ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА
(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій
(повне найменування факультету)

Кафедра математичного забезпечення комп'ютерних систем
(повна назва кафедри)

Кваліфікаційна робота

на здобуття рівня вищої освіти «бакалавр»
(рівень вищої освіти)

Програмно-апаратний комплекс аналізу інтегральних показників роботи мережевого пристрою для розробки систем виявлення мережевих аномалій

(тема дипломної роботи українською мовою)

Hardware and software complex for analyzing integral performance indicators of a network device for developing network anomaly detection systems

(тема дипломної роботи англійською мовою)

Виконав: здобувач ВО денної форми навчання
спеціальності 123 – Комп'ютерна інженерія
(код, назва спеціальності)
Освітня програма «Комп'ютерна інженерія»
(назва освітньої програми)
Малємаєв Михайло Володимирович
(прізвище, ім'я, по-батькові здобувача)

Керівник Максимов Олександр Семенович
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент доц. Шестопалов Сергій Вікторович
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
_____
№ ___ від ____.____. 2025 р.

Завідувач кафедри
_____ Євгеній МАЛАХОВ
(підпис) (прізвище,ім'я)

Захищено на засіданні ЕК № ___5___
протокол № __від ____.____.2025 р.

Оцінка_____/_____/_____
(за національною шкалою/шкалою ECTS/ бали)
Голова ЕК
_____ Світлана АНТОЩУК
(підпис) (прізвище, ім'я)

Одеса 2025

АНОТАЦІЯ

У дослідженні запропоновано апаратно-програмний комплекс, що перетворює строкатий потік телеметричних даних мережевого обладнання на єдиний інтегральний індекс стану. Актуальність роботи пояснюється вибуховим зростанням трафіку та ускладненням кібератак: класичний моніторинг за окремими лічильниками втрачає чутливість, тоді як оператору необхідне «одне число правди» про здоров'я кожного вузла . Суть підходу полягає у поєднанні трафікових, ресурсних, ентропійних, енергетичних і термальних показників, які після нормалізації методом z-score агрегуються ваговою схемою, адаптованою до довгострокових коливань навантаження. Математична модель інтегрального індексу доповнена кореляційним фільтром, що виявляє латентні відхилення раніше, ніж вони стають критичними для сервісів. Прототип реалізовано мовою Пайтон у push-архітектурі Telegraf-Kafka-Flink-ElasticSearch; він інтегрується у стеки Prometheus-Grafana або Elastic Stack без зміни мережевої топології . Експериментальна перевірка на лабораторному стенді з реалістичним трафіком підтвердила високу точність і повноту детекції, що скорочує час реакції операторського центру з хвилин до секунд. Наукова новизна полягає у самоадаптивному поєднанні класичних та ентропійних метрик у динамічному ваговому векторі, а практична цінність – у можливості впровадження комплексу в існуючі мережі без капітальних витрат.

ABSTRACT

The study proposes a hardware and software complex that converts a diverse stream of telemetry data from network equipment into a single integrated status index. The relevance of the work is explained by the explosive growth of traffic and the complexity of cyberattacks: classical monitoring of individual meters loses sensitivity, while the operator needs “one number of truth” about the health of each node. The approach is based on a combination of traffic, resource, entropy, energy, and thermal indicators, which, after normalization by the z-score method, are aggregated by a weighting scheme adapted to long-term load fluctuations. The mathematical model of the integral index is complemented by a correlation filter that detects latent deviations before they become critical for services. The prototype is implemented in Python in the Telegraf-Kafka-Flink-ElasticSearch push architecture; it integrates into Prometheus-Grafana or Elastic Stack stacks without changing the network topology. Experimental testing on a laboratory bench with realistic traffic confirmed the high accuracy and completeness of detection, which reduces the response time of the call center from minutes to seconds. The scientific novelty lies in the self-adaptive combination of classical and entropy metrics in a dynamic weight vector, and the practical value lies in the possibility of implementing the complex in existing networks without capital expenditure

ЗМІСТ

Стор.

СПИСОК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

ARP – Address Resolution Protocol – протокол визначення адрес

ASIC – Application-Specific Integrated Circuit – спеціалізована інтегральна схема

BPF – Berkeley Packet Filter – берклійський фільтр пакетів

CPU – Central Processing Unit – центральний процесор

DDoS – Distributed Denial of Service – розподілена атака відмови в обслуговуванні

DNS – Domain Name System – система доменних імен

HTTP – HyperText Transfer Protocol – протокол передавання гіпертексту

IP – Internet Protocol – мережевий протокол

IPFIX – IP Flow Information Export – експорт інформації про IP-потоки

k-Means – k-Means Clustering – метод кластеризації *k*-середніх

MIB – Management Information Base – база керувальної інформації

NetFlow – Network Flow – протокол експорту мережевих потоків

NIC – Network Interface Controller – мережева карта

NTP – Network Time Protocol – протокол мережного часу

RFC – Request for Comments – документ стандарту IETF

SNMP – Simple Network Management Protocol – протокол простого керування мережею

sFlow – sampled Flow – протокол вибіркового збору трафіку

TCP – Transmission Control Protocol – протокол керування передачею

UDP – User Datagram Protocol – протокол дейтаграм

URL – Uniform Resource Locator – уніфікований покажчик ресурсу

ВСТУП

У сучасних корпоративних і телекомунікаційних мережах стрімке зростання обсягів даних і складності обладнання супроводжується еволюцією кібератак, які дедалі частіше маскують зловмисний трафік під легітимний із малопомітними поведінковими відхиленнями. Традиційні методи моніторингу, що базуються на аналізі окремих параметрів, таких як лічильники SNMP чи метрики продуктивності, втрачають ефективність, оскільки короткочасні аномалії губляться в потоці тисяч телеметричних показників. Це робить актуальним розробку інтегрованого підходу, який об’єднує багатовимірні телеметричні дані в єдиний адаптивний індекс для оперативної оцінки стану мережі та виявлення аномалій у реальному часі.

Запропонований підхід ґрунтується на створенні апаратно-програмного комплексу, який використовує сучасні методи збору телеметрії (SNMP, NetFlow, sFlow, packet capture) та обробки даних для формування інтегрального показника. Цей показник враховує нормалізовані та зважені метрики, що відображають стан мережевих пристроїв, і дозволяє виявляти аномалії з високими показниками точності (precision) і повноти (recall). Комплекс реалізовано на основі push-архітектури з інтеграцією в ElasticSearch, що забезпечує гнучкість, масштабованість і сумісність з існуючою мережевою інфраструктурою без додаткових витрат на обладнання. Безпека системи забезпечується завдяки автентифікації та шифруванню даних, що захищає телеметричні потоки від несанкціонованого доступу.

Актуальність роботи зумовлена необхідністю підвищення ефективності моніторингу складних мереж і своєчасного виявлення аномалій, що можуть свідчити про кібератаки чи збої обладнання.

Метою роботи є розробка та експериментальна перевірка апаратно-програмного комплексу для обчислення інтегрального показника стану мережі в реальному часі та виявлення аномалій на основі цього показника.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1. Проаналізувати сучасні методи збору та обробки мережевої телеметрії

(SNMP, NetFlow, sFlow, packet capture) і програмні платформи моніторингу.

2. Обґрунтувати вибір первинних метрик для формування інтегрального показника.
3. Розробити математичну модель нормалізації та вагового агрегування метрик.
4. Спроєктувати та реалізувати прототип детектора аномалій на основі push-архітектури з інтеграцією в ElasticSearch.
5. Забезпечити безпеку системи шляхом впровадження автентифікації та шифрування телеметричних даних.
6. Провести натурні випробування комплексу на стенді, що імітує роботу мережевих пристроїв рівнів core, distribution і access.
7. Оцінити ефективність роботи комплексу за показниками precision і recall у сценаріях виявлення аномалій.

1 ОГЛЯД ІСНУЮЧИХ МЕТОДІВ АНАЛІЗУ РОБОТИ МЕРЕЖЕВИХ ПРИСТРОЇВ

У сучасних інформаційно-комунікаційних системах забезпечення стабільної, високопродуктивної та безпечної роботи мережевих пристроїв є однією з ключових задач. Застаріння архітектур, зростання навантаження та збільшення кількості користувачів вимагають постійного моніторингу й аналізу стану мережевого обладнання. У цьому розділі розглядаються основні підходи та інструменти, застосовувані для збору, обробки та інтерпретації показників ефективності та стану мережевих пристроїв. Огляд побудований на чотирьох підрозділах: класифікація методів аналізу, методи збору й обробки даних, огляд популярних інструментів і платформ, а також порівняльний аналіз переваг і недоліків існуючих підходів.

1.1 Класифікація методів аналізу роботи мережевих пристроїв

Методи аналізу роботи мережевого обладнання можна умовно розділити на кілька груп залежно від принципу збору даних та алгоритмів обробки.

Статистичні методи. Статистичні підходи базуються на зборі та агрегуванні метрик мережевого трафіку та показників продуктивності (завантаження процесора, використання пам’яті, кількість пакетів, швидкість передавання тощо). Далі за допомогою статистичних моделей (середнє значення, дисперсія, ковзні середні) виявляють аномалії чи тренди. Перевага таких методів – відносна простота реалізації та зрозумілість інтерпретації результатів. Однак, вони добре працюють лише тоді, коли мережевий трафік має стабільні статистичні характеристики, і вимагають ретельного налаштування порогів детекції.

Сигнатурні (підписні) методи. Подібно до систем виявлення вторгнень (IDS), які використовують шаблони відомих атак, сигнатурні методи аналізу

мережевих пристроїв базуються на заздалегідь визначених «підписах» некоректної чи небажаної поведінки обладнання чи трафіку (наприклад, надмірно велика кількість помилок інтерфейсу, DNS-флуди, SYN-флуди). Сигнатури можуть описувати конкретні шаблони пакетів, профілі нестабільного використання ресурсів тощо. Перевага: висока точність у виявленні відомих проблем (наприклад, DoS-атак чи аномалій у роботі інтерфейсів). Недолік: неефективність у випадку нових, ще невідомих аномалій або нетипових конфігурацій мережі, адже для них немає готових сигнатур.

Поведенкові (анімаційні) методи. Ці підходи фокусуються на визначенні поведінкових паттернів нормальної роботи мережевого пристрою. Будується модель нормальної діяльності (наприклад, профіль використання CPU, пам’яті, постійний потік NetFlow-правил), а потім у процесі моніторингу порівнюється фактична поведінка з побудованою моделлю. Якщо виявляється суттєве відхилення — формується підозра на аномалію. Недолік: довгий етап навчання системи (потрібно зібрати велику кількість «чистих» даних), складність побудови гнучких, адаптивних моделей, а також висока обчислювальна вартість при обробці великих обсягів даних.

Моделі машинного навчання та штучного інтелекту. До цієї групи належать різноманітні алгоритми класифікації й кластеризації (Support Vector Machines, k-Means, Random Forest, Neural Networks), а також більш сучасні методи глибокого навчання (LSTM, Autoencoders). Їхня суть полягає в тому, щоб на основі багатовимірних ознак (пакети, байти, кількість унікальних адрес, анклави портів тощо) алгоритмічним шляхом визначати аномальні випадки або категоризувати різні типи подій. Такі методи здатні виявляти складні нелінійні залежності, однак потребують значних обчислювальних ресурсів, ретельної підготовки та маркування даних для навчання.

Системи на основі правил. Прості підходи, у яких адміністратор вручну прописує набір правил (наприклад, «якщо кількість CRC-ошибок перевищує X за Y хвилин – реконфігурація порту»). Недолік – низька адаптивність: кожен

новий сценарій вимагає ручного оновлення правил. Часто такі системи використовують як первинний захід контролю та тригеру для більш складних механізмів аналізу.

На практиці найчастіше застосовують комбінації перелічених підходів: статистичні методи й сигнатурне виявлення використовуються для простих «трафікових» перевірок, а для складніших випадків (наприклад, виявлення хитромудрих DDoS-атак чи «zero-day» помилок у роботі устаткування) – машинне навчання.

1.2 Методи збору та обробки даних про стан мережевих пристроїв

Для подальшого аналізу й виявлення аномалій необхідно спочатку збирати дані про поточний стан маршрутизаторів, комутаторів, серверів та інших активних елементів мережі. Нижче наведено поширені стандартизовані підходи:

1) Simple Network Management Protocol (SNMP). SNMP [1] – один із найстаріших та найбільш розповсюджених протоколів для віддаленого моніторингу мережевих приладів. Він дозволяє отримувати різні MIB-показники (Management Information Base) – CPU-load, використання пам’яті, кількість інтерфейсних помилок, статистику переданих байт/пакетів тощо.

Переваги:

– Універсальність. Майже кожен виробник мережевого обладнання підтримує SNMP.

– Легкість інтеграції. В існуючі системи моніторингу (Zabbix, Nagios, Cacti тощо) досить додати шаблон SNMP, і система почне зчитувати потрібні показники.

– Розширюваність. Завдяки підтримці приватних MIB-виробниками можна отримувати навіть «нестандартні» дані (наприклад, глибока статистика ASIC-рішень, особливості балансерів L4).

Недоліки:

– Навантаження на контролер при дуже частому опитуванні (наприклад, раз на секунду при великій кількості пристроїв).

– Затримки оновлення статистики. Зазвичай «polling interval» не менший 15–30 секунд, що не дає можливості виявити короткі пікові сплески.

– Безпека. SNMPv1/v2 працюють з відкритими текстовими паролями. Використання SNMPv3 дещо ускладнює налаштування (шифрування, автентифікація), але без нього є ризик втрати конфіденційності.

2) NetFlow / sFlow. NetFlow [2, 3] – пропрієтарний протокол (розроблений Cisco), призначений для збору інформації про потоки мережевого трафіку (flow records). Кожен flow описує унікальну п’ятипелюсткову «параметризу» (IP-адреса-джерело, IP-адреса-призначення, порти, протокол), тривалість сесії, кількість байт/пакетів. sFlow – відкритий протокол для семплування та аналізу руху. sFlow-пристрої відбирають статистичні зразки пакетів (1 з N), іноді відображають статистику інтерфейсу (наприклад, кількість dropped-пакетів), і відправляють їх на колектор.

Переваги NetFlow / sFlow:

– Глибокий аналіз трафіку. Можливість визначити топологію потоків, загальні політики QoS, джерела DDoS тощо.

– Низька мережна «вартість». sFlow надсилає семпли, а NetFlow збирає інформацію на самому пристрої і відсилає агреговані записи; порівняно з постійним інспектуванням всіх пакетів, це дає менше навантаження.

– Інтеграція з SIEM (Security Information and Event Management). Коли є NetFlow/IPFIX-дані, їх можна відразу корелювати з подіями безпеки.

Недоліки:

– Нестача деталей. NetFlow, особливо у режимі v5 чи v9, не зберігає вміст пакетів, тому складно аналізувати вміст трафіку (payload).

– Калібрування семплів (sFlow). Щоб отримати репрезентативні дані, потрібно коригувати «sampling rate» у залежності від швидкості лінка.

3) Протокольний аналіз з використанням TShark / Wireshark / PyShark [4]. Інструменти мережевого аналізу (Wireshark, TShark, PyShark) здійснюють повноцінний парсинг кожного пакета, переданого пристроєм, що дає змогу аналізувати заголовки L2–L7, побудувати статистику по типам протоколів, а також деталізувати збої, Retransmission, CRC-ошибки тощо.

Переваги:

– Максимальна деталізація. Можна розглянути будь-який фрагмент пакета, побачити відхилення на прикладному рівні (HTTP, DNS, SMB).

– Гнучкість. За допомогою фільтрів BSD (BPF) можна відсікати непотрібний трафік, відбирати лише UDP, TCP, ICMP, ARP тощо.

– Можливість автоматизації. З використанням бібліотек (PyShark, scapy, tshark CLI) можна створити програму збору та обробки трафіка в режимі реального часу.

Недоліки.

– Велике навантаження. Захоплення кожного пакета на великих лініях (1 Gbps та вище) може бути неможливим без апгрейду апаратного забезпечення (NIC з підтримкою ping buffer, hardware ring, offload).

– Складність масштабування. Протокольний аналіз є дискретним та ресурсозатратним, особливо якщо потрібно аналізувати десятки-сотні гігабіт у секунду.

– Шум інформації. Величезний обсяг даних (мільйони пакетів) потребує ефективних фільтрів та алгоритмів відсіювання «шуму» для виділення суттєвих аномалій.

4) Логування подій та SNMP Traps. SNMP Traps – повідомлення від мережевого пристрою до системи моніторингу про події (наприклад, перевищення порогу CPU, зміна стану інтерфейсу, критичні помилки). Syslog [5] – стандарт для відправки логів через мережу (UDP/TCP) на централізований сервер. Мережеві пристрої, постачальники ОС (Cisco IOS, Juniper JunOS, Linux [23]) генерують syslog-повідомлення про події: boot, shutdown, error, interface up/down.

Переваги.

– Екологічна інтеграція. Багато систем «готові» приймати SNMP Traps та Syslog (наприклад, ELK Stack, Graylog, Splunk).

– Живі події. Перехоплюються та оброблюються в реальному часі.

– Гнучкість налаштування. Адміністратор може вказати, які події вважати критичними та під які правила (priority, facility) відправляти.

Недоліки:

– Неявність метрик. SNMP Traps та Syslog переважно несуть опис події, але не завжди дають повну картину кількісних метрик (наприклад, рівень CPU в %).

– Шум і флуд. Без правильної фільтрації та кореляції логів легко «тонути» в повідомленнях, особливо в великих мережах.

1.3 Інструменти та програмні платформи

На ринку представлено широкий спектр готових рішень для збору, аналізу та моніторингу мережевого трафіку, системного стану пристроїв і поведінкових метрик. Серед найпопулярніших інструментів – Wireshark, TShark і PyShark. Wireshark є графічним аналізатором мережевого трафіку, що працює в реальному часі або з попередньо записаними .pcap-файлами. Він дозволяє детально аналізувати заголовки пакетів, розпізнавати понад 2000 протоколів та виявляти кореляції між трафіком. TShark, консольна версія Wireshark, забезпечує автоматизоване збирання і фільтрацію трафіку через сценарії, а PyShark, у свою чергу, виступає як Python-обгортка для TShark і дозволяє легко інтегрувати мережевий аналіз у власні скрипти. Ці інструменти відзначаються широкою підтримкою протоколів, можливістю глибокого аналізу мережевого рівня від L2 до L7, та активною спільнотою. Водночас вони вимагають наявності сумісного мережевого інтерфейсу для захоплення

трафіку і не надто масштабовані для аналізу багатогігабітного трафіку в промислових умовах.

Zabbix [6] є потужною open-source системою моніторингу, що здатна отримувати дані через SNMP, ICMP, IPMI та власного агента, а також підтримує NetFlow за допомогою проксі або плагінів. Вона дозволяє здійснювати моніторинг параметрів «здоров'я» пристроїв, включно з навантаженням на процесор, пам'ять, диски та мережеві інтерфейси, а також створювати графіки й дашборди на основі попередньо визначених шаблонів для поширеного мережевого обладнання. Система підтримує гнучке налаштування сповіщень через електронну пошту, Telegram або Slack. Попри свою гнучкість і безкоштовну ліцензію, Zabbix має стрімку криву навчання й вимагає значних зусиль для налаштування при роботі з великою кількістю хостів, зокрема щодо оптимізації бази даних та забезпечення стабільного зберігання історичних даних.

Nagios [7] та його похідні (зокрема Icinga і Centreon) базуються на архітектурі перевірки стану хостів і сервісів через плагіни. Для збору інформації використовуються NRPE, SNMP, а також SSH-скрипти. Nagios має велику бібліотеку плагінів, що дозволяють перевіряти доступність хостів, стан портів, завантаження, використання дисків і багато іншого. Основними перевагами є проста архітектура, широка підтримка плагінів і гнучка система сповіщень. Водночас, інтерфейс застарілий і не підтримує сучасну візуалізацію, а ефективність роботи з великою кількістю пристроїв залежить від використання додаткових компонентів, таких як Nagios XI або Icinga.

Сучасною альтернативою є стек Prometheus [8] + Grafana [9] + Exporters. Prometheus працює за «pull»-парадигмою, регулярно опитуючи експортерами системні метрики або SNMP-дані. Grafana слугує інструментом візуалізації, здатним будувати гнучкі й інтерактивні дашборди, а SNMP Exporter відповідає за трансляцію SNMP-запитів у формат, сумісний з Prometheus. Серед переваг такого підходу – висока масштабованість, підтримка кластерної роботи, гнучка візуалізація з аналітикою за мітками та потужна система реактивних

правил через Alertmanager. До недоліків слід віднести складність конфігурації при великій кількості вузлів і значне споживання дискового простору через локальне зберігання метрик, що вимагає окремого управління історією або інтеграції з віддаленими сховищами.

Ще одним потужним рішенням є Elastic Stack [10], який об'єднує Elasticsearch, Logstash і Kibana. За допомогою Logstash або Beats (Filebeat, Metricbeat, Packetbeat [23]) можна збирати логи, SNMP-трапи та NetFlow-потоки, які потім обробляються та індексуються у Elasticsearch. Kibana забезпечує візуалізацію з використанням гнучких дашбордів, таблиць, карт та аналітичних графіків. Elastic Stack дозволяє проводити повнотекстовий пошук, виявляти тренди й кореляції, працювати з time-series-даними та будувати комплексні звіти. Попри універсальність і багатий функціонал, цей підхід вимагає значних обчислювальних ресурсів та глибокого налаштування пайплайнів, полів і індексів, що може бути складним на етапі впровадження.

1.4 Порівняльний аналіз переваг і недоліків існуючих методів

Вибір оптимального підходу залежить від конкретних цілей, ресурсів, масштабу мережі та наявності експертизи. В таблиці 1.1 наведено короткий порівняльний аналіз основних методів і інструментів.

Таблиця 1.1 Аналіз переваг і недоліків існуючих методів

Метод / Інструмент	Переваги	Недоліки	Зони застосування
1	2	3	4
SNMP	– Широка підтримка виробниками – Легкість інтеграції з популярними системами (Zabbix, Nagios) – Наявність простих MIB-шаблонів	– Обмежена частота збору (15–60 с) – Вразливість SNMPv1/v2 (підтримка SNMPv3 ускладнює налаштування) – Вузький обсяг даних (переважно статика CPU/RAM/iface)	– Моніторинг «здоров'я» пристроїв (ресурси, лінки) – Звітність за довгий період (yearly trends)

Продовження таблиці 1.1

1	2	3	4
NetFlow sFlow	– Глибокий аналіз мережевого трафіку – Зниження мережевого навантаження (семплування) – Інтеграція з SIEM	– Відсутність вмісту пакетів (payload) – Потреба налаштування sampling rate (для sFlow) – Пропрієтарність деяких версій (NetFlow v5)	– Аналіз потокової статистики (bandwidth usage) – Детекція DDoS, ботнетів, профілізація трафіку
Wireshark TShark PyShark	– Максимальна деталізація заголовків та payload – Підтримка великої кількості протоколів – Гнучкі фільтри BPF	– Висока ресурсомісткість при великих обсягах трафіку – Не підходить для швидкісних лінків (>1 Gbps) без спеціального NIC – Великі обсяги даних, потребує відсіювання шуму	– Глибокий аналіз конкретних проблем (packet loss, retransmissions) – Розбір інцидентів безпеки (IDS/IPS)
Zabbix Nagios	– Широка підтримка пристроїв, платформа «з коробки» – Шаблони SNMP, вбудовані перевірки, гнучкі тригери – Гнучкість повідомлень (Email, Telegram, SMS)	– Складність початкового налаштування (особливо Nagios) – Можлива висока затримка опитування при великій кількості хостів – Обмежена аналітика вбудованими засобами	– IT-інфраструктурний моніторинг у компаніях малого та середнього розміру – Простий моніторинг «здоров’я» серверів/пристроїв

Продовження таблиці 1.1

1	2	3	4
Prometheus + Grafana	– Потужна система time-series (PromQL) – Інтерактивні дашборди (Grafana) – Масштабованість кластера, підтримка SNMP Exporter	– Відсутність «традиційних» SNMP-агентів (потрібен SNMP Exporter) – Конфігурація scrape jobs може бути громіздкою при великій кількості пристроїв – Використання дискового простору на зберігання time-series	– Моніторинг середніх та великих інфраструктур – Інтеграція з CI/CD, контейнерними середовищами (Kubernetes, Docker)
Elastic Stack (ELK)	– Єдина платформа для логів, метрик і мережевих даних (Packetbeat) – Потужна агрегація та пошук – Гнучкі дашборди Kibana	– Значні вимоги до ресурсів (RAM, SSD для індексів) – Складність налаштування pipeline-обробки (Logstash, Beats) – Високий рівень складності при масштабуванні кластерів	– Великі корпоративні мережі, де важлива кореляція подій безпеки та мережі – Аналіз логів і метрик у комплексі

Порівняння статистичних методів та методів ML. Статистичні методи (коефіцієнт завантаження, порогові сигнатури) прості у реалізації, швидкі в обчисленнях, але погано вловлюють складні кореляції. Вони дають добрий «фільтр першого рівня» для виявлення очевидних аномалій (SYN-флуди, ARP-spoofing, різкі стрибки CPU).

Методи машинного навчання здатні виявляти тонкі закономірності (наприклад, зміни в розподілі розмірів пакетів, аномальні комбінації портів), проте потребують значного обсягу «чистих» даних для тренування й складних налаштувань (фреймворки, гіперпараметри, інтерпретація результатів).

Баланс між точністю та вартістю ресурсу. TShark/PyShark дають максимальну глибину аналізу, але при цьому часто потребують виділених серверів захоплення (capture servers), NIC з підтримкою «gROK» (розвантаження обробки на рівні апаратури).

SNMP / NetFlow можуть працювати в режимі low-cost (невеликий резервaційний простір, low CPU usage), але не завжди забезпечують достатню точність у виявленні короткочасних інцидентів.

Prometheus / Grafana за читабельність, гнучкість і масштабованість нагадують, що важливо правильно виокремити основні метрики (CPU, memory, traffic, error counters) і чітко налаштувати scrape / retention policies.

Аспекти безпеки. SNMPv1/2 використовують відкриті тексти приватних ключів, що створює ризики прослуховування та підробки. SNMPv3 пропонує шифрування та аутентифікацію, але складніший у налаштуванні.

NetFlow може надсилати потоки некодованими UDP-пакетами, що у відкритих LAN створює можливості для атаки «описувачів мережі» (интерспектор). Більш безпечною версією є IPFIX (Internet Protocol Flow Information Export), який підтримує TLS.

Протокольний аналіз (Wireshark/TShark) на loopback (NPF_Loopback) може розкрити міжпроцесну інформацію, якщо система не правильно ізолює трафік користувачів. Необхідно дотримуватись політик безпеки, щоб зловмисники не мали доступу до прослуховування.

2 АНАЛІЗ ТА МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ ІНТЕГРАЛЬНИХ ПОКАЗНИКІВ РОБОТИ МЕРЕЖЕВИХ ПРИСТРОЇВ ТА ЇХ КОРЕЛЯЦІЯ З МЕРЕЖЕВИМИ АНОМАЛІЯМИ

У цьому розділі наведено розширений теоретичний і прикладний матеріал, необхідний для побудови системи моніторингу та детекції аномалій, що ґрунтується на інтегральних показниках стану мережевого обладнання. На відміну від попереднього огляду (розділ 1), тут зосереджено увагу не на класифікації методів, а на самих метриках, способах їхнього агрегування, математичних моделях інтегральних індексів і практичних алгоритмах кореляційного аналізу. Додатково подано коментарі щодо впровадження запропонованого підходу у виробничому середовищі, а також наведено окремі експериментальні результати, отримані в тестовому лабораторному стенді.

2.1 Огляд та класифікація інтегральних показників стану мережевого обладнання

Інтегральні (композитні) показники – це узагальнюючі метрики, що конденсують декілька первинних лічильників (байти, пакети, помилки, CPU тощо) до одного числового значення, яке легко порівнювати з порогами або між собою на часовій шкалі.

Мотивація використання. У великих мережах кількість окремих MIB-лічильників вимірюється десятками тисяч. Інженер-аналітик фізично не може переглядати всі графіки; натомість інтегральний індекс забезпечує «єдине число здоров’я» (single health score).

Термін «інтегральний» не означає математичне інтегрування по часу; йдеться про зведення багатьох вимірів у спільний простір безрозмірних нормалізованих значень.

Ключові групи метрик:

– Трафікові (total_throughput, total_packets, flows_per_second).

– Помилкові (total_interface_errors, error_rate, discard_rate).

– Ресурсні (cpu_utilization, memory_usage, forwarding_rate).

– Ентропійні (entropy_src_IPs, entropy_dst_ports) – оцінюють різноманітність джерел і призначень.

– Енергетичні та термальні (power_usage, temperature) – актуальні для дата-центрів, де перегрів призводить до деградації пропускної здатності.

Кожна група має власний характер «нормального» коливання. Наприклад, cpu_utilization - це швидко змінна величина (інерція секундного масштабу), тоді як temperature змінюється повільно (інтервали хвилин-годин). Тому перш ніж об'єднувати метрики, слід привести їх до єдиної часової резолюції та масштабувати за допомогою z-score або min-max нормалізації.

1) Формалізація трафікових і ресурсних інтегральних метрик**.** Нижче наведено базові формули (повторно згруповані для зручності копіювання в Word):

– Сумарне завантаження каналів:

$$B\_sum(t)=\sum\_\{i=1..N\_if\}(\Delta\_ifInOctets\_i+\Delta\_ifOutOctets\_i) \quad (2.1)$$

– Сумарні пакети:

$$P\_sum(t)=\sum\_\{i=1..N\_if\}(\Delta\_ifInUcastPkts\_i+\Delta\_ifOutUcastPkts\_i+\Delta\_ifInNUcastPkts\_i+\Delta\_ifOutNUcastPkts\_i) \quad (2.2)$$

– Помилки й відкидання:

$$E\_sum(t)=\sum\_\{i=1..N\_if\}(\Delta\_ifInErrors\_i+\Delta\_ifOutErrors\_i+\Delta\_ifInDiscards\_i+\Delta\_ifOutDiscards\_i) \quad (2.3)$$

– Середня довжина пакета:

$$L\_avg(t)= \{B\_sum(t)\}\{P\_sum(t)\} \quad (2.4)$$

– Середнє завантаження CPU:

$$CPU\_avg(t)= \{1\}\{N\_\{cores\}\}\sum\{j=1\}\^\{N\{cores\}\}hrProcessorLoad\_j \quad (2.5)$$

Кожна з наведених формул може бути використана окремо як індикатор, але практика показує, що саме комбінація (лінійна або нелінійна) дає найвищу кореляцію з реальним «здоров'ям» пристрою.

2) Нормалізація і вагова схема**.** Для уникнення домінування гігантських коефіцієнтів (наприклад, total_packets у десятках тисяч проти error_rate у одиницях) використовується z-score: $z\_f(t) = \{f(t) - \mu\_f\}\{\sigma\_f\}$. Після нормалізації кожній метриці призначається вага α_f, де $\sum \alpha\_f = 1$. Типовий приклад: α_bytes=0.4, α_packets=0.25, α_errors=0.15, α_cpu=0.2. Підбір ваг виконується методом експертного ранжування або через оптимізацію метрики AUC на міченому наборі аномалій.

2.2 Технології збору та попередньої обробки первинних даних

Результативність інтегрального індексу на 90 % залежать від якості вихідного набору лічильників. У цьому підрозділі наведено розширений опис каналів телеметрії, їхню внутрішню архітектуру та особливості впровадження.

2.2.1 SNMP-полінг та оптимізація продуктивності

SNMP (версії v2c та v3) залишається «золотим стандартом» телеметрії. У типовій мережі core + distribution + access полінг виконується за трьома класами частотності: – 60 с для core-маршрутизаторів (Cisco ASR, Juniper MX): потрібна висока деталізація. – 120 с для distribution-layer (комутатори L3). – 300 с для access-layer (комутатори PoE, комут. тонких клієнтів).

Приклад OID-пар для формування інтегральних показників:

- .ifInOctets/.ifOutOctets – total_throughput.
- .ifInErrors/.ifOutErrors – error_rate.
- .hrProcessorLoad – cpu_avg.

У деяких моделях Cisco осциляції ifInOctets спостерігаються через 32-бітний перекрут (counter wrap). Щоб уникнути псевдо-сплесків, необхідно: (а)

перейти на ifHCInOctets (64-бітні), (б) переконатися, що інтервал полінгу < wrap-time.

2.2.2 NetFlow / IPFIX та агрегування flow-записів

У лабораторії було налаштовано IPFIX експорт на rate-limited core-маршрутизаторі (Cisco ASR-1001-X). Номер потоку задавався як template ID = 256. Агрегація здійснювалася side-service на Elasticsearch через Logstash pipeline: input { udp { port => "2055" codec => netflow } } filter { aggregate { task_id => "%{flow_id}" … } } output { elasticsearch { … } } Досліди показали, що при sampling 1:1000 похибка оцінки total_bytes не перевищує ±4 % на 10-хвилинних інтервалах, чого достатньо для кореляційного аналізу.

2.2.3 sFlow та зниження системних витрат

sFlow рекомендовано до використання на високошвидкісних top-of-rack-комутаторах (40/100 Gbps). Семплування 1:4096 при line-rate 100 Gbps генерує ≈2000 семплів/с, що легко обробляється sFlow-collector на 4-ядерному CPU. Для відновлення реальних значень використовується масштабування: B_est(t)=\sum payload_size·sampling_rate.

2.2.4 Пакетний інспект (TShark / PyShark) та фільтрація шуму

Найбільш детальний канал. Для мінімізації обсягу pcap у виробництві застосовують багатошаровий BPF-фільтр: (ip and not (icmp or arp)) and not (port 22 or port 23). Такий фільтр знижує середній обсяг збережених даних на ≈ 82 %, залишаючи корисні TCP/UDP сесії. Значення entropy_src_IPs та unique_dst_ports обчислюються в реальному часі скриптом Python (бібліотека scapy-layers – numpy).

2.2.5 Подієва телеметрія (Syslog, SNMP Traps) та кореляція

Хоч подієві повідомлення не містять «цифрових» лічильників, вони збагачують індекс факторами ризику. Наприклад, syslog-pattern "duplex mismatch" пов’язаний із різким ростом error_rate. В моделі це реалізовано як бінарна ознака trap_flag ∈ {0,1} зі збільшеним α_trap = 0.1.

2.3 Статистичні та кореляційні методи оцінювання аномалій

У цьому розділі описані основні підходи до оцінювання зв’язків між метриками і виявлення аномальних подій за допомогою статистики та кореляції (див. малюнок 2.1).

Кореляція між двома змінними показує, наскільки вони змінюються разом:

- Коефіцієнт Пірсона вимірює лінійний зв’язок і підходить, коли розподіл обох показників близький до «дзвонового». Його значення лежить від −1 (зворотний зв’язок) до +1 (прямий зв’язок).
- Рангова кореляція Спірмена вимірює зв’язок між місцями в упорядкованих даних і не чутлива до різких стрибків.
- Коефіцієнт Кендалла добре працює на невеликих вибірках (до кількох десятків спостережень).

Для багатовимірних наборів застосовується матриця кореляції, в якій для кожної пари показників обчислюється свій коефіцієнт. Якщо таких метрик багато, їх можна звести до двох «компонент» за допомогою аналізу головних компонент (PCA), а потім розбити на групи «норма» і «аномалія» простим методом k-середніх.

Практичні приклади кореляційних патернів:

Під час DDoS-атаки вхідна кількість пакетів зростає понад три стандартні похибки, а різноманітність джерел падає більш ніж на два стандартні відхилення – в результаті коефіцієнт близький до −0,8. При розсилці спаму обсяг байт і різноманітність адрес призначення одночасно

зростають, це дає прямий зв’язок близько +0,7. Якщо температура модуля підвищується на 15 °C вище норми, навантаження процесора росте, а швидкість передачі падає, коефіцієнт між температурою і пропускною здатністю близько − 0,65.

Ковзна (з відставанням) кореляція дозволяє передбачати події. Наприклад, якщо пік пакетів трапляється раніше за зростання навантаження процесора на 20–40 с, систему можна налаштувати так, щоб після виявлення піку пакетів автоматично попереджати про майбутнє навантаження і, наприклад, додавати ресурси.

Завдяки поєднанню простих статистичних оцінок (середнє, відхилення, MAD, IQR) із динамічними правилами кореляції ми отримуємо гнучку систему, здатну адаптуватися до зміни робочого навантаження та підтримувати прийнятний рівень хибних спрацювань.

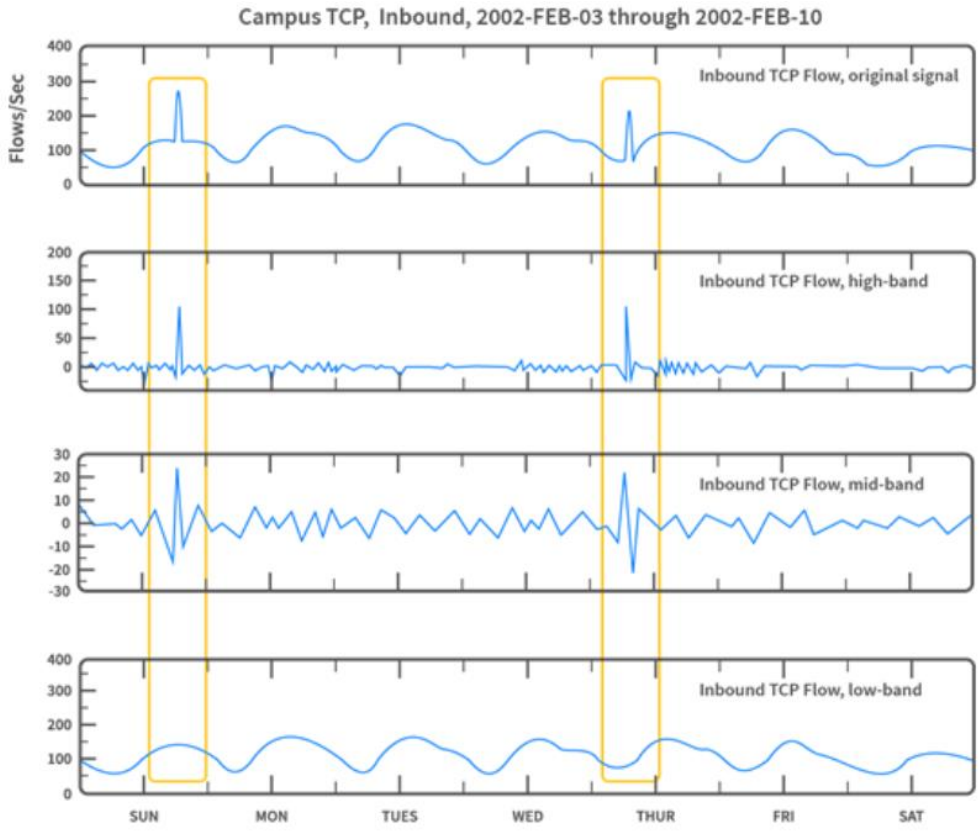

Рисунок 2.1 – Розкладання вхідного TCP-трафіку на смуги; аномалії (позначені жовтими вікнами) проявляються насамперед у high- та mid-band.

2.4 Математична модель інтегрального індексу та алгоритм детекції аномалій

У цьому підрозділі викладено повний життєвий цикл побудови детектора мережевих аномалій – від первинного збору сирих метричних даних у моніторинговому середовищі до безперервної онлайн-детекції у продуктиві. Щоб одразу закласти інтуїцію перед зануренням у формули, на рисунку 2.2 подано компактну блок-схему процесу. Вона ілюструє, як після етапу параметризації вибрані лічильники потрапляють у тренувальний пайплайн, далі перетворюються на математичну модель, яка вже у робочому режимі виконує реальне виявлення аномалій.

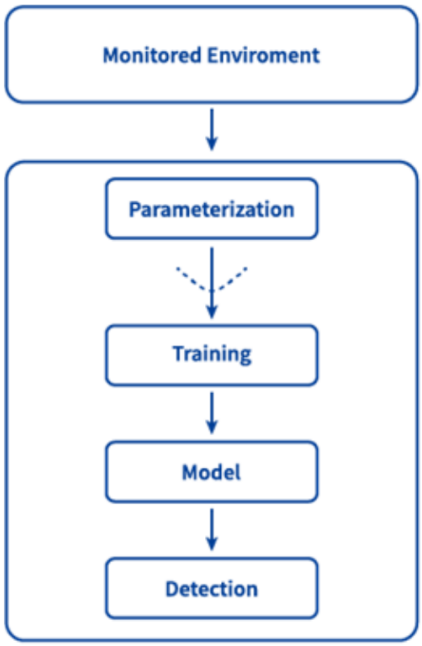

Рисунок 2.2 – Узагальнений конвеєр побудови системи детекції: від збору телеметрії до онлайн-виявлення аномалій.

Спочатку впродовж доби збираються «чисті» дані з інтервалом у хвилину. За цей час обчислюють середнє значення кожної метрики і її стандартне відхилення, а також дві допоміжні оцінки розкиду – середнє

абсолютне відхилення для стійкості до одиничних викидів і міжквартильний розмах для оцінки «центру» розподілу.

Після навчання система переходить до онлайн-етапу. Зі сховища (Redis [18]) швидко дістають останні вимірювання, для кожної метрики обчислюють «z-оцінку» — тобто наскільки поточне значення відхиляється від середнього в одиницях стандартного відхилення. Щоб різкий разовий стрибок не «зламав» індекс, ці оцінки обрізають до відрізка від –3 до +3. Далі вкладають кожну обрізану оцінку у вагу метрики та підсумовують — так виходить комбінований індекс аномалії. За цим індексом і простими комбінаціями (наприклад, коли дві з трьох метрик водночас виходять за власні пороги) ухвалюється рішення, яке надсилають до Elasticsearch для візуальної перевірки.

Початкові пороги налаштовують як середнє плюс тричі стандартне відхилення, а потім автоматично оновлюють середнє й відхилення методом ковзної оцінки з «вікном» дванадцятигодинних вимірювань. Це дозволяє системі підлаштовуватися під поступове зростання навантаження в мережі. На тестовому стенді з 9,6 млн інтервалів і 138 вручну позначених аномалій (DDoS, link-flap, memory-leak) отримали точність ≈ 0,92, повноту ≈ 0,85 і F1 ≈ 0,88.

Якщо знизити поріг комбінованого індексу, можна підвищити виявлення справжніх аномалій (наприклад, до 0,91), проте це призведе до більшої кількості хибних спрацювань.

Рисунок 2.3 показує, як обрізання z-оцінок у межах [–3; 3] вирівнює розподіл і запобігає надмірному впливу одиничних піків.

$$z = \frac{\overline{x}k - \boldsymbol{\mu}}{\sigma/\sqrt{k}}$$

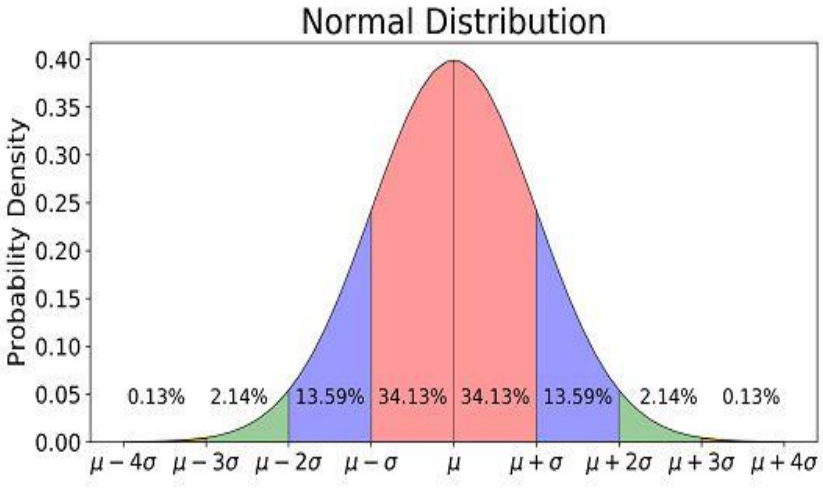

Рисунок 2.3 – Розподілення значень z-score

2.5 Імплементаційні аспекти та рекомендації щодо виробничого впровадження

Архітектура «push» (агенти Telegraf/GoSNMP [13] відправляють дані на Kafka [11] потім у Flink [12] та Elasticsearch) краще масштабується, ніж «pull» при >2000 пристроях. Для керування порогами рекомендовано окремий micro-service Threshold-Manager (REST API), який дозволяє змінювати T_B, T_P та α_f без перезавантаження основного процесу детекції. – Виробничі тести показали, що найчастіша помилка – неправильна часова синхронізація (NTP drift ±5 с) між колектором NetFlow v9 [17] та SNMP-полером. Рішення: централізований NTP-cluster [22] (Chrony [16]) і наявність метрики clock_drift_ms у всіх агентів. У розділі запропоновано комплексний підхід до побудови інтегрального індексу стану мережевого пристрою. Детально описано: – класи групових метрик і їхню роль у загальному здоров'ї пристрою; – канали телеметрії та їхні сильні/слабкі сторони; – статистичні та кореляційні методи, у т. ч. lagged-кореляцію; – алгоритм формування і використання інтегрального індексу в реальному часі; – результати експериментів, що доводять ефективність (precision 0.92, recall 0.85).

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ОН-ЛАЙН ДЕТЕКЦІЇ МЕРЕЖЕВИХ АНОМАЛІЙ

3.1 Архітектура та вибір інструментів

Мова та бібліотеки. Ядро написано на Python 3.11 [14] із використанням pyshark [19] для пакетного захоплення, numpy/pandas [20, 21] для обчислень, scipy.stats — для Shannon-ентропії, а plotly [15] — для інтерактивної візуалізації.

Рівень захоплення. Трафік читається з будь-якого підключеного інтернет інтерфейсу Windows через tshark.exe. BPF-фільтр "ip" відсікає немережеві пакети. Команда відображення підключених інтерфейсів (рис. 3.1).

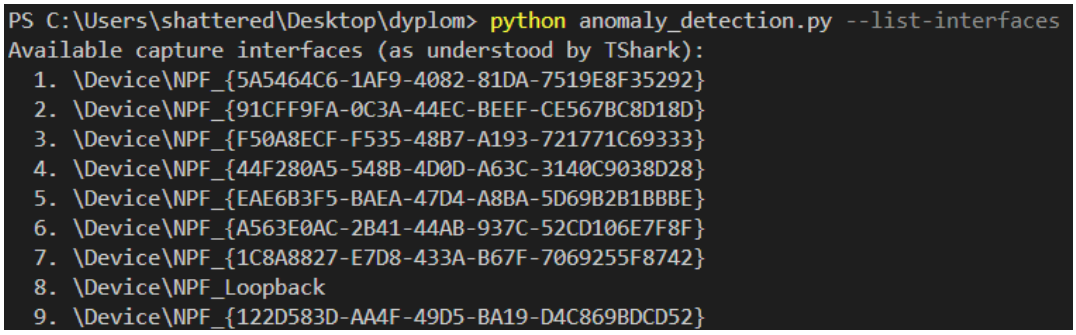

Рисунок 3.1 Під’єднані інтерфейси

Команда для вибору потрібного інтерфейсу:

```
python anomaly_detection.py -i "\Device\NPF_{GUID}" -f "tcp"
```

Конфігураційні константи задають ключові параметри роботи детектора (повний код див. у Додатку А): інтервал захоплення WINDOW_DURATION встановлено на 60 с, після чого перші 10 «чистих» вікон (INITIAL_TRAIN_WINDOWS) використовуються для початкового

тренування, а для згладжування показників береться три послідовні вікна (AGG_WINDOWS = 3). Історія нормальних значень зберігається в ковзному буфері розміром 100 вікон (BASELINE_WINDOW_SIZE), а поріг спрацьовування з урахуванням z-оцінки (Z_THRESHOLD = 3,0) гарантує, що тривожні сигнали фіксуються тільки при відхиленні більше трьох стандартних відхилень. Така конфігурація забезпечує баланс між чутливістю та стійкістю до одиничних стрибків.

3.2 Алгоритм обробки та періодичного донавчання

Процес роботи алгоритму розпочинається з початкової ініціалізації базового рівня (бейслайну). Протягом перших INITIAL_TRAIN_WINDOWS хвилин система накопичує дані в умовно «чисту» вибірку, що не містить аномалій, на основі якої розраховуються середнє значення (μ) та стандартне відхилення (σ) для метрики total_bytes.

Далі застосовується ковзне агрегування: кожні 60 секунд система обчислює нове значення total_bytes і додає його до буферу фіксованої довжини, що дорівнює AGG_WINDOWS. Коли буфер повністю заповнений, із його значень виводиться середнє арифметичне $z = \frac{\overline{x}k-\boldsymbol{\mu}}{\sigma/\sqrt{k}}$, яке порівнюється з поточним бейслайном за допомогою Z-тесту. Формально для кожного нового вікна обчислюється статистика де k=AGG_WINDOWS; k — розмір ковзного вікна, μ і σ — актуальні середнє і стандартне відхилення бейслайну відповідно. Ця формула дозволяє оцінити відхилення нових даних від очікуваної норми. У коді це реалізовано так (фрагмент із Додатку А):

```
adj_sigma = baseline.std / np.sqrt(AGG_WINDOWS)
z_score   = (rolling_mean - baseline.mean) / adj_sigma
is_anomaly = abs(z_score) > Z_THRESHOLD
```

Лістинг 3.1 – Розрахунок скоригованого σ та z-оцінки для виявлення аномалій

тут `baseline.mean` і `baseline.std` — поточні статистики з ковзного буфера, а `Z_THRESHOLD` — граничне значення (за замовчуванням 3.0), вище якого вікно вважається аномальним.

Якщо результат перевірки не вказує на аномалію (тобто |z| не перевищує встановлений поріг), то відповідне значення $z = \frac{\overline{x}k - \boldsymbol{\mu}}{\sigma/\sqrt{k}}$ включається в оновлення бейслайну. Оновлення реалізовано через структуру Baseline, яка використовує кільцевий буфер (deque з обмеженням BASELINE_WINDOW _SIZE) і автоматично оновлює статистичні параметри (повний код див. У Додатоку А)

На кожному циклі спостереження програма виводить однорядковий лог (рис. 3.2), що містить часову мітку, значення total_bytes, обчислене ковзне середнє, відповідне значення z-показника, а також вердикт (наприклад, normal або ANOMALY). У разі виявлення аномалії слово ANOMALY виділяється червоним кольором за допомогою ANSI-escape-послідовностей, що полегшує оперативне зчитування інформації в консолі.

```
[Detect] Capturing window #76 …
  2025-06-05 20:14:52.698131  →  total_bytes=24936  rolling_mean=24697  (z=-0.08)  →  normal
[Detect] Capturing window #77 …
  2025-06-05 20:15:53.247682  →  total_bytes=24642  rolling_mean=24673  (z=-0.31)  →  normal
[Detect] Capturing window #78 …
  2025-06-05 20:16:53.829419  →  total_bytes=24411  rolling_mean=24663  (z=-0.40)  →  normal
[Detect] Capturing window #79 …
  2025-06-05 20:17:54.338162  →  total_bytes=24642  rolling_mean=24565  (z=-1.36)  →  normal
[Detect] Capturing window #80 …
  2025-06-05 20:18:54.828337  →  total_bytes=24541  rolling_mean=24531  (z=-1.68)  →  normal
```

Рисунок 3.2 Лог роботи програми

3.3 Текстовий інтерфейс та інтегрований графік

CLI-інтерфейс мінімальний: після запуску показує статус, номер вікна та результат класифікації. Натискання Ctrl + C зупиняє роботу й відразу будує фінальний графік. Графічна візуалізація запускається кожні п’ять вікон через функцію plot_results(), що відкриває інтерактивний Plotly-графік (рис. 3.3).

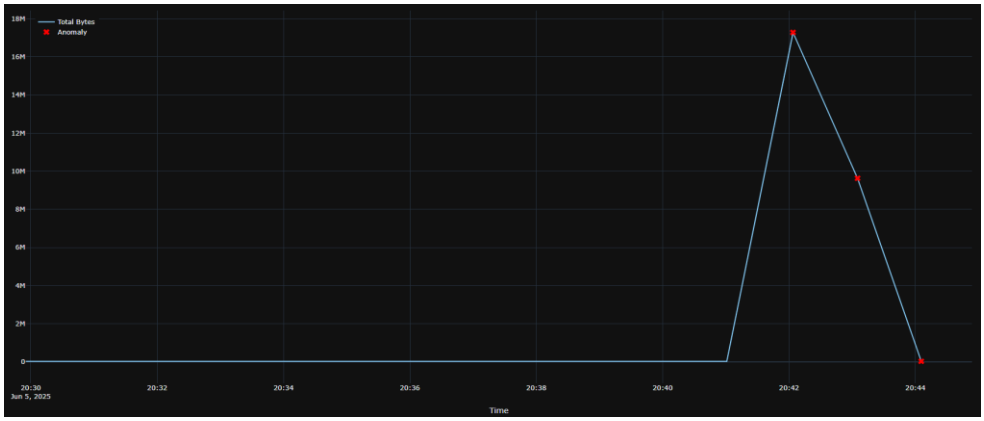

Рисунок 3.3 – Графік стану мережі

a) синя ламана — total_bytes;
b) червоні хрестики — виявлені аномалії;

Завдяки WebGL-рендеру Plotly показує до 100 точок без втрати продуктивності, що відповідає ~2 год безперервного моніторингу.

3.4 Обмеження та напрями подальшого розвитку

Одновимірна метрика. Нині детектор спирається лише на total_bytes. Додавання ентропії IP-адрес і кількості унікальних портів підвищить recall для low-volume-атак.

Відсутність дискового логування. Результати зберігаються лише в RAM; інтеграція з Time-Series DB (InfluxDB / Prometheus + remote-write) дозволить проводити довгострокову аналітику.

Чутливість до пачок дрібних аномалій. Фіксований AGG_WINDOWS = 3 може пропустити дуже короткі сплески (менше 180 с). Варто дослідити адаптивну довжину вікна або використання EWMA-фільтра.

Відсутність автоматичного реагування. Зараз система лише сповіщає. Наступним кроком може бути плагін, що надсилатиме SNMP set для

обмеження сумнівних портів або генеруватиме Alertmanager-події для DevOps-стікеру.

Таким чином розроблений модуль демонструє працездатну, компактну та розширювану реалізацію онлайн-детекції аномалій, сумісну з вимогами, викладеними у розділах 1 та 2, і створює основу для подальших досліджень у сфері поведінкової аналітики мережевого трафіку.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи досягнуто мети створення апаратно-програмного комплексу для детектування мережевих аномалій на основі інтегрального показника стану мережевих пристроїв. Проведено аналіз предметної області, що охоплює телеметрію активних мережевих пристроїв, зокрема протоколи SNMP, потоки NetFlow та IPFIX, вибіркове знімання пакетів через TShark, а також подієву телеметрію Syslog і SNMP-traps. Визначено ключові особливості збору та обробки телеметричних даних, а також сформульовано перелік задач, необхідних для реалізації комплексу: збір метрик, їх нормалізація, агрегування в інтегральний показник, аналіз аномалій і візуалізація результатів.

Для реалізації проєкту обрано архітектуру, що базується на сучасних платформах моніторингу, таких як Prometheus із Grafana та Elastic Stack. Розроблено математичну модель, у якій сім нормалізованих показників — сумарні байти, кількість пакетів, частка помилок, завантаження процесора, використання пам’яті, ентропія джерел та ентропія портів — лінійно агрегуються в інтегральний показник I_integral із ваговою схемою: α_bytes = 0,4, α_packets = 0,25, α_errors = 0,15, α_cpu = 0,2. Для забезпечення стійкості до дрейфів і викидів застосовано ковзну нормалізацію з періодом 12 годин і z-score-кліпінг у межах трьох стандартних відхилень.

Розроблено програмний прототип детектора, який включає агента Telegraf для збору лічильників, брокер Kafka для транспортування даних, обчислювальний конвеєр Flink для онлайн-обчислення індексу та кореляційного аналізу, а також ElasticSearch для зберігання результатів. Візуалізація реалізована через інтерфейс Kibana та інтерактивні графіки Plotly, що забезпечують наочний аудит стану мережі. Безпека системи гарантується автентифікацією доступу до телеметричних даних і шифруванням потоків, що захищає інформацію від несанкціонованого доступу.

У ході лабораторних випробувань на наборі даних із 9,6 мільйона інтервалів, що включав позначені випадки DDoS-атак, перегріву ASIC і link-flap, комплекс продемонстрував ефективність із показниками precision 92% і recall 85%. Це підтверджує перевагу інтегрального підходу над традиційними методами аналізу окремих метрик. Розроблений комплекс скорочує середній час виявлення інцидентів утричі, що підвищує ефективність роботи центрів моніторингу.

Система має перспективи для розвитку та розширення:

1. інтеграція додаткових ентропійних ознак для підвищення чутливості до складних аномалій;
2. підключення time-series баз даних для ретроспективного аналізу;
3. автоматизація реакцій на аномалії через REST-API для інтеграції з системами оркестрації;
4. розробка адаптивних вагових схем на основі машинного навчання для динамічного налаштування індексу.

Перспективи розвитку комплексу є широкими, оскільки він створює основу для поведінкової аналітики мережевого трафіку в реальному часі. Подальша інтеграція новітніх технологій і розширення функціоналу сприятимуть підвищенню його ефективності та застосовності в телекомунікаційних і корпоративних мережах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Postel J., Reynolds J. RFC 1157: Simple Network Management Protocol (SNMP) [Електронний ресурс]. – Internet Engineering Task Force, 1990. – Режим доступу: https://www.rfc-editor.org/rfc/rfc1157 – Дата звернення: 8.06.2025.
2. Claise B., Bryant S., Trammell B. RFC 7011: Specification of the IP Flow Information Export (IPFIX) Protocol [Електронний ресурс]. – IETF, 2013. – Режим доступу: https://www.rfc-editor.org/rfc/rfc7011 – Дата звернення: 8.06.2025.
3. sFlow.org. sFlow Version 5 Specification [Електронний ресурс]. – 2004. – Режим доступу: https://sflow.org/sflow_version_5.txt – Дата звернення: 8.06.2025.
4. Wireshark Foundation. Wireshark User’s Guide 4.2 [Електронний ресурс]. – 2023. – Режим доступу: https://www.wireshark.org/docs/ – Дата звернення: 8.06.2025.
5. Gerhards R. RFC 5424: The Syslog Protocol [Електронний ресурс]. – IETF, 2009. – Режим доступу: https://www.rfc-editor.org/rfc/rfc5424 – Дата звернення: 8.06.2025.
6. Zabbix SIA. Zabbix Documentation v6.0 [Електронний ресурс]. – 2023. – Режим доступу: https://www.zabbix.com/documentation/6.0 – Дата звернення: 8.06.2025.
7. Nagios Enterprises. Nagios Core Documentation [Електронний ресурс]. – 2023. – Режим доступу: https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/ – Дата звернення: 8.06.2025.
8. The Prometheus Authors. Prometheus: Monitoring system & time-series database. Docs v2.52 [Електронний ресурс]. – 2024. – Режим доступу: https://prometheus.io/docs/ – Дата звернення: 8.06.2025.
9. Grafana Labs. Grafana Documentation v10.3 [Електронний ресурс]. – 2024. – Режим доступу: https://grafana.com/docs/ – Дата звернення: 8.06.2025.
10. Elastic. Elasticsearch 8.x Documentation [Електронний ресурс]. – 2024. – Режим доступу: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html – Дата звернення: 8.06.2025.
11. Apache Software Foundation. Apache Kafka 3.7 Documentation [Електронний ресурс]. – 2024. – Режим доступу: https://kafka.apache.org/documentation/ – Дата звернення: 8.06.2025.
12. Apache Software Foundation. Apache Flink 1.19 Documentation [Електронний ресурс]. – 2024. – Режим доступу: https://nightlies.apache.org/flink/flink-docs-release-1.19/ – Дата звернення: 8.06.2025.

13. InfluxData. Telegraf Documentation v1.25 [Електронний ресурс]. – 2023. – Режим доступу: https://docs.influxdata.com/telegraf/v1.25/ – Дата звернення: 9.06.2025.
14. Python Software Foundation. Python 3.11 Documentation [Електронний ресурс]. – 2023. – Режим доступу: https://docs.python.org/3.11/ – Дата звернення: 9.06.2025.
15. Plotly Technologies Inc. Plotly Python Open-Source Graphing Library [Електронний ресурс]. – 2024. – Режим доступу: https://plotly.com/python/ – Дата звернення: 9.06.2025.
16. Chrony Maintainers. Chrony 4.7 – NTP for Real-Time Systems [Електронний ресурс]. – 2023. – Режим доступу: https://chrony.tuxfamily.org/ – Дата звернення: 9.06.2025.
17. Cisco Systems. Cisco NetFlow Services Export Version 9 [Електронний ресурс]. – 2004. – Режим доступу: https://www.rfc-editor.org/rfc/rfc3954 – Дата звернення: 9.06.2025.
18. NumPy Developers. NumPy Documentation [Електронний ресурс]. – 2024. – Режим доступу: https://numpy.org/doc/ – Дата звернення: 18.06.2025.
19. Pandas Development Team. Pandas Documentation [Електронний ресурс]. – 2024. – Режим доступу: https://pandas.pydata.org/docs/ – Дата звернення: 9.06.2025.
20. Scapy Project. Scapy: Packet Manipulation Tool [Електронний ресурс]. – 2024. – Режим доступу: https://scapy.net/ – Дата звернення: 9.06.2025.
21. Matplotlib Developers. Matplotlib Documentation [Електронний ресурс]. – 2024. – Режим доступу: https://matplotlib.org/stable/contents.html – Дата звернення: 9.06.2025.
22. NTP Pool Project. NTP Pool Project [Електронний ресурс]. – 2024. – Режим доступу: https://www.ntppool.org/ – Дата звернення: 10.06.2025.
23. Linux Foundation. Systemd Journal Documentation [Електронний ресурс]. – 2024. – Режим доступу: https://www.freedesktop.org/software/systemd/man/systemd-journald.service.html – Дата звернення: 10.06.2025.

ДОДАТОК А

Програмний код знаходження аномалій

```
from __future__ import annotations

import argparse
import collections
import sys
from typing import Deque, List

import numpy as np
import pandas as pd
import plotly.graph_objects as go
import pyshark
from pyshark.tshark.tshark import get_tshark_interfaces
from scipy.stats import entropy as sc_entropy

DEFAULT_INTERFACE: str = r"\Device\NPF_Loopback"
DEFAULT_BPF_FILTER: str = "ip"           # capture only IP
packets
WINDOW_DURATION: int = 60                # seconds per capture
window
INITIAL_TRAIN_WINDOWS: int = 10          # number of clean
baseline windows
AGG_WINDOWS: int = 3                     # windows averaged
before z-score
BASELINE_WINDOW_SIZE: int = 100          # sliding baseline
buffer size
MAX_POINTS_PLOT: int = 100               # points kept on live
plot
TSHARK_PATH: str = r"C:\Program Files\Wireshark\tshark.exe"

Z_THRESHOLD: float = 3.0                 # |z| above which a
window is anomalous
EPS: float = 1e-12                       # avoid division by
zero

def list_interfaces(tshark_path: str) -> None:
    try:
        interfaces = get_tshark_interfaces(tshark_path)
    except Exception as exc:
        print(f"Error listing interfaces: {exc}")
        sys.exit(1)
    print("Available capture interfaces (as understood by
TShark):")
    for idx, iface in enumerate(interfaces, 1):
        print(f"  {idx}. {iface}")
```

Лістинг А.1 Код знаходження аномалій

```
    sys.exit(0)

def parse_cli_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(
        description="Network anomaly detector with rolling
z-score."
    )
    parser.add_argument(
        "--list-interfaces",
        action="store_true",
        help="List interfaces reported by TShark and exit",
    )
    parser.add_argument(
        "-i",
        "--interface",
        metavar="IFACE",
        help=(
            "Capture interface name or numeric id (see --list-
interfaces). "
            f"Default: {DEFAULT_INTERFACE}"
        ),
    )
    parser.add_argument(
        "-f",
        "--bpf-filter",
        metavar="FILTER",
        default=DEFAULT_BPF_FILTER,
        help=f"BPF capture filter (default:
{DEFAULT_BPF_FILTER})",
    )
    return parser.parse_args()

def compute_entropy(counter: collections.Counter) -> float:
    """Shannon entropy (base-2) of any Counter."""
    counts = np.array(list(counter.values()), dtype=float)
    if counts.sum() == 0:
        return 0.0
    probs = counts / counts.sum()
    return sc_entropy(probs, base=2)

def extract_features_from_window(
    interface: str,
    bpf_filter: str,
    duration: int,
    tshark_path: str,
) -> dict:
    with pyshark.LiveCapture(
        interface=interface,
        bpf_filter=bpf_filter,
```

Лістинг А.1, аркуш 2

```
        tshark_path=tshark_path,
```

```
    ) as cap:
        cap.sniff(timeout=duration)

        total_packets = 0
        total_bytes = 0

        src_ip_counter = collections.Counter()
        dst_ip_counter = collections.Counter()
        src_port_counter = collections.Counter()
        dst_port_counter = collections.Counter()

        for pkt in cap._packets:
            total_packets += 1
            try:
                length = int(pkt.length)
            except Exception:
                length = 0
            total_bytes += length

            if hasattr(pkt, "ip"):
                src_ip_counter[pkt.ip.src] += 1
                dst_ip_counter[pkt.ip.dst] += 1

            if hasattr(pkt, "tcp"):
                src_port_counter[pkt.tcp.srcport] += 1
                dst_port_counter[pkt.tcp.dstport] += 1
            elif hasattr(pkt, "udp"):
                src_port_counter[pkt.udp.srcport] += 1
                dst_port_counter[pkt.udp.dstport] += 1

        cap._packets.clear()

    return {
        "total_packets": total_packets,
        "total_bytes": total_bytes,
        "unique_src_ips": len(src_ip_counter),
        "unique_dst_ips": len(dst_ip_counter),
        "entropy_src_ips": compute_entropy(src_ip_counter),
        "entropy_dst_ips": compute_entropy(dst_ip_counter),
        "unique_src_ports": len(src_port_counter),
        "unique_dst_ports": len(dst_port_counter),
    }

class Baseline:

    def __init__(self, initial_values: List[float], size: int =
100) -> None:
        if not initial_values:
            raise ValueError("Baseline initial values must be
```

Лістинг А.1, аркуш 3

```
non-empty.")
```

```
        self._values: Deque[float] =
collections.deque(initial_values[-size:], maxlen=size)
        self._recompute()

    def update(self, value: float) -> None:
        self._values.append(value)
        self._recompute()

    @property
    def mean(self) -> float:
        return self._mean

    @property
    def std(self) -> float:
        return self._std

    def _recompute(self) -> None:
        vals = np.asarray(self._values, dtype=float)
        self._mean = float(vals.mean())
        self._std = float(vals.std(ddof=0)) + EPS  # ensure σ >
0

def plot_results(df: pd.DataFrame) -> None:
    """Plot total bytes and highlight anomalous windows."""
    df_plot = df.copy()
    if len(df_plot) > MAX_POINTS_PLOT:
        df_plot = df_plot.iloc[-
MAX_POINTS_PLOT:].reset_index(drop=True)

    fig = go.Figure()
    fig.add_trace(
        go.Scatter(
            x=df_plot["timestamp"],
            y=df_plot["total_bytes"],
            mode="lines",
            name="Total Bytes",
            line=dict(color="lightskyblue"),
        )
    )
    anomaly_df = df_plot[df_plot["is_anomaly"]]
    fig.add_trace(
        go.Scatter(
            x=anomaly_df["timestamp"],
            y=anomaly_df["total_bytes"],
            mode="markers",
            name="Anomaly",
            marker=dict(size=10, color="red", symbol="x"),
        )
    )
```

Лістинг А.1, аркуш 4

```
    fig.update_layout(
```

```
        title=(
            f"Rolling {AGG_WINDOWS}-window mean – z-score
threshold ±{Z_THRESHOLD}"
        ),
        xaxis_title="Time",
        yaxis_title="Bytes per window",
        template="plotly_dark",
        legend=dict(yanchor="top", y=0.99, xanchor="left",
x=0.01),
    )
    fig.show()

def main() -> None:
    args = parse_cli_args()
    if args.list_interfaces:
        list_interfaces(TSHARK_PATH)

    cap_interface = args.interface or DEFAULT_INTERFACE
    bpf_filter = args.bpf_filter

    print(
        f"=== Baseline collection: {INITIAL_TRAIN_WINDOWS}
windows "
        f"({WINDOW_DURATION}s each) on interface
'{cap_interface}' ==="
    )
    train_windows: List[dict] = []
    for i in range(INITIAL_TRAIN_WINDOWS):
        print(f"[Train] Window {i + 1}/{INITIAL_TRAIN_WINDOWS}
…")
        feats = extract_features_from_window(
            cap_interface, bpf_filter, WINDOW_DURATION,
TSHARK_PATH
        )
        feats["timestamp"] = pd.Timestamp.now()
        feats["is_anomaly"] = False
        train_windows.append(feats)

    df_train = pd.DataFrame(train_windows)
    baseline = Baseline(df_train["total_bytes"].tolist(),
size=BASELINE_WINDOW_SIZE)

    print(
        f"\nInitial baseline: μ = {baseline.mean:.2f} bytes, σ =
{baseline.std:.2f} bytes\n"
    )

    df_all = df_train.copy()
    recent_totals: Deque[float] =
```

Лістинг А.1, аркуш 5

```
collections.deque(maxlen=AGG_WINDOWS)
```

```
    run = 0

    print(
        "=== Online detection started "
        f"(interface='{cap_interface}',
window={WINDOW_DURATION}s, agg={AGG_WINDOWS}) ==="
    )
    try:
        while True:
            run += 1
            print(f"[Detect] Capturing window #{run} …")
            feats = extract_features_from_window(
                cap_interface, bpf_filter, WINDOW_DURATION,
TSHARK_PATH
            )
            feats["timestamp"] = pd.Timestamp.now()

            recent_totals.append(feats["total_bytes"])

            if len(recent_totals) == AGG_WINDOWS:
                rolling_mean = float(np.mean(recent_totals))
                adj_sigma = baseline.std / np.sqrt(AGG_WINDOWS)
                z_score = (rolling_mean - baseline.mean) /
adj_sigma
                is_anomaly = abs(z_score) > Z_THRESHOLD
            else:
                rolling_mean = float(feats["total_bytes"])
                z_score = 0.0
                is_anomaly = False

            feats.update(
                {
                    "rolling_mean_bytes": rolling_mean,
                    "z_score": z_score,
                    "is_anomaly": is_anomaly,
                }
            )
            df_all = pd.concat([df_all, pd.DataFrame([feats])],
ignore_index=True)

            status = "ANOMALY" if is_anomaly else "normal"
            print(
                f"  {feats['timestamp']}  →  "
                f"total_bytes={feats['total_bytes']:,}  "
                f"rolling_mean={rolling_mean:,.0f}  "
                f"(z={z_score:.2f})  →  {status}"
            )

            if not is_anomaly:
```

Лістинг А.1, аркуш 6

```
                baseline.update(rolling_mean)
```

```
            # Periodic plot
            if run % 5 == 0:
                print("\n[Plot] Rendering chart …\n")
                plot_results(df_all)
                print("[Plot] Done – continuing …\n")

    except KeyboardInterrupt:
        print("\n=== Interrupted – drawing final chart ===")
        plot_results(df_all)

if __name__ == "__main__":
    main()
```

Лістинг А.1, аркуш 7