МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

**КОЛОДЮК Дмитро Олексійович**

# Програмно-апаратний модуль мережевої телеметрії та виявлення аномалій / Software–Hardware Module for Network Telemetry and Anomaly Detection

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-41
Д. О. Колодюк

Науковий керівник:
к.т.н., доцент, О.Р. Осолінський

Кваліфікаційну роботу допущено до захисту
«___»_____ 2026 р.

Завідувач кафедри
_____ Н. В. Дзюбановська

**Тернопіль – 2026**

**Факультет комп'ютерних інформаційних технологій**
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.В. Дзюбановська
«____» _____20 р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**
**КОЛОДЮКУ Дмитру Олексійовичу**
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Програмно-апаратний модуль мережевої телеметрії та виявлення аномалій / Software–Hardware Module for Network Telemetry and Anomaly Detection
керівник роботи О.Р. Осолінський, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 01 грудня 2025 року № 894.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
- провести аналіз предметної області мережевої телеметрі;
- провести огляд і аналіз існуючих рішень аналогів;
- сформувати постановку задачі та вимоги до модуля;
- розробити архітектуру програмно апаратного рішення;
- розробити алгоритмічне забезпечення модуля;
- розробити інформаційне забезпечення;
- реалізувати програмне забезпечення модуля;
- розробити інтерфейс користувача та засоби представлення результатів;
- провести тестування програмно – апаратного модуля.

5. Перелік графічного матеріалу в роботі:
- архітектура програмно-апаратного модуля;
- Загальний алгоритм функціонування програмно-апаратного модуля мережевої телеметрії та виявлення аномалій.

6. Консультанти розділів кваліфікаційної роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 01 грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів кваліфікаційної роботи | Примітка |
|---|---|---|---|
| 1 | Огляд літературних джерел відповідно до предметної області та складання плану роботи | до 01.01.2026 р. | |
| 2 | Написання 1 розділу кваліфікаційної роботи | до 01.02.2026 р. | |
| 3 | Написання 2 розділу кваліфікаційної роботи | до 15.03.2026 р. | |
| 4 | Написання 3 розділу кваліфікаційної роботи | до 01.05.2026 р. | |
| 5 | Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником | до 15.05.2026 р. | |
| 6 | Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів | до 25.05.2026 р. | |
| 7 | Перевірка кваліфікаційної роботи на оригінальність тексту | до 30.05.2026 р. | |
| 8 | Оформлення кваліфікаційної роботи та отримання допуску до захисту | до 10.06.2026 р. | |
| 9 | Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії | до 10.06.2026 р. | |

Студент_____ Д. О. Колодюк
( підпис ) (прізвище та ініціали)

Керівник кваліфікаційної роботи_____ О.Р. Осолінський
( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи: 53 сторінки, 14 ілюстрацій, 3 таблиці, 4 додатки, 30 використаних джерел.

Метою кваліфікаційної роботи є розробка програмно-апаратного модуля мережевої телеметрії та виявлення аномалій на основі flow-даних, який забезпечує збір, агрегацію, аналіз, збереження результатів і подання їх оператору в зручному вигляді.

В роботі використано методи системного та структурно-функціонального аналізу, логіко-алгоритмічного моделювання, статистичної обробки даних у часових вікнах, побудови ознак, а також підходи машинного навчання для безнаглядного виявлення аномалій. Реалізовано конвеєр обробки flow-даних з накопиченням в базі даних, двома детекторами аномалій на основі базових правил і модуля машинного навчання, механізмом об’єднання повторних подій у інциденти та керуванням сповіщеннями з паузою між повторними повідомленнями. Працездатність модуля перевірено на контрольованих сценаріях, зокрема для нормального трафіку та сканування портів; у тестовому запуску базовий детектор сформував 5 подій, модуль машинного навчання сформував 2 події, після об’єднання повторів отримано 3 інциденти і 3 сповіщення.

Практичне значення роботи полягає у створенні прототипу, який може бути використаний як основа для моніторингу невеликих мереж, навчальних стендів і лабораторних досліджень.

Ключові слова: МЕРЕЖЕВА ТЕЛЕМЕТРІЯ, FLOW-ТЕЛЕМЕТРІЯ, NETFLOW, IPFIX, SFLOW, АГРЕГАЦІЯ У ЧАСОВИХ ВІКНАХ, ОЗНАКИ ТРАФІКУ, ЕНТРОПІЯ ПОРТІВ, ВИЯВЛЕННЯ АНОМАЛІЙ, ІНЦИДЕНТИ, СПОВІЩЕННЯ, ВЕБ-ІНТЕРФЕЙС.

# ANNOTATION

Explanatory note to the qualification thesis: 53 pages, 14 figures, 3 tables, 4 annexes, 30 references.

The aim of this work is to develop a software–hardware module for network telemetry and anomaly detection based on flow data that supports collection, time-window aggregation, analysis, persistent storage of results, and operator-friendly presentation. A distributed architecture is proposed in which the collection node performs flow reception and aggregation, while the analytics node computes features, detects deviations, and produces events, incidents, and alerts.

The work employs methods of system and structural–functional analysis, logical and algorithmic modeling, statistical processing in time windows, feature extraction, and unsupervised machine-learning approaches for anomaly detection. A processing pipeline is implemented with database storage, two complementary detectors based on baseline rules and a machine-learning module, incident consolidation to merge repeated events, and alert throttling using a cooldown interval. The prototype is tested on controlled scenarios including normal traffic and port scanning; during the test run the baseline detector produced 5 events, the machine-learning module produced 2 events, and the consolidation stage resulted in 3 incidents and 3 alerts.

The practical value of the work is a functioning prototype suitable for monitoring small networks and laboratory testbeds.

Keywords: NETWORK TELEMETRY, FLOW TELEMETRY, NETFLOW, IPFIX, SFLOW, TIME-WINDOW AGGREGATION, TRAFFIC FEATURES, PORT ENTROPY, ANOMALY DETECTION, INCIDENTS, ALERTS, WEB INTERFACE.

ЗМІСТ

# ВСТУП

Сучасні комп’ютерні мережі стали основою роботи підприємств, навчальних закладів, сервісів електронної комерції та хмарних платформ. Зростання кількості цифрових сервісів, дистанційної роботи та мережевих пристроїв призводить до того, що мережа працює в режимі постійного навантаження і часто змінює свій профіль трафіку протягом дня. Паралельно з цим збільшується кількість інцидентів інформаційної безпеки, а також випадків деградації якості сервісу через помилки конфігурації, перевантаження або відмови обладнання. В таких умовах для підтримки стабільної роботи сервісів потрібен безперервний моніторинг мережі та своєчасне виявлення відхилень у поведінці трафіку.

Традиційний спосіб аналізу мережі на основі повного захоплення пакетів часто є надто дорогим з точки зору зберігання та обробки даних, особливо, якщо потрібна тривала історія. Тому на практиці широко використовується підхід flow телеметрії, де мережеві взаємодії описуються компактними записами про потоки. До таких підходів належать NetFlow, IPFIX та sFlow, які дозволяють отримувати узагальнену інформацію про те, хто з ким взаємодіє, який обсяг трафіку передається і як змінюється картина в часі. Flow телеметрія дає змогу будувати часові ряди метрик та ознак, на основі яких можна виявляти аномалії, такі як сканування, різкі сплески трафіку, нетипові напрямки комунікацій або прояви DoS атак.

Під час побудови системи на основі flow даних виникає ряд проблем. По перше, потрібно забезпечити стабільний прийом потокових записів і зменшити втрати через UDP, буфери та обмеження ресурсів на вузлі збору. По друге, потрібна швидка агрегація за часовими вікнами і формування ознак, щоб детектор працював у близькому до реального часу режимі. По третє, необхідно мінімізувати хибні спрацювання, оскільки надмірні повідомлення швидко знижують довіру оператора до системи. По четверте, важливо мати зручне збереження і візуалізацію, щоб оператор міг не лише отримати сповіщення, а й швидко зрозуміти причину події та контекст.

Тому доцільно розробити програмно-апаратний модуль, який в одному рішенні поєднує збір потокової телеметрії, її аналіз і формування сповіщень для оператора. В цій роботі розглядається система, де Raspberry Pi використовується як колектор та агрегатор, що працює поруч із мережевим обладнанням і виконує прийом, нормалізацію та первинне зведення потоків. Аналітичний вузол реалізується на персональному комп'ютері з GPU, де доступне прискорення CUDA для ресурсоємних обчислень, зокрема для обробки ознак та роботи моделей машинного навчання. Такий поділ завдань зменшує навантаження на периферійний пристрій і дає змогу швидше обробляти дані на центральному вузлі, що важливо для своєчасного реагування.

Метою кваліфікаційної роботи є розробка програмно-апаратного модуля мережевої телеметрії та виявлення аномалій на основі flow даних, який забезпечує збір, агрегацію, аналіз, збереження результатів і подання їх оператору у зручному вигляді.

Для досягнення поставленої мети необхідно виконати такі завдання:

– провести аналіз предметної області мережевої телеметрії та підходів flow-обліку, а також визначити вимоги до систем телеметрії за показниками швидкодії, затримок, втрат і обсягу зберігання;

– виконати огляд і аналіз існуючих рішень для збору та обробки flow-даних та підходів до виявлення аномалій у мережах;

– сформувати постановку задачі та вимоги до програмно-апаратного модуля;

– розробити архітектуру програмно-апаратного рішення для збору flow-даних, агрегації на вузлі збору та аналітики на обчислювальному вузлі;

– розробити алгоритмічне забезпечення модуля;

– розробити інформаційне забезпечення, визначити вхідні та вихідні дані, побудувати концептуальну і логічну модель даних;

– виконати вибір програмних засобів і технологій реалізації модулів збору, зберігання, аналітики та веб-інтерфейсу;

– реалізувати ключові модулі системи;

– реалізувати інтерфейс користувача у вигляді веб-дашборду та API для перегляду інцидентів, подій, агрегатів і сповіщень;

– провести тестування працездатності модуля на контрольованих сценаріях.

Об’єкт дослідження — процеси збору, агрегації та аналізу мережевої телеметрії у вигляді записів про потоки в комп’ютерних мережах.

Предмет дослдження — методи та засоби, виявлення аномалій, формування подій і сповіщень в мережі.

В роботі використано системний аналіз предметної області, проєктування модульної архітектури, статистичну обробку даних в часових вікнах, формування ознак, застосування правил і підходів машинного навчання для виявлення відхилень.

Результати кваліфікаційної роботи апробовані та опубліковані в матеріалах XI Міжнародної наукової конференції «Традиційні та інноваційні підходи до наукових досліджень» (м. Луцьк, Україна).

Практичне значення роботи полягає у створенні працездатного прототипу, який може бути використаний як основа для моніторингу невеликих мереж, навчальних стендів і лабораторних досліджень.

Структура роботи. Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел та додатків.

# 1 СТАН І АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Опис предметної області мережевої телеметрії

Мережева телеметрія дає змогу відстежувати стан мережі та своєчасно виявляти відхилення в її роботі. В найпростішому варіанті телеметрія зводиться до лічильників на інтерфейсах і базових графіків навантаження. Але для задач безпеки та діагностики цього часто замало, тому що важливо розуміти не тільки скільки трафіку пройшло, а й між ким і ким він ходив, якими протоколами, якими портами, як довго тривали з’єднання і як змінювалася картина в часі. Саме для цього використовується flow телеметрія, тобто збір даних у вигляді записів про потоки. Такий підхід широко застосовується в практичних системах моніторингу і реагування, тому що дає хороший баланс між обсягом даних і корисною інформацією для аналізу [1, 2].

Перевага потокового підходу полягає в тому, що він поєднує помірний обсяг даних із достатньою інформативністю для аналізу. З одного боку він значно легший за зберігання повних дампів пакетів, тому підходить для тривалого збору і для мереж з великим обсягом трафіку. З іншого боку він містить більше змісту, ніж прості лічильники, бо дозволяє будувати картину взаємодій в мережі. На практиці потокові дані зазвичай перетворюють на часові ряди метрик, наприклад інтенсивність байтів і пакетів, кількість потоків, кількість унікальних адрес у вікні, структуру портів і протоколів. На таких часових рядах зручно будувати правила і моделі детекції аномалій, як це показано в рішеннях реального часу на основі flow даних [3] (рисунок 1.1).

Потік в контексті мережевої телеметрії це група пакетів, які належать до однієї і тієї ж взаємодії. Найпоширеніше визначення потоку спирається на п’яти-польний ключ, який включає адресу джерела, адресу призначення, порт джерела, порт призначення і транспортний протокол. В документах стандартів NetFlow та IPFIX ця ідея використовується як базова для ідентифікації комунікації між вузлами [1, 2]. В окремих реалізаціях до ключа додають інші ознаки, наприклад інтерфейс входу або виходу, клас сервісу DSCP, VLAN або next hop. Такі поля

потрібні для локалізації, де саме в мережі виникла проблема, або відстежити, як трафік маршрутизується [4]


Рисунок - 1.1 Відношення між пакетами, потоками, записами і часовими рядами.

Запис потоку в більшості містить дві основні групи даних. Перша група описує, хто з ким спілкувався, тобто адреси, порти і протокол. Друга група описує, як саме відбувався обмін, тобто кількість пакетів, кількість байтів, часові межі або тривалість. В IPFIX це формалізується через інформаційні елементи, які мають однозначні назви і типи, наприклад octetDeltaCount, packetDeltaCount, flowStartMilliseconds, flowEndMilliseconds

Для TCP корисними є контрольні біти, які відображають прапори SYN, ACK, RST та інші. Підходи детекції на основі таких ознак використовуються в практичних системах аналізу потоків та у високошвидкісних flow IDS рішеннях. [3, 5-7]

Важливо враховувати, що запис потоку формується експортером за власними правилами та часовими обмеженнями. Експортер тримає потік відкритим, поки бачить активність, а потім закриває його за таймаутом бездіяльності або за таймаутом активності. Через це один довгий сеанс може бути розбитий на декілька записів, а короткі події можуть зникати при агресивному sampling. Такі практичні нюанси розглядаються у матеріалах з переходу від пакетів до flow і далі до аналізу,

де підкреслюється вплив параметрів експорту та підготовки даних на якість результату.

У потоковій телеметрії найчастіше використовують три основні формати. Це NetFlow, IPFIX і sFlow. Вони схожі тим, що дають структуровані дані для аналізу, але різняться підходом і наслідками для точності та навантаження.

Ключова ідея NetFlow версії 9 [1] полягає в тому, що формат записів задається шаблонами. Спочатку експортер надсилає шаблон в якому перелічені поля і їх порядок, а далі надсилає дані, які відповідають цьому шаблону. Шаблонний підхід зручний, бо дозволяє додавати нові поля без зміни базового протоколу. Для системи виявлення аномалій це важливо томущо деколи потрібні додаткові поля, наприклад TCP прапори, інтерфейси або додаткові часові мітки [1-7].

IPFIX є стандартизованим розвитком підходу NetFlow v9. Він також використовує шаблони і розділяє ролі на експортер і колектор. IPFIX задає не тільки протокол, а і інформаційну модель полів [4].

Повний перелік елементів підтримує IANA у вигляді реєстру, де наведені назви, ідентифікатори і типи, що дозволяє однозначно інтерпретувати значення полів у конвеєрі обробки [8]. Це потрібно, коли система повинна приймати дані з різних джерел і все одно будувати однакові метрики.

Особливість sFlow [9] полягає в орієнтації на вибірковий збір даних. Замість того, щоб експортувати кожен потік повністю, sFlow передає вибірки пакетів і окремо лічильники інтерфейсів. Перевага sFlow у тому, що він добре масштабується на високих швидкостях, бо обсяг даних керується коефіцієнтом вибірки. Недолік у тому, що детекція дрібних подій стає складнішою, а втрати UDP датаграм фактично зменшують ефективну кількість семплів і погіршують статистичну точність.

Практичні рішення часто використовують також споріднені джерела даних, які за змістом близькі до flow. Наприклад в [10] формується журнал з’єднань conn log, який містить ключові поля сеансу і метрики тривалості та обсягу, і це використовується для побудови ознак і моделей аномалій. Аналогічно проект Argus

[11] створює записи на основі трафіку і дає набір полів, близький до семантики flow, що теж можна застосувати у задачах моніторингу.

Записи потоків зручні для аналізу, однак у великих мережах їх кількість швидко збільшується. Тому типовий підхід полягає у переході від сирих записів до агрегованих метрик. Агрегація означає, що за певний інтервал часу збираються статистики по ключах, наприклад по адресі джерела, по адресі призначення або по порту. В RFC 7015 потоки можуть об’єднуватися за правилами, щоб зменшити обсяг і зосередитися на потрібних метриках [12].

Агрегування також спрощує подальший аналіз, оскільки потоки перетворюються на часові ряди, придатні для роботи детектора [3].

В [13] підкреслюється, що перехід від пакетів до flow і далі до аналізу складається з кількох кроків і кожен крок може впливати на якість, тобто якщо десь виникає пропуск або некоректний час, то часові ряди виходять спотвореними, і детектор починає помилятися. Через це завжди враховують синхронізацію часу, контроль цілісності, а також стабільність конвеєра обробки.

## 1.2 Огляд і аналіз існуючих рішень аналогів

Більшість розглянутих підходів мають подібну логіку обробки даних. Спочатку в мережі формується потік даних про трафік у вигляді flow записів або близьких за змістом подій. Далі ці записи приймає колектор, зберігає їх і за потреби агрегує. Після цього дані або одразу аналізуються правилами і моделями, або спочатку потрапляють у сховище і візуалізацію, де оператор бачить картину, а детектор працює як окремий модуль.

Одним із найпоширеніших класів рішень є інструменти, що приймають NetFlow та IPFIX і зберігають ці дані у файлах для подальшого аналізу. Прикладом такого підходу є nfcapd, який є демоном захоплення потоку записів і зберігає дані у файлах з автоматичною ротацією за часом [14]. Така модель зручна для лабораторних стендів і невеликих інсталяцій, оскільки її легко перевіряти та відтворювати. Дані лежать у файловій структурі, їх легко переносити, архівувати і

повторно проганяти через аналізатор. Додатково цей підхід добре працює тоді, коли потрібно мати історію для досліджень, а не тільки миттєві графіки. В nfcapd зазначено, що підтримуються кілька версій NetFlow і також IPFIX, а ще описано важливий практичний момент із sampling. Якщо пристрій експортує sampling, то інструмент може автоматично враховувати це при підрахунках байтів і пакетів.

Окремий клас рішень це програмні експортери, які можуть створювати NetFlow або IPFIX не на маршрутизаторі, а на звичайному комп'ютері чи на edge вузлі. Прикладом є softflowd [15], який читає трафік із мережевого інтерфейсу або з файлів pcap і будує потоки як комунікацію між IP адресами, а для TCP і UDP ще й на рівні кортежів адреса порт. Після цього softflowd може експортувати дані в кількох версіях NetFlow та у IPFIX.

Ще один практичний підхід до формування потоків це використання спеціалізованих flowmeter рішень, які споживають пакети і випускають потоки в стандартизованому вигляді. В наборі файлів таким рішенням є YAF [16]. Він споживає пакети і перетворює їх в потоки, щоб нижчі за конвеєром інструменти могли робити ситуаційну обізнаність по мережі. YAF випускає записи в форматі IPFIX і всі поля описуються інформаційними елементами IPFIX з фіксованими типами. Це підсилює сумісність з подальшими сховищами та аналітикою. Також YAF підтримує збагачення потоків додатковими полями і метаданими глибшого аналізу бо дає більше ознак для детекції.

Для великих інсталяцій, де важливо мати швидкі запити по історії та інструменти для аналітика, часто використовують SiLK [17] - це набір командних інструментів для обробки flow записів, які створюються системою пакування, а центральним елементом аналізу є rwfilter для запитів до репозиторію. Це показує інший підхід порівняно з простим складанням файлів. В SiLK є чіткий акцент на те, що дані організуються як централізований репозиторій, а аналіз будується комбінацією утиліт і конвеєрів через UNIX pipes. Перевагою такого рішення це можливість робити складні вибірки та групування без написання власного коду.

Якщо порівнювати ці підходи на рівні ролей, то nfcapd зручний як приймач і архіватор потоку даних, softflowd зручний як джерело потоків у софт середовищі,

YAF зручний як потужний flowmeter з можливістю збагачення IPFIX записів, а SiLK зручний як аналітичний шар з репозиторієм і утилітами запитів

Окремо також треба розглянути питання зберігання даних і подання результатів оператору. В цьому випадку дані доцільно поділити на два типи. Перший вид це сирі flow записи або збагачені потоки, які є подіями про комунікації. Другий вид це агреговані метрики та ознаки, які вже нагадують часові ряди. В більшості сирі потоки зберігають обмежений час або тільки вибірково, а агреговані ряди метрик зберігають довше і саме їх показують на дашбордах.

Prometheus є прикладом системи, яка працює з часовими рядами. Він зберігає все як часові ряди, тобто потоки значень з мітками часу, які належать до однієї метрики і набору міток [18]. Модель з мітками дозволяє будувати вимірювання з багатьма вимірами, наприклад метрика flows_per_second з мітками exporter або interface або vlan, і потім фільтрувати та агрегувати у запитах. Це підходить для метрик колектора, для ознак детектора і для сервісних показників, таких як затримка обробки і кількість втрачених пакетів телеметрії. Але Prometheus не працює як сховище для сирих flow записів. Він може зберігати агреговані метрики і стан системи.

Elastic підхід [19] корисний тоді, коли потрібно приймати різні типи даних і швидко по них шукати. Даний пакет може приймати дані з Elastic Agent, Beats, колекторів OpenTelemetry та Logstash, а також є механізми додаткової обробки даних перед індексацією.

Kibana є типовим інтерфейсом до такого сховища - це інтерфейс з відкритим кодом для запитів, аналізу, візуалізації та керування даними, які зберігаються в Elasticsearch [20]. Для задач телеметрії це зручно тим, що оператор може будувати дашборди, робити швидке дослідження подій, фільтрувати за полями і працювати з даними як з журналом і як з часовими рядами.

Отже, Prometheus доцільно використовувати для зберігання метрик і агрегованих ознак, тоді як Elasticsearch і Kibana краще підходять для роботи з подіями, пошуку та аналізу інцидентів. В реальних (рисунок 1.2) системах часто поєднують ці два підходи, коли Prometheus відповідає за метрики стану системи і

агреговані ознаки, а Elastic стек відповідає за події, алерти і розслідування інцидентів.


Рисунок 1.2 - Архітектура системи виявлення аномального трафіку на основі потоків

Крім цього, існують класичні IDS-системи, які традиційно аналізують пакети та використовують сигнатурні правила. Через це виник великий клас рішень, де детекція атак і аномалій робиться саме на flow даних або на ознаках, які з них будуються. Наприклад в роботі [21] запропоновано підхід, де детекція складається з частини, яка аналізує заголовки потоків, і частини, яка аналізує трафік як шаблони у часі. На рівні реалізації автори описують прототип системи, яка використовує flow інформацію від моніторингової платформи і будує конвеєр у вигляді пайплайна з послідовними фазами, які можуть розподілятися на кілька систем для балансування навантаження. Також в роботі є приклад, що для різних типів атак можуть застосовуватися різні порогові значення і ваги параметрів, а це показує, що практичний детектор часто вимагає адаптації під середовище.

В роботі [22] задачу розглянуто з іншого боку. У цьому підході аналізується шаблон окремої атаки та поведінка кінцевих вузлів і відхилення від їхньої типової комунікації. В роботі відзначається, що сигнатурні технології добре детектують відомі атаки, але нові типи можуть лишатися непоміченими, тому вони роблять акцент на машинному навчанні та пошуку аномальної мережевої активності кінцевих користувачів. Рішення в цій роботі описується як фреймворк (рисунок 1.3), який будує профілі звичних контактів вузла і відмічає відхилення, а також виділяє характерні патерни трафіку для подальшого розбору. Превагою такого підходу є те що він може знаходити нетипові ситуації навіть без точних сигнатур. До недоліків можна віднести те, що потрібно коректно налаштовувати пороги і фільтри, інакше можна отримати багато хибних спрацювань, особливо коли користувацька активність різко змінюється через робочі процеси.

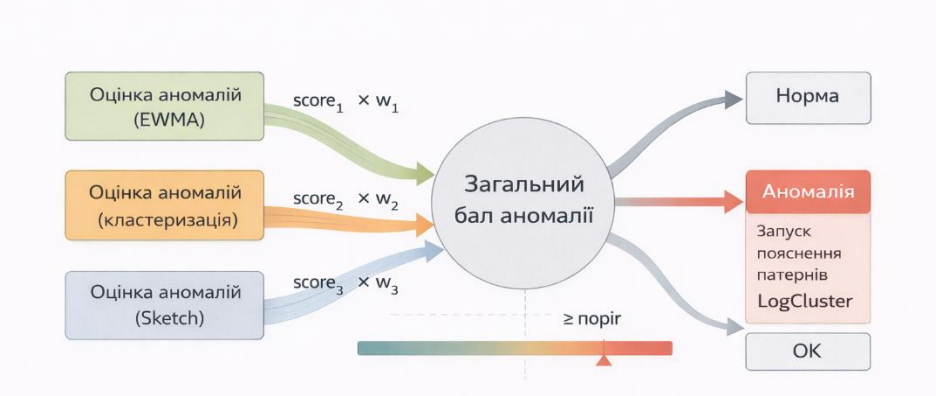



Рисунок 1.3 - Схема фреймворку виявлення аномальних кінцевих вузлів на основі NetFlow та профілів поведінки.

У простих системах часто недооцінюють затримку, яка виникає через таймаути формування потоків. В досліджені [23] пояснено, що в типових архітектурах аналіз flow даних відбувається після спрацювання таймаутів і тому детекція може затримуватися на кілька хвилин. Для великих flooding атак це критично, бо за цей час атака вже встигає нанести суттєву шкоду. У відповідь на це пропонується функціональне розширення для експортерів NetFlow та IPFIX, щоб забезпечити більш своєчасну детекцію і можливість швидшої реакції. Крім того в роботі описано прототип, реалізований як модуль для платформи FlowMon, де

архітектура побудована на плагінах для прийому, обробки, фільтрації і експорту, а модуль детекції інтегрується як обробний плагін. Це показує ще один шлях побудови системи, коли частина логіки переноситься ближче до джерела і до моменту експорту, а не тільки працює на архівних файлах (рисунок 1.4).

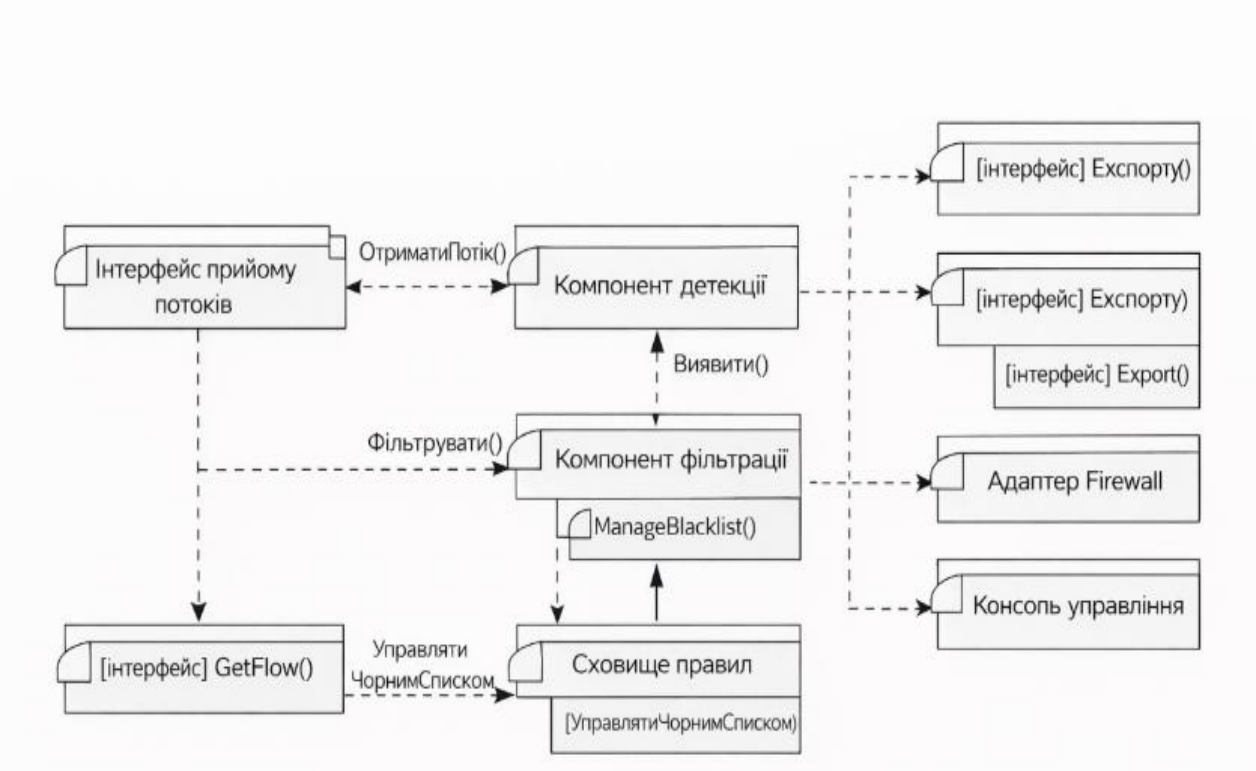



Рисунок 1.4 - Архітектура прототипу для своєчасної детекції на основі NetFlow та IPFIX з модулем фільтрації та консоллю подій.

Ще один приклад практичної реалізації наведено в роботі [24]. Там ідея полягає в тому, щоб використовувати flow статистику безпосередньо на мережевому обладнанні та будувати детекцію навколо метрик, які можна зібрати з flow кешу і обробити в межах можливостей пристрою. В роботі було дано інформацію про результати оцінки навантаження і показано, що при реалізації на IOS важливо контролювати ресурсні обмеження і підбирати метрики так, щоб детектор був достатньо швидким і не заважав основним функціям маршрутизації. Такий підхід добрий коли детекція переноситься на рівень мережевого пристрою, і дає практичні аргументи, чому в інших архітектурах детекцію виносять на окремий вузол або на аналітичний сервер.

## 1.3 Постановка задачі та вимоги до модуля

На основі аналізу джерел, розглянутих у попередніх підрозділах, можна сформулювати задачу для розроблюваної системи. Існуючі рішення показують, що flow телеметрія є зручним і масштабованим способом спостереження за мережею. Водночас значна частина реалізацій або обмежується архівуванням потоків, або використовує лише один підхід до виявлення аномалій. Частина рішень працює як автономні аналізатори історичних даних, інші як модулі у складі великих комерційних платформ. Тому постає завдання розробити власний програмно-апаратний модуль, який поєднує збір потоків, аналітику та механізм реагування в єдиній узгодженій архітектурі.

Модуль мережевої телеметрії та виявлення аномалій доцільно побудувати з трьох логічних частин. Перша частина відповідатиме за прийом і початкову обробку flow даних. Друга частина виконуватиме аналітику і обчислення ознак та реалізує алгоритми виявлення відхилень. Третя частина забезпечує сповіщення, збереження результатів і надання інформації через інтерфейс для оператора.

Архітектура системи базуєтиметься на розподіленні ролей між двома вузлами. Raspberry Pi, який буде виконувати функцію колектора та агрегатора потоків. Персональний комп’ютер з GPU виконуватиме функцію аналітичного вузла, на якому запускаються ресурсоємні обчислення. Такий підхід відповідає сучасним принципам розділення edge збору і централізованої обробки, тому що модульна структура з окремими компонентами прийому, детекції та фільтрації дозволить гнучко керувати потоком даних і впроваджувати алгоритми без зміни всієї системи.

### 1.3.1 Функціональні вимоги

Модуль збору повинен приймати flow записи у форматах NetFlow або IPFIX через мережевий інтерфейс. Він має підтримувати одночасний прийом з кількох джерел і вести облік часу надходження записів.

Система повинна виконувати агрегацію потоків за часовими вікнами. Модуль аналітики повинен обчислювати ознаки і формувати числову оцінку аномальності. Система повинна порівнювати загальний бал із заданим порогом і в разі перевищення створювати подію аномалії. Результати повинні зберігатися у базі подій і бути доступними через веб інтерфейс або програмний API.

### 1.3.2 Нефункціональні вимоги

Модуль збору на Raspberry Pi повинен приймати потоки без суттєвих втрат навіть при підвищеному навантаженні. Аналітичний вузол з GPU повинен забезпечувати обробку агрегованих даних в майже реальному часі, щоб затримка між появою аномалії і її фіксацією була мінімальною.

Якщо аналітичний вузол тимчасово недоступний, модуль збору повинен зберігати потоки локально до відновлення зв’язку. Оскільки система працюватиме з чутливою інформацією про мережеві взаємодії, необхідно передбачити захист доступу та передавання даних. Доступ до інтерфейсу повинен бути обмежений механізмами автентифікації. Передача даних між модулями має бути захищена. В разі помилки у модулі детекції повинна бути можливість перезапуску без втрати історії.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1. Розробити архітектуру програмно-апаратного рішення для збору flow-даних, агрегації на вузлі збору та аналітики на обчислювальному вузлі;

2. Розробити алгоритмічне забезпечення модуля;

3. Розробити інформаційне забезпечення, визначити вхідні та вихідні дані, побудувати концептуальну і логічну модель даних;

4. Виконати вибір програмних засобів і технологій реалізації модулів збору, зберігання, аналітики та веб-інтерфейсу;

5. Реалізувати ключові модулі системи;

6. Реалізувати інтерфейс користувача у вигляді веб-дашборду та API для перегляду інцидентів, подій, агрегатів і сповіщень;

7. Провести тестування працездатності модуля на контрольованих сценаріях.

# 2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ МОДУЛЯ МЕРЕЖЕВОЇ ТЕЛЕМЕТРІЇ ТА ВИЯВЛЕННЯ АНОМАЛІЙ

## 2.1 Архітектура програмно апаратного рішення

Архітектуру модуля побудовано у вигляді конвеєра обробки, у якому дані послідовно проходять від мережевого обладнання до колектора, аналітичного вузла, сховища та інтерфейсу оператора.

На першому етапі джерело формує записи потоків з узагальненими відомостями про мережеві взаємодії. Потім колектор приймає ці записи і готує їх до аналізу. Після цього аналітичний модуль перетворює потоки в ознаки, знаходить відхилення і формує події. На завершення оператор отримує результат у вигляді дашборду та сповіщень, а система накопичує історію для подальшого аналізу.

Запропоноване рішення є розподіленим (рисунок 2.1). Роль вузла збору виконує Raspberry Pi, оскільки він може працювати як постійно увімкнений edge пристрій, який легко встановити поряд із мережевим обладнанням.

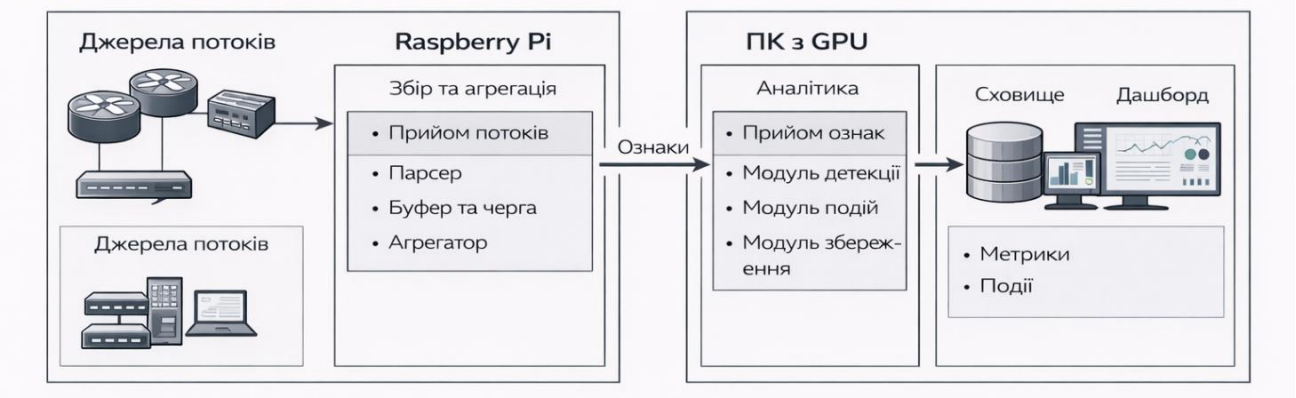



Рисунок 2.1 - Структурна схема програмно - апаратного рішення для збору flow телеметрії.

На ньому зручно організувати прийом NetFlow або IPFIX, а також прийом sFlow у випадку, якщо джерелом є комутатор з вибірковою телеметрією. Raspberry Pi виконує функції колектора та агрегатора, тобто приймає дані, перевіряє їхню коректність, буферизує, виконує первинне зведення за часовими вікнами та передає підготовлені дані далі.

Аналітичним вузлом виступає персональний комп’ютер з GPU, де доступне прискорення CUDA для ресурсоємних обчислень, зокрема для навчання моделей і швидкого інференсу. Винесення інтенсивної обробки на GPU дозволяє обробляти великі потоки записів швидше та зменшувати затримку між появою події і реакцією системи.

Загальна структура системи починається з джерел. Джерелами можуть бути маршрутизатор, комутатор або програмний агент. Маршрутизатор формує NetFlow або IPFIX, комутатор формує sFlow, а агент може імітувати або формувати потоки на лабораторному стенді. Для системи важливо не прив’язуватися до одного формату. На рівні прийому на Raspberry Pi  можна мати два незалежні канали. Перший канал призначений для NetFlow та IPFIX, де потоки приходять як записи із шаблонами та даними. Другий канал призначений для sFlow, де надходять вибіркові семпли та лічильники. Після прийому дані переводяться у внутрішній уніфікований формат. Це дозволить далі однаково працювати з потоками незалежно від того, як саме вони були отримані.

Структуру колектора на базі Raspberry Pi треба робити простою та стійкою до збоїв. Перший підкомпонент це приймач телеметрії, який слухає мережеві порти і приймає UDP повідомлення. Основне завдання цього компонента — забезпечити стабільний прийом даних навіть під час короткочасних піків навантаження. Далі йде парсер, який витягує поля потоків, перевіряє базову коректність, приводить часові мітки до узгодженого вигляду і формує внутрішній запис потоку. Після цього дані потрапляють в буфер.

Після буферизації дані обробляє агрегатор, розгорнутий на Raspberry Pi. Його задача підготувати дані так, щоб аналітичний вузол отримував не сирий потік подій, а вже впорядковані за часом та ключами дані. Агрегація виконується у вікнах часу, наприклад в секундах або хвилинах, залежно від того, яка потрібна оперативність. Для кожного вікна агрегатор рахує базові статистики, такі як кількість потоків, суму байтів, суму пакетів, кількість унікальних адрес або портів. Також формуються допоміжні показники, які важливі для виявлення сканувань і

флуд атак, наприклад частку коротких потоків або частку потоків із певними TCP ознаками, якщо ці поля доступні.

Передача даних з Raspberry Pi на ПК виконується через логічну чергу. У найпростішому варіанті передавання може виконуватися через надсилання пакетів ознак із підтвердженням доставки на рівні застосунку або через використання брокера повідомлень у локальній мережі. Якщо мережа між вузлами переривається, Raspberry Pi повинен продовжувати буферизацію і після відновлення зв'язку доганяти відставання.

Аналітичний вузол на ПК приймає агреговані ознаки і виконує основну обробку. Для аналітичного вузла передбачено два режими роботи. Перший режим це поточний інференс, коли кожне нове вікно ознак одразу обробляється і система вирішує, чи є ознаки аномалії. Другий режим це навчання або переналаштування моделі, коли накопичена історія використовується для побудови профілю норми та підбору порогів. Режим інференсу має бути максимально швидким, щоб зменшувати затримку. Режим навчання може запускатися рідше і використовувати більш довгі горизонти даних.

В межах аналітичного вузла можна (рисунок 2.2) виділити кілька логічних компонентів. Перший компонент це приймач ознак, який гарантує впорядкування за часом і базову перевірку, щоб у конвеєр не потрапляли дублікати або записи з некоректним часом.

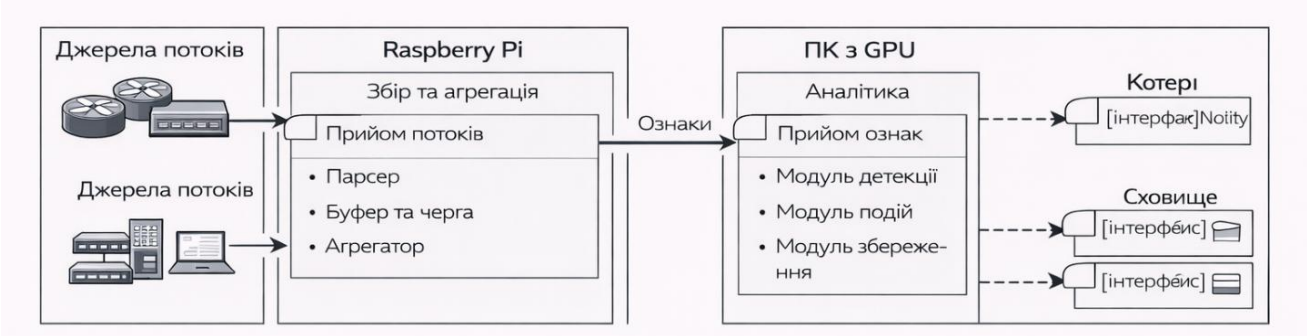



Рисунок 2.2 - Компонентна діаграма прототипу архітектури

Другий компонент це модуль детекції, який реалізує вибрані підходи. На практиці кращий результат часто забезпечує поєднання кількох простих методів,

які компенсують обмеження один одного. Наприклад, один метод добре ловить різкі сплески, інший добре ловить зміну структури трафіку, третій добре ловить накопичувальні відхилення. Тому в архітектурі передбачено можливість кількох детекторів і об’єднання їх оцінок у загальний бал.

Третій компонент це модуль подій, який перетворює результат детекції у подію з рівнем важливості, часовою міткою і набором пояснювальних полів.

Четвертий компонент - модуль збереження, який записує події та агреговані метрики у сховище.

Сховище на ПК розділене за призначенням. Метрики та часові ряди зберігаються в сховищі, орієнтованому на часові ряди, оскільки оператору важливо бачити графіки і тренди. Події аномалій та супутні записи зберігаються у сховищі, яке підтримує фільтрацію за полями і швидкий пошук, тому що для дослідження потрібно знаходити інциденти за адресою, портом, часом, типом аномалії.

Завершальним елементом архітектури є модуль сповіщень та інтерфейсу користувача. Він має два обов’язкових виходи. Перший вихід це сповіщення, наприклад повідомлення оператору або запис у журнал, щоб зафіксувати інцидент і не пропустити його. Другий вихід це веб інтерфейс або API, через який оператор бачить поточний стан, список подій і графіки метрик. Важливо, щоб інтерфейс не вимагав доступу до внутрішніх компонентів системи. Оператор працює з даними через API, а сама система приховує деталі.

## 2.2 Алгоритмічне забезпечення модуля

Алгоритмічне забезпечення побудовано відповідно до обраної архітектури, де Raspberry Pi виконує прийом, нормалізацію і первинну агрегацію, а ПК з GPU виконує обчислення ознак, детекцію та формування подій.

Мета алгоритму збору та нормалізації flow даних полягає в стабільному прийомі NetFlow або IPFIX та, за потреби, sFlow, з подальшим перетворенням у внутрішній уніфікований формат. На цьому етапі потрібно мінімізувати втрати записів, коректно обробляти часові мітки та відкидати помилкові дані.

Нижче представлено алгоритм збору та нормалізації flow даних:

1. Запустити приймач телеметрії і відкрити мережеві порти для прийому повідомлень.

2. Для кожного отриманого UDP повідомлення визначити тип формату за заголовком або за налаштованим портом прийому.

3. Якщо повідомлення належить NetFlow або IPFIX, витягнути ідентифікатор джерела та службові поля, які потрібні для розбору структури.

4. Якщо формат підтримує шаблони, оновити таблицю шаблонів для відповідного джерела, коли у повідомленні присутні записи шаблонів.

5. Зберігати шаблони з прив'язкою до exporter id, observation domain або іншого стабільного ідентифікатора джерела.

6. Якщо у повідомленні присутні записи даних, розібрати їх відповідно до актуального шаблону.

7. Для кожного запису витягнути мінімальний набір полів, який потрібен для подальшої обробки.

8. Якщо повідомлення належить sFlow, розібрати семпли і лічильники.

9. Для зразків даних сформувати спрощене подання, максимально наближене до формату потокових даних.

10. Для лічильників сформувати окремий запис метрик інтерфейсу, який може бути використаний для контролю цілісності та загальної картини навантаження.

11. Виконати валідацію полів. Перевірити, що адреси мають коректний формат, порти належать діапазону, протокол визначений, лічильники не мають від'ємних значень, а часові поля не суперечать одне одному.

12. Виконати нормалізацію часових міток.

13. Якщо у записі є час початку і час завершення, перевірити порядок і у разі аномальних значень помітити запис як підозрілий.

14. Якщо запис містить лише тривалість, обчислити час завершення відносно часу отримання або відносно часу початку.

15. Виконати проставлення часових міток на стороні колектора.

16. Додати до внутрішнього запису поле часу прийому на Raspberry Pi.

17. Перетворити дані у внутрішній уніфікований формат..

18. Помістити нормалізований запис у буфер черги.

19. Якщо буфер переповнений, зафіксувати факт втрати в лічильниках системи і застосувати політику обмеження, наприклад скидання найстаріших записів або тимчасове зменшення детальності через агрегацію.

20. Періодично записувати метрики прийому, такі як кількість отриманих повідомлень, кількість розібраних записів, кількість відкинутих записів, а також оцінку можливих втрат через переповнення.

Після виконання алгоритму система отримує уніфіковані записи, придатні для подальшої агрегації та побудови ознак.

Наступний етап роботи модуля пов’язаний з агрегацією потоків і побудовою ознак. Мета алгоритму - перетворити сирі записи потоків на набір ознак у часових вікнах. Ознаки повинні бути достатньо простими для стабільного обчислення на Raspberry Pi, але водночас інформативними для детекції аномалій на ПК.

Алгоритм виконується так.

1. Вибрати тривалість вікна агрегації.

2. Визначити ключі агрегації.

3. Для кожного нормалізованого flow запису визначити, до якого часового вікна він належить.

4. Додати запис до агрегатора вікна за відповідним ключем.

5. Оновити накопичувальні лічильники.

6. Збільшити суму байтів і суму пакетів.

7. Збільшити кількість потоків у вікні.

8. Для кожного ключа вікна окремо рахувати приблизну або точну кількість унікальних значень.

9. Обчислювати top N.

10. Підтримувати для вікна рейтинги найбільш активних співрозмовників або портів.

11. Для кожного ключа вікна сформувати частоти для обраної ознаки, наприклад для портів призначення або для адрес призначення.

12. Далі обчислити ентропію як міру розпорошення.

13. Для кожного ключа і вікна формувати середні значення, максимуми, мінімальні значення і дисперсію там, де це доречно.

14. У випадку доступності TCP ознак, накопичувати частки потоків з певними прапорцями або ознаками.

15. По завершенню вікна сформувати запис ознак.

16. У записі вказати час вікна, ключ, і набір розрахованих значень.

17. Передати сформований запис ознак в чергу до аналітичного вузла.

18. Якщо передача тимчасово недоступна, зберігати записи локально у буфері, щоб потім догнати.

19. Очистити структури завершеного вікна і перейти до наступного, зберігши лише те, що потрібно для ковзних оцінок.

Окремий етап алгоритмічного забезпечення становить виявлення аномалій.

Мета цього алгоритму — за сформованими ознаками визначити наявність підозрілої мережевої активності та зменшити кількість хибних спрацювань.

Далі представлено основні кроки алгоритму:

1. Отримати запис ознак за поточне вікно і визначити, для якого ключа він належить.

2. Завантажити базову норму для цього ключа.

3. Якщо норма ще не сформована, система працює у режимі навчання і накопичує статистику до мінімального обсягу.

4. Для кожної ключової ознаки порівняти поточне значення з нормою.

5. Якщо значення перевищує встановлений поріг, врахувати його внесок у базову оцінку.

6. Сформувати коротке пояснення для базової оцінки.

7. Зафіксувати, які саме ознаки перевищили поріг і наскільки.

8. Якщо базова оцінка дорівнює нулю або нижча за мінімальний рівень, віднести стан до норми і завершити обробку, але зберегти ознаки у сховище метрик.

9. Якщо базова оцінка має підозріле значення, передати запис ознак у основний ML модуль.

10. В ML модулі виконати нормалізацію ознак.

11. Обчислити оцінку, отриману за допомогою моделі машинного навчання.

12. Якщо використовується один метод, він повертає один бал.

13. Якщо використовується ансамбль, кожен метод повертає власний бал, а потім бали об’єднуються у загальний бал за вагами.

14. Порівняти загальний бал із порогом спрацювання.

15. Перевірити, чи не належить ключ до списку відомих легітимних високих навантажень, чи не відбувається планове резервне копіювання, чи не спостерігається загальне підняття трафіку у всій мережі.

16. Застосувати перевірку стійкості.

17. Якщо умови виконані, сформувати рішення про аномалію і передати його в модуль подій.

18. Якщо умови не виконані, позначити запис як підозрілий без ескалації і продовжити накопичення статистики.

В результаті система або фіксує нормальний стан, або формує підтверджену аномалію з оцінкою та коротким поясненням причини.

Наступний алгоритм описує порядок формування подій та сповіщень.

Мета алгоритму — перетворити виявлену аномалію на подію, зберегти її, відобразити на панелі моніторингу та, за потреби, надіслати сповіщення оператору, не створюючи надмірного потоку повідомлень під час тривалого інциденту. Для цього вводяться рівні важливості, дедуплікація, кореляція і режим охолодження.

Далі представлено кроки виконання алгоритму:

1. Отримати рішення детектора, яке містить ключ, час вікна, загальний бал аномалії та пояснювальні ознаки.

2. Якщо бал значно перевищує поріг, присвоїти високий рівень.

3. Якщо бал близький до порога, присвоїти середній рівень.

4. Якщо це повторюваний слабкий сигнал, присвоїти низький рівень або залишити як подію для журналу без сповіщення.

5. Перевірити, чи вже існує активний інцидент для цього ключа в межах останнього проміжку часу.

6. Якщо існує, не створювати нову подію, а оновити лічильник повторів та час останнього спостереження.

7. Перевірити, чи одночасно з цією аномалією виникають пов’язані події для інших ключів в межах однієї підмережі або до одного сервісу.

8. Якщо так, об’єднати події у один інцидент або створити зв’язки між ними.

9. Якщо ключа cooldown або інциденту недавно вже надсилалося сповіщення, то нове сповіщення не надсилати, а лише оновити запис у базі подій.

10. Сформувати об’єкт події який має містити ідентифікатор, час початку, час останнього оновлення, рівень важливості, тип аномалії або клас, ключ, основні метрики і коротке пояснення.

11. Записати подію у сховище подій.

12. Якщо подія є оновленням, виконати оновлення запису.

13. Передати подію в модуль сповіщень.

14. Вибрати канали сповіщення.

15. Сформувати повідомлення для оператора та вказати що сталося, для якого ключа, коли, який рівень, і які ознаки стали причиною.

16. Оновити дашборд через API, щоб інтерфейс міг показати її у списку інцидентів і на часовій шкалі.

17. Якщо протягом певного часу сигнал не повторюється, помітити інцидент як завершений, зафіксувати час завершення і підсумкові лічильники.

18. Зафіксувати кількість подій, кількість сповіщень, кількість дедуплікованих оновлень і середній час реакції, щоб контролювати стабільність системи.

В результаті формується впорядкований потік подій і сповіщень: оператор отримує потрібну інформацію, але не перевантажується повторними повідомленнями.

## 2.3 Інформаційне забезпечення

Для системи потокової телеметрії доцільно одразу розділити дані на два класи. Перший клас це потоки та семпли, які приходять з мережі і можуть бути досить об’ємними. Другий клас це агрегати, ознаки, події та службові метрики, які є значно компактнішими і саме вони найчастіше потрібні оператору та детектору (рисунок 2.3).

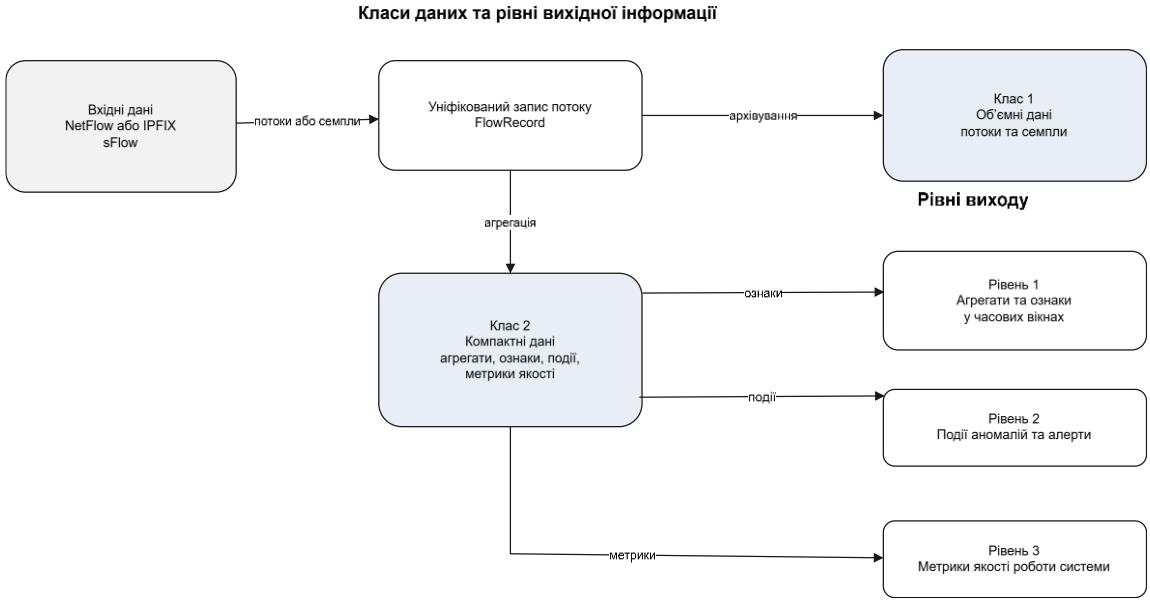



Рисунок 2.3 – Класи даних та рівні вихідної інформації в модулі мережевої телеметрії

Вхідна інформація для модуля складається з потокових записів NetFlow або IPFIX, а також з вибіркових даних sFlow, якщо джерелом є комутатор з вибірковою телеметрією. Усі ці формати зводяться до внутрішнього уніфікованого запису потоку, щоб далі система працювала однаково незалежно від джерела.

Основним вхідним об’єктом є запис потоку. Він містить ідентифікаційні поля, які відповідають на питання хто з ким взаємодіяв, і вимірювальні поля, які відповідають на питання скільки даних передано і коли це відбулося.

Додатково можуть надходити службові поля, наприклад ідентифікатор експортера, домен спостереження, ознаки TCP, індекс інтерфейсу входу, індекс інтерфейсу виходу, ознаки сервісу DSCP.

Вихідну інформацію системи можна поділити на три рівні. Перший рівень це агрегати і ознаки у часових вікнах. Другий рівень це події аномалій і алерти. Третій рівень це метрики якості роботи самої системи, які потрібні для контролю надійності (рисунок 2.4).

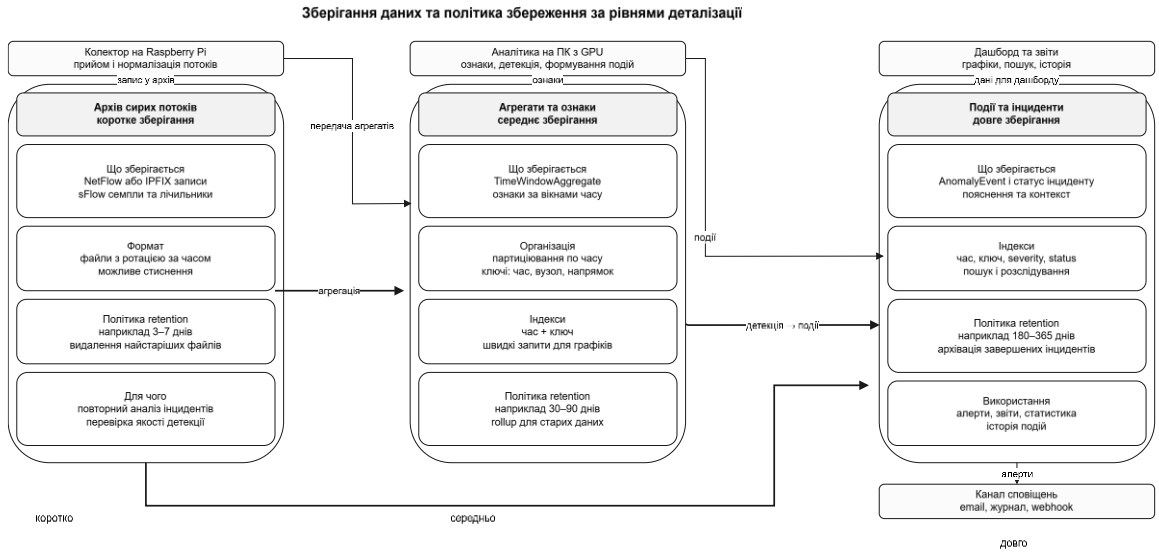



Рисунок 2.4 – Схема зберігання даних та політика збереження за рівнями деталізації

Агрегати і ознаки формуються по часових вікнах і по ключу, який вибраний для моніторингу. Наприклад ключем може бути адреса вузла або пара адрес. У записі агрегату зберігається час початку вікна, час завершення вікна, ідентифікатор ключа і набір числових значень. Значеннями є кількість потоків, сума байтів, сума пакетів, середній розмір потоку, кількість унікальних адрес призначення, кількість унікальних портів призначення, ентропія розподілу портів, а також короткий

перелік top значень, наприклад top адреси або top порти за байтами чи за кількістю потоків.

Подія аномалії є узагальненим об’єктом, що описує зафіксоване відхилення. Вона містить час, ключ, підсумковий бал або оцінку аномальності, рівень важливості, тип або категорію, а також пояснювальні поля. Пояснювальні поля потрібні для оператора. Вони показують, які саме ознаки стали причиною спрацювання, і наскільки поточне значення відрізняється від норми. Повідомлення про небезпеку є формою події, яка була відправлена у канал повідомлень, наприклад у журнал, у повідомлення оператору або в системний webhook.

До метрик якості, які описують роботу всієї системи як сервісу входить кількість отриманих потоків за хвилину, кількість відкинутих записів через валідацію, оцінка втрат через переповнення буфера, затримка між часом події і часом обробки, тривалість обробки вікна, завантаження CPU і пам’яті на Raspberry Pi та на ПК, а також статуси компонентів. Такі метрики потрібні для того, щоб відрізнити реальну аномалію трафіку від збою або перевантаження самої системи збору.

# 3 ПРОГРАМНО ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ ТА РЕЗУЛЬТАТИ РЕАЛІЗАЦІЇ

## 3.1 Реалізація програмного забезпечення модуля

Реалізацію модуля виконано на Raspberry Pi OS. як колектора та окремого аналітичного вузла з GPU, а обчислення, які можуть бути прискорені CUDA, реалізовано на CPU з максимально збереженою структурою конвеєра. Це дозволяє протестувати працездатність підходу та перенести частину обчислень на прискорювач без зміни логіки даних.

Збір flow-даних організовано як послідовність двох інструментів. Формування потоків виконано softflowd, який аналізує трафік на loopback інтерфейсі і експортує потоки у форматі NetFlow. Прийом та буферизацію потоків забезпечено nfcapd, який слухає UDP порт 2055 та записує дані у файли з ротацією в каталозі /var/cache/nfdump (рисунок 3.1).

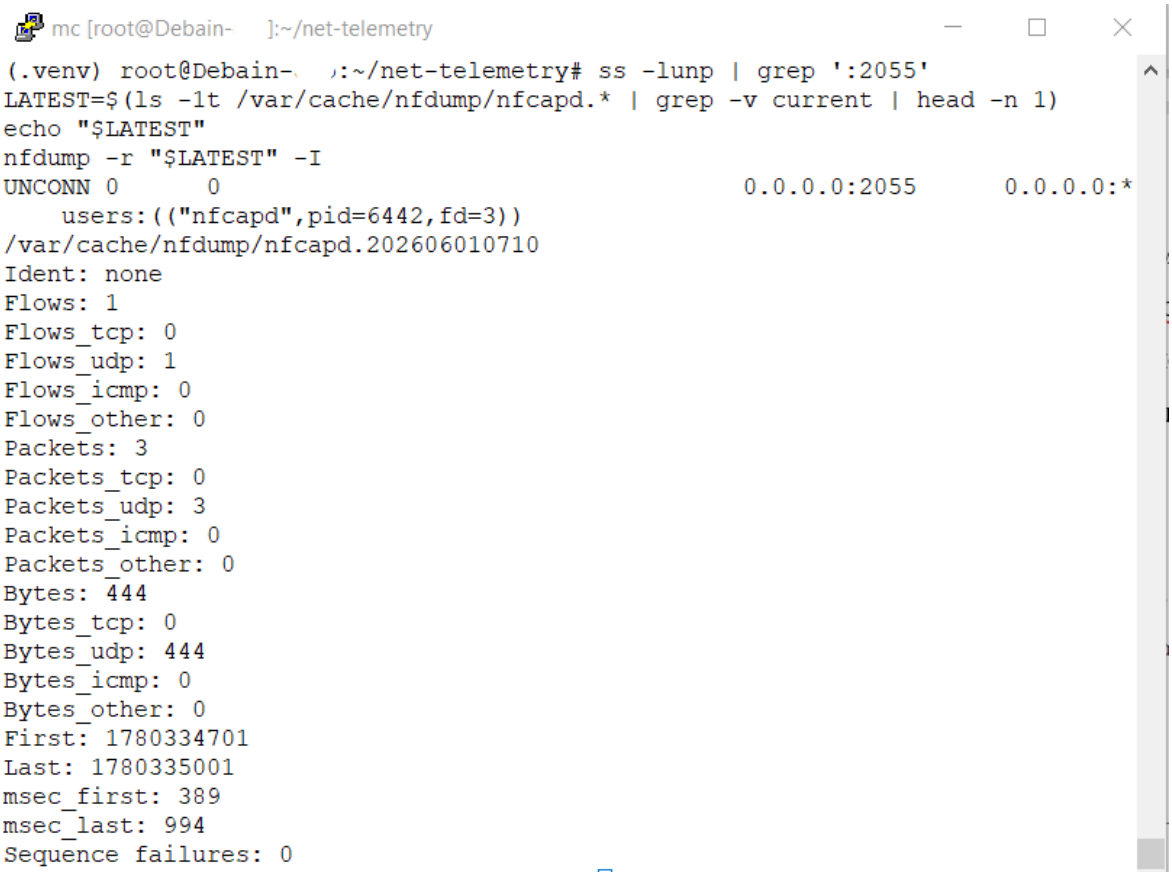



Рисунок 3.1 - Підтвердження збору flow і наявності записів у файлі

Такий вибір забезпечує стабільний прийом потоків і не ускладнює власний код низькорівневим захопленням трафіку, а також забезпечує відтворюваність тестів через наявність архіву файлів потоків.

Далі реалізовано етап нормалізації та перенесення даних у внутрішнє сховище. Для цього використано утиліту nfdump як інструмент перетворення файлів nfcapd у машинно-читаний формат CSV. На основі CSV реалізовано скрипт імпорту import_flows.py, який записує дані у SQLite базу telemetry.db в таблицю flow_record. Для кожного запису потоку фіксуються час першого спостереження, тривалість, протокол, адреси та порти, а також лічильники пакетів і байтів. Додатково створено індекси за часом і за ключовими полями, що забезпечує швидкий відбір даних для подальших етапів.

На наступному етапі виконано агрегацію потоків у часові вікна та побудову ознак. Для цього реалізовано скрипт aggregate_windows.py, який читає flow_record та формує таблицю time_window_aggregate. Вікно агрегації обрано рівним 60 секунд, що дає баланс між стабільністю статистики та оперативністю реакції. Для кожного вікна обчислюються кількість потоків, сума байтів, сума пакетів, кількість різних портів призначення та ентропія розподілу портів. Саме кількість різних портів і ентропія їх розподілу є важливими ознаками сканування, оскільки під час такої активності зростає різноманітність портів і розпорошеність розподілу. В результаті агрегації сформовано 13 часових вікон, в яких можна виділити періоди з нетипово високою кількістю різних портів та високою ентропією.

Детекція аномалій реалізована у двох підходах. Перший підхід є базовим і використовує rolling порівняння з попередніми вікнами. Він реалізований в baseline_detect.py. Алгоритм порівнює поточне вікно з кількома попередніми, обчислює відхилення та формує подію при перевищенні порога. Окремо додано явне правило сканування. Якщо кількість різних портів у вікні достатньо велика та ентропія висока при достатній кількості потоків, подія формується незалежно від відхилення по байтах. Завдяки цьому оператор отримує зрозуміле пояснення спрацювання: ознаки сканування, багато різних портів і висока ентропія.

Другий підхід подано як основний ML-детектор. Модуль ml_detect.py використовує безнаглядне оцінювання, яке визначає міру відхилення поточних ознак від типових значень, сформованих за попередні часові вікна. Для стабільності виконано нормалізацію через log1p для потоків і байтів, а також врахування лише додатних відхилень, що усуває спрацювання на падіння активності. Такий підхід є практичним, оскільки зосереджує увагу на небажаних сплесках і нетипових змінах структури трафіку. ML-детектор також підтримує правило сканування і формує подію з поясненням, яка ознака зробила найбільший внесок у score. За результатами запуску baseline-детектор сформував 5 подій, а ML-детектор сформував 2 події, які відповідають сценарію сканування.

Оскільки події можуть повторюватися у сусідніх часових вікнах, реалізовано модуль event_manager.py для керування інцидентами та сповіщеннями. Він перетворює набір подій детекторів у узагальнені інциденти. Інцидент має час початку, час останнього прояву, тип, ключ, рівень важливості, перелік детекторів, що підтвердили подію, та лічильник оновлень. Для зменшення кількості повторних сповіщень застосовано правило паузи між повідомленнями. В журналі сповіщень фіксується час відправки і короткий текст. В тестовому запуску сформовано 3 інциденти, виконано 4 оновлення інцидентів та відправлено 3 сповіщення з урахуванням паузи між повторними повідомленнями.

Для перевірки результатів і використання їх у звіті реалізовано експорт у CSV за допомогою export_results.py. Сформовано файли events.csv, incidents.csv, alerts.csv та агрегатні таблиці для демонстрації найвиразніших вікон за кількістю різних портів і за ентропією.

## 3.2 Інтерфейс користувача та засоби представлення результатів

Щоб спростити перегляд і аналіз результатів, реалізовано веб-інтерфейс, який працює поверх telemetry.db. Інтерфейс створено на базі FastAPI, Uvicorn та Jinja2, що дозволяє відображати дані у вигляді сторінок з таблицями та графіками. Інтерфейс запускається як веб-сервер у VM та відкривається в браузері на стороні

хост-системи. Для запуску використано порт 8081, що дозволяє паралельно тримати інші сервісні порти без конфліктів (рисунок 3.2).

```
(.venv) root@Debain-  :~/net-telemetry# ss -ltnp | grep ':8081'
curl -i "http://127.0.0.1:8081/incidents?limit=3" | head -n 12
LISTEN 0      2048         0.0.0.0:8081      0.0.0.0:*    users:(("uvicorn",pi
d=9136,fd=6))
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
t
                                 Dload  Upload   Total   Spent    Left  Speed
100  2792  100  2792    0     0   2523      0  0:00:01  0:00:01 --:--:--  2524
HTTP/1.1 200 OK
date: Mon, 01 Jun 2026 18:10:35 GMT
server: uvicorn
content-length: 2792
content-type: text/html; charset=utf-8

<!doctype html>
<html lang="uk">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Інциденти</title>
```

Рисунок 3.2 - Підтвердження роботи графічного інтерфейсу

Веб-інтерфейс містить кілька сторінок для перегляду різних груп даних. Головна сторінка (рисунок 3.3) відображає коротку статистику, а також графіки ключових показників за останні часові вікна.

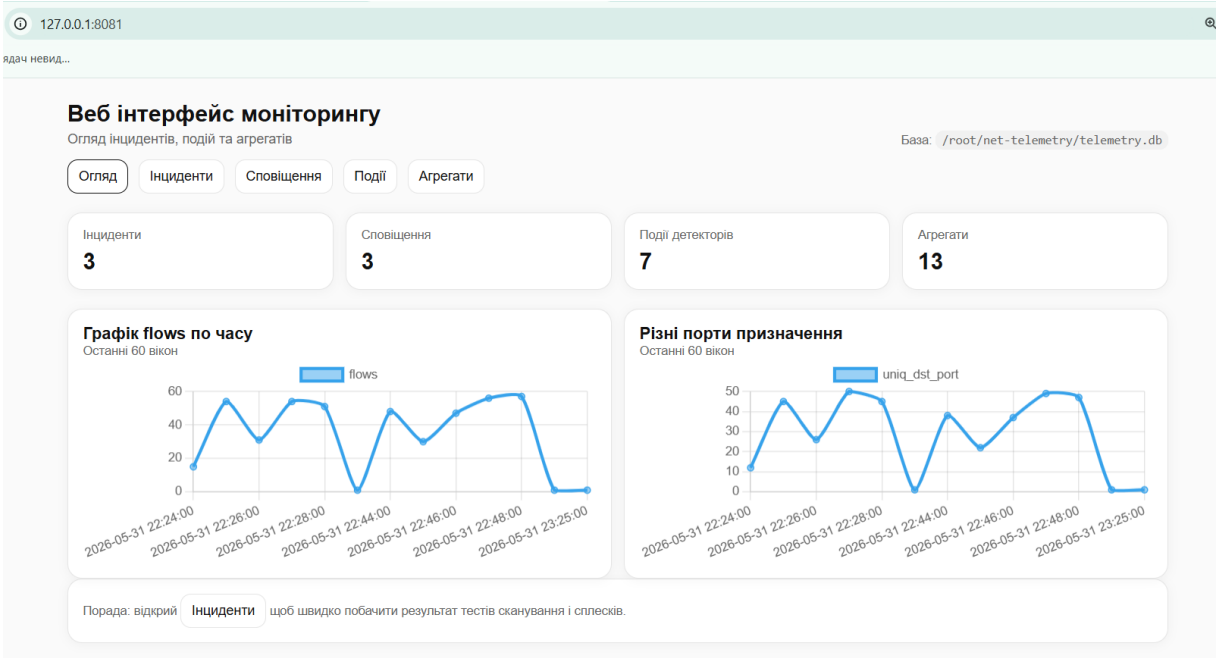



Рисунок 3.3 – Головна сторінка веб-інтерфейсу з короткою статистикою та графіками

Для графіків використано Chart.js, а дані подаються через простий API-ендпоїнт, який повертає часові ряди з таблиці агрегатів.

Окремо створено сторінку інцидентів (рисунок 3.4), де відображаються узагальнені записи після об’єднання повторних подій.

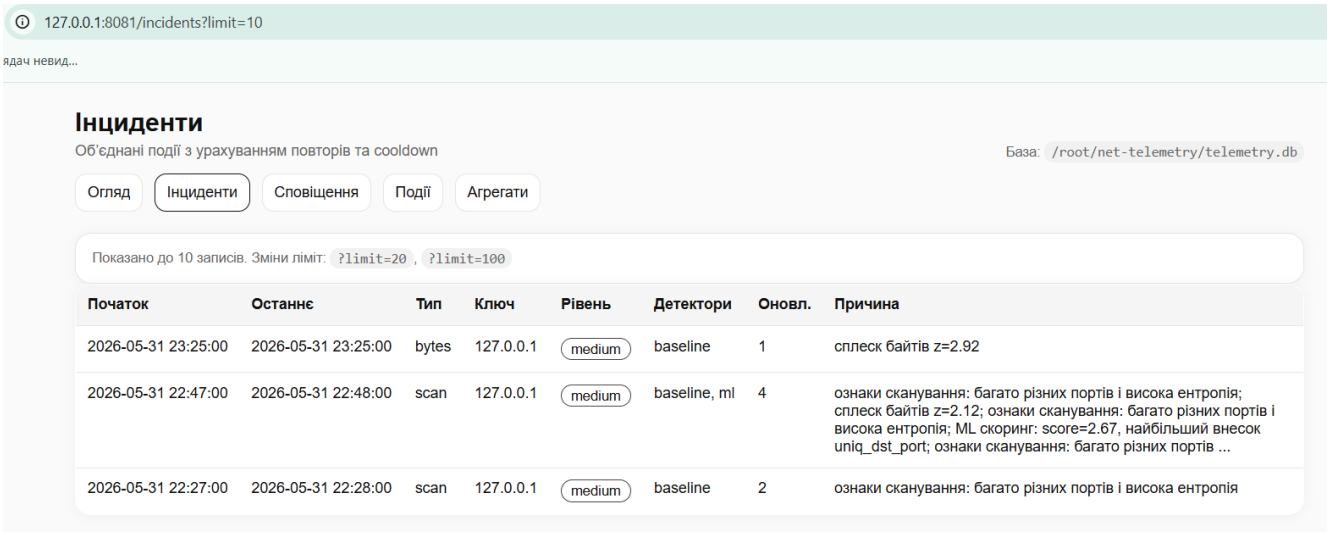



Рисунок 3.4 – Сторінка інцидентів з результатами виявлення аномалій та поясненням причин

На цій сторінці відображається тип інциденту, рівень важливості, перелік детекторів, кількість оновлень і коротке пояснення причини. Також є сторінка сповіщень (рисунок 3.5), яка відображає журнал відправлених повідомлень з урахуванням паузи між повторними спрацюваннями.

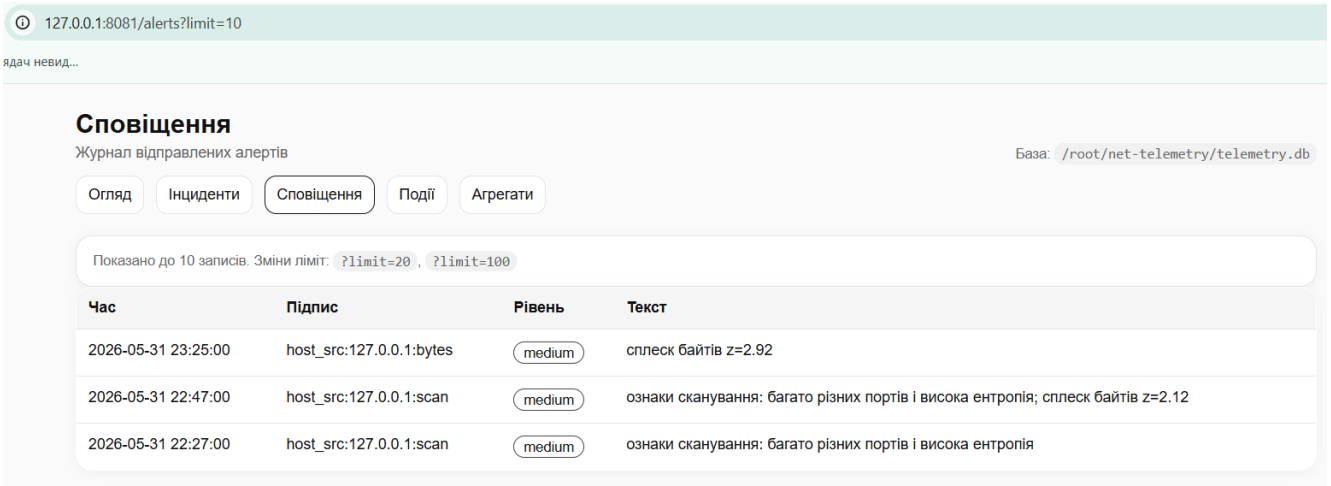



Рисунок 3.5 – Сторінка сповіщень з журналом повідомлень про інциденти

Додаткові сторінки подій і агрегатів дозволяють переглядати первинні записи детекторів та значення ознак у вікнах часу.

## 3.3 Тестування програмно – апаратного модуля

Тестування проведено за двома сценаріями: нормальна робота мережі та аномальна активність. Для нормального сценарію використано утиліту iperf3, яка генерує контрольований обмін даними між клієнтом і сервером. Такий трафік імітує стабільну передачу даних і формує потоки з типовими параметрами. Для аномального сценарію використано утиліту nmap, яка виконує сканування діапазону TCP-портів. В такому випадку в flow-даних з'являється велика кількість коротких з'єднань до різних портів, що добре відображається в ознаках uniq_dst_port та entropy_dst_port.

Перевірка збору даних підтвердила, що nfcapd створює файли та приймає потоки, а nfdump дозволяє переглядати їх вміст. Для нормального трафіку в файлах формуються потоки, пов'язані з тестовим з'єднанням. Для сканування формуються сотні потоків, що підтверджує правильність стенду і придатність даних для подальшої обробки.

Перевірка імпорту підтвердила коректність перетворення файлів потоків у базу SQLite. Кількість записів у flow_record зростає після імпорту, а запити по таблиці дозволяють отримувати часові межі, розподіл протоколів і найбільш часті порти. Перевірка агрегації показала, що ознаки вікон правильно реагують на різні сценарії.

Таблиця 3.1 – Приклад агрегованих ознак у часових вікнах під час сканування портів.

| Час вікна | Flows | Bytes | Різні порти призначення | Ентропія портів |
|---|---|---|---|---|
| 2026-05-31 22:27:00 | 54 | 3420 | 50 | 5.593 |
| 2026-05-31 22:47:00 | 56 | 3900 | 49 | 5.522 |
| 2026-05-31 22:48:00 | 57 | 3540 | 47 | 5.412 |
| 2026-05-31 22:28:00 | 51 | 3120 | 45 | 5.393 |
| 2026-05-31 22:25:00 | 54 | 3420 | 45 | 5.371 |

В вікнах сканування значно зростає кількість різних портів та зростає ентропія розподілу портів. У вікнах нормального трафіку ці значення мають інший характер і не формують такої розпорошеної картини. Отримані значення підтвердили, що вибрані ознаки можна використовувати для роботи детекторів.

Перевірка базового детектора показала, що правило сканування коректно формує події. Події містять пояснення ознаки сканування, багато різних портів і високу ентропію, що робить результат зрозумілим без складних інтерпретацій. Окремо фіксувались події типу bytes, якщо спостерігався значний сплеск байтів. В тестовому запуску baseline сформував 5 подій.

Перевірка ML-детектора показала, що після стабілізації оцінювання та врахування лише додатних відхилень модуль на основі машинного навчання формує події переважно для сканування і не створює зайвих спрацювань на зниження інтенсивності трафіку. В тестовому запуску ML сформував 2 події, які відповідають періодам сканування і підтверджують baseline результат.

Перевірка модуля інцидентів показала, що події від різних детекторів коректно об’єднуються в узагальнені інциденти, а повтори не перетворюються на надлишковий потік повідомлень.

Таблиця 3.2 – Узагальнені інциденти після об’єднання повторних подій.

| Початок інциденту | Останній прояв | Тип | Ключ | Рівень | Детектори | Кількість оновлень |
|---|---|---|---|---|---|---|
| 2026-05-31 22:27:00 | 2026-05-31 22:28:00 | scan | 127.0.0.1 | medium | baseline | 2 |
| 2026-05-31 22:47:00 | 2026-05-31 22:48:00 | scan | 127.0.0.1 | medium | baseline, ml | 4 |
| 2026-05-31 23:25:00 | 2026-05-31 23:25:00 | bytes | 127.0.0.1 | medium | baseline | 1 |

В тестовому запуску сформовано 3 інциденти і створено 3 сповіщення (рисунок 3.6).

```
(.venv) root@Debain-   :~/net-telemetry# python event_manager.py --reset
sqlite3 telemetry.db "DELETE FROM anomaly_event;"

python baseline_detect.py
python ml_detect.py
python event_manager.py

sqlite3 telemetry.db "SELECT first_seen_at, last_seen_at, kind, key_value, sev
erity, detectors, update_count FROM incident ORDER BY first_seen_at;"
sqlite3 telemetry.db "SELECT sent_at, signature, severity FROM alert_log ORDER
 BY sent_at;"
Reset OK: incident, incident_event, alert_log очищено.
Baseline events inserted: 5
ML events inserted: 2
Incidents created: 3
Incidents updated: 4
Alerts sent: 3
2026-05-31 22:27:00|2026-05-31 22:28:00|scan|127.0.0.1|medium|baseline|2
2026-05-31 22:47:00|2026-05-31 22:48:00|scan|127.0.0.1|medium|baseline, ml|4
2026-05-31 23:25:00|2026-05-31 23:25:00|bytes|127.0.0.1|medium|baseline|1
2026-05-31 22:27:00|host_src:127.0.0.1:scan|medium
2026-05-31 22:47:00|host_src:127.0.0.1:scan|medium
2026-05-31 23:25:00|host_src:127.0.0.1:bytes|medium
```

Рисунок 3.6 - Результат детекції + інциденти + сповіщення

Таблиця 3.3 – Журнал сповіщень з урахуванням паузи між повторними повідомленнями

| Час сповіщення | Підпис | Рівень | Текст |
|---|---|---|---|
| 2026-05-31 22:27:00 | host_src:127.0.0.1:scan | medium | ознаки сканування: багато різних портів і висока ентропія |
| 2026-05-31 22:47:00 | host_src:127.0.0.1:scan | medium | ознаки сканування: багато різних портів і висока ентропія; сплеск байтів z=2.12 |
| 2026-05-31 23:25:00 | host_src:127.0.0.1:bytes | medium | сплеск байтів z=2.92 |

Отримані результати показують, що система може використовуватися як інструмент оперативного моніторингу, надаючи оператору короткі узагальнені сигнали, а детальні дані залишаючи доступними у подіях і агрегатах.

Перевірка веб-інтерфейсу показала, що інциденти, події, сповіщення та агреговані дані коректно відображаються у браузері на стороні хост-системи. На сторінці огляду відображаються графіки часових рядів, що дозволяє швидко помітити аномальні періоди. На сторінці інцидентів видно тип scan, ключ і пояснення причини. Журнал сповіщень відображає факти відправки повідомлень.

# ВИСНОВКИ

Під час підготовки бакалаврської роботи було отримано наступні результати:

1. Проаналізовано предметну область мережевої телеметрії та підхід потокового обліку, показано роль потокових записів для довготривалого моніторингу і виявлення відхилень у трафіку.

2. Розглянуто формати NetFlow, IPFIX та sFlow, а також практичні особливості формування потоків і вплив параметрів експорту на якість подальшого аналізу.

3. Виконано огляд наявних інструментів і підходів для збору, архівації та обробки flow-даних а також підходів до збереження метрик і подій та їх візуалізації.

4. Сформульовано постановку задачі та вимоги до модуля, обґрунтовано архітектуру з розподілом ролей між вузлом збору і вузлом аналітики з можливістю обчислювального прискорення на центральному вузлі.

5. Розроблено алгоритмічне забезпечення модуля, яке охоплює прийом і нормалізацію flow-даних, агрегацію у часових вікнах, побудову ознак, виявлення аномалій, формування подій, об’єднання повторів та керування сповіщеннями з паузою між повторними повідомленнями.

6. Реалізовано прототип конвеєра обробки, який накопичує дані в базі та експортує результати у файли для подальшого аналізу і включення в звіт.

7. Реалізовано веб-інтерфейс для подання результатів на основі FastAPI, Uvicorn та Jinja2 з відображенням таблиць і графіків та сторінок огляду інцидентів і сповіщень.

8. Проведено тестування модуля на контрольованих сценаріях нормального трафіку та аномальної активності.

9. За результатами експериментів базовий детектор сформував 5 подій, модуль на основі машинного навчання сформував 2 події, а після об’єднання повторів отримано 3 інциденти і 3 сповіщення, що підтвердило працездатність підходу та керованість повідомлень.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. RFC 3954. Cisco Systems NetFlow Services Export Version 9. URL: https://datatracker.ietf.org/doc/html/rfc3954 (дата звернення: 09.03.2026).
2. RFC 7011. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. URL: https://datatracker.ietf.org /doc/html/rfc7011 (дата звернення: 09.03.2026).
3. Münz G., Carle G. Real-time Analysis of Flow Data for Network Attack Detection. URL: https://www.net.in.tum.de/fileadmin /TUM/members/muenz/documents/muenz07real-time-analysis.pdf (дата звернення: 09.03.2026).
4. RFC 7012. Information Model for IP Flow Information Export (IPFIX). URL: https://datatracker.ietf.org/doc/html/rfc7012 (дата звернення: 09.03.2026).
5. Wu W. та ін. G-NetMon: A GPU-accelerated Network Performance Monitoring System. URL: https://arxiv.org/abs/1108.1785 (дата звернення: 09.03.2026).
6. Deri L. Realtime High-Speed Network Traffic Monitoring Using ntopng. URL: https://www.usenix.org/system/files/conference/lisa14/lisa14-paper-deri.pdf (дата звернення: 09.03.2026).
7. Erlacher F., Dressler F. FIXIDS: A High-Speed Signature-based Flow Intrusion Detection System. URL: https://www.ccs-labs.org/bib/erlacher
8. 2018fixids/erlacher2018fixids.pdf (дата звернення: 09.03.2026).
9. IPFIX Information Elements registry. IANA. URL: https://www.iana.org
10. /assignments/ipfix/ipfix-information-elements.csv (дата звернення: 09.03.2026).
11. sFlow Version 5 specification. sFlow.org. URL: https://sflow.org/sflow_version_5.txt (дата звернення: 09.03.2026).
12. Conn log documentation. Zeek Project. URL: https://docs. zeek.org/en/master/logs/conn.html (дата звернення: 09.03.2026).

13. Argus manual. QoSient. URL: https://qosient.com /argus/man/man8/argus.8.pdf (дата звернення: 09.03.2026).

14. RFC 7015. Flow Aggregation for the IPFIX Protocol. URL: https://datatracker.ietf.org/doc/html/rfc7015 (дата звернення: 09.03.2026).

15. From Packet Capture to Data Analysis with NetFlow and IPFIX : tutorial. Univ. of Twente. URL: https://ris.utwente.nl/ws/files/6519118/tutorial.pdf (дата звернення: 09.03.2026).

16. nfcapd — netflow capture daemon : Ubuntu manpages. URL: https://manpages.ubuntu.com/manpages/bionic/man1/nfcapd.1.html (дата звернення: 09.03.2026).

17. softflowd(8) — softflowd : Debian manpages. URL: https://manpages.debian.org/unstable/softflowd/softflowd.8.en.html (дата звернення: 09.03.2026).

18. YAF documentation. CERT NetSA. URL: https://tools.netsa.cert.org /yaf/docs.html (дата звернення: 09.03.2026).

19. SiLK documentation. CERT NetSA. URL: https://tools.netsa.cert.org /silk/docs.html (дата звернення: 09.03.2026).

20. Data model. Prometheus documentation. URL: https://prometheus.io

21. /docs/concepts/data_model/ (дата звернення: 09.03.2026).

22. Ingesting time series data. Elastic Docs. URL: https://www.elastic.co

23. /docs/manage-data/ingest/ingesting-timeseries-data (дата звернення: 09.03.2026).

24. Kibana: Visualize, explore, and manage data in Elasticsearch. Elastic. URL: https://www.elastic.co/kibana (дата звернення: 09.03.2026).

25. A flow-based method for abnormal network traffic detection. URL: https://scispace.com/pdf/a-flow-based-method-for-abnormal-network-traffic-detection-3egspnwrzd.pdf (дата звернення: 09.03.2026).

26. NetFlow Based Framework for Identifying Anomalous End User Nodes. URL: https://ccdcoe.org/uploads/2020/07/NetFlow-Based-Framework-for-Identifying-Anomalous-End-User-Nodes-final-PDF.pdf (дата звернення: 09.03.2026).

27. Hofstede R. та ін. Towards real-time intrusion detection for NetFlow and IPFIX. URL: https://opendl.ifip-tc6.org/db/conf/cnsm/cnsm2013/HofstedeBSP13.pdf (дата звернення: 09.03.2026).

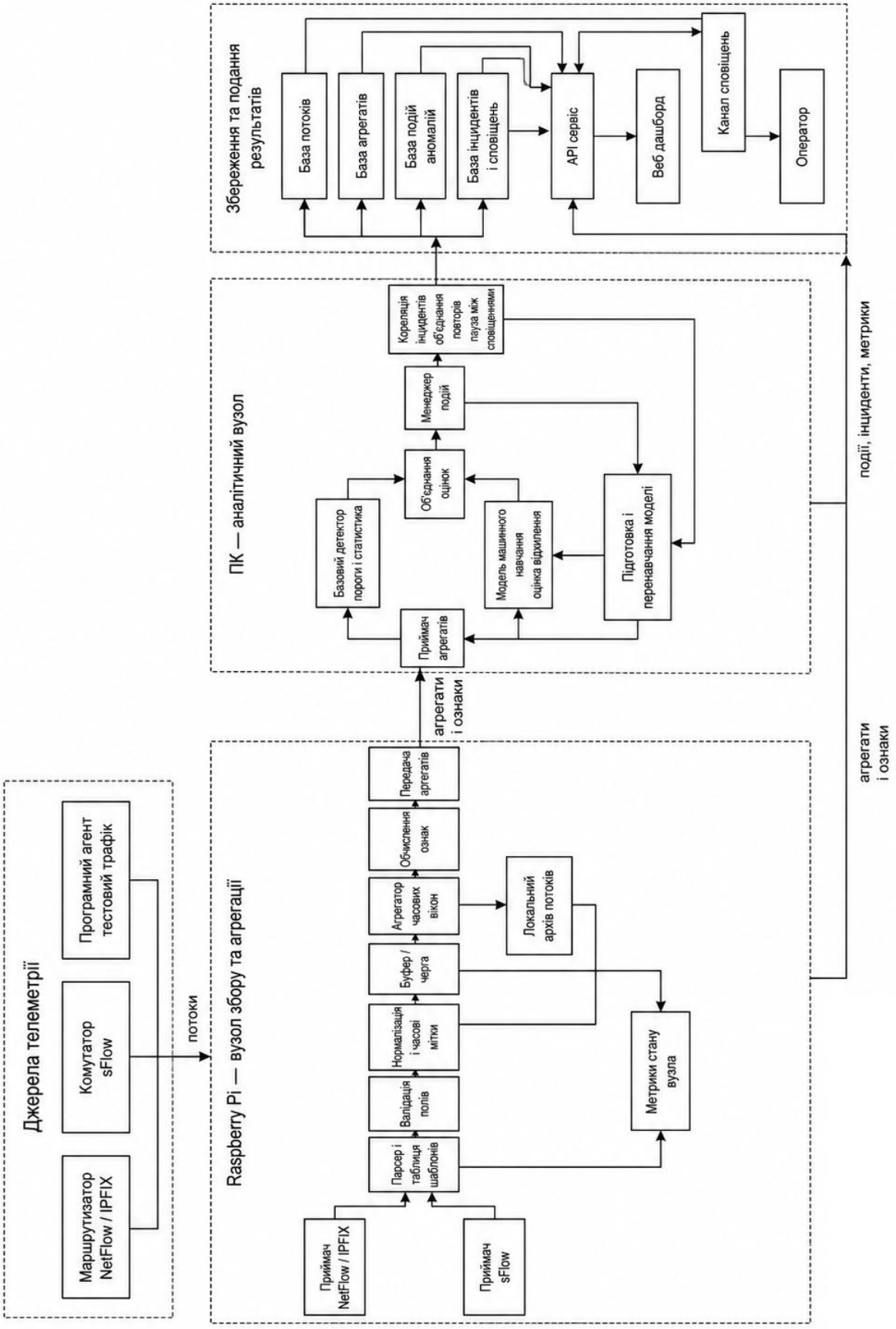
28. Steeg M. Flow based DDoS detection on Cisco IOS : master thesis. URL: https://essay.utwente.nl/65875/1/steeg_MA_ewi.pdf (дата звернення: 09.03.2026).

29. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп'яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

30. Колодюк Д. О. Програмно-апаратний модуль мережевої телеметрії та виявлення аномалій // Традиційні та інноваційні підходи до наукових досліджень : матеріали XI Міжнар. наук. конф., м. Луцьк, 19 черв. 2026 р. Луцьк, 2026.

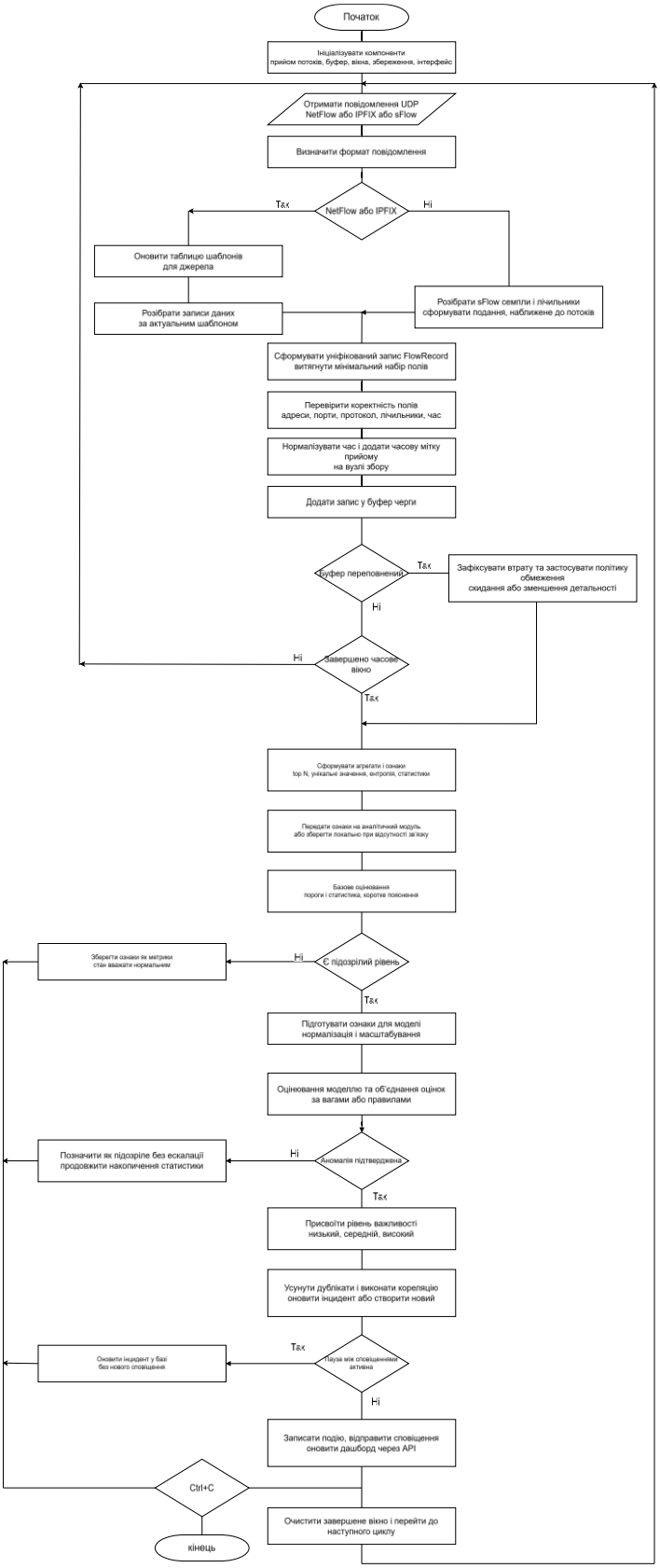
Додаток А

Архітектура програмно-апаратного модуля



Додаток Б

Загальний алгоритм функціонування програмно-апаратного модуля мережевої телеметрії та виявлення аномалій



# Додаток В

## Структура файлів проекту та їх взаємодія

```
/root/net-telemetry/
├─ telemetry.db
├─ import_flows.py
├─ aggregate_windows.py
├─ baseline_detect.py
├─ ml_detect.py
├─ event_manager.py
├─ export_results.py
├─ web_app.py
├─ out/
|  ├─ flows_sample.csv
|  ├─ aggregates_top_by_ports.csv
|  ├─ aggregates_top_by_entropy.csv
|  ├─ events.csv
|  ├─ incidents.csv
|  └─ alerts.csv
├─ templates/
|  ├─ base.html
|  ├─ home.html
|  ├─ incidents.html
|  ├─ alerts.html
|  ├─ events.html
|  └─ aggregates.html
└─ static/
   └─ style.css
```

В.2 Призначення основних файлів

telemetry.db — Локальна база даних SQLite. Містить таблиці flow_record, time_window_aggregate, anomaly_event, incident, alert_log.

import_flows.py — Імпорт потоків з файлів nfcapd у таблицю flow_record.

aggregate_windows.py — Агрегація потоків у часові вікна 60 секунд і формування ознак у таблицю time_window_aggregate.

baseline_detect.py — Базовий детектор на основі порогів і rolling порівняння з попередніми вікнами. Додає події у anomaly_event.
ml_detect.py — Детектор на основі машинного навчання без зовнішніх бібліотек. Виконує безнаглядне оцінювання відхилення ознак і додає події у anomaly_event з позначкою detector=ml.
event_manager.py — Менеджер інцидентів. Об'єднує повторні події в інциденти, застосовує паузу між сповіщеннями, веде alert_log.
export_results.py — Експортує події, інциденти, сповіщення та агрегати у CSV файли в каталозі out.
web_app.py — Веб-інтерфейс на FastAPI. Відображає дані з telemetry.db у вигляді сторінок та графіків.
templates/*.html — HTML шаблони сторінок веб-інтерфейсу.
static/style.css — Файл оформлення веб-інтерфейсу.
out/*.csv — Результати для звіту: витяги подій, інцидентів, сповіщень та приклади агрегатів.

В.3 Логіка взаємодії модулів
В.3.1 Ланцюжок обробки даних
1) Файли потоків nfcapd.* у /var/cache/nfdump
2) import_flows.py → таблиця flow_record
3) aggregate_windows.py → таблиця time_window_aggregate
4) baseline_detect.py і ml_detect.py → таблиця anomaly_event
5) event_manager.py → таблиці incident і alert_log
6) export_results.py → файли CSV у каталозі out
7) web_app.py → відображення даних у браузері

В.3.2 Взаємодія через базу даних
Зв'язок між модулем агрегації, детекторами, менеджером інцидентів та веб-інтерфейсом реалізовано через спільну базу telemetry.db.
В.3.3 Зв'язок модулів і таблиць

| Модуль | Читає | Пише |
|---|---|---|
| import_flows.py | файли nfcapd.* через nfdump | flow_record |
| aggregate_windows.py | flow_record | time_window_aggregate |
| baseline_detect.py | time_window_aggregate | anomaly_event |
| ml_detect.py | time_window_aggregate, anomaly_event | anomaly_event |
| event_manager.py | anomaly_event | incident, incident_event, alert_log |
| export_results.py | усі таблиці | out/*.csv |
| web_app.py | incident, alert_log, anomaly_event, time_window_aggregate | не записує |

Додаток Г
Апробація отриманих результатів

# ДОВІДКА

## ПРО ПРИЙНЯТТЯ НАУКОВОЇ РОБОТИ ДО ПУБЛІКАЦІЇ

10.06.2026

**Шановний(і) авторе(и):**

Колодюк Дмитро Олексійович ,

Організаційний комітет з радістю повідомляє, що наукову роботу «ПРОГРАМНО-АПАРАТНИЙ МОДУЛЬ МЕРЕЖЕВОЇ ТЕЛЕМЕТРІЇ ТА ВИЯВЛЕННЯ АНОМАЛІЙ» прийнято до публікації в збірнику за матеріалами XI Міжнародної наукової конференції «Традиційні та інноваційні підходи до наукових досліджень» (19.06.2026; м. Луцьк, Україна).

**Опублікована робота буде доступна з 19.06.2026 за посиланням:**

https://archives.mcnd.org.ua/index.php/conference-proceeding/issue/view/19.06.2026

. . . . . .

Електронні сертифікати учасників конференції будуть доступні з 19 червня.

*Конференцію зареєстровано в базі даних науково-технічних заходів УкрІНТЕІ (Посвідчення № 176 від 26.01.2026). Матеріали конференції знаходитимуться в відкритому доступі (Open Access) на умовах ліцензії CC BY-SA 4.0 International.*

З повагою,

Віце-президент МЦНД
Голова оргкомітету конференції
**НАСТАСІЯ РАБЕЙ**

# ПРОГРАМНО-АПАРАТНИЙ МОДУЛЬ МЕРЕЖЕВОЇ ТЕЛЕМЕТРІЇ ТА ВИЯВЛЕННЯ АНОМАЛІЙ

**Колодюк Дмитро Олексійович**
Бакалавр спеціальності 122 – Комп'ютерні науки
Західноукраїнський національний університет, Україна
**Науковий керівник: Осолінський Олександр Романович**
ORCID ID: 0000-0002-0136-395X
к.т.н., доцент кафедри інформаційно-обчислювальних систем і управління
Західноукраїнський національний університет
*Україна*

## Вступ

Сучасні комп'ютерні мережі забезпечують роботу підприємств, навчальних закладів, хмарних сервісів і систем електронної взаємодії. Зростання кількості сервісів та мережевих пристроїв призводить до постійної зміни профілю трафіку, тому адміністратору важливо своєчасно бачити не лише загальне навантаження та нетипові взаємодії між вузлами. Особливої уваги потребують ситуації, пов'язані з скануванням портів, різкими сплесками передавання даних, нестандартними напрямками комунікації та ознаками атак на доступність.

Повне захоплення пакетів дає детальну інформацію, але потребує значних ресурсів для зберігання та аналізу. Тому для тривалого спостереження доцільно застосовувати потокову телеметрію, в якій мережеві взаємодії подаються компактними записами про потоки. До таких підходів належать NetFlow, IPFIX та sFlow [1–3]. Вони дають змогу будувати часові ряди метрик, обчислювати ознаки трафіку та використовувати їх для виявлення аномалій.

Отже є потреба розробки модуля мережевої телеметрії та виявлення аномалій на основі даних про потоки трафіку, який забезпечує збір, агрегацію, аналіз, збереження результатів і подання їх оператору у зручному вигляді.

## Архітектура та реалізація модуля

Запропоноване рішення побудовано, як розподілений конвеєр обробки. Вузол збору на базі Raspberry Pi розміщується поряд з мережевим обладнанням і виконує прийом, перевірку, буферизацію та первинне зведення потоків. Аналітичний вузол на персональному комп'ютері з GPU приймає підготовлені ознаки, виконує виявлення відхилень, формує події, інциденти та сповіщення. Такий поділ ролей зменшує навантаження на периферійний пристрій і залишає ресурсоємні обчислення центральному вузлу.


Рис. 1. **Структурна схема програмно-апаратного рішення для збору потокової телеметрії**

В реалізованому прототипі формування потоків виконано за допомогою softflowd, а прийом і буферизацію - за допомогою nfcapd, який зберігає записи у файлах з ротацією [4, 5]. Далі утиліта nfdump перетворює накопичені дані в машинно-читаний формат, після чого скрипт імпорту записує їх у базу SQLite. Це спрощує відтворення експериментів, тому що вихідні файли потоків можна повторно обробляти під час перевірки алгоритмів.

Після імпорту дані агрегуються в часові вікна тривалістю 60 секунд. Для кожного вікна обчислюються кількість потоків, сума байтів і пакетів, кількість різних портів призначення, а також ентропія розподілу портів. Кількість портів і ентропія є важливими ознаками сканування, оскільки під час такої активності зростає різноманітність цільових портів і розпорошеність розподілу.

Виявлення аномалій реалізовано двома взаємодоповнювальними способами. Базовий детектор порівнює поточне вікно з попередніми та застосовує явне правило для сканування. Модуль машинного навчання виконує

безнаглядне оцінювання відхилення ознак від типової поведінки та формує власну оцінку аномальності. Після цього менеджер подій об'єднує повторні спрацювання в інциденти й застосовує паузу між повторними повідомленнями, щоб не перевантажувати оператора.

### Інтерфейс та результати тестування

Для перегляду результатів створено веб-інтерфейс на основі FastAPI, Uvicorn та Jinja2. Він працює поверх бази і відображає оглядові графіки, список інцидентів, журнал сповіщень, первинні події детекторів та агреговані ознаки у часових вікнах.

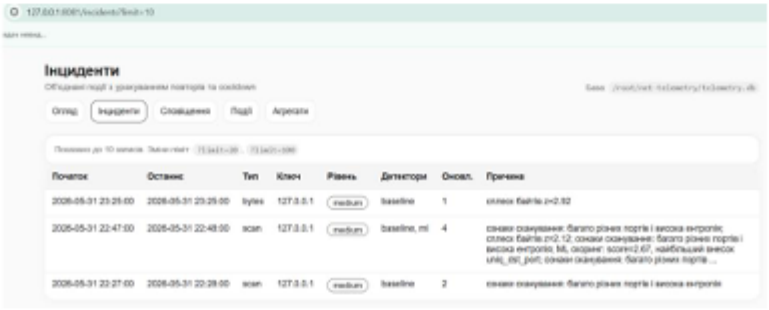



Рис. 2. **Сторінка інцидентів з результатами виявлення аномалій та поясненням причин**

Тестування виконано за двома сценаріями. Для нормального трафіку використано iperf3, який генерує контрольований обмін даними між клієнтом і сервером. Для аномальної активності використано nmap, що виконує сканування TCP-портів.

За результатами тестового запуску базовий детектор сформував 5 подій, а модуль машинного навчання - 2 події, які відповідали періодам сканування. Після об'єднання повторних спрацювань отримано 3 узагальнені інциденти і 3 сповіщення. Це показало, що система фіксує аномалії та зменшує кількість дубльованих повідомлень не перевантажуючи оператора.

Окремо в інтерфейсі передбачено журнал сповіщень. Він показує час повідомлення, підпис інциденту, рівень важливості та коротке пояснення причини.

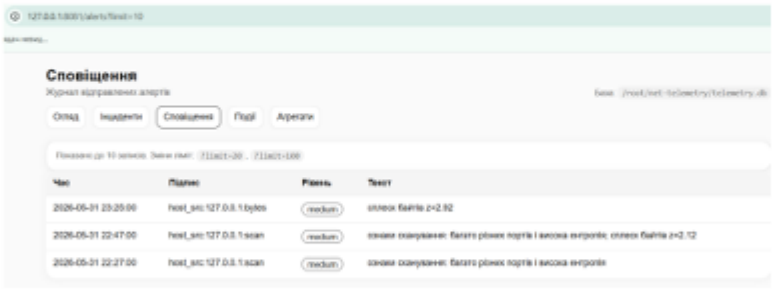



Рис. 3. **Сторінка сповіщень з журналом повідомлень про інциденти**

**Висновки.** Було реалізовано програмно-апаратний модуль мережевої телеметрії який поєднує модуль Raspberry Pi в якості модуля збору мережевого трафіку та можливості GPU де можна розширювати обчислювальну частину, налаштовувати пороги та додавати нові ознаки.

**Список використаних джерел:**


1. RFC 3954. Cisco Systems NetFlow Services Export Version 9. URL: https://datatracker.ietf.org/doc/html/rfc3954.
2. RFC 7011. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. URL: https://datatracker.ietf.org/doc/html/rfc7011.
3. sFlow Version 5 specification. sFlow.org. URL: https://sflow.org/sflow_version_5.txt.
4. nfcapd — netflow capture daemon : Ubuntu manpages. URL: https://manpages.ubuntu.com/manpages/bionic/man1/nfcapd.1.html.
5. softflowd(8) — softflowd : Debian manpages. URL: https://manpages.debian.org/unstable/softflowd/softflowd.8.en.html.