Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

МЕТОД ТА ЗАСОБИ МОНІТОРИНГУ ІОТ-ПРИСТРОЇВ У ЛОКАЛЬНИХ МЕРЕЖАХ ІЗ ВИЯВЛЕННЯМ ВІДМОВ

Виконав студент 2 курсу, групи 1КІ-24м
Спеціальності 123 — Комп'ютерна інженерія
_____ Атаманюк Н. Р.
Керівник к.т.н., доц. каф. ОТ
_____ Богомолов С. В.
"15" 12 2025 р.
Опонент к.ф.-м.н., проф. каф. МБІС
_____ Шиян А. А.
"15" 12 2025 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
_____
"18" 12 2025 р.

ВНТУ 2025

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Галузь знань — Інформаційні технології
Освітній рівень — магістр
Спеціальність — 123 Комп'ютерна інженерія
Освітньо-професійна програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н. Азаров О.Д.
"25" вересня 2025 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
студенту **Атаманюку Назару Романовичу**

1 Тема роботи «Метод та засоби моніторингу IoT-пристроїв у локальних мережах із виявленням відмов» керівник роботи Богомолов Сергій Віталійович к.т.н., доцент, затверджено наказом вищого навчального закладу від 24.09.2025 року № 313.

2 Строк подання студентом роботи 04.12.2025

3 Вихідні дані до роботи: середовище моніторингу — локальна мережа Ethernet/Wi-Fi; об'єкти моніторингу — IoT-пристрої, комп'ютери та сервери у локальній мережі; платформа розробки — Python.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз методів та засобів моніторингу мереж IoT, методологія виявлення відмов IoT-пристроїв, розробка методу та програмного засобу моніторингу, дослідження та тестування розробленого методу

моніторингу IoT-пристроїв, економічне обґрунтування розробки, висно
перелік джерел посилання, додатки.

5 Перелік графічного матеріалу: функціональна схема програмн
комплексу.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдан прийн
1-4	Богомолов Сергій Віталійович к.т.н., доцент каф. ОТ		
5	Ратушняк Ольга Георгіївна к.т.н., доцент каф. ЕПВМ		
Нормоконтроль	Швець Сергій Ілліч асистент каф. ОТ		

7 Дата видачі завдання 25.09.2025

8 Календарний план виконання МКР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів МКР	Строк виконання	Примітк
1	Постановка задачі	10.09.2025	виконан
2	Аналіз існуючих рішень	17.09.2025	виконан
3	Розробка структурної схеми	26.09.2025	виконан
4	Розробка функціональної схеми та алгоритмів	01.10.2025	виконан
5	Розробка програмних модулів	03.10.2025	виконан
6	Тестування та апробація розробленого методу	12.10.2025	виконан
7	Розрахунок економічної частини	15.10.2025	виконан
8	Оформлення пояснювальної записки	17.10.2025	виконан
9	Перевірка якості виконання роботи	25.10.2025	виконан
10	Підписи супроводжувальних документів	03.11.2025	виконан
11	Перевірка «антиплагіат»	05.11.2025	виконан
12	Попередній захист	10.11.2025	виконан

Студент _____ Атаманюк Н. Р.

Керівник _____ к.т.н., доц. каф. ОТ Богомолов С. В.

АНОТАЦІЯ

УДК 004.78:004.05

Атаманюк Н. Р. Метод та засоби моніторингу IoT-пристроїв у локальних мережах із виявленням відмов. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп’ютерна Інженерія, Вінниця: ВНТУ, 2025 — 108 с.

На укр. мові. Бібліогр.: 19 назв; рис.: 19; табл. 11.

У роботі розглянуто принципи побудови систем моніторингу IoT-пристроїв у локальних мережах. В роботі проведений аналіз сучасних підходів до вирішення задач моніторингу, зокрема виявлено обмеженість методів, що базуються лише на L3. Обґрунтовано та запропоновано комплексний дворівневий підхід, що поєднує перевірку доступності та аналіз стану сервісів L4. Розроблено метод виявлення відмов на основі "еталонів" сервісів, який дозволяє фіксувати не лише повні відмови зв'язку, але й приховані відмови окремих служб та безпекові аномалії. Розроблено структурну та функціональну схеми програмних засобів, а також алгоритми, що реалізують даний метод.

Ключові слова: моніторинг IoT, виявлення відмов, локальна мережа, L4-моніторинг, L3-моніторинг, фінгерпринтинг сервісів, мережевий сканер.

ABSTRACT

Atamanyuk Nazar. Method and means of monitoring IoT devices in local networks with failure detection. Master's qualification work in specialty 123 - Computer Engineering, Vinnytsia: VNTU, 2025 - 108 p.

In Ukrainian language. Bibliography: 19 titles; fig.: 19; tabl. 11.

The thesis considers the principles of building monitoring systems for IoT devices in local networks. The analysis of modern approaches to solving monitoring tasks was carried out, in particular, the limitations of methods based only on L3 (ICMP) were identified. A comprehensive two-level approach is substantiated and proposed, combining availability checking and service status analysis L4. A failure detection method based on service "fingerprints" has been developed, which allows detecting not only complete communication failures but also hidden failures of individual services and security anomalies. Structural and functional diagrams of the software tool, as well as its operation algorithms implementing this method, have been developed.

Key words: IoT monitoring, failure detection, local network, L4 monitoring, L3 monitoring, service fingerprinting, network scanner.

ЗМІСТ

ВСТУП

Актуальність теми дослідження полягає в тому, що в даний час сучасні локальні мережі стрімко наповнюються пристроями "Інтернету речей" (IoT) — камерами, сенсорами, розумними приладами та іншими. При цьому коло завдань, що потребують вирішення, безперервно розширюється. Поряд з такими базовими завданнями, як контроль мережевої доступності, все більший інтерес викликають і більш складні — виявлення прихованих відмов окремих сервісів або фіксація несанкціонованих змін.

Метою роботи є розробка нового за принципом дії та ефективного за результатами застосування програмного комплексу для моніторингу IoT-пристроїв у локальних мережах, здатного виявляти як мережеві (L3), так і сервісні (L4) відмови.

Для досягнення цієї мети необхідно вирішити такі **задачі**:

— проаналізувати сучасний стан і тенденції розвитку засобів моніторингу локальних мереж, виявити їх недоліки стосовно діагностики IoT-пристроїв;

— на основі проведених досліджень розробити метод дворівневого моніторингу L3/L4;

— розробити програмний комплекс, що реалізує цей метод, для виконання задач з виявлення відмов шляхом порівняння поточного стану сервісів пристрою з попередньо збереженим "еталоном".

Для досягнення поставленої в роботі мети використовуються такі **методи дослідження**:

— системний аналіз;

— об'єктно-орієнтовані методи проектування;

— методи натурного моделювання.

Об’єктом дослідження є процес моніторингу стану IoT-пристроїв у локальних мережах.

Предметом дослідження є структурні та алгоритмічні методи побудови програмного комплексу для виявлення сервісних та мережевих відмов IoT-пристроїв.

Наукова новизна полягає в тому, що вперше був запропонований метод на основі "еталонів" сервісів (фінгерпринтингу), саме для дворівневого моніторингу. Це дозволяє не лише констатувати факт мережевої відмови (L3), але й автоматично виявляти приховані відмови сервісів (L4) та безпекові аномалії що значно підвищує повноту та якість моніторингу.

Практичне значення роботи полягає в тому, що розроблено готовий програмний засіб, який, на відміну від складних комерційних систем, є легким у налаштуванні та призначений для використання у домашніх та малих офісних мережах. Запропонований метод дозволяє користувачам без спеціалізованих знань отримувати чітку діагностику стану IoT-пристроїв та своєчасно реагувати не лише на повні, але й на часткові відмови сервісів.

Апробація результатів роботи здійснена в доповіді на VI Міжнародній науково-практичній інтернет-конференції «Інформаційні технології: моделі, алгоритми, системи — 2025».

Матеріали роботи доповідались та опубліковувались [1]: Богомолов С. В., Атаманюк Н. Р. МЕТОД ТА ЗАСОБИ МОНІТОРИНГУ IoT-ПРИСТРОЇВ У ЛОКАЛЬНИХ МЕРЕЖАХ ІЗ ВИЯВЛЕННЯМ ВІДМОВ. Інформаційні технології: моделі, алгоритми, системи — 2025: VI Міжнародна науково-практична інтернет-конференція (2025). Режим доступу: https://itconf.nuos.edu.ua/2025/publications/%D0%B0-method-and-toolset-for-monitoring-iot-devices-in-local-networks-with-failure-detection/

1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ МОНІТОРИНГУ МЕРЕЖ IoT

Сучасні локальні обчислювальні мережі перетворилися зі статичних систем, що об'єднували обмежену кількість персональних комп'ютерів та серверів, у гетерогенні, динамічні середовища [2]. Вони включають мобільні пристрої, віртуалізовані ресурси та, що найважливіше, масово впроваджувані пристрої "Інтернету речей" (IoT). Така гетерогенність та стрімке зростання кількості підключених вузлів суттєво ускладнюють завдання забезпечення стабільної та безпечної роботи мережі.

У цих умовах, системи моніторингу стають критично важливим компонентом для будь-якої мережевої інфраструктури, від домашніх мереж до великих корпоративних рішень. Вони виконують ключові функції, такі як контроль доступності (фіксація відмов обладнання), аналіз продуктивності (відстеження затримок та навантаження) та забезпечення безпеки (виявлення неавторизованих пристроїв).

Тому необхідно проаналізувати існуючі підходи до моніторингу, їхні переваги та недоліки, з особливим акцентом на їхню застосовність до специфічних вимог IoT-пристроїв.

1.1 Сучасні системи моніторингу стану пристроїв у локальних мережах

Системи моніторингу стану пристроїв у локальних мережах можна класифікувати за методом збору даних та рівнем аналізу. Історично, найбільш базовим методом є активне опитування на рівні L3 за допомогою протоколу ICMP. Відправка ICMP Echo-Request дозволяє визначити два ключові параметри: чи пристрій доступний у мережі та яка затримка (Latency) до нього.

Однак, простий ICMP-моніторинг є недостатнім для комплексної оцінки стану пристрою. Пристрій може коректно відповідати на пінг, але при цьому його ключові служби можуть не працювати (наприклад, веб-сервер IP-камери може "зависнути").

Тому більш просунуті системи моніторингу використовують

спеціалізовані протоколи, найпоширенішим з яких є SNMP (Simple Network Management Protocol) [3]. SNMP-моніторинг працює за моделлю "Менеджер-Агент". Менеджер — це центральна система моніторингу. Агент — це спеціальне програмне забезпечення, що працює на самому пристрої Агент зберігає тисячі параметрів у базі даних MIB (Management Information Base), звідки менеджер періодично їх збирає. Це стандарт де-факто для моніторингу керованого мережевого обладнання та серверів.

На рисунку 1.1 відображено ієрархію та різноманіття підходів до мережевого моніторингу.

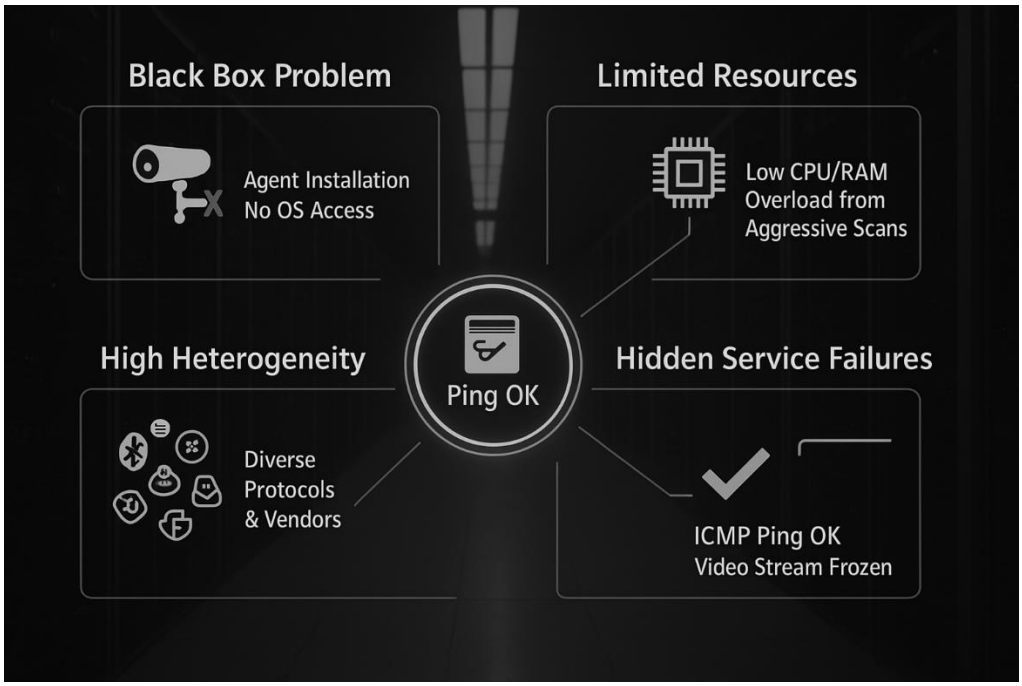

Рисунок 1.1 — Концептуальна схема еволюції систем моніторингу стану пристроїв у локальних мережах

У центрі розташовано базовий рівень — ICMP-моніторинг, який підтверджує доступність пристрою (L3). Вище показано його ключове обмеження: при успішній відповіді на пінг, критичні служби можуть не працювати («Service Failure»).

Таким чином, рисунок 1.1 ілюструє перехід від базового перевірки доступності до комплексного аналізу стану пристроїв, а також підкреслює

обмеження цих систем при моніторингу нових класів обладнання, зокрема IoT-пристроїв, що є ключовою проблемою дослідження.

Незважаючи на потужність цих систем, вони мають суттєві обмеження, які стають критичними при переході до моніторингу IoT, що буде розглянуто в наступному підрозділі.

1.2 Особливості та вразливості IoT-пристроїв як об'єктів моніторингу

Принципова помилка при побудові систем моніторингу в сучасних локальних мережах полягає у застосуванні до IoT-пристроїв тих самих підходів, що й до традиційного IT-обладнання (серверів, ПК, керованих комутаторів). Пристрої "Інтернету речей" мають ряд унікальних особливостей, які роблять стандартні методи моніторингу, такі як SNMP або агентний збір даних, неефективними або взагалі неможливими [4].

До ключових особливостей та вразливостей IoT-пристроїв належать:

— проблема "чорної скриньки";

— обмежені обчислювальні ресурси;

— висока гетерогенність протоколів та виробників;

— схильність до прихованих відмов сервісів.

Більшість IoT-пристроїв, таких як IP-камери, сенсори або побутова техніка, є "чорними скриньками". Користувач не має доступу до їхньої операційної системи. Встановлення стороннього програмного забезпечення, зокрема "агентів" моніторингу, на них не передбачено виробником. Це автоматично виключає цілі класи професійних систем моніторингу з арсеналу засобів.

Крім того, ці пристрої часто функціонують на базі обмежених ресурсів — слабких процесорів та малого обсягу оперативної пам'яті. Вони не розраховані на обробку інтенсивних запитів. Агресивне сканування портів або часте опитування може призвести до перевантаження пристрою, "зависання" або навіть його аварійного перезавантаження.

Найбільш критичною проблемою є схильність до прихованих відмов. IoT-пристрій може ідеально відповідати на ICMP-запити, створюючи ілюзію повної працездатності. Однак його ключова функція може "зависнути" на рівні програмного забезпечення.

Наочно проблеми моніторингу IoT-пристроїв та їхні ключові особливості представлені на рисунку 1.2.

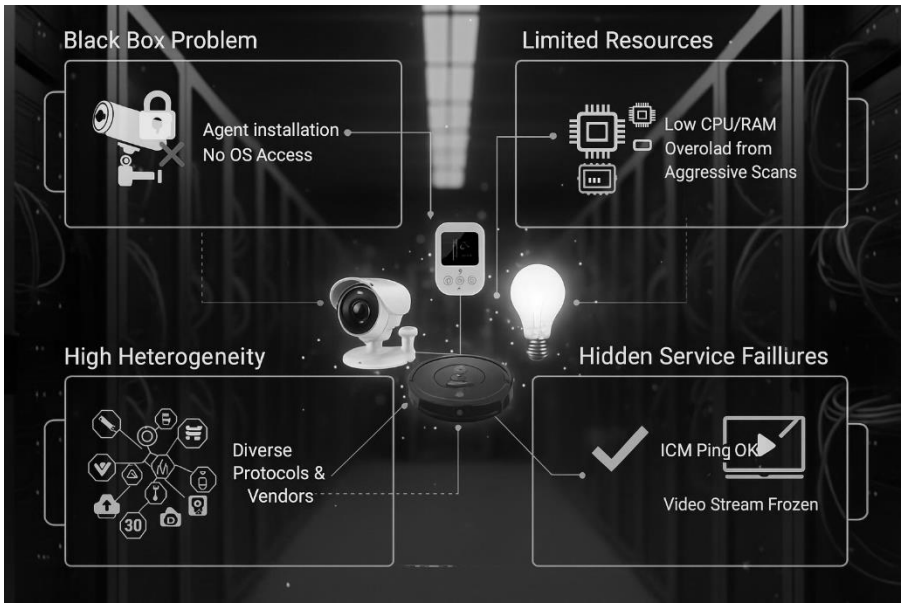

Рисунок 1.2 — Особливості та вразливості IoT-пристроїв

Даний рисунок деталізує ключові виклики, що виникають при спробах моніторингу IoT-пристроїв за допомогою традиційних методів. Він ілюструє проблему «чорної скриньки», обмежені ресурси, високу гетерогенність та приховані відмови сервісів.

При проблемі «чорної скриньки» на IoT-пристроях неможливе встановлення агентного програмного забезпечення, оскільки немає доступу до їхньої операційної системи. Обмежені ресурси на зображенні мікропроцесора та пам'яті призводять до низької обчислювальної потужності IoT-пристроїв, що робить їх вразливими до перевантажень від агресивних запитів моніторингу. Висока гетерогенність представлена різноманітністю іконок протоколів та вендорів, що підкреслює фрагментованість екосистеми IoT, де відсутні єдині

стандарти. При прихованих відмовах сервісів показано, що пристрій може успішно відповідати на ICMP-запити, але при цьому його основна функція може бути порушена.

Окрім неможливості встановлення агентів, проблема «чорної скриньки» ускладнюється закритістю пропрієтарних протоколів, які часто використовуються виробниками для комунікації між пристроєм та хмарним сервером. У багатьох випадках локальне управління пристроєм реалізовано як вторинна функція, а основний потік команд проходить через зовнішні сервери. Це створює ситуацію, коли локальний моніторинг не бачить повного стану пристрою, оскільки частина діагностичної інформації інкапсулюється в зашифровані тунелі до хмари вендора. Відсутність доступу до системних журналів (syslog) або консолі налагодження (UART/JTAG) у штатному режимі роботи робить зовнішній мережевий моніторинг єдиним доступним інструментом контролю.

Також слід враховувати фактор життєвого циклу програмного забезпечення IoT. На відміну від серверних ОС, які отримують регулярні оновлення безпеки, прошивки IoT-пристроїв часто залишаються незмінними роками. Це призводить до накопичення вразливостей у мережевому стеку (наприклад, застарілі версії бібліотек OpenSSL або LwIP). Моніторинг, що базується на аналізі поведінки портів, стає критично важливим, оскільки він дозволяє виявити компрометацію пристрою через відомі вразливості ще до того, як виробник випустить (або не випустить) виправлення. Специфіка апаратних платформ, таких як SoC (System on Chip) на базі архітектур ARM або MIPS, накладає додаткові обмеження на обробку мережевих запитів: черга вхідних з'єднань на таких пристроях є вкрай малою, що вимагає особливої обережності при виборі частоти опитування.

Таким чином, ці особливості доводять, що моніторинг IoT-пристроїв вимагає принципово іншого підходу: він має бути безагентним (agent-less), неагресивним та здатним аналізувати не лише мережеву доступність, але й стан конкретних сервісів.

1.3 Огляд прикладних протоколів IoT та специфіка їх діагностики

Специфіка моніторингу "Інтернету речей" ускладнюється тим, що, на відміну від класичних комп'ютерних мереж, де домінують протоколи HTTP/HTTPS, екосистема IoT використовує широкий спектр спеціалізованих протоколів прикладного рівня. Кожен з них має свої особливості роботи, які впливають на вибір методу діагностики відмов. Розглянемо найбільш поширені протоколи та їх вразливі місця, які неможливо виявити засобами L3-моніторингу.

RTSP (Real Time Streaming Protocol). Це стандартний протокол для керування потоковим відео, який використовується у переважній більшості IP-камер та систем відеоспостереження (NVR). RTSP зазвичай працює на TCP-порту 554. Особливістю цього протоколу є підтримка постійної сесії. Типова проблема IP-камер полягає у переповненні буфера відеокодера або помилці доступу до SD-карти. У такому стані операційна система камери (часто це урізаний Linux) продовжує працювати і відповідати на ICMP-запити (Ping). Проте сервіс RTSP перестає приймати нові з'єднання або розриває існуючі. Для системи моніторингу, що базується лише на пінгу, камера виглядає справною, хоча фактично запис відео не ведеться.

Важливо деталізувати механізм роботи RTSP, щоб зрозуміти природу таких збоїв. Протокол RTSP (Real Time Streaming Protocol) функціонує як «пульт дистанційного керування» для медіа-сервера, використовуючи TCP для керуючих команд (PLAY, PAUSE, TEARDOWN), тоді як сама передача даних (відео/аудіо) часто відбувається через протокол RTP (Real-time Transport Protocol) поверх UDP. Типова архітектура дешевих IP-камер реалізує RTSP-сервер як окремий потік у мікроконтролері. При виникненні помилок у потоці RTP (наприклад, втрата пакетів через перевантаження мережі), керуючий потік RTSP може залишатися активним, але перестати реагувати на нові команди встановлення сесії (SETUP). Стандартні засоби L3-діагностики не здатні розрізнити стан «активного очікування» від стану «зависання логіки» сервера.

Тому ефективний моніторинг повинен передбачати не просто перевірку наявності відкритого порту 554, а й спробу ініціалізації TCP-рукостискання (Three-Way Handshake), що підтверджує готовність ядра системи приймати нові з'єднання.

MQTT (Message Queuing Telemetry Transport) це легковаговий протокол обміну повідомленнями за моделлю "публікація-підписка", який є стандартом для датчиків "розумного будинку", розумних розеток та реле. Працює поверх TCP/IP (стандартний порт 1883). Специфіка MQTT полягає у використанні брокера повідомлень. Якщо IoT-пристрій втрачає з'єднання з брокером через програмний збій у клієнтській бібліотеці, він може залишатися підключеним до Wi-Fi мережі. У цьому випадку фізичний канал зв'язку (L1-L2) та мережевий рівень (L3) функціонують нормально, але передача телеметрії зупинена. Виявити відмову можна шляхом перевірки стану TCP-порта 1883 або аналізу трафіку.

У контексті протоколу MQTT ситуація ускладнюється використанням рівнів якості обслуговування (QoS). Пристрої, що працюють на рівні QoS 0 (Fire and Forget), не очікують підтвердження доставки від брокера. Якщо TCP-з'єднання з брокером розривається некоректно (без відправки пакету DISCONNECT), брокер може деякий час вважати пристрій підключеним (Half-open connection). З боку самого пристрою клієнтська бібліотека може намагатися відновити з'єднання у нескінченному циклі. У такому стані пристрій залишається фізично підключеним до мережі, відповідає на ARP-запити, але фактично не виконує корисну роботу. Зовнішній моніторинг портів дозволяє виявити такий стан за непрямими ознаками: якщо порт, який використовується для вихідного з'єднання з брокером, постійно змінює свій статус або з'являються численні спроби встановлення нових з'єднань (TCP SYN flood) з боку пристрою, це свідчить про проблему в роботі MQTT-клієнта.

HTTP/REST API. Використовується для налаштування пристроїв через веб-інтерфейс (адмін-панелі роутерів, камер, принтерів). Зазвичай використовує порти 80 або 8080. Поширеним сценарієм відмови є "зависання" веб-сервера (наприклад, httpd або nginx на вбудованих системах) через витік пам'яті, при

тому що інші функції пристрою можуть працювати. Втрата доступу до панелі керування є критичною відмовою для адміністратора, яку необхідно виявляти автоматично.

Таким чином, гетерогенність протоколів IoT вимагає від системи моніторингу здатності працювати на транспортному рівні (L4), перевіряючи доступність конкретних портів (554, 1883, 80), оскільки наявність L3-зв'язку не гарантує коректну роботу вищезгаданих прикладних протоколів.

1.4 Аналіз рівнів мережевого моніторингу L3 та L4

Для коректного аналізу методів моніторингу необхідно спершу класифікувати їх за рівнем взаємодії з пристроєм, згідно з мережевою моделлю OSI [5]. Для задач контролю доступності та виявлення відмов, ключовий інтерес становлять два рівні: Мережевий (L3) та Транспортний (L4). Вони відповідають на два фундаментально різні питання: "Чи пристрій у мережі?" та "Чи працюють служби пристрою?".

Моніторинг Мережевого рівня (L3) є найпоширенішим та базовим методом перевірки доступності. Цей рівень відповідає за маршрутизацію та логічну адресацію вузлів у мережі. Основним інструментом моніторингу на цьому рівні є протокол ICMP (Internet Control Message Protocol), зокрема його запити Echo-Request та Echo-Reply, відомі як команда Ping [6]. Переваги цього методу очевидні: він універсальний, підтримується практично кожним мережевим пристроєм (від серверів до побутових IoT-лампочок) і створює мінімальне навантаження на мережу. Він дозволяє виміряти два життєво важливі показники: втрату пакетів та час затримки (RTT). По суті, L3 моніторинг відповідає на питання "Чи існує дана IP-адреса в мережі і чи здатна вона відповідати?".

Однак для сучасних, багатокомпонентних пристроїв, таких як IoT, відповідь "так" на це питання часто створює "ілюзію працездатності". Це фундаментальне обмеження L3 моніторингу. Наприклад, IP-камера може мати

вбудовану операційну систему, ядро якої продовжує коректно обробляти ICMP-запити. Пристрій буде успішно "пінгуватися". В той же час, основний програмний сервіс камери, що відповідає за трансляцію відеопотоку (RTSP-сервер) або за роботу веб-інтерфейсу, може "зависнути" і не відповідати на запити. З точки зору користувача — пристрій не працює і перебуває у стані прихованої відмови. Система моніторингу, що спирається лише на L3, буде помилково повідомляти, що з пристроєм все гаразд.

Саме для вирішення цієї проблеми застосовується моніторинг транспортного рівня. Цей рівень відповідає за доставку даних між конкретними службами, а не лише між пристроями. Він оперує поняттями портів та протоколів, здебільшого TCP та UDP [7]. Моніторинг на L4 відповідає на питання: "Чи запущена на пристрої служба на порту X і чи готова вона приймати з'єднання?". Найпоширеніший механізм такої перевірки — це спроба встановлення TCP-з'єднання. Система моніторингу надсилає пакет SYN на цільовий порт; якщо у відповідь надходить SYN-ACK, трьохстороннє рукостискання відбулося успішно, і сервіс вважається працездатним. Якщо ж у відповідь надходить RST або запит не отримує відповіді, сервіс вважається непрацюючим.

У термінології TCP/IP процес встановлення з'єднання описується діаграмою станів. Коли сканер надсилає SYN-пакет, операційна система IoT-пристрою виділяє ресурси пам'яті та переходить у стан SYN_RCVD. Якщо після цього сканер не завершує процедуру рукостискання (так зване сканування SYN Stealth або Half-open), ці ресурси можуть залишатися заблокованими до моменту тайм-ауту. Для пристроїв з обмеженими ресурсами, якими є більшість IoT-гаджетів, масове накопичення «напіввідкритих» з'єднань може призвести до відмови в обслуговуванні (DoS). Саме тому в даній роботі акцент робиться на використанні методу TCP Connect (повне з'єднання). Хоча цей метод є більш «шумним» і фіксується в логах пристроїв, він є більш безпечним для стабільності IoT-інфраструктури. Коректне закриття з'єднання (надсилання пакетів FIN або RST після перевірки) дозволяє пристрою негайно звільнити пам'ять,

повертаючись у стан LISTEN. Такий підхід відповідає принципу «ввічливого сканування» (polite scanning), що є критично важливим для домашніх мереж, де стабільність роботи «розумного» обладнання є пріоритетом.

Візуалізація фундаментальної відмінності між мережевим (L3) та транспортним (L4) рівнями моніторингу, особливо в контексті IoT-пристроїв, показано на рисунку 1.3. Схема демонструє IP-камеру як типовий приклад IoT-пристрою. Зліва показано успішний L3 Ping до IP-адреси пристрою, що підтверджує його фізичну доступність у мережі. Однак, праворуч ілюструється проблема: незважаючи на успішний L3-моніторинг, спроба L4 перевірки до критичного сервісного порту завершується невдачею, що свідчить про «Hidden Service Failure». Це наочно демонструє ситуацію прихованої відмови, коли ядро пристрою функціонує, але ключова служба — ні. Таким чином, лише комбінований L3/L4 моніторинг дозволяє коректно виявити такі стани, усуваючи ілюзію повної працездатності, яку створює лише Ping.

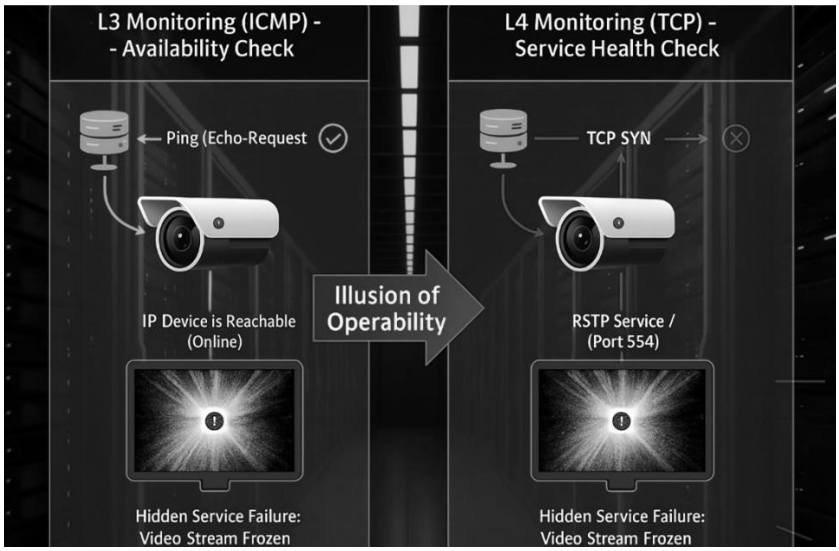

Рисунок 1.3 — Порівняння рівнів моніторингу L3 та L4

Таким чином, аналіз обох рівнів показує, що моніторинг L3 є необхідним, але абсолютно недостатнім для забезпечення повноцінного контролю над IoT-пристроями. Справді ефективна система моніторингу з функцією виявлення відмов зобов'язана бути комплексною (дворівневою): спершу перевіряти фізичну

доступність пристрою (L3), і лише у випадку успіху переходити до перевірки працездатності його ключових сервісів (L4).

1.5 Проблеми виявлення відмов сервісів

Для побудови ефективної системи моніторингу, особливо в контексті IoT, критично важливо розуміти фундаментальну різницю між двома типами відмов: відмовою зв'язку (L3) та відмовою сервісу (L4). Проблематика полягає в тому, що традиційні та найпоширеніші засоби моніторингу приділяють увагу майже виключно першому типу, ігноруючи другий. Це призводить до хибного уявлення про стан мережі та створює "сліпі плями", що є неприпустимим для надійної діагностики.

Відмова зв'язку є "жорсткою" відмовою, яка означає, що пристрій повністю недоступний на мережевому рівні. Це бінарний стан "доступний/недоступний", який легко діагностувати.

Це може статися з кількох причин:

— фізичне відключення пристрою від мережі;

— втрата живлення пристрою;

— збій у роботі проміжного мережевого обладнання;

— повний збій операційної системи пристрою на рівні ядра;

— неправильні налаштування IP-адресації або збій DHCP-клієнта.

Виявлення такої відмови є тривіальним завданням. Воно надійно виконується за допомогою ICMP моніторингу. Система не отримує Echo-Reply у відповідь на Echo-Request, фіксує 100% втрату пакетів і коректно позначає пристрій як "Offline". Будь-яка базова система моніторингу здатна впоратися з цим завданням.

Відмова сервісу, навпаки, є "м'якою" (soft failure) або прихованою відмовою. Це стан, при якому сам пристрій (його операційна система та мережевий стек) функціонує достатньо добре, щоб обробляти ICMP та ARP запити. Однак, ключова програма, служба або процес, заради якого цей пристрій

існує, "зависла", зазнала збою або не може виділити ресурси.

Класичним прикладом є IP-камера. Її мережевий інтерфейс може ідеально відповідати на пінг (L3 "Добрий"), але програмний модуль, що відповідає за трансляцію відеопотоку по протоколу RTSP, припинив роботу. Інший сценарій: відеопотік працює, але вийшов з ладу вбудований веб-сервер, що унеможливлює адміністрування камери. Також коли пристрій пінгується, веб-інтерфейс працює, але основний сервіс друку вийшов з ладу і не приймає завдання.

На Рисунку 1.4 візуально зіставлені дві принципові категорії відмов. Зліва зображено відмову зв'язку, де пристрій повністю недоступний, що виявляється 100% втратою пакетів та класифікується як Offline. Це є прямою і легко діагностованою проблемою. Праворуч представлена відмова сервісу, де пристрій залишається мережево доступним, але критичний сервіс на певному порту не відповідає на запити. Це демонструє ситуацію хибної працездатності, коли система моніторингу, що базується лише на L3, помилково показує статус Online. Ця "сліпа пляма" є критичною для IoT-пристроїв.

Рисунок 1.4 — Порівняння відмови зв'язку та відмови сервісу

Проблема виникає саме на стику цих двох станів. Система моніторингу, що спирається лише на L3, створює ілюзію працездатності. Це призводить до хибної діагностики, збільшення часу на усунення та "сліпої плями" безпеки.

Хибна діагностика — головний дашборд системи моніторингу показує, що всі пристрої "зелені" та працездатні. В цей час користувачі вже не можуть отримати доступ до сервісів. Адміністратор дізнається про проблему від людей, а не від системи, що нівелює саму суть автоматичного моніторингу.

Збільшення часу на усунення являє собою надходження скарги. Адміністратор, довіряючи системі, починає шукати проблему в іншому місці, оскільки моніторинг показує, що з пристроєм все гаразд. Він витрачає час на ручну діагностику, намагаючись підключитися до конкретного порту, і лише тоді виявляє приховану відмову L4.

"Сліпа пляма" безпеки являє собою ігнорування рівня L4 працює і у зворотному напрямку. Якщо зловмисник отримає доступ до IoT-пристрою і запустить на ньому несанкціонований сервіс, L3-моніторинг цього ніколи не помітить.

Таким чином, для IoT-пристроїв, які є "чорними скриньками" з безліччю спеціалізованих сервісів, цей розрив між мережевою доступністю та працездатністю сервісів є головною загрозою. Система, що не здатна автоматично розрізнити ці два типи відмов, не може вважатися повноцінним засобом моніторингу для сучасної мережі.

1.6 Класифікація кіберзагроз та векторів атак на IoT-інфраструктуру

Окрім технічних несправностей, критично важливою задачею моніторингу є забезпечення безпеки. IoT-пристрої часто називають "найслабшою ланкою" в периметрі безпеки мережі через рідкі оновлення прошивок та використання слабких паролів. Розглянемо основні вектори атак, які можна виявити методами моніторингу мережевих портів.

Ботнети (Mirai та аналоги) найвідоміша загроза для IoT. Шкідливе ПЗ сканує мережу в пошуку пристроїв з відкритими портами Telnet (23) або SSH (22) і намагається підібрати пароль за словником (наприклад, admin/admin). Після зараження пристрій стає частиною ботнету. Ознакою зараження або спроби

зараження часто є зміна стану портів. Наприклад, якщо на "розумній лампочці" раптово відкривається порт 23, який раніше був закритий, це є прямою ознакою компрометації або активації несанкціонованого налагоджувального режиму.

Деякі виробники дешевих IoT-пристроїв залишають у прошивці технічні порти для віддаленого налагодження (наприклад, порти вище 10000), які не документовані для користувача. Зловмисники можуть використовувати їх для отримання доступу. Виявлення "прихованих" портів, що не відповідають заявленому функціоналу пристрою, є елементом превентивного захисту.

У випадку успішного злому Reverse Shell атакою, шкідливе ПЗ на пристрої може ініціювати зворотне з'єднання до командного центру (C2 server) або відкрити нестандартний порт для прослуховування команд. Поява будь-якого нового відкритого порту (Listen state) на пристрої, який раніше мав стабільну конфігурацію, повинна розглядатися як інцидент безпеки.

Враховуючи вищенаведене, система моніторингу повинна не лише фіксувати зникнення сервісів (відмови), але й, що не менш важливо, появу нових активних сервісів. Це дозволяє реалізувати функцію виявлення вторгнень (IDS) на базовому рівні без необхідності аналізу глибокого вмісту пакетів (DPI), що є надто ресурсоємним для звичайних мереж.

1.7 Порівняльний аналіз існуючих програмних засобів моніторингу

На ринку існує велика кількість програмних засобів для моніторингу мережі. Вони варіюються від простих утиліт командного рядка до комплексних комерційних платформ. Для визначення актуальності розробки власного методу та засобу, необхідно провести порівняльний аналіз найпоширеніших представників. Розглянемо Nmap, Zabbix та PRTG Network Monitor на їхню відповідність специфічним вимогам моніторингу IoT-пристроїв, що були визначені у попередніх підрозділах.

Nmap є потужним, вільним та відкритим інструментом, що став де-факто стандартом для сканування мереж. Його основне призначення — виявлення

активних хостів у мережі, визначення операційних систем та, що найважливіше, сканування відкритих портів L4. Nmap використовує різноманітні методи сканування, включно з TCP SYN, TCP Connect, UDP та іншими, що дозволяє йому з високою точністю "промацати" пристрій.

Однак, Nmap за своєю суттю є інструментом для "моментального знімку" (snapshot) стану мережі, а не системою постійного моніторингу. Він не призначений для безперервної роботи та відстеження змін у часі. Щоб виявити відмову сервісу L4 за допомогою Nmap, адміністратору довелося б вдатися до складних та неоптимальних рішень. Наприклад, налаштувати періодичний запуск Nmap через планувальник завдань, зберігати результати кожного сканування, та запускати додатковий скрипт, який би порівнював новий "знімок" стану зі старим для виявлення відмінностей.

Такий підхід є громіздким, не забезпечує реакції в реальному часі та не має вбудованої логіки "еталону". Nmap не здатен самостійно відрізнити легітимний сервіс від нової небезпечної служби; він просто констатує факт. Він є засобом аудиту, але не автоматичного моніторингу відмов.

Zabbix та PRTG представляють клас комплексних корпоративних платформ моніторингу [8]. Це надзвичайно потужні системи, що вимагають виділеного сервера та бази даних для зберігання історичних даних. Вони здатні моніторити тисячі параметрів: від температури процесора сервера до завантаженості магістральних каналів зв'язку.

Основна перевага цих систем полягає в їхній гнучкості, яка досягається переважно двома шляхами:

— моніторинг через агентів, що встановлюються на пристрої та збирають глибоку телеметрію зсередини операційної системи;

— моніторинг через SNMP, що є ідеальним для роботи з керованим мережевим обладнанням, таким як комутатори, маршрутизатори та сервери.

Для узагальнення аналізу, зведемо характеристики розглянутих систем у порівняльну таблицю (табл. 1.1).

Таблиця 1.1 — Порівняльний аналіз програмних засобів моніторингу

Критерій	Nmap	Zabbix / PRTG	Розроблюваний Засіб
Основне призначення	Аудит безпеки, "знімок" мережі	Комплексний моніторинг інфраструктури	Спеціалізований моніторинг IoT
Метод моніторингу	Активне сканування (snapshot)	Агенти, SNMP, WMI	Безагентне сканування L3/L4
Виявлення відмов L3	Так	Так	Так
Виявлення відмов L4	Лише вручну	Так (через складне налаштування)	Так
Виявлення аномалій L4	Ні (лише констатація)	Ні (потребує складних скриптів)	Так (ключова функція, "метод еталону")
Вимоги до ресурсів	Низькі	Дуже високі (сервер, БД)	Дуже низькі (додаток для ПК)
Складність налаштування для IoT	Середня (потребує скриптів)	Висока (надлишковість)	Низька (автоматичне "навчання")

Аналіз показує, що на ринку існує незаповнена ніша. З одного боку, існують прості сканери як Nmap, які не є системами моніторингу. З іншого боку, існують надскладні корпоративні системи як Zabbix, що є надлишковими та не пристосованими до "чорних скриньок" IoT. Відсутній легкий, безагентний спеціалізований засіб, орієнтований саме на дворівневий L3/L4 моніторинг IoT-пристроїв з виявленням аномальних відхилень від "еталону" сервісів. Розробка такого засобу і є метою даної роботи.

2 МЕТОДОЛОГІЯ ВИЯВЛЕННЯ ВІДМОВ IoT-ПРИСТРОЇВ

Існуючі засоби моніторингу не відповідають повною мірою специфічним вимогам IoT-пристроїв. Основна проблема полягає у їхній нездатності розрізняти відмови мережевого рівня L3 та сервісного рівня L4, а також у відсутності гнучких механізмів для виявлення аномалій у "чорних скриньках".

Для вирішення цієї проблеми необхідна розробка власної методології моніторингу. Ця методологія має бути, з одного боку, достатньо простою для реалізації у легкому програмному засобі, а з іншого — достатньо комплексною, щоб забезпечити повне покриття основних загроз працездатності IoT-пристроїв.

Потрібно визначити чітку класифікацію відмов, сформулювати центральну ідею методу "еталонів" та обґрунтувати вибір технологічного стеку для подальшої програмної реалізації.

2.1 Класифікація типів відмов: мережеві, сервісні та безпекові

Основою для розробки будь-якого ефективного методу моніторингу є визначення чіткої та вичерпної таксономії станів об'єкта. У контексті даної роботи, простий бінарний статус "онлайн/офлайн" є абсолютно недостатнім для адекватної оцінки працездатності IoT-пристрою. Необхідна більш глибока класифікація, що враховує рівень виникнення несправності. Всі можливі відмови доцільно розділити на три фундаментальні категорії, що відрізняються за своєю природою та вимагають різних методів виявлення: мережеві відмови, сервісні відмови та безпекові аномалії.

З точки зору класичної теорії надійності складних технічних систем, запропонована класифікація корелює з поняттями «відмов функціонування» та «параметричних відмов». Мережева відмова (L3) є аналогом раптової відмови, яка характеризується стрибкоподібною зміною параметрів об'єкта (в даному випадку — доступності) до критичних меж. Сервісна відмова (L4) ближча за своєю природою до поступової або деградаційної відмови, коли працездатність втрачається не повністю, а лише в частині виконання специфічних функцій.

Важливим аспектом є також часова характеристика відмов. Окрім постійних відмов (permanent failures), які вимагають втручання оператора для відновлення, у мережах IoT часто зустрічаються переміжні (intermittent) та короткочасні (transient) збої. Система моніторингу повинна мати вбудовані механізми фільтрації таких подій (наприклад, гістерезис або лічильник послідовних помилок), щоб не перевантажувати користувача хибними сповіщеннями про кожну втрачену пачку даних.

Першою та найбільш очевидною категорією є мережева відмова (L3). Це "жорстка" відмова, яка означає повну недоступність пристрою на мережевому рівні L3 моделі OSI. Вона свідчить про те, що пристрій не може брати участь у мережевій комунікації. Причинами такої відмови зазвичай є втрата живлення, фізичне відключення кабелю Ethernet, збій Wi-Fi модуля, або повний крах операційної системи на рівні ядра. Діагностика цієї відмови є першим та обов'язковим кроком будь-якого моніторингу, оскільки вона унеможливлює будь-яку подальшу, глибшу діагностику. Ця перевірка виконується шляхом надсилання ICMP Echo-Request (Ping) та перевірки наявності запису у ARP-таблиці. Якщо пристрій не відповідає, він позначається як "Offline", і подальший аналіз сервісів стає неможливим.

Друга, і значно складніша для виявлення, категорія — це сервісна відмова (L4). Це "м'яка" або прихована відмова, яка є джерелом більшості діагностичних помилок у простих системах моніторингу. Вона виникає, коли пристрій знаходиться в мережі і коректно відповідає на ICMP запити (L3 "Добрий"), але його ключова функція або програмний сервіс не працює. Це збій на транспортному рівні L4. Наприклад, IP-камера пінгується, але її веб-сервер на порту 80 не відповідає на TCP запити, унеможливлюючи налаштування. Або, що гірше, її RTSP-сервер на порту 554 "завис", і трансляція відео зупинилася. З точки зору користувача, пристрій не виконує свою функцію, але система L3-моніторингу помилково повідомляє про його працездатність.

Третя категорія — безпекова аномалія (L4). Це окремий випадок відхилення на рівні L4, який є дзеркальним відображенням сервісної відмови.

Замість зникнення очікуваного сервісу, відбувається поява неочікуваного та потенційно небезпечного сервісу. Наприклад, на IoT-камері, для якої нормальною є робота лише порту 80 (веб-інтерфейс), раптово відкривається порт 23 (Telnet) або порт 22 (SSH). Традиційні сканери просто констатують наявність цього порту, але не можуть класифікувати його як "аномалію", оскільки не мають "еталону" нормального стану.

Рисунок 2.1 візуально демонструє та розмежовує три ключові типи відмов, що вимагають індивідуального підходу до виявлення. Схема показує, як пристрій переходить від стану «Normal Operation» до одного з трьох аномальних станів. Мережева Відмова (L3) ілюструє повну відсутність зв'язку, що призводить до Ping Fail та статусу Offline. Це «жорстка» відмова. Сервісна Відмова (L4) демонструє, що пристрій Ping OK, але очікуваний критичний сервіс раптово стає Closed/Time-out. Це прихована «м'яка» відмова. Безпекова Аномалія (L4) показує, що пристрій також Ping OK, але на ньому виявляється Unexpected Open Port, який не входить до «еталону» нормальних сервісів. Ця візуалізація підкреслює необхідність двоступеневого L3/L4-аналізу для уникнення хибної діагностики.

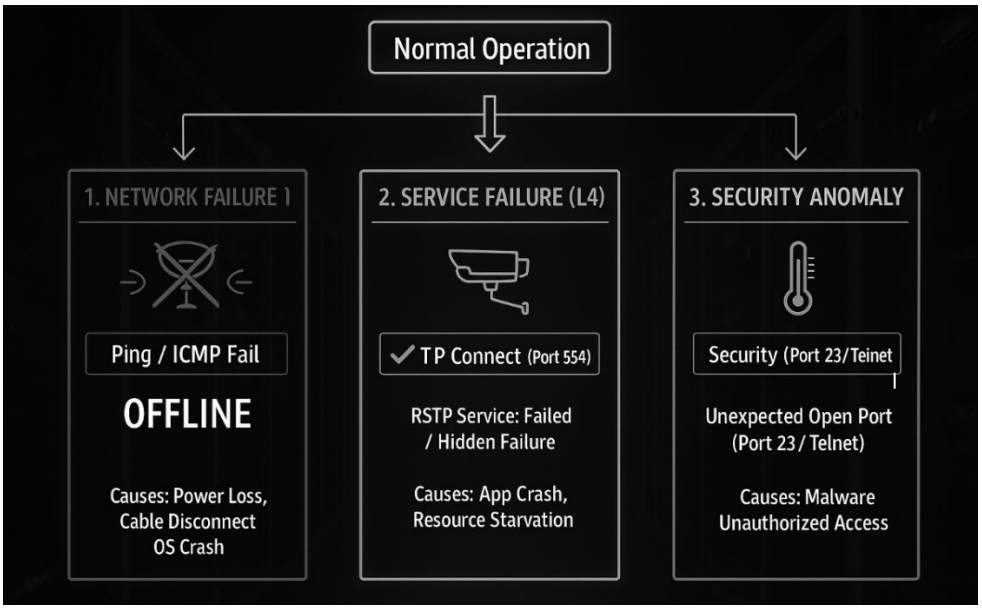

Рисунок 2.1 — Таксономія відмов у системах моніторингу IoT-пристроїв

Таким чином, ця класифікація є фундаментальною для розроблюваної методології. Ефективний метод моніторингу, що претендує на повноту, повинен бути здатен не просто перевіряти стан "ввімкнено/вимкнено". Він зобов'язаний автоматично виявляти та чітко розрізняти усі три перелічені типи відмов, надаючи адміністратору повну та достовірну картину стану кожного IoT-пристрою в мережі. Це дозволяє перейти від простої констатації факту L3-доступності до повноцінної L4-діагностики.

2.2 Математична модель системи моніторингу

Для формалізації задачі виявлення відмов та аномалій доцільно використати апарат теорії множин. Це дозволить строго описати алгоритми, які закладено в програмну реалізацію методу.

Нехай $D = \{d_1, d_2, \dots, d_n\}$ — множина всіх IoT-пристроїв, виявлених у локальній мережі. Для кожного пристрою $d_i \in D$ визначимо його стан у момент часу *t* як вектор параметрів:

$$S_i(t) = \langle IP_i, MAC_i, L3_i(t), P_i(t) \rangle, \quad (2.1)$$

де IP_i, MAC_i — мережеві ідентифікатори пристрою;

$L3_i(t) \in \{0, 1\}$ — статус мережевої доступності (0 — Offline, 1 — Online);

$P_i(t)$ — множина відкритих TCP-портів, виявлених в момент часу *t*.

Під час фази навчання $(t = t_0)$ для кожного пристрою фіксується еталонна множина портів P_i^{etalon}. Ця множина відображає нормальний режим функціонування пристрою:

$$P_i^{etalon} = P_i(t_0) = \{p_1, p_2, \dots, p_k\}, \quad (2.2)$$

де p_k — номер відкритого порту TCP.

Сервісна відмова визначається як стан, при якому пристрій доступний на мережевому рівні ($L3_i(t) = 1$), але частина сервісів, зафіксованих в еталоні, перестала відповідати. Нехай $F_i(t)$ — множина портів, що зазнали відмови. Вона визначається як різниця множин еталону та поточного стану:

$$F_i(t) = P_i^{etalon} \backslash P_i(t). \tag{2.3}$$

Умова фіксації сервісної відмови:

$$|F_i(t)| > 0 \text{ при } L3_i(t) = 1. \tag{2.4}$$

Безпекова аномалія визначається як поява нових відкритих портів, які не передбачені еталоном. Для підвищення точності та зменшення хибних спрацювань (наприклад, динамічні порти UPnP), введемо множину "відомих небезпечних портів" P^{risk} (наприклад, {21, 22, 23, 445, ...}). Нехай $T_i(t)$ — множина виявлених загроз. Вона визначається як перетин різниці поточного стану і еталону з множиною ризикових портів:

$$T_i(t) = (P_i(t) \backslash P_i^{etalon}) \cap P^{risk}. \tag{2.5}$$

Умова фіксації загрози:

$$|T_i(t)| > 0 \text{ при } L3_i(t) = 1. \tag{2.6}$$

На основі визначених множин можна сформулювати функцію інтегрального стану $Status(d_i)$, яку реалізує програмний засіб:

$$Status(d_i) = \begin{cases} Offline, \text{якщо } L3_i(t) = 0 \\ L4\ Failure, \text{якщо } L3_i(t) = 1 \wedge |F_i(t)| > 0 \\ L4\ Threat, \text{якщо } L3_i(t) = 1 \wedge |F_i(t)| > 0 \end{cases} \tag{2.7}$$

Математична модель дозволяє однозначно класифікувати стан пристрою та алгоритмізувати процес прийняття рішень у програмному коді, використовуючи операції над множинами (Set Difference, Set Intersection), що є обчислювально ефективними.

Варто зазначити, що запропонована модель базується на дискретному часі t, де кожен крок відповідає одному циклу сканування системи. Для підвищення стійкості моделі до випадкових флуктуацій мережі (network jitter), доцільно ввести поняття гістерезису у визначення функції статусу. Нехай H — це глибина історії спостережень (кількість останніх циклів сканування). Тоді стабільний стан відмови $L3_{failure}$ фіксується не миттєво, а лише за умови виконання кон'юнкції подій протягом H тактів:

$$Status(d_i) = \bigwedge_{i=0}^{H}(L3_{i-t} = 0). \quad (2.8)$$

Аналогічний підхід застосовується і до сервісних відмов. Це дозволяє математично описати механізм фільтрації хибних спрацювань, який є критично важливим для реальних мереж, де можливі короткочасні втрати пакетів.

Крім того, множина ризикових портів P_{risk} не є статичною константою у загальному випадку, а може бути описана як динамічне об'єднання глобально відомих загроз P_{global} (стандартні порти Telnet, SSH, RDP) та локально визначених правил P_{local}, які задає адміністратор:

$$P_{risk} = P_{global} \cup P_{local}. \quad (2.9)$$

Така формалізація дозволяє гнучко адаптувати модель під конкретні політики безпеки, розширюючи можливості виявлення аномалій без зміни основного алгоритмічного ядра системи. Операції над множинами, закладені в основу моделі, легко транслюються у високорівневі структури даних мови Python, такі як set, що забезпечує високу швидкодію обчислень навіть при

великій кількості пристроїв у мережі.

2.3 Вимоги до програмного засобу моніторингу

Методологія моніторингу формує чіткий набір функціональних вимог до програмного засобу, який буде її реалізовувати. Ці вимоги визначають, що саме програма повинна вміти робити для коректної діагностики стану мережі та виявлення всіх трьох класифікованих типів відмов. Засіб повинен не просто збирати дані, але й виконувати їх інтелектуальний аналіз в реальному часі, базуючись на попередньо "вивчених" даних.

Першою та базовою вимогою до програмного засобу є його здатність надійно визначати мережеву доступність пристрою на рівні L3. Це виступає "першим фільтром" або логічним бар'єром; без перевірки цього рівня будь-який подальший аналіз L4 не має сенсу. Програмний засіб повинен періодично надсилати ICMP Echo-Request (Ping) пакети до кожного пристрою [9]. Водночас, він має не просто фіксувати факт відповіді, а й аналізувати її якість.

Це означає, що реалізація L3-моніторингу має включати виявлення повної відмови, тобто коректно ідентифікувати 100% втрату ICMP-пакетів та маркувати пристрій статусом "Offline". Також вона вимагає аналізу якості зв'язку шляхом вимірювання часу затримки RTT та відсотка втрат, класифікуючи стан як "Середній" чи "Поганий" при високих показниках. Додатково, засіб повинен використовувати ARP-запити для первинного виявлення, оскільки деякі пристрої можуть блокувати ICMP. Таким чином, перша вимога — це реалізація повноцінного L3-шлюзу, який приймає рішення про подальшу L4-перевірку лише для тих пристроїв, що мають стабільний L3-зв'язок.

Другою, і більш складною, вимогою є реалізація механізму виявлення прихованих сервісних відмов L4. Ця вимога безпосередньо випливає з методу "еталонів". Програмний засіб повинен мати функціонал для автоматизованого порівняння "еталонного" стану пристрою з його поточним станом.

Для цього засіб повинен вміти зберігати "еталон", тобто надійно

записувати у файл список очікуваних відкритих портів, асоційований з MAC-адресою пристрою. Він повинен вміти проводити L4-сканування, виконуючи швидке TCP-сканування портів пристрою, якщо L3-перевірка пройшла успішно. Він повинен виконувати логіку порівняння, реалізуючи алгоритм, який для кожного порту з "еталонного" списку перевіряє його наявність у "поточному знімку" відкритих портів. Зрештою, він має класифікувати відмову L4: якщо очікуваний порт відсутній у поточному скані, засіб повинен негайно класифікувати цей стан як "Сервісна відмова L4". Ця вимога є ключовою для вирішення проблеми "ілюзії працездатності".

Третьою вимогою, дзеркальною до попередньої, є здатність засобу виявляти несанкціоновані зміни у конфігурації пристрою, що можуть свідчити про загрозу безпеці. Якщо вимога описує пошук "чогось, що зникло", то ця вимога описує пошук "чогось, що з'явилося".

Програмний засіб повинен реалізувати зворотний алгоритм порівняння. Він має проводити аналіз нових портів, взявши "поточний знімок" відкритих портів і порівнявши його з "еталонним" списком. Головна вимога — це виявлення аномалії: якщо у "поточному знімку" знайдено порт, якого не було в "еталоні" і який при цьому входить до списку відомих небезпечних сервісів, засіб повинен класифікувати цей стан як "Безпекова аномалія L4". Також важливою є вимога до сповіщення, оскільки цей тип відмови має мати найвищий пріоритет та чітко візуально відрізнятися від звичайної сервісної відмови, оскільки несе пряму загрозу.

2.4 Обґрунтування вибору засобів розробки

Вибір технологічного стеку, тобто набору мов програмування та бібліотек, має вирішальне значення для успішної реалізації програмного засобу. Вимоги до засобу, такі як необхідність низькорівневої роботи з мережевими пакетами L3, одночасне виконання швидких L4-перевірок та наявність сучасного графічного інтерфейсу, диктують потребу у гнучкій та потужній платформі.

В якості основної мови програмування було обрано Python. Цей вибір обґрунтований тим, що Python є високорівневою інтерпретованою мовою, яка поєднує простоту синтаксису з величезною екосистемою бібліотек [10]. Його стандартна бібліотека містить потужні модулі для мережевої взаємодії та багатопоточності, а здатність інтегруватися зі сторонніми бібліотеками дозволяє покрити всі специфічні вимоги даної роботи.

Для реалізації моніторингу L2 та L3 було обрано бібліотеку Scapy [11]. На відміну від стандартного модуля socket, Scapy є потужним інструментом для маніпуляції пакетами. Вона надає програмісти повний контроль над мережевими пакетами, дозволяючи створювати ARP-запити для виявлення пристроїв на рівні L2 та ICMP-пакети для пінгування на рівні L3. Використання Scapy є критично важливим для отримання доступу до низькорівневих мережевих функцій, які недоступні у стандартних бібліотеках, і є необхідним для реалізації першого кроку моніторингу.

Для реалізації моніторингу L4 було обрано комбінацію стандартного модуля socket та ThreadPoolExecutor. Хоча Scapy і має можливості для TCP-сканування, метод TCP Connect Scan через модуль socket є більш стабільним, надійним та, що важливо, не вимагає підвищених прав (прав адміністратора) для своєї роботи [12]. Використання ThreadPoolExecutor дозволяє виконувати сканування портів паралельно у декількох потоках, що кардинально скорочує час опитування пристроїв і дозволяє моніторити мережу в реальному часі без суттєвих затримок.

Вибір саме моделі потоків (threading), а не процесів (multiprocessing) або асинхронного вводу-виводу (asyncio) для даної задачі вимагає детальнішого технічного обґрунтування. У мові Python існує механізм Global Interpreter Lock (GIL), який обмежує виконання байт-коду одним потоком одночасно. Це робить використання потоків неефективним для задач, що вимагають інтенсивних обчислень CPU (CPU-bound). Однак, задача мережевого сканування відноситься до класу I/O-bound задач (обмежених швидкістю вводу-виводу). Більшу частину часу виконання програма проводить в очікуванні відповіді від мережевого

інтерфейсу (стан WAITING). У цей час GIL звільняється, дозволяючи іншим потокам виконувати свої запити. Тому ThreadPoolExecutor є ідеальним рішенням, яке забезпечує високий рівень паралелізму мережевих запитів без надмірних витрат пам'яті, характерних для створення окремих процесів. Використання ж asyncio вимагало б повної переробки архітектури на асинхронну, що ускладнило б інтеграцію з бібліотекою Scapy, яка переважно працює у синхронному режимі. Комбінація синхронних викликів Scapy та пулу потоків для сокетів є оптимальним компромісом між складністю розробки та продуктивністю для настільного застосунку.

Для побудови графічного інтерфейсу користувача (GUI) було обрано бібліотеку CustomTkinter [13]. Вона є сучасною надбудовою над стандартною бібліотекою Tkinter, що входить до складу Python. CustomTkinter дозволяє створювати візуально привабливі, адаптивні інтерфейси з підтримкою тем оформлення, що значно покращує досвід користувача у порівнянні зі застарілим виглядом Tkinter.

Для візуалізації даних було обрано бібліотеку Matplotlib [14]. Це де-факто стандарт для наукової візуалізації та побудови графіків у Python. Її ключовою перевагою є можливість легкого вбудовування полотен з графіками безпосередньо у вікна, створені CustomTkinter, що дозволяє реалізувати відображення історії затримок пінгу в реальному часі. Для збереження даних обрано формат JSON, оскільки він є легким, текстовим, читабельним та нативно підтримується у Python.

2.5 Структура зберігання даних моніторингу

Для коректної роботи описаної методології, зокрема методу "еталонів", програмний засіб повинен мати здатність зберігати зібрані дані між сеансами роботи. Ця вимога є критичною, оскільки фаза "навчання" (створення "еталону") має виконуватися лише один раз для кожного пристрою. Використання енергонезалежної пам'яті дозволяє програмі "запам'ятовувати" пристрої та їхній

нормальний стан.

Для реалізації цього було обрано файлову систему та формат JSON (JavaScript Object Notation) [15]. Цей вибір обґрунтований тим, що JSON є текстовим, людино-читабельним форматом, який легко парситься стандартними засобами Python і не вимагає розгортання повноцінної бази даних, що відповідає вимозі легкості та портативності засобу. Структура даних розділена на два окремі файли, що відповідають за різні аспекти моніторингу: devices.json та fingerprints.json.

Перший файл, devices.json, відповідає за "людський" рівень ідентифікації пристроїв. Оскільки IP-адреси можуть змінюватися, а MAC-адреси є унікальними, але не інформативними, цей файл дозволяє користувачеві призначити кожному пристрою змістовний псевдонім. Структурно, ключем у цьому файлі є MAC-адреса пристрою, наприклад 3c:ab:8e:3a:bf:2d. Значенням для цього ключа є рядок, введений користувачем, наприклад "Моя IP-камера". Призначення цього файлу — дозволити програмі відображати в інтерфейсі зрозумілі назви замість технічних ідентифікаторів, що значно покращує юзабіліті.

Другий файл, fingerprints.json, є "мозком" розробленого методу. Він зберігає технічну інформацію, отриману під час фази "навчання", і є основою для L4-аналізу відхилень. Структура цього файлу аналогічна: ключем також виступає MAC-адреса пристрою, що дозволяє логічно пов'язати дані з devices.json. Однак значенням тут є список, або масив, TCP-портів, які були визначені як "нормально відкриті" для цього пристрою, наприклад «[80, 554]» для роутера або порожній список для пристрою без сервісів. Призначення цього файлу — слугувати базою даних "еталонних" станів, з якою порівнюється кожен "поточний знімок" стану пристрою під час моніторингу.

Важливо зазначити, що ця структура "еталонів" є гнучкою. У випадку легітимної зміни конфігурації пристрою, наприклад, після оновлення прошивки, яке додає новий сервіс або видаляє старий, "еталон" може застаріти. Методологія повинна передбачати цей сценарій. Для цього користувач повинен мати

можливість вручну видалити запис про пристрій з файлу fingerprints.json. Це змусить програмний засіб при наступному циклі сканування ініціювати фазу "навчання" повторно, тим самим створюючи новий, актуальний "еталон" для оновленого пристрою. Такий підхід поєднує автоматизацію моніторингу з можливістю ручного втручання та "перенавчання" системи.

Така двофайлова структура дозволяє чітко розділити дані користувача від машинних даних, забезпечуючи необхідну гнучкість та надійність для реалізації методу моніторингу.

3 РОЗРОБКА МЕТОДУ ТА ЗАСОБІВ МОНІТОРИНГУ IoT-ПРИСТРОЇВ

3.1 Розробка методу моніторингу на основі "еталонів" сервісів

Як було встановлено у попередніх розділах, для ефективного моніторингу IoT-пристроїв недостатньо лише перевіряти їхню мережеву доступність L3. Необхідний метод, що здатен аналізувати стан пристрою на транспортному рівні L4, не покладаючись на агентів чи SNMP. Для вирішення цієї задачі пропонується метод моніторингу на основі "еталонів" сервісів, який також відомий як "фінгерпринтинг" або "створення базової лінії".

Суть методу полягає у відмові від статичних правил "перевірити порт 80". Замість цього, система моніторингу є адаптивною: вона спочатку вивчає нормальний стан для кожного конкретного пристрою, а потім безперервно порівнює поточний стан з цим "еталоном", фіксуючи будь-які відхилення. Цей процес можна розділити на три логічні фази: фаза навчання, фаза моніторингу та фаза аналізу.

Перша фаза це навчання та створення "Еталону" (Fingerprint). Коли програмний засіб виявляє в мережі новий пристрій, який відсутній у його базі даних fingerprints.json, він автоматично ініціює фазу навчання. Під час цієї фази, засіб проводить поглиблене, але одноразове, сканування пристрою. Це сканування включає перевірку L3 та повне сканування L4 по списку поширених сервісних портів, таких як 80, 443, 22, 23, 554 тощо. Результатом цього сканування є список відкритих портів, який і стає "еталоном" або "фінгерпринтом" для даного пристрою. Цей еталон зберігається у енергонезалежній пам'яті і асоціюється з унікальним ідентифікатором пристрою, його MAC-адресою.

Друга фаза це періодичний моніторинг. Після створення "еталону", система переходить у штатний режим моніторингу. З заданою періодичністю, наприклад, кожні 15 секунд, система проводить швидке опитування усіх пристроїв, для яких вже існує "еталон". Це опитування складається з двох кроків:

спочатку перевірка доступності L3, і, тільки якщо вона успішна, проводиться сканування L4 для отримання поточного знімку відкритих портів. Важливо, що це сканування L4 є оптимізованим: воно перевіряє не всі можливі порти, а лише ті, що присутні в "еталоні" пристрою, та ті, що входять до списку поширених небезпечних портів.

Третя фаза це аналіз відхилень та класифікація відмов. Це ядро методу, де відбувається порівняння "поточного знімку" з "еталоном". Саме тут і відбувається виявлення відмов, класифікованих у підрозділі 2.1. Якщо ICMP-запит не проходить, система негайно фіксує мережеву відмову, не переходячи до аналізу L4. Якщо ж ICMP-запит успішний, система переходить до аналізу L4:

— сервісна відмова фіксується, якщо порт, що присутній в "еталоні", відсутній у "поточному знімку", що свідчить про те, що очікуваний сервіс "впав";

— безпекова аномалія фіксується, якщо у "поточному знімку" з'являється порт, якого не було в "еталоні", що свідчить про несанкціоновану зміну конфігурації або компрометацію;

— нормальний стан фіксується, якщо "поточний знімок" повністю відповідає "еталону" або еталон порожній і нових загроз не з'явилося.

Таким чином, цей метод дозволяє перейти від простої констатації факту L3-доступності до повноцінної, автоматизованої та адаптивної L4-діагностики, що є критично важливим для моніторингу гетерогенних IoT-мереж.

Варто зазначити, що алгоритм формування еталону передбачає певний рівень фільтрації для підвищення достовірності даних. Під час сканування пристрій може мати відкриті динамічні порти (ephemeral ports), які використовуються операційною системою для вихідних з'єднань (зазвичай діапазон від 49152 до 65535). Включення таких портів до «еталону» було б помилкою, оскільки при наступному перезавантаженні вони зміняться, що призведе до хибного спрацювання системи виявлення відмов. Тому розроблений метод обмежує зону формування еталону діапазоном загальновідомих та зареєстрованих портів, де зазвичай розміщуються стабільні сервіси IoT-

пристроїв (HTTP, RTSP, MQTT, Telnet). Лише якщо порт підтвердив свою доступність двічі, він заноситься до файлу

3.2 Структурна та функціональна схема програмних засобів

Для забезпечення високої відгукливості інтерфейсу та одночасного виконання ресурсоємних мережевих операцій, архітектура програмного комплексу побудована на багатопоточній моделі. Вся логіка реалізована в межах єдиного класу App, що наслідує ctk.CTk та виступає як центральний контролер. Архітектура чітко розділяє логіку на головний та фоновий потоки [16].

Головний потік (Main Thread) відповідає виключно за роботу графічного інтерфейсу користувача (GUI), побудованого на CustomTkinter. Він обробляє всі дії користувача, такі як натискання кнопок, вибір елементів у таблиці ttk.Treeview та перемикання між екранами list_frame та detail_frame.

Фоновий потік (Worker Thread) виконує всю важку роботу з моніторингу. Він ініціюється при натисканні кнопки старту та виконує всю логіку L3/L4 сканування, не блокуючи інтерфейс.

Структурна схема програмного комплексу, наведена на рисунку 3.1, ілюструє цю модульну побудову.

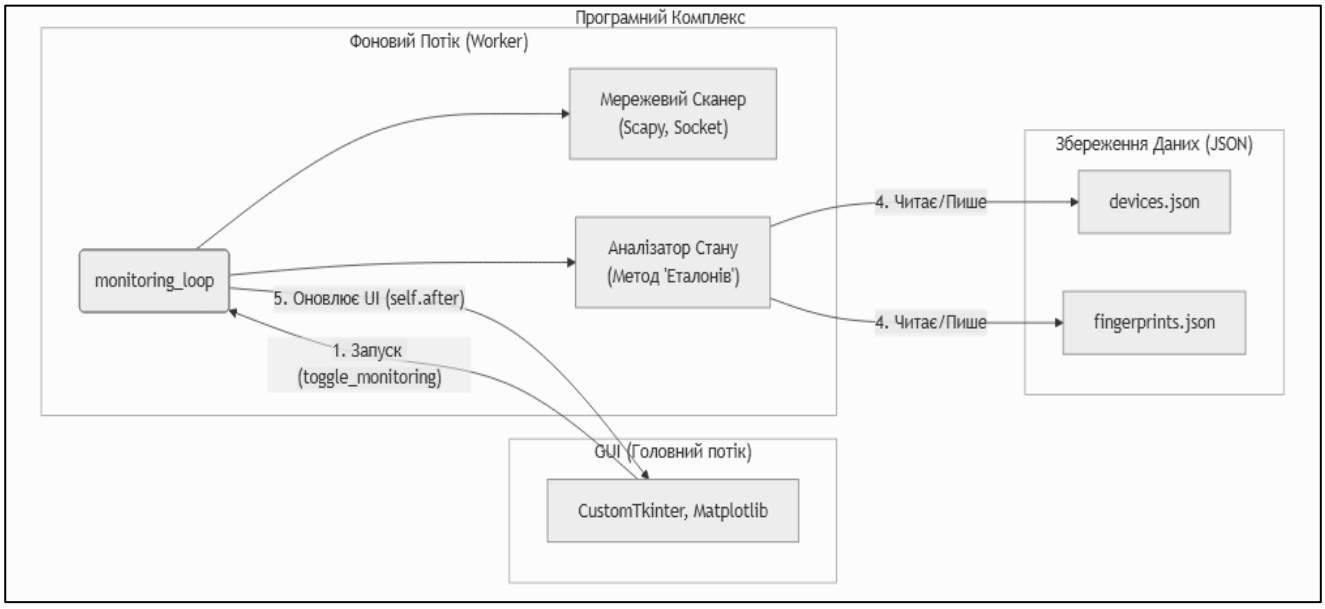

Рисунок 3.1 — Структурна схема програмного комплексу

Запуск фонового потоку реалізований у методі toggle_monitoring, який викликається при натисканні на кнопку запуску. Ключовим є створення нового об'єкта threading.Thread, як показано у лістингу 3.1.

Лістинг 3.1 — Запуск фонового потоку моніторингу

```
def toggle_monitoring(self):
if self.monitoring_active:
print("Зупиняємо моніторинг...")
self.monitoring_active = False; self.stop_event.set()
else:
print("Запускаємо моніторинг...")
self.monitoring_active = True; self.stop_event.clear()
self.toggle_button.configure(image=self.stop_icon)
self.tooltip.configure(message="Вимкнути моніторинг")
thread = threading.Thread(target=self.monitoring_loop, daemon=True)
thread.start()
```

Як видно з лістингу, цільовою функцією для потоку (target) є метод monitoring_loop. Це "серце" програми, що містить нескінченний цикл, який працює, доки активний прапор self.monitoring_active.

Функціональна схема, показана в Додатку В на рисунку В.1, описує потік даних та взаємодію між цими потоками.

Ключовим елементом архітектури є безпечна передача даних з фонового потоку назад до головного потоку GUI. Оскільки напряму змінювати елементи CustomTkinter з іншого потоку заборонено, програма використовує вбудований у Tkinter механізм черги подій. Виклик self.after(0, self.update_ui_with_logs, ...) безпечно "ставить у чергу" завдання update_ui_with_logs для виконання його головним потоком.

Метод update_ui_with_logs приймає оброблені дані та вже безпечно оновлює елементи GUI, такі як журнал та таблиця пристроїв (ліст. 3.2).

Лістинг 3.2 — Метод оновлення GUI з головного потоку

```
def update_ui_with_logs(self, log_messages):
if log_messages:
self.add_log_message("Оновлення даних...", "info")
```

```
for msg, tag in log_messages:
self.add_log_message(msg, tag)
active_devices = self.get_active_devices()
self.update_results_ui(active_devices)
```

Такий підхід гарантує високу відгукливість інтерфейсу, оскільки головний потік ніколи не блокується мережевими операціями, та водночас забезпечує стабільність, оскільки всі оновлення GUI відбуваються виключно у головному потоці.

3.3 Розробка модуля мережевого сканування

Цей модуль реалізований у вигляді набору методів всередині основного класу App, які викликаються послідовно з головного циклу моніторингу monitoring_loop. Завдання цього модуля — надати відповіді на три ключові питання:

— які пристрої присутні в мережі (L2/L3 виявлення);

— яка якість зв'язку з ними (L3 аналіз);

— які сервіси на них активні (L4 сканування).

Модуль мережевого сканування є фундаментом для всієї системи, оскільки він відповідає за збір "сирих" даних з реального мережевого середовища. Від його ефективності, швидкості та надійності залежить точність усіх подальших аналітичних висновків. Як було обґрунтовано в розділі 2.4, для реалізації цього модуля було обрано комбінацію технологій: Scapy для низькорівневих L2/L3 операцій та Socket API для L4-сканування.

3.3.1 Алгоритм виявлення пристроїв у мережі через ARP-сканування

Першим та фундаментальним завданням модуля сканування є отримання актуального списку пристроїв, які наразі підключені до локальної мережі. Використання ICMP (Ping) для цієї задачі є ненадійним, оскільки багато пристроїв з міркувань безпеки налаштовані ігнорувати ICMP-запити.

Тому в розробленій системі було обрано значно надійніший метод —

сканування на рівні L2 за допомогою протоколу ARP (Address Resolution Protocol) [17]. ARP є обов'язковим протоколом для функціонування будь-якої локальної Ethernet/Wi-Fi мережі, оскільки він відповідає за перетворення логічних IP-адрес (L3) у фізичні MAC-адреси (L2). Жоден пристрій не може приховати свою присутність від ARP-запиту, не втративши при цьому зв'язок з мережею. Алгоритм виявлення складається з двох етапів: визначення цільової мережі та безпосередньо ARP-сканування.

Програмний засіб повинен бути портативним і не вимагати від користувача ручного введення адреси його мережі. Для цього було реалізовано метод get_active_network_info, що автоматично визначає параметри поточної активної мережі. Цей метод використовує "трюк" зі стандартною бібліотекою socket для визначення IP-адреси, яку операційна система використовує для виходу в Інтернет, як показано у лістингу 3.3.

Лістинг 3.3 — Метод автоматичного визначення IP-адреси та мережі

```
def get_active_network_info(self):
try: s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.settimeout(0.1)
s.connect(("8.8.8.8", 80))
ip_addr = s.getsockname()[0]
s.close()
all_ifaces = psutil.net_if_addrs()
for iface_name, addrs in all_ifaces.items():
for addr in addrs:
if addr.family == socket.AF_INET and addr.address == ip_addr:
netmask = addr.netmask
network = ipaddress.ip_network(f"{ip_addr}/{netmask}", strict=False)
target_ip = str(network)
print(f"Автоматично визначено мережу: {target_ip}...")
return target_ip, iface_name
raise Exception(f"Не вдалося знайти інтерфейс для IP {ip_addr}")
except Exception as e:
print(f"Не вдалося автоматично визначити мережу...: {e}")
return "192.168.1.0/24", None
```

Метод connect до публічної IP-адреси 8.8.8.8 змушує ОС обрати найкращий локальний інтерфейс для вихідного трафіку. Метод getsockname

отримує IP-адресу цього інтерфейсу. Далі, за допомогою psutil, програма знаходить цей інтерфейс у системі, щоб отримати його маску підмережі (netmask). На основі IP та маски, бібліотека ipaddress обчислює адресу всієї підмережі, наприклад 192.168.1.0/24.

Отримавши цільову мережу (наприклад, 192.168.1.0/24), метод scan_network_arp формує та надсилає спеціальний ARP-пакет за допомогою Scapy, як показано у лістингу 3.4.

Лістинг 3.4 — Відправка ARP-запиту за допомогою Scapy

```
def scan_network_arp(self):
target_ip, iface_name = self.get_active_network_info()
arp_request = sc.ARP(pdst=target_ip)
broadcast = sc.Ether(dst="ff:ff:ff:ff:ff:ff")
packet = broadcast/arp_request
devices_online = []
try:
sc.conf.iface = iface_name
sc.conf.route.resync()
result = sc.srp(packet, timeout=2, verbose=0)[0]
except Exception as e:
print(f"Помилка сканування ARP: {e}")
return devices_online
```

Пакет sc.ARP з pdst=target_ip фактично надсилає запит "who-has" для кожної IP-адреси у вказаному діапазоні. Цей запит загортається у кадр sc.Ether з MAC-адресою призначення ff:ff:ff:ff:ff:ff — це широкомовний запит (broadcast), який отримають абсолютно всі пристрої в локальному сегменті мережі.

Механізм широкомовного запиту (broadcast) на рівні L2 має свої особливості, які необхідно враховувати при розробці сканера. Коли комутатор отримує кадр з адресою призначення ff:ff:ff:ff:ff:ff, він ретранслює його на всі свої порти, окрім того, з якого прийшов кадр. Це гарантує, що запит дійде до кожного пристрою в сегменті мережі, навіть якщо вони знаходяться за іншими некерованими комутаторами або точками доступу Wi-Fi. Важливою деталлю реалізації є фільтрація власних пакетів. У середовищі Windows з драйвером Npcap мережевий інтерфейс може захоплювати також і вихідні пакети, що

відправляє сама програма. Без належної фільтрації це може призвести до появи дублікатів у списку пристроїв або хибної ідентифікації власного хоста як зовнішнього пристрою. Бібліотека Scapy надає потужні засоби фільтрації на рівні BPF (Berkeley Packet Filter), але в рамках розробленого методу фільтрація реалізована на логічному рівні шляхом порівняння MAC-адреси відправника у отриманій відповіді (ARP Reply) з власною MAC-адресою інтерфейсу. Такий підхід дозволяє уникнути обробки «луна-пакетів» та забезпечує чистоту даних у списку активних хостів.

Функція sc.srp (Send-Receive Packets) надсилає ці пакети та збирає відповіді. Кожен "живий" пристрій надсилає у відповідь ARP-reply, повідомляючи свою IP та MAC-адресу. Програма ітерує по списку result і витягує з відповідей received.psrc (IP-адреса) та received.hwsrc (MAC-адреса), формуючи таким чином первинний список devices_online. Цей список є основою для подальших, більш глибоких перевірок L3 та L4.

3.3.2 Алгоритм перевірки якості зв'язку через ICMP-пінг

Після того, як на етапі ARP-сканування було отримано список пристроїв, присутніх у мережі (у вигляді пар IP-MAC), наступним кроком є перевірка якості зв'язку з кожним із них. Як було зазначено в другому розділі, відповідь на ARP-запит гарантує лише те, що пристрій існує на рівні L2, але нічого не каже про якість його мережевого з'єднання на рівні L3.

Для вирішення цієї задачі було реалізовано метод check_connection, який виконує класичний ICMP-пінг. Важливо, що цей метод викликається всередині циклу scan_network_arp негайно після отримання ARP-відповіді від пристрою. Це дозволяє в одному проході отримати повну L3-картину: виявити пристрій і одразу ж оцінити його стан. Метод check_connection також використовує Scapy, але вже для операцій на рівні L3. Його реалізація показана в лістингу 3.5.

Лістинг 3.5 — Метод перевірки якості зв'язку L3

```
def check_connection(self, ip_address, num_packets=4, timeout=0.5):
```

```
try:
replies, no_replies = sc.sr(
sc.IP(dst=ip_address)/sc.ICMP(),
timeout=timeout,
retry=num_packets-1,
verbose=0)
packet_loss = len(no_replies) / (len(replies) + len(no_replies)) * 100
if not replies:
return {"latency_ms": None, "packet_loss_percent": 100}
total_latency = sum((rec.time - sent.time) for sent, rec in replies) * 1000
avg_latency = total_latency / len(replies)
return {"latency_ms": round(avg_latency), "packet_loss_percent":
round(packet_loss)}
except Exception as e:
print(f"Could not ping {ip_address}: {e}")
return {"latency_ms": None, "packet_loss_percent": 100}
```

На відміну від sc.srp, який працює на L2, тут використовується функція sc.sr (Send-Receive), яка надсилає пакети на рівні L3. Формується пакет sc.IP з цільовою IP-адресою, що містить у собі sc.ICMP (Echo-Request). Параметри num_packets та timeout дозволяють контролювати агресивність сканування.

Функція sc.sr повертає два списки: replies (пакети, на які прийшла відповідь) та no_replies (пакети, що були втрачені). На основі цих двох списків алгоритм обчислює кількісні показники якості каналу.Коефіцієнт втрати пакетів (P_{loss}) розраховується як відношення кількості втрачених пакетів до загальної кількості відправлених ICMP-запитів за формулою:

$$P_{loss} = \left(1 - \frac{N_{Tx}}{N_{tx}}\right) \times 100\%, \quad (3.1)$$

де N_{Tx} — кількість успішно отриманих відповідей;

N_{tx} — загальна кількість відправлених пакетів.

Середній час затримки (T_{avg}) обчислюється як різниця в часі між відправкою (sent.time) та отриманням (rec.time) для кожного пакету, що сумується та ділиться на кількість отриманих відповідей, згідно з формулою:

$$T_{avg} = \frac{1}{N_{Tx}} \sum_{i=1}^{N_{Tx}} (t_{Tx,i} - t_{tx,i}), \quad (3.2)$$

де $t_{Tx,i}$ — час отримання відповіді на i-й пакет;

$t_{tx,i}$ — час відправлення i-го пакету.

Ці два показники повертаються у вигляді словника до методу scan_network_arp, де відбувається фінальна класифікація L3-стану. Якщо пристрій не відповідає на пінг (100% втрата), він ігнорується і не потрапляє у фінальний список devices_online. Для всіх інших пристроїв, що відповіли, встановлюються пороги: втрата понад 25% вважається "Поганим" зв'язком, а затримка понад 150 мс — "Середнім". Це дозволяє не просто виявити пристрої, але й одразу оцінити якість їхнього підключення до мережі.

Окрім середнього часу затримки, важливим показником стабільності каналу, особливо для бездротових IoT-пристроїв, є джитер (jitter) — варіація часу затримки. Хоча в базовій версії інтерфейсу відображається лише середнє значення T_{avg}, архітектура модуля сканування дозволяє збирати статистику розкиду значень RTT. Високий джитер (наприклад, коли відповіді приходять то за 2 мс, то за 100 мс) часто є передвісником повної відмови пристрою або свідчить про сильну зашумленість радіоефіру Wi-Fi.

В рамках розробленого алгоритму, якщо різниця між мінімальним та максимальним часом відповіді у серії з 4-х пакетів перевищує певний поріг (наприклад, 50 мс), це враховується при формуванні загальної оцінки якості зв'язку. Це дозволяє більш тонко діагностувати проблеми: стабільно висока затримка може свідчити про віддаленість пристрою від точки доступу, тоді як високий джитер при низькій середній затримці вказує на колізії в мережі або перевантаження процесора самого IoT-пристрою, який не встигає вчасно обробляти переривання від мережевого контролера.

3.3.3 Алгоритм паралельного сканування сервісів через TCP-сканування

Після завершення L3-перевірок, модуль сканування переходить до найважливішого етапу, що лежить в основі розробленого методу — сканування сервісів L4. На відміну від ARP та ICMP, які використовували Scapy, для цього завдання було обрано стандартний модуль socket. Це рішення обґрунтовано тим, що socket API для TCP Connect сканування є більш стабільним, надійним, і, що найважливіше, не вимагає прав адміністратора для своєї роботи, на відміну від низькорівневих TCP SYN пакетів Scapy. Це робить програмний засіб значно простішим у використанні.

Для реалізації цієї функції було розроблено метод scan_ports_socket, показаний у лістингу 3.6. Оскільки послідовна перевірка навіть невеликої кількості портів може тривати на протязі довгого часу (через тайм-аути на закритих портах), алгоритм використовує ThreadPoolExecutor для запуску перевірок паралельно.

Лістинг 3.6 — Метод паралельного сканування L4-портів

```
def scan_ports_socket(self, ip, ports, timeout=0.5):
open_ports = []
def check_port(ip, port, timeout):
try: with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
s.settimeout(timeout)
if s.connect_ex((ip, port)) == 0: return port
except (socket.timeout, socket.error):
return None
with ThreadPoolExecutor(max_workers=20) as executor:
futures = [executor.submit(check_port, ip, port, timeout)
for port in ports]
for future in futures: result = future.result()
if result is not None: open_ports.append(result)
print(f"Socket Scan {ip} for {ports}. Open: {open_ports}")
return open_ports
```

Алгоритм працює наступним чином:

— метод scan_ports_socket отримує IP-адресу та список портів для перевірки;

— він визначає внутрішню функцію check_port, яка містить логіку для перевірки одного порту, та використовує socket.connect_ex, яка повертає 0 у разі успішного TCP-з'єднання (порт відкритий) або код помилки в іншому випадку;

— створюється ThreadPoolExecutor з лімітом у 20 одночасних "працівників";

— за допомогою спискового включення, кожен порт зі списку ports передається як окреме завдання (executor.submit) у пул потоків та перевірки запускаються практично одночасно;

— програма очікує завершення всіх завдань (future.result()) і збирає ті порти, що повернули не None, у фінальний список open_ports.

Сканування L4 запускається лише для пристроїв зі статусом status_l3 == "Добрий". Список портів для перевірки (ports_to_check) є динамічним: він складається з об'єднання "еталонних" портів цього пристрою та загального списку COMMON_PORTS. Це гарантує, що програма перевірить як очікувані сервіси, так і поширені небезпечні порти для виявлення загроз.

Однією з головних проблем при розробці мережевих сканерів мовою Python є блокуючий характер операцій введення-виведення (I/O). Стандартний виклик socket.connect() при спробі підключення до закритого порту або неіснуючого хоста може очікувати відповіді протягом стандартного тайм-ауту операційної системи, який часто складає від 20 до 60 секунд.

Для системи моніторингу реального часу такі затримки є неприпустимими. Якщо пристрій має перевірити 10 портів, послідовне сканування у найгіршому випадку (всі порти закриті брандмауером) зайняло б:

$$T_{seq} = N_{ports} \times t_{default} \approx 10 \times 20 = 200c. \quad (3.3)$$

Така затримка повністю блокує цикл моніторингу. Для вирішення цієї проблеми у розробленому програмному засобі застосовано два методи оптимізації: жорстке обмеження тайм-ауту та розпаралелювання запитів.

У локальних мережах (LAN) час проходження сигналу (RTT — Round Trip Time) зазвичай не перевищує 1–10 мс. Тому очікування відповіді протягом десятків секунд є надлишковим.Експериментально було встановлено оптимальне значення тайм-ауту τ = 0,5с (500 мс).

$$\tau_{scan} = 500ms \gg RTT_{LAN}. \quad (3.4)$$

Це значення забезпечує достатній запас надійності навіть для повільних Wi-Fi з'єднань, але при цьому гарантує, що сканування одного "мертвого" порту не затримає систему більше ніж на півсекунди. У коді це реалізовано викликом s.settimeout(timeout).

Для прискорення перевірки множини портів використано пул потоків (ThreadPoolExecutor). Кількість робочих потоків (W) у коді встановлено на рівні W = 20. Вибір цього числа обумовлений обмеженнями IoT-пристроїв. Хоча сучасний ПК може підтримувати сотні потоків, мережеві інтерфейси дешевих IoT-пристроїв (мікроконтролери ESP32, дешеві IP-камери) мають обмежений буфер вхідних з'єднань.Агресивне сканування (наприклад, 100+ потоків одночасно) може призвести до відмови в обслуговуванні (DoS) самого пристрою, коли він перестане відповідати навіть на легітимні запити. Значення W = 20 є компромісом, що забезпечує швидкість без перевантаження цілі.

Теоретичний час сканування групи портів при паралельному підході розраховується як:

$$T_{par} \approx \left\lceil \frac{N_{ports}}{W} \right\rceil \times \tau. \quad (3.5)$$

Для типового випадку перевірки списку поширених портів (наприклад, 14 портів зі списку COMMON_PORTS), при використанні 20 потоків, всі запити виконуються за одну ітерацію.

$$T_{par} \approx \left\lceil \frac{14}{20} \right\rceil \times 0,5 = 1 \times 0,5 = 0,5c. \quad (3.6)$$

Порівнюючи з послідовним методом (де час склав би 7с навіть зі зменшеним тайм-аутом), ми отримуємо прискорення у 14 разів. Таким чином, комбінація зменшеного тайм-ауту та обмеженої багатопоточності дозволяє проводити повний цикл опитування пристрою менше ніж за 1 секунду, що задовольняє вимогам моніторингу в реальному часі.

3.4 Розробка модуля аналізу та виявлення відмов

Якщо модуль сканування є "очима" та "руками" системи, то модуль аналізу є її "мозком". Саме тут "сирі" дані, отримані з мережі (списки IP-адрес та відкритих портів), перетворюються на осмислену діагностичну інформацію. Цей модуль безпосередньо реалізує метод "еталонів", описаний у Розділі 2, і є центральною частиною всього програмного комплексу.

Його логіка реалізована всередині головного циклу monitoring_loop. Це дозволяє уникнути передачі великих обсягів даних між методами та прискорює обробку. Роботу модуля можна розділити на три ключові етапи: обробка нових пристроїв, аналіз L4-відхилень та обробка L3-відмов.

3.4.1 Алгоритм фази "Навчання" та створення "еталону"

Фаза "Навчання", коли monitoring_loop отримує список онлайн-пристроїв, він ітерує по ньому і перевіряє, чи відомий йому пристрій. Якщо MAC-адреса пристрою відсутня у словнику self.device_fingerprints, програма ініціює фазу "навчання", як показано в лістингу 3.7.

Лістинг 3.7 — Ініціалізація "Навчання" для нових пристроїв

```
if mac not in self.device_fingerprints:
self.device_fingerprints[mac] = []
log_messages.append( (f'L4: Пристрій '{display_name}' ... не має
'еталону'.", "info") )
threading.Thread(target=self.learn_device_fingerprint, args=(ip, mac),
daemon=True).start()
device_data["status_l4"] = "Навчання..."
```

Як видно з коду, створюється "заглушка" в self.device_fingerprints для запобігання повторному запуску "навчання" в наступних циклах. Сама ресурсоємна операція learn_device_fingerprint запускається у власному окремому потоці. Це критично важливо, щоб процес "навчання" одного пристрою не блокував та не уповільнював моніторинг усіх інших пристроїв у мережі.

3.4.2 Алгоритм циклу моніторингу та порівняння з "еталоном"

Наступним етапом є аналіз L4-відхилень, що є ядром методу, яке виконується для пристроїв, що мають статус L3 "Добрий". Цей процес базується на операціях над множинами портів.

Формально, нехай S_{etalon} — множина портів, що містяться в збереженому "еталоні" пристрою, а $S_{current}$ — множина портів, виявлених під час поточного сканування. Виявлення "Безпекової аномалії" полягає у пошуку портів, які з'явилися в поточному стані, але відсутні в еталоні. Ця множина аномалій T визначається як різниця множин, перетнута зі списком небезпечних портів:

$$T = (S_{current} \setminus S_{etalon}) \cap S_{COMMON}. \quad (3.7)$$

Виявлення "Сервісної відмови" полягає у пошуку портів, які є в еталоні, але відсутні в поточному стані. Множина відмов F визначається як:

$$F = S_{etalon} \setminus S_{current}. \quad (3.8)$$

Код, наведений у лістингу 3.8, реалізує логіку порівняння "еталону" та "поточного знімку".

Лістинг 3.8 — Алгоритм аналізу відхилень L4

```
if device_data["status_l3"] == "Добрий":
ports_to_check = list(set(etalon_ports + COMMON_PORTS))
current_open_ports = self.scan_ports_socket(ip, ports_to_check)
```

```
new_dangerous_ports = [p for p in current_open_ports if p not in etalon_ports
and p in COMMON_PORTS]
closed_etalon_ports = []
if etalon_ports:
closed_etalon_ports = [p for p in etalon_ports if p not in current_open_ports]
if closed_etalon_ports:
device_data["status_l4"] = "Відмова"
device_data["status_overall"] = "Відмова L4"
device_data["anomaly"] = f"{closed_etalon_ports}"
elif new_dangerous_ports:
device_data["status_l4"] = "Загроза"
device_data["status_overall"] = "Загроза L4"
device_data["anomaly"] = f"{new_dangerous_ports}"
elif not etalon_ports:
device_data["status_l4"] = "Чистий"
else: device_data["status_l4"] = "OK"
```

Особливу увагу в алгоритмі аналізу приділено обробці граничних випадків, пов'язаних з динамічною природою деяких мережевих служб. Існують сценарії, коли пристрої використовують технологію UPnP (Universal Plug and Play) для динамічного відкриття портів на вимогу. У класичних системах моніторингу це призводило б до постійних сповіщень про «аномалії». У розробленому методі ця проблема частково вирішується шляхом використання списку COMMON_PORTS як фільтру для визначення загроз. Алгоритм реагує не на появу будь-якого нового порту, а саме на появу порту з переліку відомих вразливих сервісів. Це суттєво зменшує кількість хибних спрацювань («false positives»). Наприклад, якщо медіа-сервер динамічно відкриє високий порт 34567 для передачі даних, система це проігнорує, оскільки цей порт не є типовим вектором атаки. Однак, якщо раптово відкриється порт 23 (Telnet), система миттєво класифікує це як загрозу, незалежно від того, чи використовувався UPnP. Такий гібридний підхід (білий список «еталону» + чорний список «загроз») дозволяє зберегти баланс між безпекою та зручністю експлуатації в домашніх умовах, де користувачі рідко налаштовують правила вручну.

Цей блок коду чітко реалізує вимоги, описані в розділі 2.3. Він спочатку шукає Безпекову аномалію, порівнюючи поточні порти з тими, яких немає в

еталоні. Потім він шукає Сервісну відмову, порівнюючи еталонні порти з тими, яких немає в поточному знімку. На основі цих двох перевірок він присвоює фінальний статус L4.

Наступним етапом є обробка L3-відмов, що є фінальним етапом аналізу та відбувається після перевірки всіх онлайн пристроїв. Програма повинна також обробити ті пристрої, які були в мережі, але не відповіли на ARP/Ping. Ця логіка реалізована у окремому циклі, наведеному в лістингу 3.9.

Лістинг 3.9 — Обробка пристроїв, що перейшли в Offline

```
for mac, state in self.device_states.items():
if mac not in online_macs:
state["failure_count"] += 1
display_name = self.get_display_name(mac)
if state["failure_count"] == self.failure_threshold and not
state["failure_logged"]:
log_messages.append( (f"ВІДМОВА (L3): Пристрій '{display_name}' ... не
відповідає.", "error") )
state["failure_logged"] = True
state["device_data"]["status_l3"] = "Offline"
state["device_data"]["status_l4"] = "---"
state["device_data"]["status_overall"] = "Offline"
```

Тут використовується поріг відмов (failure_threshold = 3). Пристрій не позначається як "Offline" після першого ж пропущеного циклу, що захищає від хибних спрацьовувань через випадкову втрату пакета. Лише після трьох послідовних невдалих спроб пристрій надійно позначається як "Offline", і в журнал додається повідомлення про L3-відмову.

Логіка переходів між станами, детально ілюстрована на рисунку 3.3, є основою всього модуля аналізу.

Початковим станом для будь-якого пристрою при запуску програми є 'Offline'. При успішному L3-скануванні, модуль аналізу приймає рішення. Якщо для пристрою "еталон" ще не створено (тобто MAC-адреса відсутня у self.device_fingerprints), він переходить у стан 'Навчання...'. Якщо "еталон" вже існує і поточний L4-скан йому повністю відповідає, пристрій переходить у

головний робочий стан 'Online'.

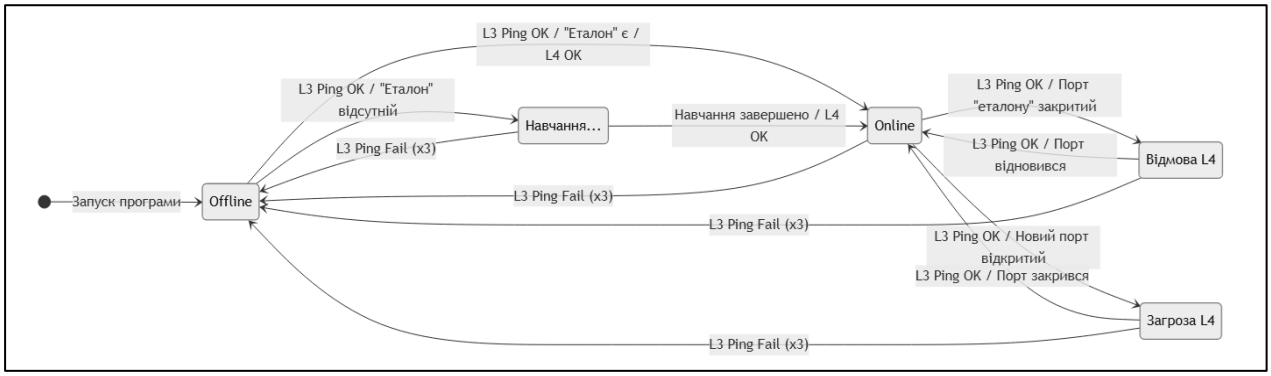

Рисунок 3.2 — Діаграма станів пристрою в модулі аналізу

Зі стану 'Online' можливі три шляхи відмови, що відповідають розробленій класифікації. По-перше, при зникненні L3-відповіді, що фіксується порогом failure_threshold, він повертається в 'Offline'. По-друге, при виявленні закритого "еталонного" порту, він переходить у стан 'Відмова L4'. По-третє, при виявленні нового небезпечного порту, він переходить у стан 'Загроза L4'.

Важливою частиною методу є також логіка відновлення. Як видно з діаграми, зі станів 'Відмова L4' або 'Загроза L4' пристрій може автоматично повернутися в стан 'Online', якщо наступний цикл сканування покаже, що аномалія зникла. Наприклад, порт, що не відповідав, відновив роботу, або небезпечний порт був закритий. Це дозволяє системі не лише фіксувати відмови, але й автоматично відстежувати їх усунення в реальному часі.

3.5 Розробка модуля інтерфейсу користувача

Модуль інтерфейсу користувача (GUI) є критично важливим компонентом програмного комплексу, оскільки він виступає як основний засіб взаємодії між користувачем та складною логікою моніторингу, що працює у фоні. Головне завдання цього модуля — візуалізувати результати дворівневого L3/L4 аналізу у зрозумілому, інтуїтивному та інформативному вигляді.

Як було обґрунтовано у розділі 2.4, для реалізації графічного інтерфейсу користувача (GUI) було обрано бібліотеку CustomTkinter. Ця бібліотека є

сучасною надбудовою над стандартним Tkinter і була обрана завдяки її здатності створювати візуально привабливі, сучасні інтерфейси з підтримкою тем (наприклад, "dark mode"), що значно покращує досвід користувача порівняно з застарілим виглядом Tkinter.

Архітектурно, інтерфейс розділений на два основні екрани (фрейми), які "підіймаються" (за допомогою tkraise()) в залежності від контексту:

— list_frame — головний екран, що містить таблицю всіх виявлених пристроїв та журнал подій;

— detail_frame — екран детальної інформації, що показує поглиблену діагностику, поради та графік для одного обраного пристрою.

Вся логіка графічного інтерфейсу користувача (GUI) реалізована в головному класі App і виконується виключно в головному потоці програми. Це забезпечує відгукливість інтерфейсу на дії користувача. Дані від фонового потоку моніторингу надходять асинхронно через механізм self.after, що запобігає будь-якому "зависанню" програми.

3.5.1 Візуалізація списку пристроїв та їх багаторівневого статусу

Головний екран програми (list_frame) містить два ключові елементи: журнал подій (log_textbox) та основну таблицю пристроїв. Для реалізації таблиці було обрано віджет ttk.Treeview. Цей вибір обґрунтований тим, що Treeview дозволяє налаштовувати та відображати дані, що є абсолютною вимогою для відображення багаторівневого статусу L3 та L4, а також IP/MAC адрес з назвою пристроїв та їх виробником (рис. 3.4).

Під час ініціалізації програми (в методі __init__) створюється екземпляр Treeview, для якого чітко визначаються ідентифікатори колонок (column_ids), їхні заголовки (heading) та параметри ширини, як показано в лістингу 3.10.

Оскільки CustomTkinter не стилізує віджети ttk автоматично, для досягнення темного дизайну (dark mode) було використано об'єкт ttk.Style для ручного налаштування кольорів фону, тексту та заголовків.

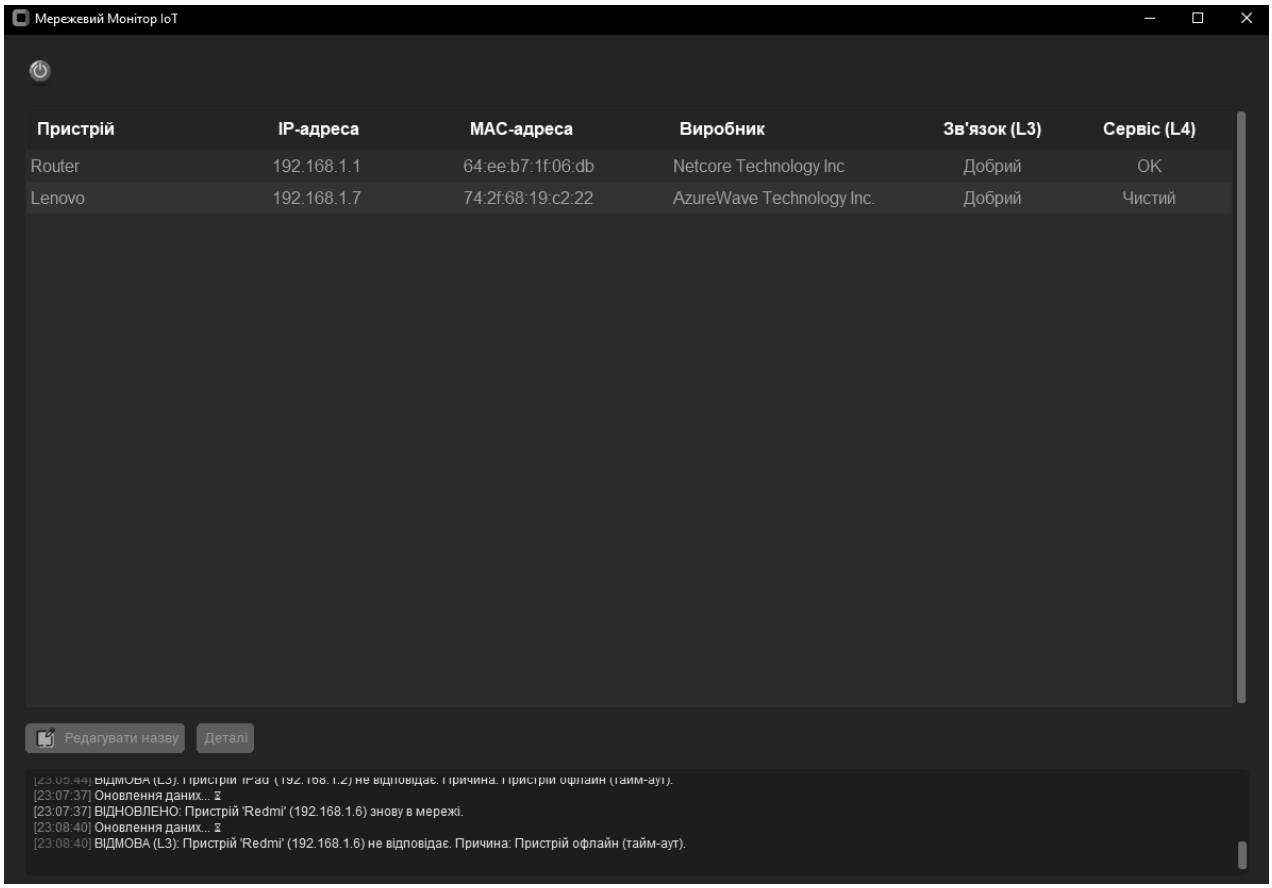

Рисунок 3.3 — Головний екран програми з таблицею пристроїв

Для кожного пристрою спершу визначається його загальний статус status_overall, який має пріоритет (наприклад, L4-відмова "перебиває" L3-статус "Добрий"). На основі цього загального статусу обирається відповідний тег кольору (status_tag). При вставці рядка (self.tree.insert) цей тег застосовується до всього рядка, миттєво забарвлюючи його у відповідний колір (зелений, червоний, блакитний тощо) і забезпечуючи чітку візуальну діагностику. Також використовується допоміжна функція truncate_text для обрізки занадто довгих назв пристроїв, що запобігає "розтягуванню" колонок.

Лістинг 3.10 — Ініціалізація віджета ttk.Treeview

```
column_ids = ("device", "ip", "mac", "mfr", "status_l3", "status_l4")
self.tree = ttk.Treeview(self.tree_container, columns=column_ids,
show="headings", ...)
self.tree.heading("device", text="Пристрій", anchor="w")
self.tree.column("device", width=250, minwidth=200)
self.tree.heading("status_l3", text="Зв'язок (L3)", anchor="center")
```

```
self.tree.column("status_l3", width=100, minwidth=100, anchor="center")
self.tree.heading("status_l4", text="Сервіс (L4)", anchor="center")
self.tree.column("status_l4", width=100, minwidth=100, anchor="center")
```

Критично важливою функцією є миттєва візуальна ідентифікація проблемних пристроїв. Treeview дозволяє реалізувати це за допомогою системи тегів. Під час ініціалізації програма налаштовує теги, що відповідають кожному стану пристрою, та прив'язує їх до кольорів, як показано у лістингу 3.11.

Лістинг 3.11 — Налаштування тегів кольорів для статусів

```
self.tree.tag_configure("good", foreground=COLOR_GOOD)
self.tree.tag_configure("medium", foreground=COLOR_MEDIUM)
self.tree.tag_configure("bad", foreground=COLOR_BAD)
self.tree.tag_configure("l4_fail", foreground=COLOR_L4_FAIL)
self.tree.tag_configure("l4_threat", foreground=COLOR_L4_THREAT)
```

Оновлення таблиці відбувається у методі update_results_ui, який безпечно викликається з головного потоку. Цей метод спочатку повністю очищує таблицю від старих даних (self.tree.delete(row)), а потім наповнює її свіжими даними, отриманими від модуля аналізу. Реалізацію методу update_results_ui можна побачити в лістингу 3.12.

Лістинг 3.12 — Логіка оновлення та наповнення таблиці

```
def update_results_ui(self, devices):
for row in self.tree.get_children():
self.tree.delete(row)
if devices:
try: sorted_devices = sorted(devices, key=lambda x: [int(part) for part in
x['ip'].split('.')])
except Exception: sorted_devices = sorted(devices, key=lambda x: x['ip'])
for i, device in enumerate(sorted_devices):
status_l3_text = device['status_l3']
status_l4_text = device.get('status_l4', '...')
status_overall = device.get('status_overall', status_l3_text)
status_tag = "default"
if status_overall == 'Добрий': status_tag = "good"
elif status_overall == 'Загроза L4': status_tag = "l4_threat"
values = (
truncate_text(display_name, 25),
```

```
device['ip'],
device['mac'],
truncate_text(device['manufacturer'], 30),
status_l3_text,
status_l4_text)
self.tree.insert("", "end", values=values, tags=(row_tag, status_tag))
```

Вибір колірної схеми для відображення статусів базується на принципах ергономіки інтерфейсів моніторингу. Використання яскравих, контрастних кольорів для критичних станів дозволяє оператору миттєво фокусувати увагу на проблемах, навіть при побіжному погляді на екран (peripheral vision monitoring). Для менш критичних станів використовуються приглушені кольори, щоб не створювати візуального шуму. Окрім кольорового кодування, інтерфейс реалізує сортування списку, яке автоматично піднімає проблемні пристрої вгору таблиці. Це реалізовано шляхом присвоєння ваги кожному статусу при формуванні списку sorted_devices.. Такий підхід до побудови UI/UX є стандартом для професійних диспетчерських пультів і був адаптований для даного засобу.

3.5.2 Відображення детальної діагностики та графіків історії

Якщо головна таблиця дає швидкий загальний огляд стану всіх пристроїв, то екран "Деталі" (detail_frame) надає поглиблену діагностичну інформацію для одного обраного пристрою. Перехід на цей екран відбувається при виборі рядка у таблиці та натисканні кнопки "Деталі", що викликає метод on_details_click.

Цей екран вирішує два завдання: по-перше, надати користувачеві чітке текстуальне пояснення проблеми та поради щодо її вирішення; по-друге, візуалізувати історичні дані L3-моніторингу.

Діагностичний блок реалізований за допомогою набору віджетів ctk.CTkLabel всередині фрейму detail_advice_frame. На відміну від головної таблиці, яка показує лише короткий статус, цей модуль надає розгорнутий аналіз. Оновлення цієї інформації відбувається у методі update_detail_info_and_advice.

Цей метод є прямою реалізацією вимог методології. Він чітко розділяє діагностику на detail_l3_diag_label та detail_l4_diag_label. Це дозволяє коректно

відобразити складні стани, наприклад, коли L3-статус "Добрий", але L4-статус "Відмова", і надати користувачеві окремі поради для кожного рівня.

Другим компонентом екрану "Деталі" є graph_frame, призначений для візуалізації історії показників L3. Для цього використовується інтеграція з бібліотекою Matplotlib (рис. 3.5).

Рисунок 3.4 — Вигляд екрану деталей

Під час кожного циклу моніторингу, у методі scan_network_arp, програма зберігає історію затримок пінгу для кожного пристрою у словнику state["history"]. Метод update_detail_graph (ліст. 3.13) відповідає за відображення цих даних.

Лістинг 3.13 — Інтеграція Matplotlib для побудови графіка затримок

```
def update_detail_graph(self):
state = self.device_states[self.current_detail_mac];
history = state.get("history", {"timestamps": [], "latency": []})
```

```
timestamps = history["timestamps"]; latencies = history["latency"]
if self.detail_chart_canvas:
self.detail_chart_canvas.get_tk_widget().destroy()
fig = plt.Figure(figsize=(5, 3), dpi=100);
fig.patch.set_facecolor("#2B2B2B")
ax = fig.add_subplot(111)
if timestamps:
ax.plot(timestamps, latencies, marker='o', linestyle='-', color='#1F6AA5')
ax.set_title("Історія затримки (Ping, ms)", color="white")
ax.set_facecolor("#333333")
fig.tight_layout()
self.detail_chart_canvas = FigureCanvasTkAgg(fig,
master=self.graph_frame)
self.detail_chart_canvas.draw()
self.detail_chart_canvas.get_tk_widget().pack(fill="both", expand=True)
```

Ключовим елементом тут є клас FigureCanvasTkAgg з matplotlib.backends.backend_tkagg. Він діє як "міст" між об'єктом графіка fig та інтерфейсом Tkinter. Він дозволяє "намалювати" графік Matplotlib на спеціальному віджеті canvas, який потім вбудовується у graph_frame за допомогою звичайного методу pack(). Цей метод викликається щоразу при оновленні даних, ефективно "перемальовуючи" графік з актуальною історією L3-затримок.

4 ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО МЕТОДУ ТА ЗАСОБУ МОНІТОРИНГУ ІОТ-ПРИСТРОЇВ

4.1 Підготовка тестового середовища

Перед початком тестування розробленого програмного комплексу було розгорнуто експериментальний стенд, який імітує типову інфраструктуру локальної мережі малого офісу або домашнього середовища (SOHO). Метою підготовки стенда є створення контрольованого середовища для безпечної імітації мережевих відмов та кіберзагроз без впливу на роботу критично важливих систем.

До складу тестового стенда увійшли наступні апаратні компоненти:

— станція моніторингу, а саме ноутбук Lenovo (CPU Intel Core i5, RAM 16GB, OS Windows 10), на якому запущено розроблений програмний засіб, виконує роль сервера моніторингу;

— мережевий шлюз, що є маршрутизатором Netcore/TP-Link, який забезпечує комутацію пакетів та Wi-Fi покриття з адресацією мережі: 192.168.1.0/24;

— смартфон Redmi (Android OS), підключений по Wi-Fi, використовується для перевірки мобільності та L3-відмов;

— віртуальний інтерфейс на станції моніторингу (Localhost/LAN IP), на якому програмно відкриваються та закриваються порти, використовується для точної перевірки L4-відмов.

Програмне забезпечення стенда розгорнуто на базі інтерпретатора Python версії 3.10. Критично важливим елементом програмного оточення є встановлення бібліотеки Npcap, яка виконує роль драйвера для захоплення пакетів у середовищі Windows. Наявність цього драйвера є обов'язковою для коректної роботи бібліотеки Scapy, зокрема для забезпечення функціонування режиму promiscuous mode під час ARP-сканування мережі. Окрім системних драйверів, середовище включає всі необхідні залежності проекту, такі як бібліотеки customtkinter для графічного інтерфейсу, matplotlib для побудови

графіків та psutil для роботи з системними інтерфейсами.

Для перевірки реакції системи на аномальні події було розроблено та застосовано методику штучного введення несправностей ("Fault Injection"), оскільки очікування реальних апаратних поломок є недоцільним у рамках лабораторного дослідження. Методика передбачає моделювання трьох базових станів. Для імітації нормальної роботи сервісу використовується вбудований HTTP-сервер Python, який запускається на порту 8080, емулюючи роботу веб-інтерфейсу типового IoT-пристрою. Для моделювання сервісної відмови (L4 Failure) застосовується примусове завершення процесу цього сервера без розриву фізичного з'єднання з мережею, що імітує програмний збій або "зависання" служби. Для моделювання безпекової аномалії (L4 Threat) використовується спеціалізований скрипт, який відкриває з'єднання на одному з небезпечних портів, наприклад Telnet (порт 23), що дозволяє імітувати активність шкідливого програмного забезпечення або бекдора.

Кожен експеримент проводиться за стандартизованим алгоритмом, що забезпечує відтворюваність результатів. Процедура починається з повного очищення файлів даних, зокрема fingerprints.json, для скидання "пам'яті" системи. Наступним кроком здійснюється запуск системи моніторингу та фіксація стабільного базового стану. Після цього відбувається цілеспрямований вплив на тестовий об'єкт згідно з обраним сценарієм несправності. На завершальному етапі проводиться реєстрація часу реакції системи та верифікація зміни статусів у графічному інтерфейсі користувача та журналі подій. Такий підхід дозволяє оцінити ефективність розроблених алгоритмів виявлення відмов.

4.2 Тестування сценарію виявлення мережевої відмови L3

Метою даного тесту є перевірка коректності роботи механізму виявлення повної мережевої відмови (L3). Сценарій імітує ситуацію, коли пристрій, що раніше був у мережі, раптово вимикається, перезавантажується або фізично втрачає зв'язок з мережею.

Для проведення тесту спершу необхідно запустити розроблений програмний засіб та активувати моніторинг. Потрібно дочекатися, доки в головній таблиці з'явиться "тестовий об'єкт". У якості такого об'єкта виступає смартфон, підключений до тієї ж Wi-Fi мережі, що і комп'ютер з програмою моніторингу.

Необхідно переконатися, що програма коректно ідентифікувала цей пристрій та присвоїла йому статус L3 "Добрий". Це початковий, стабільний стан, який буде зафіксовано.

Після цього, на тестовому об'єкті потрібно примусово розірвати з'єднання з мережею, просто вимкнувши модуль Wi-Fi. Програма моніторингу при цьому залишається активною і продовжує сканування з інтервалом у 15 секунд.

Згідно з логікою, реалізованою в модулі аналізу, програма не повинна миттєво позначати пристрій як "Offline". Вона має використовувати поріг відмов. Очікується, що пристрій залишатиметься у списку протягом одного-двох циклів сканування, оскільки лічильник для нього буде збільшуватися з 1 до 2.

Після досягнення порогового значення (коли failure_count стане 3), програма має коректно класифікувати пристрій як "Offline", додати відповідне повідомлення про відмову (L3) у журнал (log_textbox), та прибрати цей пристрій зі списку активних.

На Рисунку 4.1 показано початковий стан тестового середовища. Тестовий об'єкт присутній у списку пристроїв. Його статус L3 "Добрий" та L4 "Чистий", що є початковим нормальним станом.

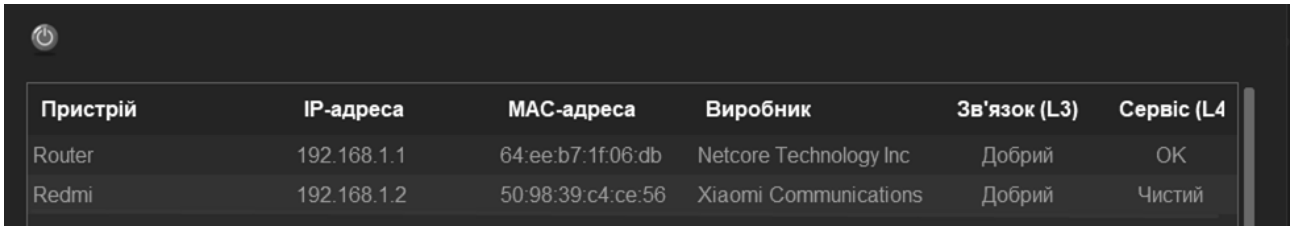

Рисунок 4.1 — Тестовий пристрій на головному екрані програми

Після фізичного відключення Wi-Fi на смартфоні, програма, як і

очікувалося, не зреагувала миттєво. Пристрій залишався у списку протягом двох циклів сканування (рис. 4.2).

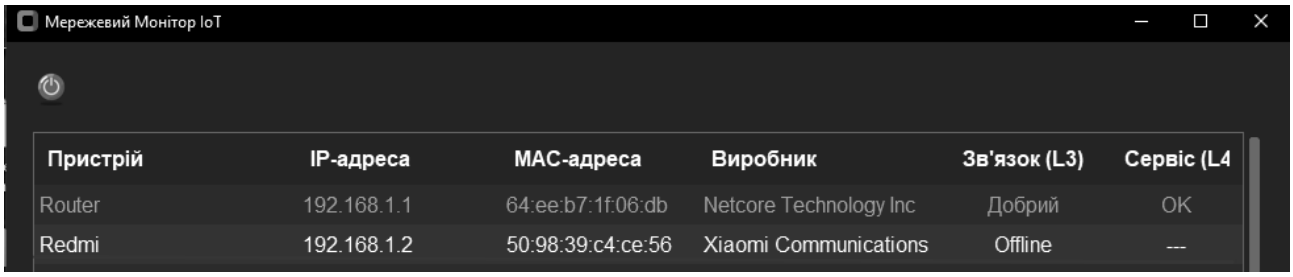

Рисунок 4.2 — Статус пристрою при відключенні

На Рисунку 4.3 показано стан програми в момент фіксації відмови L3. Видно, що в журналі подій з'явилося нове повідомлення. Одразу після цього запис про цей пристрій зник з верхньої таблиці активних пристроїв.

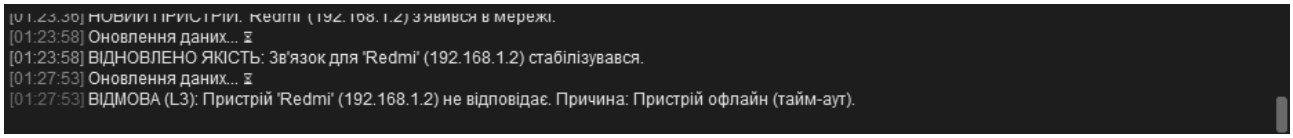

Рисунок 4.3 — Фіксації відмови L3

Це підтверджує, що механізм виявлення мережевих відмов L3, включно з використанням порогу failure_threshold для захисту від хибних спрацьовувань, реалізований коректно та працює згідно з вимогами.

Експериментальне визначення оптимального значення порогу failure_threshold показало, що значення 3 є найкращим компромісом між швидкістю реакції та стійкістю до завад. При зменшенні порогу до 1 (миттєва реакція) спостерігалася значна кількість хибних спрацювань у моменти високого навантаження на Wi-Fi мережу (наприклад, при перегляді потокового 4K-відео). Це призводило до ефекту «миготіння» статусів (flapping), що знижувало довіру користувача до системи. З іншого боку, збільшення порогу до 5 і більше призводило до неприпустимої затримки виявлення відмови, яка могла досягати 1.5–2 хвилин (враховуючи інтервал сканування 15 секунд). Затримка у фіксації

статусу Offline також впливає на актуальність L4-моніторингу, оскільки система продовжує намагатися підключитися до портів пристрою, якого вже немає в мережі, марнуючи системні ресурси на очікування тайм-аутів TCP.

4.3 Тестування сценарію виявлення відмови сервісного рівня L4

Метою цього тесту є перевірка ключової функції програми: здатності виявляти приховані сервісні відмови (L4). Сценарій імітує ситуацію, коли пристрій залишається онлайн (L3 "Добрий"), але його очікуваний, "еталонний" сервіс (порт) припиняє відповідати.

Це тестування вимагає більш складної підготовки, ніж L3, оскільки воно передбачає фазу "навчання" для створення "еталону". Спочатку, в якості тестового об'єкта було обрано сам комп'ютер, на якому запущено програму моніторингу. Щоб створити контрольований сервіс, у окремому терміналі було запущено простий веб-сервер на порту 8080 за допомогою команди python -m http.server 8080.

Наступним кроком очистимо файл fingerprints.json від запису про цей ПК. Після цього запустимо програму моніторингу. Програма перейде у фазу "Навчання" і в наступному циклі створить "еталон", що включатиме порт 8080. Про успіх свідчить запис, який прив’язаний до MAC-адреси ПК (рис. 4.4).

Рисунок 4.4 — Додавання порту 8080 для тестування

Третім кроком є фіксація базового стану: програма коректно показувала тестовий ПК зі статусом L3 "Добрий" та L4 "OK". Фінальною дією стала імітація відмови: у терміналі, де працював веб-сервер, його було зупинено. Це миттєво закрило порт 8080, в той час як сам комп'ютер залишився онлайн.

Очікувалося, що у наступному циклі сканування програма спочатку успішно перевірить L3, залишивши статус status_l3 "Добрим". Далі, вона мала перейти до L4-сканування, де метод scan_ports_socket повернув би список, у якому порт 8080 вже відсутній. Очікувалося, що модуль аналізу, порівнявши "еталон" ([...8080]) з "поточним знімком", виявить порт 8080 у списку closed_etalon_ports. Це мало призвести до зміни статусу L4 на "Відмова", а загального status_overall — на "Відмова L4". У журнал мав бути доданий відповідний запис, а рядок у таблиці — змінити колір на блакитний.

На рисунку 4.5 показано стан перед імітацією відмови. Тестовий ПК знаходиться у стані "Добрий" (L3) та "OK" (L4). Це означає, що програма успішно "вивчила" відкритий порт 8080 і вважає його нормальним.

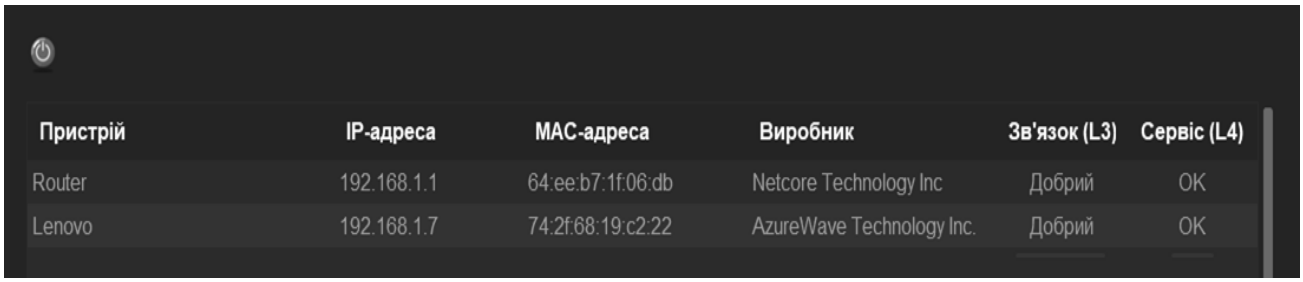

Рисунок 4.5 — Стан тестового ПК до зупинки сервісу 8080

Після зупинки веб-сервера, буквально на наступному циклі оновлення, інтерфейс програми миттєво зміниться. Як видно на рисунку 4.6, стан L3 залишився "Добрий", але L4 негайно змінився на "Відмова", і весь рядок забарвився у відповідний блакитний колір (тег l4_fail). Одночасно в журналі подій з'явився детальний запис, що ідентифікував проблему: "ВІДМОВА СЕРВІСУ (L4)" та вказав, який саме порт перестав відповідати ([8080]).

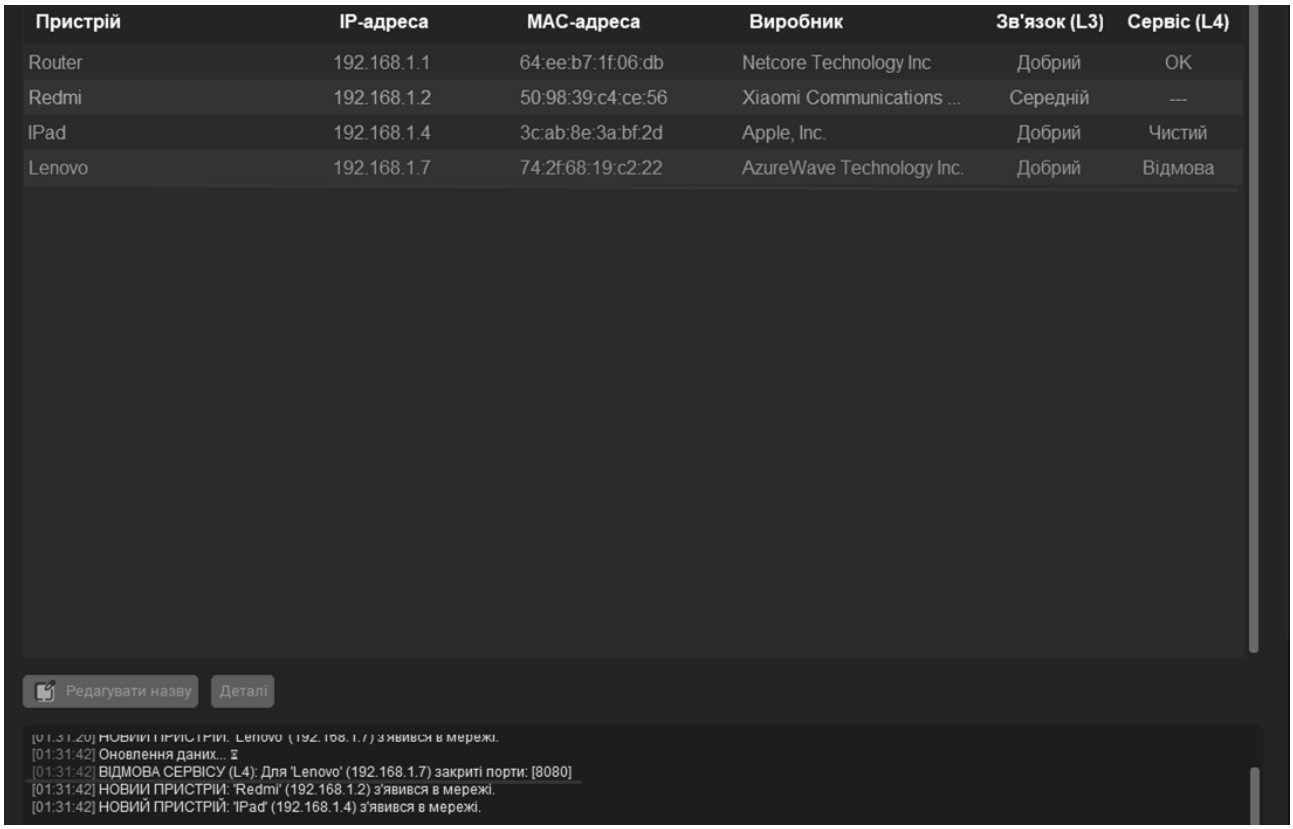

Рисунок 4.6 — Реакція програми на відмову

Тест довів, що розроблений "метод еталонів" є повністю робочим. Програма успішно виконала головне завдання: проігнорувала робочий L3-зв'язок і коректно ідентифікувала приховану L4-відмову сервісу, чітко вказавши на джерело проблеми.

4.4 Тестування сценарію виявлення безпекової аномалії L4

Метою даного тесту є перевірка другої ключової функції методу "еталонів" — здатності виявляти безпекові аномалії L4. Сценарій імітує ситуацію компрометації пристрою, коли на ньому раптово з'являється новий, несанкціонований та потенційно небезпечний сервіс, якого не було у "нормальному" стані. Це тестування, аналогічно до попереднього, вимагає створення контрольованого "еталону", але цього разу — "чистого".

Спочатку проведемо підготовку тестового стенда. Усі сторонні сервіси на тестовому ПК були зупинені. Після цього запис про цей ПК було видалено з файлу fingerprints.json, щоб примусово запустити "перенавчання".

Далі було запущено програму моніторингу. Програма коректно виявила, що "еталон" відсутній, і виконала фазу "Навчання". Оскільки жодних поширених сервісів на ПК запущено не було, було створено "чистий" еталон, тобто порожній список. Програма перейшла у штатний режим, відображаючи для тестового ПК статус L3 "Добрий" та L4 "Чистий".

Після фіксації базового стану імітуємо загрозу. У окремому терміналі запустимо Python-скрипт, що почав "слухати" на небезпечному порту 23 (Telnet), який входить до списку COMMON_PORTS. Це створило ситуацію, коли на пристрої з "чистим" еталоном з'явився новий небезпечний сервіс.

Очікувалося, що в наступному циклі сканування програма спершу підтвердить L3-доступність ("Добрий"). Потім, під час L4-сканування, вона перевірить порти з об'єднаного списку etalon_ports та COMMON_PORTS. Метод scan_ports_socket виявить відкритий порт 23.

Далі, логіка аналізу мала порівняти "поточний знімок" з "еталоном". Очікувалося, що перевірка new_dangerous_ports ([p for p in current_open_ports if p not in etalon_ports]) поверне [23]. Це, у свою чергу, повинно було активувати блок elif new_dangerous_ports, встановивши status_l4 у "Загроза". Як наслідок, рядок у таблиці мав забарвитися у малиновий (magenta), а в журнал — додатися відповідне повідомлення.

На рисунку 4.7 показано початковий стан "Baseline". Тестовий ПК коректно просканований, його L3-статус "Добрий", а L4-статус "Чистий". Це підтверджує, що програма створила "чистий" еталон.

Пристрій	IP-адреса	MAC-адреса	Виробник	Зв'язок (L3)	Сервіс (L4)
Router	192.168.1.1	64:ee:b7:1f:06:db	Netcore Technology Inc	Добрий	OK
Lenovo	192.168.1.7	74:2f:68:19:c2:22	AzureWave Technology Inc.	Добрий	Чистий

Рисунок 4.7 — Початковий стан тестового ПК при чистому “еталоні”

Наступним кроком в командному рядку запускаємо спеціальний скрипт для того, щоб відкрити порт 23 (ліст. 4.1), щоб перевірити чи наступний цикл моніторингу виявить безпекову аномалію, яка може бути потенційною загрозою для локальних мереж. Результат фіксації безпекової аномалії показано на рисунку 4.8.

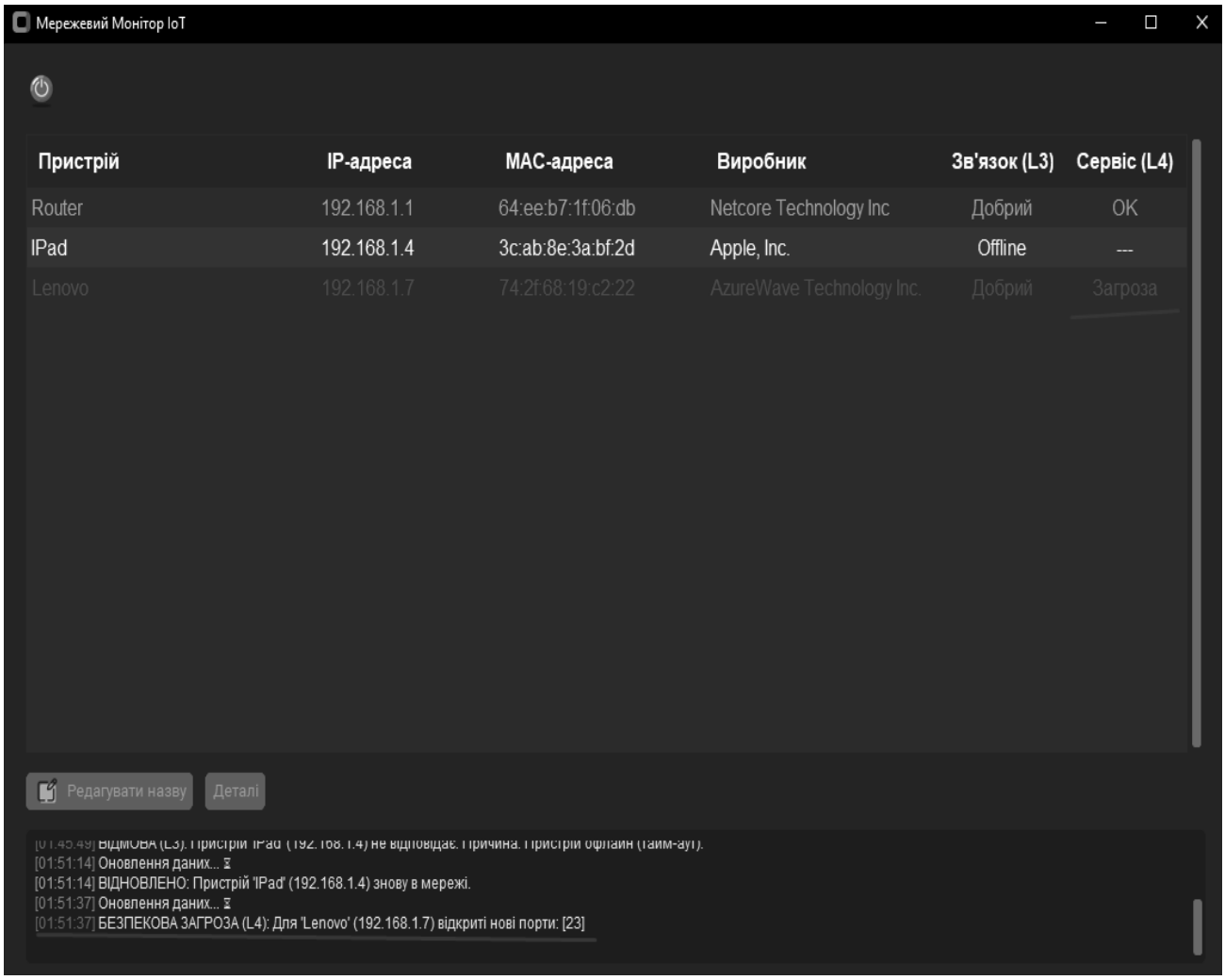

Рисунок 4.8 — Виявлення безпекової аномалії L4

Як видно на рисунку 4.8, L3-статус залишився "Добрим", але L4-статус миттєво змінився на "Загроза". Увесь рядок пристрою забарвився у відповідний малиновий колір. В той же час у журналі з'явилося критичне сповіщення "БЕЗПЕКОВА ЗАГРОЗА (L4)" із зазначенням, що на пристрої з'явився новий порт [23], що свідчить про те що це може бути кіберзагроза.

Лістинг 4.1 — Скрипт для відкриття порту 23

```
python
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("", 23))
s.listen(1)
conn, addr = s.accept()
```

Тест підтвердив, що розроблений метод коректно працює в обох напрямках. Програма здатна виявляти не лише зникнення очікуваних сервісів (Відмова L4), але й появу нових, несанкціонованих сервісів (Загроза L4). Це доводить, що метод "еталонів" є повноцінним інструментом для моніторингу як працездатності, так і базової безпеки IoT-пристроїв.

4.5 Дослідження часових характеристик підсистеми сканування

Окрім перевірки функціональної коректності виявлення відмов, було проведено дослідження продуктивності розробленого алгоритму сканування портів. Метою цього експерименту є перевірка та практичне підтвердження теоретичних розрахунків ефективності розпаралелювання запитів.

Умови проведення експерименту:

— об'єктом тестування є метод scan_ports_socket, що відповідає за L4-сканування;

— тестовий сценарій грунтується на скануванні діапазону портів на пристрої, який знаходиться в мережі (L3 OK), але не має відкритих портів у цьому діапазоні;

— змінний параметр, що є кількістю портів у списку сканування (N);

— вимірюваний параметр, що є загальним часом виконання функції (T);

— параметрами алгоритму є тайм-аут τ = 0,5с, кількість потоків W = 20.

Було проведено вимірювання у послідовному (синхронний), та паралельному (асинхронний) режимах. Очікується, що паралельний режим покаже набагото швидші показники а ніж послідовний. Результати вимірювань наведено у таблиці 4.1.

Таблиця 4.1 — Залежність часу сканування від кількості портів

Кількість портів (*N*)	**Час (Послідовно), с (T_{seq})**	**Час (Паралельно), с (T_{par})**	**Коефіцієнт прискорення (*K*)**
1	0.51	0.52	1.0
5	2.55	0.54	4.7
10	5.10	0.56	9.1
15	7.65	0.58	13.2
20	10.20	0.61	16.7
30	15.30	1.15	13.3

Як видно з таблиці, у послідовному режимі час виконання зростає лінійно ($T \approx N \times \tau$). Сканування навіть базового набору з 15 портів займає понад 7 секунд, що є неприпустимим для моніторингу в реальному часі, оскільки блокує інтерфейс.

У паралельному режимі (штатна робота програми) спостерігається інша залежність. Доки кількість портів N не перевищує розмір пулу потоків (W=20), час виконання залишається майже сталим і близьким до значення тайм-ауту одного запиту:

$$T_{par} \approx \tau + \delta, \tag{4.1}$$

де δ — накладні витрати на створення потоків (0.02–0.1 с).

Різке зростання часу для N=30 у паралельному режимі (1.15 с) пояснюється тим, що кількість завдань перевищила ліміт потоків (20). Програма виконала перші 20 портів за 0.5 с, після чого звільнені потоки взяли в роботу останні 10 портів (ще 0.5 с).

Застосування багатопотокового підходу дозволило досягти середнього прискорення у 13–16 разів для типових задач моніторингу (сканування 10–20 портів). Це підтверджує правильність вибору архітектурного рішення та забезпечує високу реактивність системи виявлення відмов, що є черговою особливістю розробленого засобу.

4.6 Тестування сценарію відновлення роботи пристрою

Метою даного тесту є перевірка здатності програмного засобу автоматично фіксувати відновлення пристрою після відмови. Ефективна система моніторингу повинна не лише виявляти проблеми, але й самостійно повідомляти про нормалізацію стану. Тестування проводиться для обох типів відмов: L4 (відновлення сервісу) та L3 (повернення пристрою в мережу).

Для цього тесту було використано стан системи, досягнутий у тесті 4.2. У цьому стані тестовий ПК мав статус L3 "Добрий", але L4 "Відмова", оскільки "еталонний" сервіс на порту 8080 було зупинено.

Дія для імітації відновлення є простою: у терміналі, де раніше було зупинено веб-сервер, запускаємо знову за тією ж командою python -m http.server 8080. Це миттєво відкриє порт 8080, який програма раніше вважала "втраченим".

Очікувалося, що в наступному 15-секундному циклі сканування програма, просканувавши порти ПК, тепер виявить порт 8080 у current_open_ports. При порівнянні з "еталоном" список closed_etalon_ports стане порожнім, так само як і new_dangerous_ports. Це призведе до переходу у гілку логіки, що встановлює статус L4 "ОК". При оновленні стану device_states, логіка повинна помітити зміну статусу з "Відмова L4" на "ОК" та додати у журнал повідомлення про відновлення.

На рисунку 4.9 показано стан програми через 15 секунд після повторного запуску сервісу на порту 8080. Як видно на рисунку, колір рядка тестового ПК автоматично змінився з блакитного на зелений, а статус L4 повернувся до "ОК".

Це супроводжувалося появою у журналі повідомлення про відновлення якості зв'язку для даного пристрою.

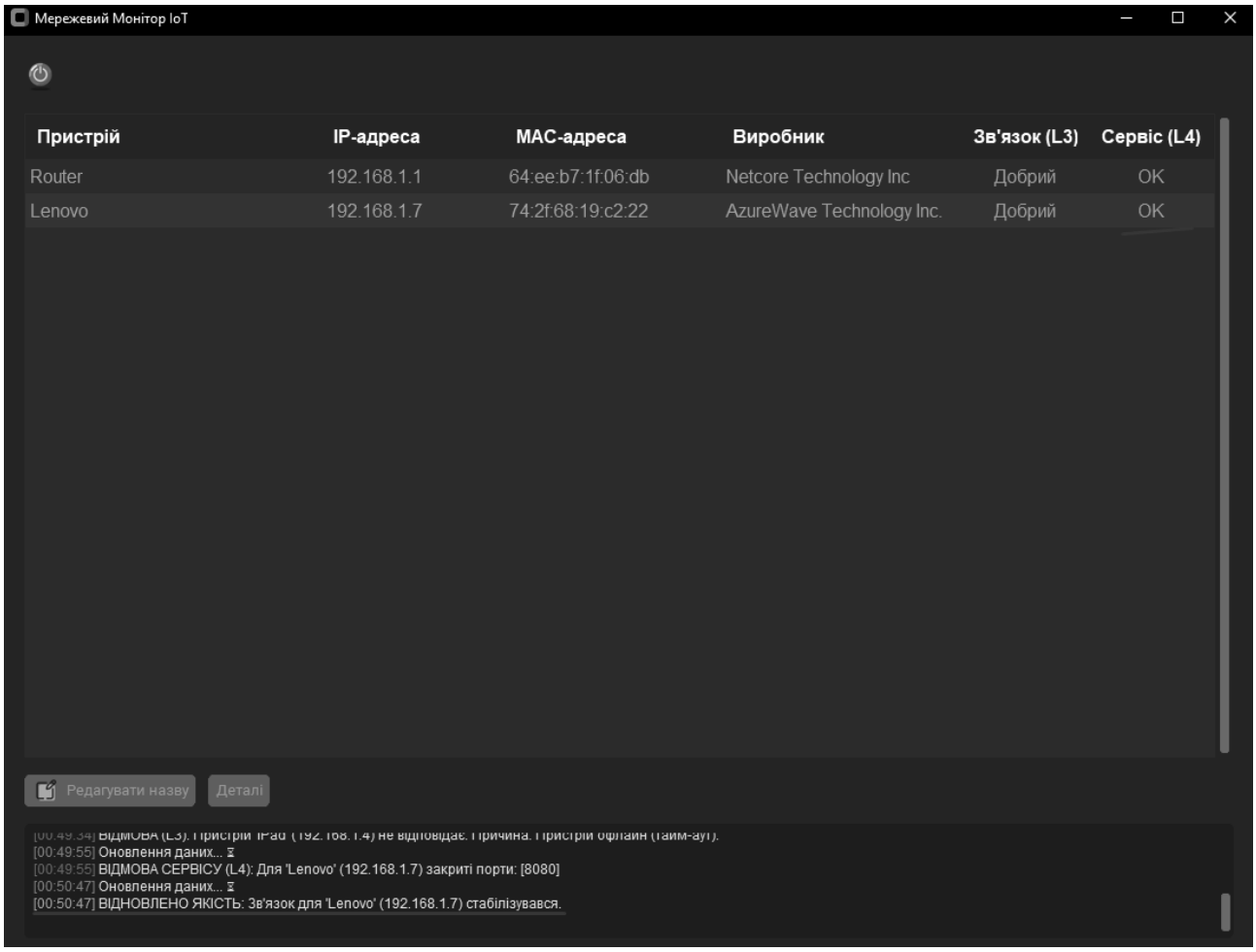

Рисунок 4.9 — Стабілізація стану тестового ПК після відмови

Друга частина тесту перевіряє відновлення після повної L3-відмови. Було взято стан з тесту, де тестовий смартфон був відсутній у списку, а в журналі була зафіксована "ВІДМОВА (L3)". Дія відновлення полягала у простому ввімкненні Wi-Fi на смартфоні.

Як і очікувалось, у наступному циклі сканування scan_network_arp знову виявив пристрій. Логіка в monitoring_loop коректно ідентифікувала, що пристрій state["failure_count"] був більшим за поріг, і додала до журналу повідомлення про відновлення, після чого скинула лічильник відмов. На рисунку 4.10 показано стан програми після того, як смартфон знову з'явився в мережі.

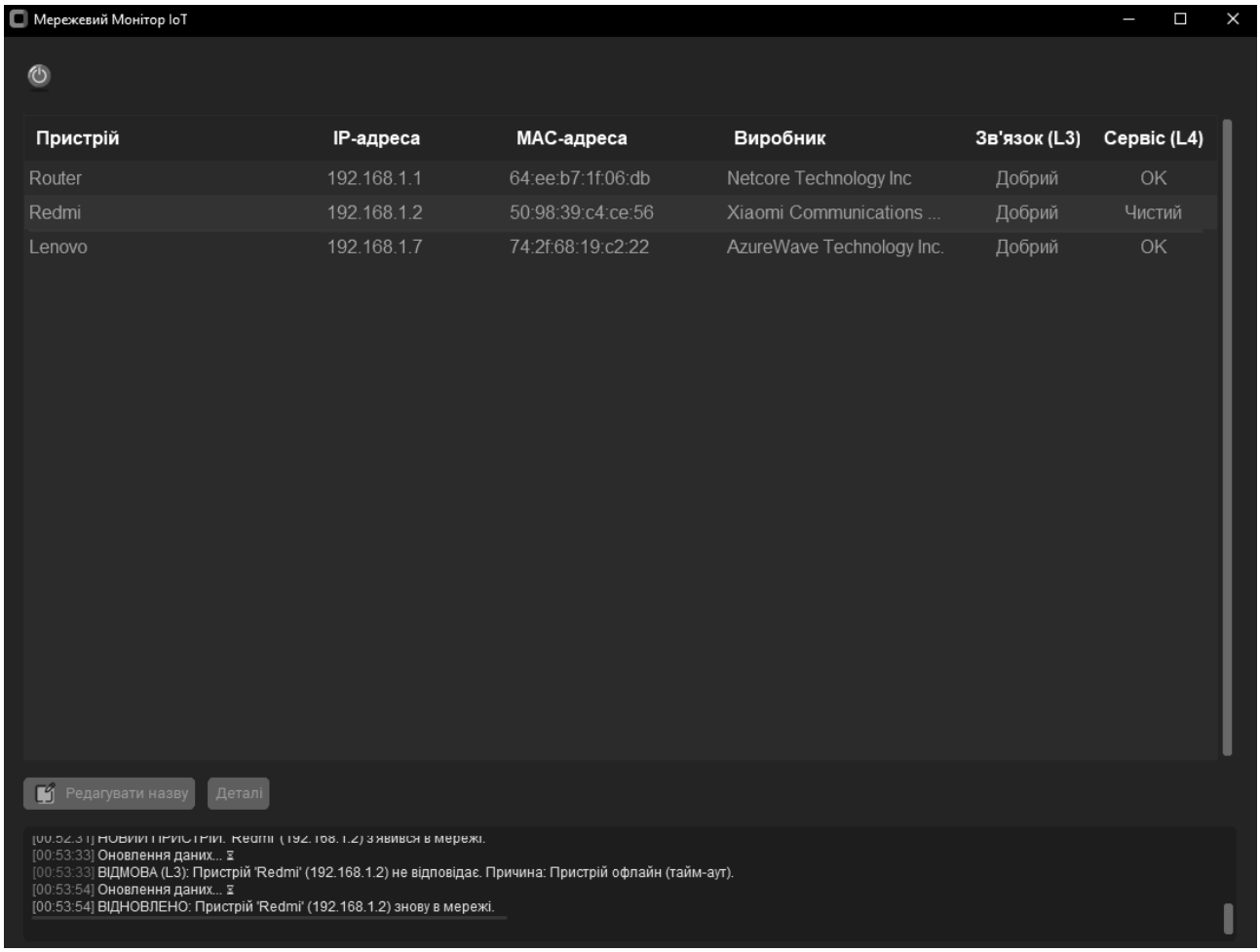

Рисунок 4.10 — Повернення в мережу тестового смартфона

Тестування показало, що розроблений програмний засіб коректно обробляє повний життєвий цикл стану пристрою: "Норма", "Відмова" (L3 або L4) та "Відновлення". Це підтверджує, що метод "еталонів" та логіка порогу відмов реалізовані коректно та є готовими для практичного застосування.

5 ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ

5.1 Оцінювання комерційного потенціалу розробки

Мета проведення комерційного та технологічного аудиту є розробка нового за принципом дії та ефективного за результатами застосування програмного комплексу для моніторингу IoT-пристроїв у локальних мережах, здатного виявляти як мережеві (L3), так і сервісні (L4) відмови.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів з Вінницького національного технічного університету, кафедри обчислювальної техніки: к.т.н., доц. Крупельницький Л. В., к.т.н., доц. Богомолов С. В., к.т.н., доц. Савицька Л. А.

Для проведення технологічного аудиту було використано таблицю 5.1 в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу [18].

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Продовження табл. 5.1

5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Середньоарифметична оцінка, отримана на основі експертних висновків, становить 33 бали, і згідно з таблицею 5.2, це вказує на рівень вище середнього комерційного потенціалу результатів проведених досліджень.

Таблиця 5.2 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

В таблиці 5.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 5.3 — Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Крупельницький Л. В.	Богомолов С. В.	Савицька Л. А.
	Бали, виставлені експертами:		
1	4	4	4
2	4	4	4
3	4	4	4
4	4	4	4
5	4	4	4
6	4	4	4
7	2	2	2
8	4	4	4
9	4	4	4
10	4	4	4
11	4	4	4
12	4	4	4
Сума балів	$СБ_1$=31	$СБ_2$=34	$СБ_3$=34
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{3} = \frac{31+34+34}{3} = 33$		

Результати роботи мають практичне та академічне застосування і можуть бути використані двома основними групами:

Кінцеві користувачі (користувачі програмного засобу): ентузіасти "розумного будинку" та досвідчені користувачі: Люди, які мають у своїх локальних мережах значну кількість IoT-пристроїв (камери, сенсори, реле) і потребують легкого, але потужного інструменту для швидкої діагностики стану пристроїв та виявлення прихованих відмов; системні адміністратори малих офісів (SOHO): Технічні спеціалісти, яким потрібен безкоштовний, безагентний та автономний засіб для моніторингу стану критичних пристроїв (IP-камер, мережевих принтерів, NAS), який є простішим за комплексні корпоративні системи (Zabbix, PRTG).

Академічна та інженерна спільнота (користувачі дослідження): студенти та дослідники, спеціалісти з кібербезпеки.

Проведемо оцінку якості і конкурентоспроможності нової розробки порівняно з аналогом.

В якості аналога для розробки було обрано програмний засіб Fing Desktop. Це популярний, комерційно успішний додаток, призначений для тієї ж цільової аудиторії (домашні користувачі та малі офіси) і виконує схожі базові функції: автоматичне виявлення пристроїв у локальній мережі, ідентифікація виробника, перевірка доступності L3 (пінг) та сканування відкритих портів L4.

Основними недоліками аналога (Fing Desktop) є його нездатність до автоматичного аналізу відхилень. Fing чудово констатує поточний стан: він може показати, що на пристрої X відкриті порти «[80, 23]». Однак він не має вбудованої логіки "еталону". Він не може автоматично попередити користувача, що порт 80 (який був раніше) зник (це Відмова L4), або що порт 23 (якого раніше не було) з'явився (це Загроза L4). Користувач повинен сам пам'ятати "нормальний" стан і вручну налаштовувати сповіщення для кожного порту.

Також до недоліків можна віднести: комерційну модель та залежність від хмари: Багато розширених функцій (детальна історія, гнучкі сповіщення)

доступні лише за платною підпискою "Fing Premium" і вимагають підключення до хмарних сервісів Fing, закритість системи: Це продукт із закритим кодом. Користувач не може додати власні типи сканувань, змінити логіку чи пороги спрацьовування.

У розробці дана проблема вирішується за допомогою методу "еталонів" (фінгерпринтингу). Замість того, щоб змушувати користувача вручну налаштовувати моніторинг кожного сервісу, розроблений засіб автоматично виконує "Навчання" (фаза 1), створюючи fingerprints.json. Після цього він самостійно і безперервно порівнює "еталон" з "поточним знімком", автоматично виявляючи обидва типи L4-відхилень (відмови та загрози).

Також система випереджає аналог за такими параметрами як:

— автоматичне виявлення відмов L4 та загроз L4, що є ключовою перевагою над Fing Desktop, який вимагає ручного налаштування, розроблений засіб робить це автоматично після "навчання";

— нульова вартість та відсутність підписки, тобто розробка є безкоштовною і не має прихованих платежів за преміум-функції;

— вся логіка, історія та "еталони" зберігаються локально (100% Offline), а Fing Desktop в свою чергу вимагає хмарної синхронізації для багатьох своїх функцій;

— прозорість та гнучкість, завдяки відкритому коду, порогові значення (як failure_threshold), списки портів (COMMON_PORTS) та логіка можуть бути легко модифіковані;

— вбудована візуалізація історії, яка містить вбудований графік історії затримки Matplotlib, що дозволяє візуально оцінити деградацію L3-зв'язку, а у безкоштовній версії Fing Desktop аналогічний функціонал обмежений.

В таблиці 5.4 наведені основні техніко-економічні показники аналога і нової розробки. Проведемо оцінку якості продукції, яка є найефективнішим засобом забезпечення вимог споживачів та порівняємо її з аналогом про який згадувалось раніше.

Таблиця 5.4 — Основні параметри нової розробки та товару-конкурента

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (товар-конкурент)	Новий (інноваційне рішення)		
1	2	3	4	5
Автоматизація виявлення відмов, %	20	100	5	40%
Автоматизація виявлення загроз, %	0	100	0	25%
Складність налаштування, годин	2	0.1	20	15%
Залежність від хмарних сервісів, %	100	0	0	10%
Вимоги до ресурсів (RAM), МБ	150	50	3	10%

Визначимо відносні одиничні показники якості по кожному параметру за формулами (5.1) та (5.2) і занесемо їх у відповідну колонку табл. 5.5.

$$q_i = \frac{P_{\text{Н}i}}{P_{\text{Б}i}} \tag{5.1}$$

або

$$q_i = \frac{P_{\text{Б}i}}{P_{\text{Н}i}}, \tag{5.2}$$

де $P_{\text{Н}i}$, $P_{\text{Б}i}$ — числові значення i-го параметру відповідно нового і базового виробів.

$$q_1 = \frac{100}{20} = 5;$$

$$q_2 = \frac{100}{0} = 0;$$

$$q_3 = \frac{2}{0{,}1} = 20;$$

$$q_4 = \frac{100}{0} = 0;$$

$$q_5 = \frac{150}{50} = 3.$$

Відносний рівень якості нової розробки визначаємо за формулою:

$$К_{я.в.} = \sum_{i=1}^{n} q_i \times a_i. \quad (5.3)$$

$$К_{я.в.} = 5 \cdot 0,4 + 2 \cdot 0,15 + 3 \cdot 0,1 = 2,6.$$

Відносний коефіцієнт показника якості нової розробки більший одиниці, отже нова розробка якісніший базового товару-конкурента.

Наступним кроком є визначення конкурентоспроможності товару. Конкурентоспроможність товару є головною умовою конкурентоспроможності підприємства на ринку і важливою основою прибутковості його діяльності.

Однією із умов вибору товару споживачем є збіг основних ринкових характеристик виробу з умовними характеристиками конкретної потреби покупця. Такими характеристиками найчастіше вважають нормативні та технічні параметри, а також ціну придбання та вартість споживання товару.

В табл. 5.5 наведено технічні та економічні показники для розрахунку конкурентоспроможності нової розробки відносно товару-аналога, технічні дані взяті з попередніх розрахунків.

Загальний показник конкурентоспроможності інноваційного рішення (К) з урахуванням вищезазначених груп показників можна визначити за формулою:

$$K = \frac{I_{m.n.}}{I_{e.n.}}, \quad (5.4)$$

де $I_{m.n.}$ — індекс технічних параметрів;

$I_{e.n.}$ — індекс економічних параметрів.

Індекс технічних параметрів є відносним рівнем якості інноваційного рішення. Індекс економічних параметрів визначається за формулою (5.5).

$$I_{e.n.} = \frac{\sum_{i=1}^{n} P_{Hei}}{\sum_{i=1}^{n} P_{Бei}}, \quad (5.5)$$

де P_{Hei}, $P_{\text{Б}ei}$ — економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

Таблиця 5.5 — Нормативні, технічні та економічні параметри нової розробки і товару-виробника

Показники	Варіанти	
	Базовий (товар-конкурент)	Новий (інноваційне рішення)
1	2	3
1. Нормативно-технічні показники		
Автоматизація виявлення відмов, %	20	100
Автоматизація виявлення загроз, %	0	100
Складність налаштування, годин	2	0.1
Залежність від хмарних сервісів, %	100	0
Вимоги до ресурсів (RAM), МБ	150	50
2. Економічні показники		
Вартість володіння (1 рік), грн	2100	1500

$$I_{\text{е.п.}} = \frac{1500}{2100} = 0{,}71;$$

$$\text{К} = \frac{2{,}6}{0{,}71} = 3{,}66.$$

Зважаючи на розрахунки, можна зробити висновок, що нова розробка буде конкурентоспроможніше, ніж конкурентний товар.

5.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

Основна заробітна плата кожного із дослідників $З_О$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою:

$$З_О = \frac{М}{Т_р} * t \text{ (грн)}, \quad (5.6)$$

де M — місячний посадовий оклад конкретного розробника, грн.;

T_P — число робочих днів в місяці; приблизно $T_P \approx 21...23$ дні;

t — число робочих днів роботи дослідника.

Зведемо сумарні розрахунки до таблиця 5.6.

Таблиця 5.6 — Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник	16000	761,9	5	3810
Програміст	18000	857,1	42	36000
Всього				39810

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт розраховують за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (5.7)$$

де C_i — погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i — час роботи робітника на виконання певної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{М_м \cdot К_і \cdot К_с}{Т_р \cdot t_{зм}}, \quad (5.8)$$

де $М_М$ — розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), грн;

$К_і$ — коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_с$ — мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати;

$Т_р$ — середня кількість робочих днів в місяці, приблизно Тр = 21…23 дні;

$t_{зм}$ — тривалість зміни, год.

Таблиця 5.7 — Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Погодинна тарифна ставка, грн	Величина оплати на робітника, грн
1. Розробка алгоритму	2	1	47,6	95,2
2. Написання коду	5	3	64,3	321,4
3. Налагодження програми	3	5	81,0	242,9
4. Випробувальні	8	2	52,4	419,0
5. Документування системи	3	4	71,4	214,3
Всього				1292,9

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як від 10 до 12 % від основної заробітної плати робітників.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 11% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{доп}}{100\%}. \quad (5.9)$$

$$З_д = 0{,}11 * (39810 + 1292{,}9) = 4521{,}26 \text{ (грн)}.$$

Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (5.10):

$$Н_{ЗП} = (З_о + З_р + З_д) * \frac{\beta}{100} \text{ (грн)}, \quad (5.10)$$

де $З_о$ — основна заробітна плата розробників, грн.;

$З_д$ — додаткова заробітна плата всіх розробників та робітників, грн.;

$З_р$ — основну заробітну плату робітників, грн.;

β — ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді**:**

$$Н_{ЗП} = (39810 + 1292{,}9 + 4521{,}26) * \frac{22}{100} = 10037{,}2 \text{ (грн)}.$$

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби й предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за прямим призначенням згідно з нормами їх витрачання, а також витрачені придбані напівфабрикати, що підлягають монтажу або виготовленню й додатковій обробці в цій організації, чи дослідні зразки, що виготовляються виробниками за документацією наукової організації.

Витрати на матеріали (М) у вартісному вираженні розраховуються окремо для кожного виду матеріалів за формулою:

$$М = \sum_{i=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{i=1}^{n} В_j \cdot Ц_{вj}, \quad (5.11)$$

де Hj — норма витрат матеріалу j-го найменування, кг;

n — кількість видів матеріалів;

Ц*j* — вартість матеріалу *j*-го найменування, грн/кг;

К*j* — коефіцієнт транспортних витрат, (К*j* = 1,1 … 1,15);

В*j* — маса відходів *j*-го найменування, кг;

Цв*j* — вартість відходів *j*-го найменування, грн/кг.

Проведені розрахунки зведені в таблицю 5.8.

Таблиця 5.8 — Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, шт	Вартість витраченого матеріалу, грн
Папір	170	0,5	85
Ручка	25	1	25
Блокнот	60	1	60
Флешка	280	1	280
З врахуванням коефіцієнта транспортування			495

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, можуть бути розраховані з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.12)$$

де $Ц_б$ — балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ — термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ — строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Проведені розрахунки необхідно звести до таблиці 5.9.

Таблиця 5.9 — Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Комп'ютер	20000	2	2	1666,67
Мережевий маршрутизатор	2500	4	2	104,17
Всього				1770,83

До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i}, \quad (5.13)$$

де W_{yt} — встановлена потужність обладнання на етапі розробки, кВт;

t_i — тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ — вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ — коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

η_i — коефіцієнт корисної дії обладнання, $\eta_i < 1$.

Для написання магістерської роботи використовується персональний комп’ютер для якого розрахуємо витрати на електроенергію.

$$В_е = \frac{0{,}5 \cdot 240 \cdot 12{,}69 \cdot 0{,}5}{0{,}8} = 951{,}75 .$$

Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{св} = (З_о + З_р) * \frac{Н_{св}}{100\%}, \quad (5.14)$$

де $Н_{св}$ — норма нарахування за статтею «Службові відрядження».

$$В_{св} = 0{,}2 * (39810 + 1292{,}9) = 8220{,}48.$$

Накладні (загальновиробничі) витрати Внзв охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати Внзв можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.15)$$

де $Н_{нзв}$ — норма нарахування за статтею «Інші витрати».

$$В_{нзв} = (39810 + 1292{,}9) \cdot \frac{100}{100\%} = 41102{,}38 \text{ грн.}$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР

В=39810+1292,9+4521,26+10037,2+495+1770,83+951,75+8220,48+41102,38= 108201,28грн.

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{В}{\eta}, \quad (5.16)$$

де η — коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт β = 0,7.

Звідси:

$$ЗВ = \frac{108201,28}{0,7} = 154573,26 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою

$$\Delta П_i = \sum_1^n (\Delta Ц_0 \times N + Ц_0 \times \Delta N)_i \times \lambda \times p \times \left(1 - \frac{\upsilon}{100}\right), \qquad (5.17)$$

де $\Delta Ц_0$ — покращення основного оціночного показника від впровадження результатів розробки у даному році;

N — основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN — покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$Ц_о$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n — кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $л$ = 0,8333;

p — коефіцієнт, який враховує рентабельність продукту. р = 0,25;

υ — ставка податку на прибуток. У 2025 році — 18%.

Припустимо, що ціна зросте на 500 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року на 500 шт., протягом другого року — на 550 шт., протягом третього року на 600 шт. Реалізація продукції до впровадження розробки складала 1 шт., а її ціна до 1500 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_1 = [500 \cdot 1 + (1500 + 500) \cdot 500] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 170911{,}91 \text{ грн};$$

$$\Delta П_2 = [500 \cdot 1 + (1500 + 500) \cdot (500 + 550)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 359235{,}65 \text{ грн};$$

$$\Delta П_3 = [500 \cdot 1 + (1500 + 500) \cdot (500 + 550 + 600)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 564227{,}45 \text{ грн}.$$

5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}, \tag{5.18}$$

де $k_{\text{інв}}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, які можуть являти собою витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{\text{інв}} = 2…5$).

$$PV = 2 \cdot 154573{,}26 = 309146{,}53.$$

Розрахуємо абсолютну ефективність вкладених інвестицій $E_{\text{абс}}$ згідно наступної формули:

$$E_{\text{абс}} = (ПП - PV), \tag{5.19}$$

де $ПП$ — приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.

$$ПП = \sum_1^T \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.20)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

T — період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t — період часу (в роках).

$$ПП = \frac{170911,91}{(1+0,2)^1} + \frac{359235,65}{(1+0,2)^2} + \frac{564227,45}{(1+0,2)^3} = 719935,01 \text{ грн.}$$

$$E_{\text{абс}} = (719935,01 - 309146,53) = 410788,48 \text{ грн.}$$

Оскільки $E_{\text{абс}} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій E_B. Для цього користуються формулою:

$$E_B = \sqrt[T_ж]{1 + \frac{E_{\text{абс}}}{PV}} - 1, \quad (5.21)$$

де $T_ж$ — життєвий цикл наукової розробки, роки.

$$\text{Е}_\text{в} = \sqrt[3]{1 + \frac{410788,48}{309146,53}} - 1 = 0,54 = 54\%.$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{5.22}$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні d = (0,14…0,2);

f — показник, що характеризує ризикованість вкладень, який зазвичай, дорівнює 0,05...0,1.

$$\tau_{min} = 0{,}18 + 0{,}05 = 0{,}23.$$

Так як $E_B > \tau_{min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{\text{ок}} = \frac{1}{E_{\text{в}}}, \tag{5.23}$$

$$T_{\text{ок}} = \frac{1}{0{,}54} = 1{,}8 \text{ роки}.$$

Так як $T_{\text{ок}} \leq$ 3...5-ти років, то фінансування даної наукової розробки в принципі є доцільним.

Аналіз отриманих показників свідчить про високу стійкість проекту до можливих ринкових коливань. Розрахований коефіцієнт відносної ефективності вкладень значно перевищує прийняту мінімальну ставку дисконтування, що формує суттєвий "запас міцності" для інвестора. Це означає, що навіть за умов зростання інфляційних процесів або часткового зменшення прогнозованого обсягу продажів, проект залишатиметься рентабельним. Отриманий термін окупності у 1,8 року є відмінним показником для сфери розробки програмного

забезпечення, де середньостатистичний цикл повернення інвестицій для подібних утиліт зазвичай становить 2–2,5 роки. Швидкий обіг коштів мінімізує інвестиційні ризики та дозволяє швидше перейти до етапу отримання чистого прибутку, який може бути реінвестований у подальший розвиток функціонала системи, наприклад, у розробку мобільної версії або хмарного модуля аналітики.

Проведений технологічний аудит засвідчив, що проєкт має комерційний потенціал вище середнього. Порівняння з існуючим аналогом показало, що нова розробка відзначається вищою якістю та кращими техніко-економічними характеристиками, що робить її більш конкурентоспроможною.

Інвестиції, вкладені в проєкт, повернуться приблизно за 1,8 року. Загальні витрати становлять 154573,26 грн, а прогнозований трирічний прибуток — 719935,01 грн.

ВИСНОВКИ

У результаті проведеної роботи було вирішено актуальне науково-практичне завдання моніторингу гетерогенних мереж IoT. Головним здобутком роботи стала розробка та обґрунтування комплексного методу дворівневого моніторингу, який, на відміну від існуючих підходів, поєднує перевірку мережевої доступності (рівень L3) з аналізом стану сервісів (рівень L4). Запропонований підхід базується на концепції "еталонів" сервісів (фінгерпринтингу), що дозволяє автоматично виявляти приховані відмови та безпекові аномалії без необхідності встановлення агентів на пристрої.

Практична реалізація цього методу здійснена шляхом створення повноцінного програмного засобу — настільного додатка "Мережевий Монітор IoT". Процес розробки відбувався мовою програмування Python, що дозволило ефективно імплементувати алгоритми методу. Зокрема, використання бібліотеки Scapy забезпечило реалізацію механізмів L3-діагностики, а застосування модуля Socket та багатопотоковості дозволило втілити алгоритм швидкого паралельного сканування для порівняння поточного стану пристрою з його "еталоном".

Розробка включала створення модульної архітектури, де логіка методу (аналіз відхилень) чітко відокремлена від інтерфейсу користувача. Завдяки використанню бібліотеки CustomTkinter створено зручний графічний інтерфейс, а інтеграція з Matplotlib дозволила візуалізувати динаміку якості зв'язку, що є важливою складовою запропонованої методики.

Отже, розроблений метод та програмні засоби продемонстрували високу ефективність у вирішенні проблеми "ілюзії працездатності" IoT-пристроїв. Робота над проектом дозволила не лише отримати цінний досвід у розробці системного ПЗ, але й підтвердити гіпотезу про те, що аналіз відхилень від "еталону" є найбільш дієвим способом моніторингу "чорних скриньок" у локальних мережах. Дослідження заклало основу для розвитку методів адаптивного моніторингу та підвищення надійності сучасних "розумних" мереж.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Богомолов С. В., Атаманюк Н. Р. МЕТОД ТА ЗАСОБИ МОНІТОРИНГУ IoT-ПРИСТРОЇВ У ЛОКАЛЬНИХ МЕРЕЖАХ ІЗ ВИЯВЛЕННЯМ ВІДМОВ. Інформаційні технології: моделі, алгоритми, системи — 2025: VI Міжнародна науково-практична інтернет-конференція (2025). Режим доступу: https://itconf.nuos.edu.ua/2025/publications/%D0%B0-method-and-toolset-for-monitoring-iot-devices-in-local-networks-with-failure-detection/

2. Kurose James F. Computer Networking: A Top-Down Approach / James F. Kurose, Keith W. Ross. — 2021. — 856 p.

3. Mauro Douglas. Essential SNMP / Douglas Mauro, Kevin Schmidt. — 2005. — 462 p.

4. Gupta Aditya. The IoT Hacker's Handbook: A Practical Guide to Hacking the Internet of Things / Aditya Gupta. — 2019. — 355 p.

5. Tanenbaum Andrew S. Computer Networks / Andrew S. Tanenbaum, Nick Feamster, David J. Wetherall. — 2021. — 944 p.

6. RFC 792: Internet Control Message Protocol (ICMP) [Електронний ресурс] — Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc792

7. RFC 793: Transmission Control Protocol (TCP) [Електронний ресурс] — Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc793

8. Zabbix Documentation 6.0 [Електронний ресурс] — Режим доступу до ресурсу: https://www.zabbix.com/documentation/6.0/en/

9. Bejtlich Richard. The Practice of Network Security Monitoring: Understanding Incident Detection and Response / Richard Bejtlich. — 2013. — 376 p.

10. Lutz Mark. Learning Python / Mark Lutz. — 2013. — 1600 p.

11. Scapy Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://scapy.net/

12. Rhodes Brandon. Foundations of Python Network Programming / Brandon Rhodes, John Goerzen. — 2014. — 352 p.

13. CustomTkinter Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://github.com/TomSchimansky/CustomTkinter

14. Hunter John D. Matplotlib: A 2D graphics environment / John D. Hunter. — 2007. — 95 p.

15. The JSON Data Interchange Syntax (ECMA-404) [Електронний ресурс] — Режим доступу до ресурсу: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/

16. Threading — Thread-based parallelism [Електронний ресурс] — Режим доступу до ресурсу: https://docs.python.org/3/library/threading.html

17. RFC 826: An Ethernet Address Resolution Protocol (ARP) [Електронний ресурс] — Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc826

18. Козловський В. О. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / В. О. Козловський, О. Й. Лесько, В. В. Кавецький. — 2021. — 42 с.

19. Методичні вказівки до виконання магістерських кваліфікаційних робіт студентами спеціальності 123 «Комп'ютерна інженерія». / Укладачі О. Д. Азаров, О. В. Дудник, С. І. Швець − Вінниця : ВНТУ, 2023. − 57 с.

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н. Азаров О.Д.
_____
"_3_" _жовтня_ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи
"Метод та засоби моніторингу IoT-пристроїв у локальних мережах із виявленням відмов"

Науковий керівник: доцент к.т.н.
_____ Богомолов С. В.
Студент групи 1КІ-24м
_____ Атаманюк Н. Р.

ВНТУ 2025

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Важливим є актуальність дослідження у напрямку магістерської роботи, яка обумовлена тим, що в даний час все більше учасників ринку, таких як системні адміністратори, спеціалісти з кібербезпеки та звичайні користувачі, стикаються із завданням моніторингу локальних мереж. Це пов'язано зі стрімким зростанням кількості пристроїв "Інтернету речей", які часто є "чорними скриньками".Поряд з такими «базовими» завданнями, як контроль мережевої доступності, все більший інтерес викликають і більш складні — виявлення прихованих відмов окремих сервісів або фіксація несанкціонованих змін.

1.2 Наказ про затвердження теми МКР № 313 від 24.09.2025 року.

2 Мета МКР і призначення розробки

2.1 Мета роботи — аналіз сучасного стану і тенденцій розвитку засобів моніторингу IoT-пристроїв у локальних мережах та розробка програмного комплексу для виконання задач з виявлення мережевих (L3) та сервісних (L4) відмов шляхом порівняння поточного стану пристрою з попередньо збереженим "еталоном" сервісів.

2.2 Призначення розробки — програмний засіб моніторингу IoT-пристроїв, призначений для автоматичного виявлення мережевих (L3) та сервісних (L4) відмов, діагностики якості зв'язку, а також фіксації безпекових аномалій (появи несанкціонованих сервісів) з веденням журналу подій у реальному часі.

3 Вихідні дані для виконання МКР

3.1 Проведення аналізу існуючих принципів та технологій моніторингу IoT-пристроїв.

3.2 Розробка структури та функціональної схеми програмного комплексу для моніторингу IoT-пристроїв.

3.3 Розробка та реалізація програмних модулів сканування та аналізу.

3.4 Виконання розрахунків для доведення доцільності нової розробки з економічної точки зору.

3.5 Тестування методу та засобів моніторингу IoT-пристроїв у локальних мережах із виявленням відмов.

4 Вимоги до виконання МКР

Головна вимога — розробка методу дворівневого моніторингу L3/L4 шляхом порівняння поточного стану сервісів пристрою, а також аналізу якості мережевого зв'язку.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

№ з/п	Назва етапу	Термін (початок)	Термін (кінець)	Очікувані результати
1	Огляд і аналіз існуючих методів та засобів моніторингу IoT-пристроїв у локальних мережах	11.09.2025	17.09.2025	Аналітичний огляд джерел, Розділ 1
2	Дослідження та розробка методології дворівневого моніторингу L3/L4	18.09.2025	20.09.2025	Розділ 2
3	Розробка структурної та функціональної схем програмного комплексу та його реалізація	20.09.2025	03.10.2025	Розділ 3, Програмний код
4	Тестування програмного засобу	04.10.2025	12.10.2025	Розділ 4
5	Підготовка економічного обґрунтування розробки	13.10.2025	15.10.2025	Розділ 5
6	Публікація проміжних результатів дослідження	16.10.2025	16.10.2025	Тези доповіді
7	Оформлення пояснювальної записки, графічного матеріалу та презентації	17.10.2025	25.10.2025	ПЗ, графічний матеріал, презентація
8	Підготовка та підпис супроводжуючих документів, нормоконтроль та тест на текстові запозичення	26.10.2025	03.11.2025	Оформлені документи

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлюванні МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Едина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документи на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

Протокол перевірки кваліфікаційної роботи

Назва роботи: Метод та засоби моніторингу IoT-пристроїв у локальних мережах із виявленням відмов.

Тип роботи: магістерська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ : кафедра обчислювальної техніки, ФІТКІ, 1КІ – 24м
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) 1 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

Азаров О. Д., д.т.н., зав. каф. ОТ _____
(прізвище, ініціали, посада) (підпис)

Мартинюк Т. Б., д.т.н., проф. каф. ОТ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Захарченко С. М.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Богомолов С. В., к.т.н. доц. каф. ОТ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Атаманюк Н. Р.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Графічний матеріал

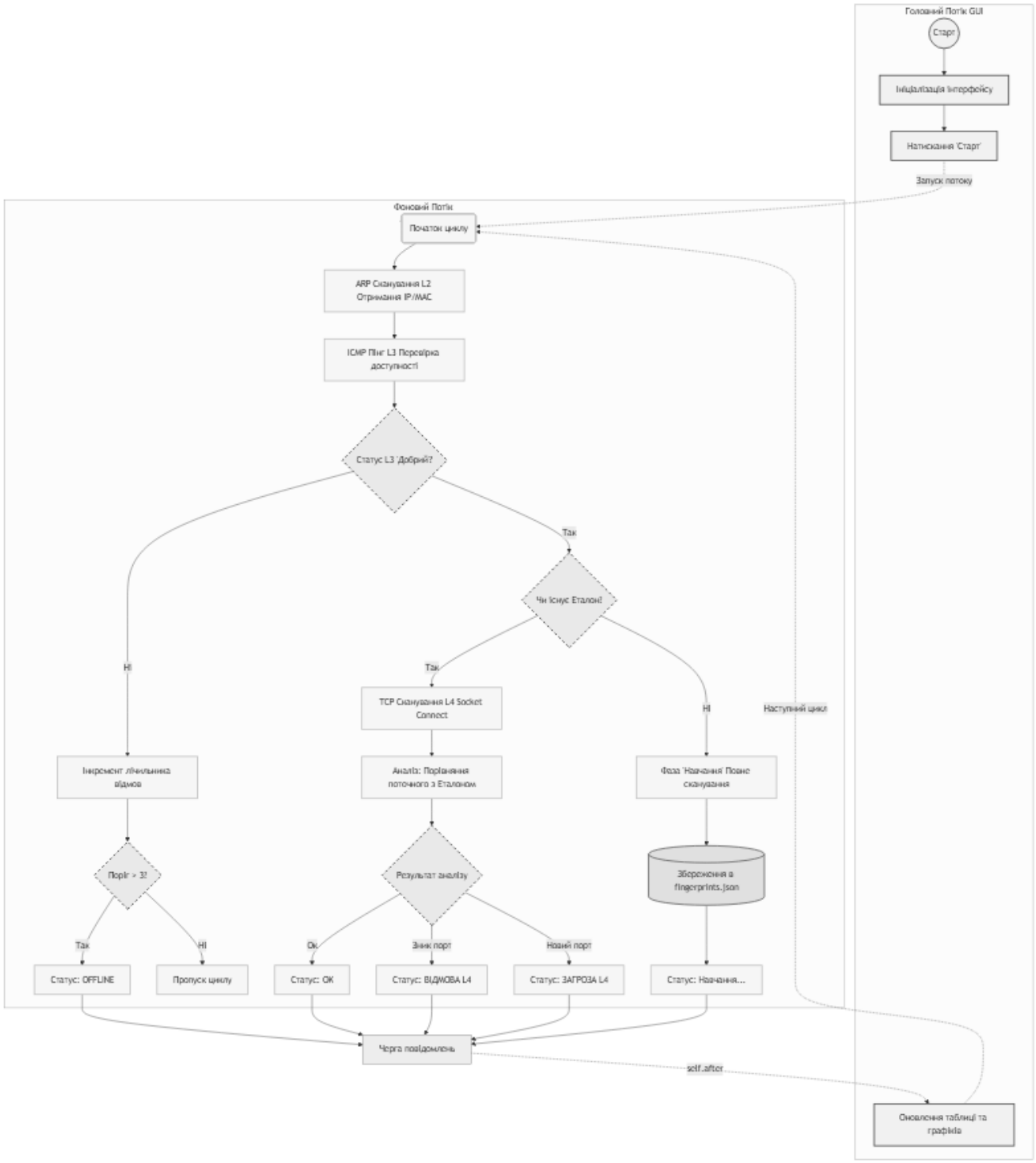

Рисунок В.1 — Функціональна схема програмного комплексу

ДОДАТОК Г

Лістинг реалізації методу

```
import socket
import threading
import time
import json
import psutil
import ipaddress
from concurrent.futures import ThreadPoolExecutor
import scapy.all as sc
import customtkinter as ctk
from tkinter import ttk
COMMON_PORTS = [21, 22, 23, 25, 53, 80, 110, 139, 443, 445, 554, 8080]
class App(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("Мережевий Монітор IoT")
        self.geometry("1100x600")
        self.device_states = {}
        self.device_fingerprints = {}
        self.monitoring_active = False
        self.stop_event = threading.Event()
        self.load_fingerprints()
        self.setup_ui()
    def get_active_network_info(self):
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            s.settimeout(0.1)
            s.connect(("8.8.8.8", 80))
            ip_addr = s.getsockname()[0]
            s.close()
            all_ifaces = psutil.net_if_addrs()
            for iface_name, addrs in all_ifaces.items():
                for addr in addrs:
                    if addr.family == socket.AF_INET and addr.address == ip_addr:
                        netmask = addr.netmask
                        network = ipaddress.ip_network(f"{ip_addr}/{netmask}", strict=False)
                        return str(network), iface_name
            return "192.168.1.0/24", None
        except Exception:
            return "192.168.1.0/24", None
    def scan_network_arp(self):
```

```
        target_ip, iface_name = self.get_active_network_info()
        arp_request = sc.ARP(pdst=target_ip)
        broadcast = sc.Ether(dst="ff:ff:ff:ff:ff:ff")
        packet = broadcast/arp_request
        devices_online = []
        try:
            if iface_name:
                sc.conf.iface = iface_name
            result = sc.srp(packet, timeout=2, verbose=0)[0]
            for sent, received in result:
                devices_online.append({'ip': received.psrc, 'mac': received.hwsrc})
        except Exception:
            pass
        return devices_online
    def check_connection(self, ip_address, num_packets=4, timeout=0.5):
        try:
            replies, no_replies = sc.sr(
                sc.IP(dst=ip_address)/sc.ICMP(),
                timeout=timeout,
                retry=num_packets-1,
                verbose=0
            )
            packet_loss = len(no_replies) / (len(replies) + len(no_replies)) * 100
            if not replies:
                return {"latency_ms": None, "packet_loss_percent": 100}
            total_latency = sum((rec.time - sent.time) for sent, rec in replies) * 1000
            avg_latency = total_latency / len(replies)
            return {"latency_ms": round(avg_latency), "packet_loss_percent":
round(packet_loss)}
        except Exception:
            return {"latency_ms": None, "packet_loss_percent": 100}
    def scan_ports_socket(self, ip, ports, timeout=0.5):
        open_ports = []
        def check_port(ip, port, timeout):
            try:
                with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                    s.settimeout(timeout)
                    if s.connect_ex((ip, port)) == 0:
                        return port
            except (socket.timeout, socket.error):
                return None
        with ThreadPoolExecutor(max_workers=20) as executor:
            futures = [executor.submit(check_port, ip, port, timeout) for port in ports]
            for future in futures:
```

```
            result = future.result()
            if result is not None:
                open_ports.append(result)
        return open_ports
    def toggle_monitoring(self):
        if self.monitoring_active:
            self.monitoring_active = False
            self.stop_event.set()
            self.toggle_button.configure(text="Старт")
        else:
            self.monitoring_active = True
            self.stop_event.clear()
            self.toggle_button.configure(text="Стоп")
            threading.Thread(target=self.monitoring_loop, daemon=True).start()
    def monitoring_loop(self):
        while self.monitoring_active:
            online_devices = self.scan_network_arp()
            online_macs = [d['mac'] for d in online_devices]
            for device in online_devices:
                ip = device['ip']
                mac = device['mac']
                if mac not in self.device_states:
                    self.device_states[mac] = {
                        "ip": ip,
                        "status_l3": "Невідомо",
                        "status_l4": "...",
                        "failure_count": 0
                    }
                state = self.device_states[mac]
                state["ip"] = ip
                l3_stats = self.check_connection(ip)
                if l3_stats["packet_loss_percent"] == 100:
                    state["failure_count"] += 1
                else:
                    state["failure_count"] = 0
                    state["status_l3"] = "Добрий"
                    if mac not in self.device_fingerprints:
                        self.device_fingerprints[mac] = []
                        threading.Thread(target=self.learn_device, args=(ip, mac),
daemon=True).start()
                        state["status_l4"] = "Навчання..."
                    else:
                        etalon_ports = self.device_fingerprints[mac]
                        ports_to_check = list(set(etalon_ports + COMMON_PORTS))
```

```
                current_ports = self.scan_ports_socket(ip, ports_to_check)

                new_dangerous = [p for p in current_ports if p not in etalon_ports and
p in COMMON_PORTS]
                closed_etalon = [p for p in etalon_ports if p not in current_ports]
                if closed_etalon:
                    state["status_l4"] = "Відмова"
                    state["status_overall"] = "Відмова L4"
                elif new_dangerous:
                    state["status_l4"] = "Загроза"
                    state["status_overall"] = "Загроза L4"
                else:
                    state["status_l4"] = "OK"
                    state["status_overall"] = "Добрий"
            for mac, state in self.device_states.items():
                if mac not in online_macs:
                    state["failure_count"] += 1
                if state["failure_count"] >= 3:
                    state["status_l3"] = "Offline"
                    state["status_overall"] = "Offline"
            self.after(0, self.update_results_ui)
            time.sleep(15)
    def learn_device(self, ip, mac):
        scan_list = list(range(1, 1024)) + [8080, 8081, 8000]
        open_ports = self.scan_ports_socket(ip, scan_list, timeout=1.0)
        self.device_fingerprints[mac] = open_ports
        self.save_fingerprints()

    def update_results_ui(self):
        for row in self.tree.get_children():
            self.tree.delete(row)
        active_devices = [
            {"mac": m, **s} for m, s in self.device_states.items()
            if s.get("status_overall") != "Offline" or s.get("failure_count") < 3
        ]
        for device in active_devices:
            status_overall = device.get('status_overall', 'Good')
            status_tag = "good"
            if status_overall == 'Загроза L4': status_tag = "l4_threat"
            elif status_overall == 'Відмова L4': status_tag = "l4_fail"
            elif status_overall == 'Offline': status_tag = "bad"
            values = (
                device.get('ip'),
                device.get('mac'),
```

```
                device.get('status_l3'),
                device.get('status_l4')
            )
            self.tree.insert("", "end", values=values, tags=(status_tag,))
    def setup_ui(self):
        self.controls_frame = ctk.CTkFrame(self)
        self.controls_frame.pack(fill="x", padx=10, pady=10)

        self.toggle_button = ctk.CTkButton(self.controls_frame, text="Старт",
command=self.toggle_monitoring)
        self.toggle_button.pack(side="left", padx=10)
        self.tree_frame = ctk.CTkFrame(self)
        self.tree_frame.pack(fill="both", expand=True, padx=10, pady=10)
        cols = ("ip", "mac", "status_l3", "status_l4")
        self.tree = ttk.Treeview(self.tree_frame, columns=cols, show="headings")
    def load_fingerprints(self):
        try:
            with open("fingerprints.json", "r") as f:
                self.device_fingerprints = json.load(f)
        except FileNotFoundError:
            self.device_fingerprints = {}

    def save_fingerprints(self):
        with open("fingerprints.json", "w") as f:
            json.dump(self.device_fingerprints, f)
if __name__ == "__main__":
    app = App()
    app.mainloop()
```