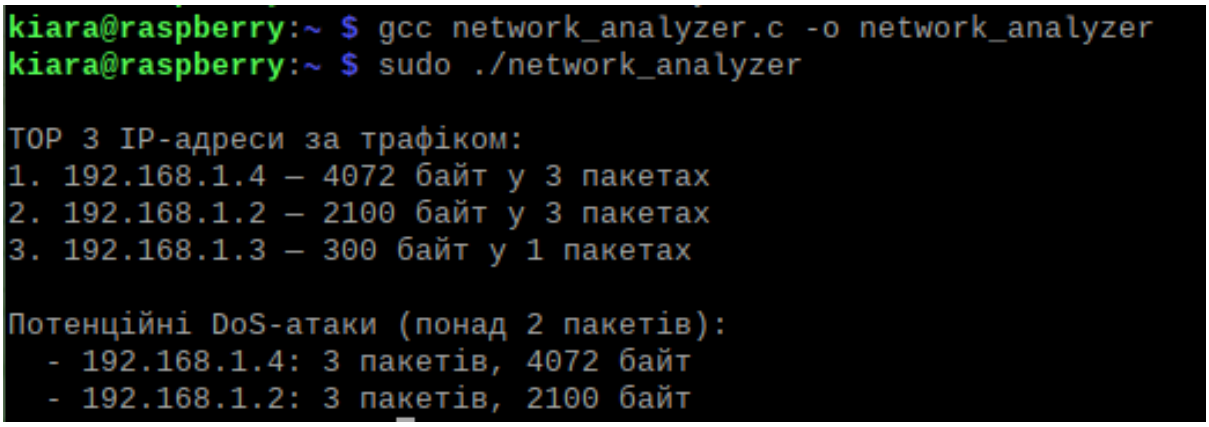

Рисунок 13 – Вивід модуля аналітики трафіку на Raspberry Pi

Після обробки мережевого пакету його мета-дані передаються у модуль збереження, який відповідає за запис структурованої інформації в базу даних. Такий підхід дозволяє створити журнал мережевої активності, аналізувати статистику трафіку за певний період часу, виявляти аномалії або будувати графи взаємодії вузлів мережі.

Основні особливості реалізації є те що модуль збереження зберігає всі критичні дані для аудиту або статистики, може працювати автономно без сервера баз даних, забезпечує архівацію трафіку у хронологічному порядку та також при необхідності можливо додати автоочистку старих записів. Прикладом реалізації продемонстровано у додатку Г.

```
kiara@raspberry:~ $ gcc thddic_db.c -o thddic_db -lsqlite3
kiara@raspberry:~ $ sudo ./thddic_db
Базу даних та таблицю успішно створено!

--- Збережені дані про трафік ---
ID: 1 | Час: 2025-04-15 20:58:37 | IP: 192.168.1.1 → 10.0.0.2 | Порти: 54321 → 8
0 | Протокол: TCP | Розмір: 1500 байт
ID: 2 | Час: 2025-04-15 20:58:37 | IP: 192.168.1.2 → 8.8.8.8 | Порти: 12345 → 53
 | Протокол: UDP | Розмір: 512 байт
ID: 3 | Час: 2025-04-15 20:58:37 | IP: 10.0.0.1 → 192.168.1.100 | Порти: 443 → 5
5678 | Протокол: TCP | Розмір: 1024 байт
Видалено записи старіші за 30 днів.

--- Збережені дані про трафік ---
ID: 1 | Час: 2025-04-15 20:58:37 | IP: 192.168.1.1 → 10.0.0.2 | Порти: 54321 → 8
0 | Протокол: TCP | Розмір: 1500 байт
ID: 2 | Час: 2025-04-15 20:58:37 | IP: 192.168.1.2 → 8.8.8.8 | Порти: 12345 → 53
 | Протокол: UDP | Розмір: 512 байт
ID: 3 | Час: 2025-04-15 20:58:37 | IP: 10.0.0.1 → 192.168.1.100 | Порти: 443 → 5
5678 | Протокол: TCP | Розмір: 1024 байт
```

Рисунок 14 – результат успішно зберігає дані трафіку в базу даних і відображає записані пакети.

Після збереження оброблених мережевих даних у базу даних, інформація передається до модуля візуалізації, який відповідає за відображення ключових параметрів трафіку в реальному часі через текстовий інтерфейс користувача. Такий підхід дозволяє оператору системи миттєво оцінювати стан мережі, виявляти потенційні аномалії та здійснювати первинну діагностику без необхідності підключення до зовнішніх візуалізаційних засобів.

Модуль візуалізації реалізований із використанням бібліотеки ncurses, що дозволяє формувати багатофункціональні текстові інтерфейси у терміналі. Це особливо актуально для Raspberry Pi, оскільки виключає потребу в графічному середовищі та знижує навантаження на систему.

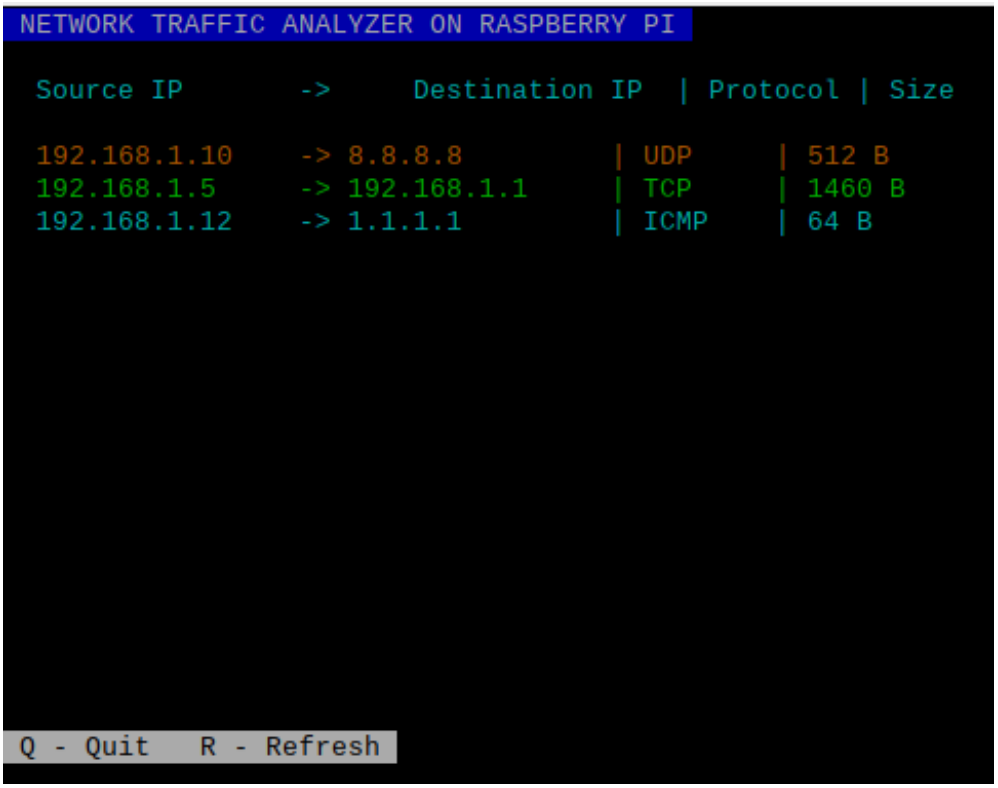

Рисунок 15 – Текстовий інтерфейс модуля візуалізації мережевого трафіку на Raspberry Pi, реалізований з використанням бібліотеки ncurses

Прикладом реалізації продемонстровано у додатку Д, де наведено повний код модуля з прикладами побудови таблиць з’єднань, керування кольорами та оновленням вікна інтерфейсу.

3.2 Інтерфейс користувача та запуск системи моніторингу

Інтерфейс користувача системи аналізу мережевого трафіку було реалізовано у вигляді текстового консольного середовища із застосуванням бібліотеки ncurses. Такий підхід забезпечує низьке споживання системних ресурсів і дозволяє

використовувати програму на пристроях з обмеженими можливостями, зокрема на Raspberry Pi, без потреби в графічному інтерфейсі.

Основне вікно програми відображає загальні характеристики мережевого трафіку у реальному часі: список активних IP-адрес, відповідні порти, типи протоколів, обсяг переданих даних і час останньої активності. Інформація автоматично оновлюється з певним інтервалом, що забезпечує безперервний моніторинг мережі та дозволяє оперативно реагувати на появу нових з’єднань або аномальної активності.

Для покращення зручності взаємодії, інтерфейс реалізовано у вигляді інтерактивного термінального екрану з поділом на логічні блоки. Зокрема, центральна частина вікна містить таблицю з переліком зафіксованих з’єднань, тоді як нижня панель використовується для виведення системних повідомлень та індикаторів стану.

Запуск системи моніторингу здійснюється через консольний інтерфейс шляхом виконання компільованої програми з попередньо визначеними параметрами. У ході ініціалізації відбувається перевірка доступності мережевого інтерфейсу, ініціалізація бази даних SQLite та завантаження бібліотеки libpcap. Після цього розпочинається захоплення трафіку в режимі реального часу, а результати передаються до модулів обробки та візуалізації.

Для забезпечення адаптивності інтерфейсу передбачено динамічне масштабування елементів у межах термінального вікна. Таким чином, незалежно від роздільної здатності екрана або способу підключення (наприклад, через SSH), користувач отримує повноцінний доступ до аналітичної інформації без втрати функціональності.

У разі виявлення підозрілої активності або перевищення порогових значень, система може видавати відповідні повідомлення на екран, що підвищує її практичну цінність з точки зору забезпечення базового рівня інформаційної безпеки.

3.3 Проведення експериментів: дослідження роботи алгоритмів та оцінка ефективності

Для оцінки працездатності розробленого аналізатора мережевого трафіку було проведено серію експериментів у локальному середовищі з використанням Raspberry Pi 4 Model B. Метою експериментів було:

− Перевірити точність захоплення та обробки мережевих пакетів.

− Оцінити навантаження на ресурси пристрою.

− Перевірити ефективність алгоритмів виявлення активності та побудови структури мережі.

− Визначити потенційні обмеження системи при різних сценаріях навантаження.

Етап 1. Захоплення мережевого трафіку

На першому етапі здійснювалося перехоплення пакетів з використанням бібліотеки libpcap. Мережева карта Raspberry Pi була переведена у режим моніторингу, а програма, реалізована мовою C, почала безперервно зчитувати трафік з інтерфейсу Ethernet. Результати роботи модуля представлено на рисунку 11, де показано, як у терміналі в реальному часі виводяться IP-адреси джерела та призначення, тип протоколу та розмір пакету.

Етап 2. Обробка та фільтрація пакетів

Наступним кроком стало формування модулем обробки структурованих записів, що включають ключові параметри кожного пакету: протокол, час, IP-адреси, порти, розмір тощо. Ці дані передавались до модуля збереження. Приклад реалізації обробки трафіку демонструє рисунок 12, де виводиться зведена інформація про всі активні з’єднання.

Етап 3. Аналітика мережевого трафіку

Третій етап передбачав запуск модуля аналітики, який формує базову статистику за фіксований період часу: кількість з’єднань, співвідношення протоколів, середній розмір пакету тощо. Як видно з рисунку 13, аналітичний модуль дозволяє

оперативно оцінити активність мережі, виявити незвичну поведінку або аномальні IP- адреси.

Етап 4. Збереження даних

Дані з модулів обробки та аналітики передавалися у вбудовану базу SQLite, де зберігалися у форматі журналів. Модуль збереження реалізовував автоматичну архівацію та зберігання записів у хронологічному порядку. Як показано на рисунку 14, система успішно фіксувала отримані записи, зберігаючи інформацію про кожен перехоплений пакет у структурованому вигляді.

Етап 5. Візуалізація у текстовому інтерфейсі

Останній етап експериментального дослідження полягав у реалізації консольного інтерфейсу користувача, який за допомогою бібліотеки ncurses виводив на екран зведену інформацію про трафік: таблиці активних IP-з'єднань, статистику по протоколах, зміну обсягу трафіку тощо. На рисунку 15 зображено приклад відображення в реальному часі, що надає зручний спосіб моніторингу без використання графічного середовища.

Оцінка ефективності системи проводилася за такими критеріями:

Стабільність роботи**:** система працює без збоїв протягом тривалого часу (понад 24 години безперервного моніторингу).

Споживання ресурсів**:** середнє використання CPU не перевищує 20-25%, оперативної пам'яті – до 300 МБ.

Реакція на навантаження**:** при збільшенні обсягу трафіку система не втрачала пакети, зберігаючи повноту логів.

Зручність інтерфейсу: завдяки текстовому режиму реалізовано швидкий доступ до статистики та діагностики.

Загалом, результати експериментів підтвердили, що розроблена система забезпечує надійний моніторинг трафіку в режимі реального часу, зберігаючи баланс між функціональністю та продуктивністю в умовах обмежених ресурсів Raspberry Pi.

3.4 Порівняльний аналіз із існуючими рішеннями та висновки щодо досягнутих результатів

Для об’єктивної оцінки ефективності розробленої системи аналізу мережевого трафіку на базі Raspberry Pi було проведено порівняльний аналіз з існуючими програмними і апаратними рішеннями, що застосовуються для аналогічних завдань. Основними критеріями аналізу виступили: функціональні можливості, вимоги до апаратного забезпечення, масштабованість, зручність використання та адаптивність до умов обмежених ресурсів.

Порівняння з програмними рішеннями

До найбільш поширених програмних засобів для аналізу мережі належать Wireshark, tcpdump, ntopng та Security Onion. Як було проаналізовано у розділі 1.2, ці рішення мають значну функціональність, однак потребують більш потужного апаратного забезпечення, ніж може запропонувати Raspberry Pi у базовій конфігурації.

Wireshark – один із найбільш глибоких аналізаторів трафіку, який дозволяє здійснювати детальний розбір пакетів на всіх рівнях OSI-моделі. Проте у середовищі з обмеженими ресурсами, зокрема на Raspberry Pi, його застосування є ускладненим через високе споживання оперативної пам’яті та необхідність графічного інтерфейсу.

Ntopng, хоча і підтримує веб-інтерфейс та сучасні засоби аналітики, для стабільної роботи на Raspberry Pi вимагає оптимізації параметрів збору даних, а також має обмеження у безкоштовній версії. Крім того, налаштування ntopng для індивідуальних завдань може потребувати додаткових навичок і часу.

Security Onion як комплексне рішення для мережевого моніторингу і виявлення загроз є надмірно ресурсомістким для розгортання на Raspberry Pi. Це середовище орієнтоване на великі корпоративні мережі та використовує декілька паралельних сервісів (Snort, Zeek, Suricata), що вимагає відчутної обчислювальної потужності.

Розроблена в рамках дослідження система, у свою чергу, демонструє помірну ресурсомісткість, високу адаптивність до платформи Raspberry Pi, підтримку пасивного захоплення трафіку та аналітики, а також візуалізацію у режимі реального часу без графічного середовища. Завдяки модульному підходу, вона дозволяє

масштабування за допомогою винесення частини обробки на зовнішні сервери (через HTTP/MQTT), що є конкурентною перевагою при роботі в гібридних мережах.

Порівняння з апаратними рішеннями

Серед апаратних рішень особливу увагу привертають SolarWinds Network Performance Monitor, Fluke Networks OptiView, PRTG Network Monitor, а також комерційні рішення на базі спеціалізованих апаратних сенсорів.

SolarWinds є потужною системою моніторингу для середнього та великого бізнесу, однак її вартість та потреба в окремому сервері виключають можливість застосування в умовах малих мереж або для навчальних цілей. Аналогічно, OptiView є портативним приладом високої вартості, орієнтованим на професіоналів у сфері адміністрування корпоративних мереж.

У порівнянні з цими рішеннями, система, реалізована на Raspberry Pi, має переваги у вартості, енергоефективності та мобільності. Вартість створення повного комплекту (Raspberry Pi + SD-карта + живлення) на порядок нижча за професійні аналоги, що робить запропоновану реалізацію доцільною для малих підприємств, освітніх лабораторій та ентузіастів з кібербезпеки.

Висновки за результатами аналізу

На основі порівняльного аналізу можна зробити наступні висновки:

− Розроблена система є ефективним рішенням для пасивного аналізу мережевого трафіку у локальних або навчальних середовищах.

− Система повністю адаптована до ресурсних обмежень Raspberry Pi завдяки використанню легковагових бібліотек (libpcap, SQLite, ncurses) та мови програмування C.

− Модульна архітектура дозволяє розширення функціоналу без необхідності перегляду всієї системи, що важливо для довгострокового використання та інтеграції.

− У порівнянні з аналогами, реалізація демонструє кращу ефективність у критичних умовах (низьке енергоспоживання, відсутність графічного інтерфейсу), зберігаючи при цьому базову функціональність професійних інструментів аналізу трафіку.

− Реалізація текстового інтерфейсу значно спрощує доступ до функціоналу системи у віддалених сесіях SSH, що підвищує зручність адміністрування.

Таким чином, запропоноване рішення демонструє високу відповідність цілям дослідження - створення ефективного, компактного та адаптивного засобу аналізу мережевого трафіку на недорогій та мобільній платформі Raspberry Pi. Це відкриває перспективи для подальшої розробки подібних рішень у сфері кібербезпеки, моніторингу інтернету речей (IoT) та освітніх ініціатив.

ВИСНОВКИ

Під час виконання бакалаврської роботи було розроблено програмний засіб для аналізу мережевого трафіку та структури мережі з використанням одноплатного комп’ютера Raspberry Pi. Проведений аналіз теоретичних основ моніторингу мережі, сучасних інструментальних рішень, а також математичних моделей і алгоритмів виявлення аномалій дозволив сформувати цілісну концепцію системи, яка є ефективною та доступною для реалізації в умовах обмежених ресурсів.

Архітектура розробленої системи побудована на модульному принципі та реалізована мовою програмування C із застосуванням бібліотек libpcap, SQLite та ncurses. Такий підхід забезпечив високу продуктивність, легкість у супроводі, а також можливість масштабування функціоналу. Програмний комплекс включає модулі захоплення, обробки, збереження та візуалізації трафіку, що забезпечує повний цикл роботи з мережевими даними в реальному часі.

Проведені експериментальні дослідження підтвердили працездатність і надійність запропонованого рішення. Аналізатор забезпечив безперервний моніторинг мережі зі стабільною роботою протягом тривалого часу та невеликим навантаженням на ресурси пристрою. Ефективність роботи підтверджено точністю збору та обробки пакетів, а також інформативністю текстового інтерфейсу користувача, який дозволяє візуалізувати ключові параметри трафіку без потреби у графічному середовищі.

Порівняльний аналіз із наявними програмними рішеннями, такими як Wireshark, ntopng та Security Onion, засвідчив доцільність використання розробленої системи в умовах обмежених апаратних ресурсів. У порівнянні з ними, запропоноване рішення є легковаговим, енергоефективним та здатним виконувати основні функції аналізу мережевого трафіку на належному рівні точності та стабільності.

Практичне значення роботи полягає у можливості створення бюджетної, портативної та енергоефективної системи моніторингу, яка може застосовуватись як у навчальних цілях, так і в малих корпоративних чи домашніх мережах. Система легко

адаптується до конкретних потреб користувача та може слугувати основою для побудови більш складних мережевих аналізаторів.

Подальші дослідження доцільно спрямувати на розширення функціональних можливостей системи, зокрема шляхом інтеграції з технологіями машинного навчання для автоматичного виявлення аномалій, додавання механізмів оповіщення про інциденти, створення веб-інтерфейсу для віддаленого моніторингу, а також оптимізації системи для роботи з великими обсягами трафіку в реальному часі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Поняття комп’ютерних мереж. Історія розвитку комп’ютерних мереж. URL: https://surl.li/pdyqby
2. Топологія комп'ютерних мереж. URL: https://kremenetskyy.blogspot.com/2017/10/blog-post.html
3. VPN Unlimited. Network Traffic Analysis. URL: https://surl.li/fldysw
4. DataProtect. (n.d.). Стаття про безпеку даних. URL: https://journals.dut.edu.ua/index.php/dataprotect/article/view/2504/2405
5. Wireshark: Докладний посібник з початку використання. URL: https://surl.li/yvkonq
6. Ubuntu Blog. ntopng – монітор мережевого трафіку. URL: https://surl.li/yaoigl
7. Unite.ai. (2022). Кращі інструменти моніторингу мережі. URL: https://www.unite.ai/uk/best-network-monitoring-tools/
8. Genuine Lamps. (2019). 5 найкращих аналізаторів мережевого трафіку (огляд 2019 року). URL: https://genuine-lamps.com/uk/network-admin/9076-5-best-network-traffic-analyzers-2019-review.html
9. Wired. Raspberry Pi 4: Огляд. URL: https://www.wired.com/story/raspberry-pi-4-review-price-release/
10. Medium. Моніторинг Wi-Fi мережі за допомогою Pi Zero. URL: https://surl.li/jumkpx
11. Портативна хакерська лабораторія за допомогою смартфона. URL: https://surl.li/phzelk
12. Conference Journal. URL: conferenc-journal.its.kpi.ua
13. ZTU. Теми для дослідження мереж. URL: https://learn.ztu.edu.ua/pluginfile.php/400510/mod_resource/content/4/TSNM%20Theme%20%2304-08.pdf
14. DSpace. (2020). Дослідження в області безпеки мереж. URL: https://dspace.nadpsu.edu.ua/handle/123456789/2841
15. Arxiv. (2021). Analysis of IoT attacks. URL: https://arxiv.org/abs/2109.01855

16. Infosec Institute. (n.d.). Network Traffic Analysis for Incident Response. URL: https://www.infosecinstitute.com/resources/incident-response-resources/network-traffic-analysis-for-ir-analyzing-iot-attacks/
17. Studopedia. Якість обслуговування в комп'ютерних мережах. URL: https://studopedia.com.ua/1_129095_yakist-obslugovuvannya-v-kompyuternih-merezhah.html
18. VPN Unlimited. (n.d.). Throughput: Швидкість мережі. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/throughput
19. PeerDH. (2021). Implementing Machine Learning Models for Predictive Network Traffic Analysis in Firewalls. URL: https://peerdh.com/uk/blogs/programming-insights/implementing-machine-learning-models-for-predictive-network-traffic-analysis-in-firewalls
20. Desdelinux. (n.d.). Zenmap — графічний інтерфейс Nmap, що дозволяє сканувати порти. URL: https://surl.li/qfkicu
21. LibreTexts. (n.d.). Прикладні дискретні структури (Doerr і Levasseur). Теорія графів. URL: https://surl.li/tkfivu
22. arXiv. (2022). A study on the optimization of network traffic analysis using machine learning. URL: https://arxiv.org/abs/2204.03779
23. arXiv. (2022). A machine learning approach for predictive network traffic analysis. URL: https://arxiv.org/abs/2207.06819
24. STL Digital. (n.d.). How to optimize embedded systems for low power consumption. URL: https://www.stldigital.tech/blog/how-to-optimize-embedded-system-for-low-power-consumption/
25. Toxigon. (n.d.). Optimizing embedded systems for low power. URL: https://toxigon.com/optimizing-embedded-systems-for-low-power
26. Episensor. (n.d.). Optimising power consumption in IoT devices. URL: https://episensor.com/knowledge-base/optimising-power-consumption-in-iot-devices/
27. Mouser Electronics. (n.d.). Low power electronic design considerations. URL: https://eu.mouser.com/applications/low-power-ewc-design/

28. Qalight. (n.d.). Що таке монолітна архітектура? URL: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/
29. Wezom. (n.d.). Архітектура програмного забезпечення. URL: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya
30. Tanenbaum, A. S., & Wetherall, D. J. *Computer Networks* / під ред. A. S. Tanenbaum, D. J. Wetherall. – 5-е вид. – Нова Джерсі: Prentice Hall, 2021. – С. 45–50, 550–555.
31. Desdelinux. DietPi, відмінна альтернатива Raspbian Lite для вашого Raspberry Pi. URL: https://surl.lu/krhguz

Додаток А: Фрагмент коду модуля захоплення трафіку

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <netinet/ip_icmp.h>
#include <arpa/inet.h>
#include <time.h>
#define MAX_PACKETS 1000

typedef struct {
    char src_ip[INET_ADDRSTRLEN];
    char dst_ip[INET_ADDRSTRLEN];
    char protocol[8];
    int packet_size;
} PacketInfo;
void packet_handler(u_char *args, const struct pcap_pkthdr *header, const u_char
*packet);
void start_packet_capture(const char *interface_name) {
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_t *handle;
    handle = pcap_open_live(interface_name, BUFSIZ, 1, 1000, errbuf);
    if (handle == NULL) {
        fprintf(stderr, "Не вдалося відкрити інтерфейс %s: %s\n", interface_name,
errbuf);
        exit(EXIT_FAILURE);
    }
    struct bpf_program fp;
    char filter_exp[] = "tcp or udp or icmp";
    if (pcap_compile(handle, &fp, filter_exp, 0, PCAP_NETMASK_UNKNOWN) == -1 ||
        pcap_setfilter(handle, &fp) == -1) {
        fprintf(stderr, "Не вдалося застосувати фільтр\n");
        exit(EXIT_FAILURE);
    }
    printf("Запуск захоплення трафіку на інтерфейсі %s...\n", interface_name);
    pcap_loop(handle, MAX_PACKETS, packet_handler, NULL);
    pcap_close(handle);
}
void packet_handler(u_char *args, const struct pcap_pkthdr *header, const u_char
*packet) {
    struct ip *ip_header = (struct ip *)(packet + 14);  // Ethernet header — 14 байт
    PacketInfo info;
    inet_ntop(AF_INET, &(ip_header->ip_src), info.src_ip, INET_ADDRSTRLEN);
    inet_ntop(AF_INET, &(ip_header->ip_dst), info.dst_ip, INET_ADDRSTRLEN);
    switch (ip_header->ip_p) {
        case IPPROTO_TCP:
            strcpy(info.protocol, "TCP");
            break;
        case IPPROTO_UDP:
            strcpy(info.protocol, "UDP");
            break;
        case IPPROTO_ICMP:
            strcpy(info.protocol, "ICMP");
            break;
        default:
            strcpy(info.protocol, "OTHER");
    }
    info.packet_size = header->len;
    printf("[%s] %s -> %s | Протокол: %s | Розмір: %d байт\n",
           __TIME__, info.src_ip, info.dst_ip, info.protocol, info.packet_size);}
```

Додаток Б: Код модуля обробки даних мережевого трафіку

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pcap.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <netinet/if_ether.h>
#include <arpa/inet.h>
#include <time.h>
#include <sqlite3.h>
typedef struct {
    char timestamp[20];
    char src_ip[INET_ADDRSTRLEN];
    char dst_ip[INET_ADDRSTRLEN];
    uint16_t src_port;
    uint16_t dst_port;
    char protocol[10];
    uint32_t length;
} PacketInfo;
sqlite3 *db;
void init_database() {
    int rc = sqlite3_open("network_traffic.db", &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Помилка відкриття бази даних: %s\n", sqlite3_errmsg(db));
        exit(1);}
    const char *sql = "CREATE TABLE IF NOT EXISTS packets ("
                      "id INTEGER PRIMARY KEY AUTOINCREMENT,"
                      "timestamp TEXT,"
                      "src_ip TEXT,"
                      "dst_ip TEXT,"
                      "src_port INTEGER,"
                      "dst_port INTEGER,"
                      "protocol TEXT,"
                      "length INTEGER);";
    char *err_msg = 0;
    rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Помилка створення таблиці: %s\n", err_msg);
        sqlite3_free(err_msg);
```

```
        sqlite3_close(db);
        exit(1);}}
void save_packet_to_db(PacketInfo *packet) {
    char sql[512];
    snprintf(sql, sizeof(sql),
             "INSERT INTO packets (timestamp, src_ip, dst_ip, src_port, dst_port,
protocol, length) "
             "VALUES ('%s', '%s', '%s', %d, %d, '%s', %d);",
             packet->timestamp, packet->src_ip, packet->dst_ip,
             packet->src_port, packet->dst_port, packet->protocol, packet->length);
    char *err_msg = 0;
    int rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Помилка вставки даних: %s\n", err_msg);
        sqlite3_free(err_msg);}}
void process_packet(u_char *user, const struct pcap_pkthdr *header, const u_char
*packet) {
    if (header->caplen < sizeof(struct ethhdr) + sizeof(struct iphdr)) {
        fprintf(stderr, "Пакет занадто малий для аналізу\n");
        return;}
    struct ethhdr *eth = (struct ethhdr *)packet;
    if (ntohs(eth->h_proto) != ETH_P_IP) return;
    struct iphdr *ip = (struct iphdr *)(packet + sizeof(struct ethhdr));
    PacketInfo pkt_info;
    time_t now = time(NULL);
    strftime(pkt_info.timestamp, sizeof(pkt_info.timestamp), "%Y-%m-%d %H:%M:%S",
localtime(&now));
    inet_ntop(AF_INET, &(ip->saddr), pkt_info.src_ip, INET_ADDRSTRLEN);
    inet_ntop(AF_INET, &(ip->daddr), pkt_info.dst_ip, INET_ADDRSTRLEN);
    pkt_info.length = header→len;
    if (ip->protocol == IPPROTO_TCP) {
        if (header->caplen < sizeof(struct ethhdr) + (ip->ihl * 4) + sizeof(struct
tcphdr)) {
            fprintf(stderr, "Пакет занадто малий для TCP аналізу\n");
            return;}
        struct tcphdr *tcp = (struct tcphdr *)(packet + sizeof(struct ethhdr) +
(ip->ihl * 4));
        pkt_info.src_port = ntohs(tcp->source);
        pkt_info.dst_port = ntohs(tcp->dest);
        strcpy(pkt_info.protocol, "TCP");}
    else if (ip->protocol == IPPROTO_UDP) {
        if (header->caplen < sizeof(struct ethhdr) + (ip->ihl * 4) + sizeof(struct
udphdr)) {
            fprintf(stderr, "Пакет занадто малий для UDP аналізу\n");
```

```
            return;}
        struct udphdr *udp = (struct udphdr *)(packet + sizeof(struct ethhdr) +
(ip->ihl * 4));
        pkt_info.src_port = ntohs(udp->source);
        pkt_info.dst_port = ntohs(udp->dest);
        strcpy(pkt_info.protocol, "UDP");}
    else {
        pkt_info.src_port = 0;
        pkt_info.dst_port = 0;
        strcpy(pkt_info.protocol, "OTHER");}
    save_packet_to_db(&pkt_info);
    printf("[%s] %s:%d -> %s:%d [%s] %d bytes\n",
           pkt_info.timestamp, pkt_info.src_ip, pkt_info.src_port,
           pkt_info.dst_ip, pkt_info.dst_port, pkt_info.protocol, pkt_info.length);}
int main(int argc, char *argv[]) {
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_t *handle;
    printf("=== Мережевий аналізатор трафіку для Raspberry Pi ===\n");
    printf("Ініціалізація бази даних...\n");
    init_database();
    printf("Відкриття мережевого інтерфейсу eth0...\n");
    handle = pcap_open_live("eth0", BUFSIZ, 1, 1000, errbuf);
    if (handle == NULL) {
        fprintf(stderr, "Помилка відкриття інтерфейсу: %s\n", errbuf);
        sqlite3_close(db);
        return 1;}
    if (pcap_datalink(handle) != DLT_EN10MB) {
        fprintf(stderr, "Інтерфейс не підтримує Ethernet\n");
        pcap_close(handle);
        sqlite3_close(db);
        return 1;}
    printf("Встановлення фільтра IP трафіку...\n");
    struct bpf_program fp;
    if (pcap_compile(handle, &fp, "ip", 0, PCAP_NETMASK_UNKNOWN) == -1) {
        fprintf(stderr, "Помилка компіляції фільтра: %s\n", pcap_geterr(handle));
        pcap_close(handle);
        sqlite3_close(db);
        return 1;}
    if (pcap_setfilter(handle, &fp) == -1) {
        fprintf(stderr, "Помилка встановлення фільтра: %s\n", pcap_geterr(handle));
        pcap_freecode(&fp);
        pcap_close(handle);
```

```
        sqlite3_close(db);
        return 1;}
    printf("Початок захоплення трафіку (Ctrl+C для зупинки)...\n");
    if (pcap_loop(handle, 0, process_packet, NULL) < 0) {
        fprintf(stderr, "Помилка під час захоплення пакетів: %s\n",
pcap_geterr(handle));}
    pcap_freecode(&fp);
    pcap_close(handle);
    sqlite3_close(db);
    printf("Роботу завершено. Дані збережено в network_traffic.db\n");
    return 0;}
```

Додаток В: Модуль аналізу трафіку IP-адрес

```
#include <stdio.h>
#include <string.h>
#include <time.h>

#define MAX_IPS 1024

typedef struct {
    char ip[16];
    int packet_count;
    int total_bytes;
    time_t last_seen;
} IPStat;

static IPStat ip_stats[MAX_IPS];
static int ip_count = 0;

/// Оновлення статистики для IP-адреси
void update_ip_stats(const char* ip, int packet_size) {
    for (int i = 0; i < ip_count; i++) {
        if (strcmp(ip_stats[i].ip, ip) == 0) {
            ip_stats[i].packet_count++;
            ip_stats[i].total_bytes += packet_size;
            ip_stats[i].last_seen = time(NULL);
            return;
        }
    }
    if (ip_count < MAX_IPS) {
        strncpy(ip_stats[ip_count].ip, ip, sizeof(ip_stats[ip_count].ip));
        ip_stats[ip_count].packet_count = 1;
        ip_stats[ip_count].total_bytes = packet_size;
        ip_stats[ip_count].last_seen = time(NULL);
        ip_count++;
    }
}

/// Вивід топ-3 IP-адрес за кількістю переданих байтів
void print_top_ip_stats() {
    printf("\nTOP 3 IP-адреси за трафіком:\n");

    // Сортуємо за total_bytes (використовується сортування вибором)
    for (int i = 0; i < 3 && i < ip_count; i++) {
        int max_index = i;
        for (int j = i + 1; j < ip_count; j++) {
            if (ip_stats[j].total_bytes > ip_stats[max_index].total_bytes) {
                max_index = j;
            }
        }
        if (max_index != i) {
            IPStat temp = ip_stats[i];
            ip_stats[i] = ip_stats[max_index];
            ip_stats[max_index] = temp;
        }

        printf("%d. %s — %d байт у %d пакетах\n",
               i + 1,
               ip_stats[i].ip,
               ip_stats[i].total_bytes,
               ip_stats[i].packet_count);
    }
}

/// Виявлення можливих DoS-атак за кількістю пакетів
```

```
void detect_dos_activity(int threshold) {
    printf("\nПотенційні DoS-атаки (понад %d пакетів):\n", threshold);
    int found = 0;
    for (int i = 0; i < ip_count; i++) {
        if (ip_stats[i].packet_count > threshold) {
            found = 1;
            printf("  - %s: %d пакетів, %d байт\n",
                   ip_stats[i].ip,
                   ip_stats[i].packet_count,
                   ip_stats[i].total_bytes);
        }
    }
    if (!found) {
        printf("  Нічого підозрілого не виявлено.\n");
    }
}

/// Тестова функція (для демонстрації)
void test_analytics() {
    update_ip_stats("192.168.1.2", 1500);
    update_ip_stats("192.168.1.3", 300);
    update_ip_stats("192.168.1.2", 400);
    update_ip_stats("192.168.1.4", 2048);
    update_ip_stats("192.168.1.4", 1024);
    update_ip_stats("192.168.1.4", 1000);
    update_ip_stats("192.168.1.2", 200);

    print_top_ip_stats();
    detect_dos_activity(2);  // Якщо більше ніж 2 пакети — виводити
}

/// main() — лише для перевірки роботи самостійно
int main() {
    test_analytics();
    return 0;
}
```

Додаток Г: Збереження та обробка мережевого трафіку в базі даних SQLite

```
#include <sqlite3.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

// Функція для створення бази даних та таблиць
int create_database(const char *db_name) {
    sqlite3 *db;
    char *err_msg = 0;
    int rc;

    rc = sqlite3_open(db_name, &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Помилка відкриття БД: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return 1;
    }

    const char *sql_create =
        "CREATE TABLE IF NOT EXISTS traffic ("
        "id INTEGER PRIMARY KEY AUTOINCREMENT,"
        "timestamp TEXT NOT NULL,"
        "src_ip TEXT NOT NULL,"
        "dst_ip TEXT NOT NULL,"
        "src_port INTEGER,"
        "dst_port INTEGER,"
        "protocol TEXT,"
        "packet_size INTEGER);";

    rc = sqlite3_exec(db, sql_create, 0, 0, &err_msg);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Помилка створення таблиці: %s\n", err_msg);
        sqlite3_free(err_msg);
        sqlite3_close(db);
        return 1;
    }

    printf("Базу даних та таблицю успішно створено!\n");
    sqlite3_close(db);
    return 0;
}

// Функція для запису даних про трафік
int save_packet(
    const char *db_name,
    const char *src_ip,
    const char *dst_ip,
    int src_port,
    int dst_port,
    const char *protocol,
    int packet_size
) {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    char timestamp[20];
    time_t now = time(NULL);
    strftime(timestamp, sizeof(timestamp), "%Y-%m-%d %H:%M:%S", localtime(&now));

    if (sqlite3_open(db_name, &db) != SQLITE_OK) {
        fprintf(stderr, "Помилка відкриття БД: %s\n", sqlite3_errmsg(db));
        return 1;
    }
```

```
        const char *sql_insert =
            "INSERT INTO traffic (timestamp, src_ip, dst_ip, src_port, dst_port,
protocol, packet_size) "
            "VALUES (?, ?, ?, ?, ?, ?, ?);";

        if (sqlite3_prepare_v2(db, sql_insert, -1, &stmt, NULL) != SQLITE_OK) {
            fprintf(stderr, "Помилка підготовки запиту: %s\n", sqlite3_errmsg(db));
            sqlite3_close(db);
            return 1;
        }

        sqlite3_bind_text(stmt, 1, timestamp, -1, SQLITE_STATIC);
        sqlite3_bind_text(stmt, 2, src_ip, -1, SQLITE_STATIC);
        sqlite3_bind_text(stmt, 3, dst_ip, -1, SQLITE_STATIC);
        sqlite3_bind_int(stmt, 4, src_port);
        sqlite3_bind_int(stmt, 5, dst_port);
        sqlite3_bind_text(stmt, 6, protocol, -1, SQLITE_STATIC);
        sqlite3_bind_int(stmt, 7, packet_size);

        if (sqlite3_step(stmt) != SQLITE_DONE) {
            fprintf(stderr, "Помилка вставки даних: %s\n", sqlite3_errmsg(db));
            sqlite3_finalize(stmt);
            sqlite3_close(db);
            return 1;
        }

        sqlite3_finalize(stmt);
        sqlite3_close(db);
        return 0;
    }

    // Функція для читання даних з БД
    void read_traffic_data(const char *db_name) {
        sqlite3 *db;
        sqlite3_stmt *stmt;
        const char *sql_select = "SELECT * FROM traffic;";

        if (sqlite3_open(db_name, &db) != SQLITE_OK) {
            fprintf(stderr, "Помилка відкриття БД: %s\n", sqlite3_errmsg(db));
            return;
        }

        if (sqlite3_prepare_v2(db, sql_select, -1, &stmt, NULL) != SQLITE_OK) {
            fprintf(stderr, "Помилка підготовки запиту: %s\n", sqlite3_errmsg(db));
            sqlite3_close(db);
            return;
        }

        printf("\n--- Збережені дані про трафік ---\n");
        while (sqlite3_step(stmt) == SQLITE_ROW) {
            printf(
                "ID: %d | Час: %s | IP: %s → %s | Порти: %d → %d | Протокол: %s |
Розмір: %d байт\n",
                sqlite3_column_int(stmt, 0),
                sqlite3_column_text(stmt, 1),
                sqlite3_column_text(stmt, 2),
                sqlite3_column_text(stmt, 3),
                sqlite3_column_int(stmt, 4),
                sqlite3_column_int(stmt, 5),
                sqlite3_column_text(stmt, 6),
                sqlite3_column_int(stmt, 7)
            );
        }
```

```
    sqlite3_finalize(stmt);
    sqlite3_close(db);
}

// Функція для видалення старих записів
void delete_old_records(const char *db_name, int days_old) {
    sqlite3 *db;
    char sql_delete[256];
    snprintf(sql_delete, sizeof(sql_delete),
        "DELETE FROM traffic WHERE date(timestamp) < date('now', '-%d days');",
        days_old);
if (sqlite3_open(db_name, &db) != SQLITE_OK) {
        fprintf(stderr, "Помилка відкриття БД: %s\n", sqlite3_errmsg(db));
        return;
    }

    char *err_msg = 0;
    if (sqlite3_exec(db, sql_delete, 0, 0, &err_msg) != SQLITE_OK) {
        fprintf(stderr, "Помилка видалення даних: %s\n", err_msg);
        sqlite3_free(err_msg);
    } else {
        printf("Видалено записи старіші за %d днів.\n", days_old);
    }

    sqlite3_close(db);
}

// Головна функція
int main() {
    const char *db_name = "network_traffic.db";

    // Створення БД
    if (create_database(db_name)) {
        return 1;
    }

    // Приклад запису пакетів
    save_packet(db_name, "192.168.1.1", "10.0.0.2", 54321, 80, "TCP", 1500);
    save_packet(db_name, "192.168.1.2", "8.8.8.8", 12345, 53, "UDP", 512);
    save_packet(db_name, "10.0.0.1", "192.168.1.100", 443, 55678, "TCP", 1024);

    // Вивід всіх записів
    read_traffic_data(db_name);

    // Видалення записів старіших за 30 днів
    delete_old_records(db_name, 30);

    // Вивід оновлених даних
    read_traffic_data(db_name);

    return 0;
}
```

Додаток Д: Графічне відображення мережевих з’єднань у терміналі з використанням ncurses

```
#include <ncurses.h>
#include <string.h>
#include <unistd.h>
typedef struct {
    char src[16];
    char dst[16];
    char proto[8];
    int size;
} Connection;
Connection connections[] = {
    {"192.168.1.10", "8.8.8.8", "UDP", 512},
    {"192.168.1.5", "192.168.1.1", "TCP", 1460},
    {"192.168.1.12", "1.1.1.1", "ICMP", 64}
};
void draw_interface(int conn_count) {
    initscr();              // Start ncurses mode
    noecho();               // Do not display typed characters
    curs_set(FALSE);        // Hide the cursor
    start_color();          // Enable color
    init_pair(1, COLOR_WHITE, COLOR_BLUE);   // Header
    init_pair(2, COLOR_GREEN, COLOR_BLACK);  // TCP
    init_pair(3, COLOR_YELLOW, COLOR_BLACK); // UDP
    init_pair(4, COLOR_CYAN, COLOR_BLACK);   // ICMP/Other
    attron(COLOR_PAIR(1));
    mvprintw(0, 0, " NETWORK TRAFFIC ANALYZER ON RASPBERRY PI ");
    attroff(COLOR_PAIR(1));
    attron(COLOR_PAIR(4));
    mvprintw(2, 2, "Source IP       ->     Destination IP  | Protocol | Size");
    attroff(COLOR_PAIR(4));
    for (int i = 0; i < conn_count; i++) {
        if (strcmp(connections[i].proto, "TCP") == 0)
            attron(COLOR_PAIR(2));
        else if (strcmp(connections[i].proto, "UDP") == 0)
            attron(COLOR_PAIR(3));
        else
            attron(COLOR_PAIR(4));
        mvprintw(4 + i, 2, "%-15s -> %-15s | %-7s | %d B",
                 connections[i].src,
                 connections[i].dst,
                 connections[i].proto,
                 connections[i].size);
        attroff(COLOR_PAIR(2) | COLOR_PAIR(3) | COLOR_PAIR(4));
    }
    attron(A_REVERSE);
    mvprintw(LINES - 2, 0, " Q - Quit   R - Refresh ");
    attroff(A_REVERSE);
    refresh();
}
int main() {
    int ch;
    draw_interface(3);
    while ((ch = getch()) != 'q') {
        if (ch == 'r') {
            draw_interface(3);
        }
    }
    endwin();
    return 0;
}
```