

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО

В.о. завідувача кафедри

_____ Олександр КОВАЛЬ

«___» _____ 2026 р.

Дипломна робота

**на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Організація моніторингу локальної мережі за допомогою
протоколу SNMP»**

Виконав:

студент IV курсу, групи TB-23

Салюк Владислав Юрійович

(прізвище, ім'я, по батькові)

(підпис)

Керівник:

доц., к.т.н Шпурик В.В.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент:

_____ (посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень із праць інших
авторів без відповідних посилань.

Студент _____
(підпис)

**Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»**

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення
інтелектуальних кібер - фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Олександр КОВАЛЬ
(підпис)

«_____» _____ 2026р.

ЗАВДАННЯ

на дипломну роботу студенту

_____ Салюку Владиславу Юрійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Організація моніторингу локальної мережі за допомогою протоколу SNMP

керівник роботи доц, к.т.н Шпурик В. В.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “__” _____ 2026 року № _____

2. Строк подання студентом роботи «10» червня 2026 р.

3. Вихідні дані до роботи: мова програмування Java, бібліотека SNMP4J, середовище розробки IntelliJ IDEA

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): Розробка програмного комплексу для організації моніторингу локальної мережі на основі протоколу SNMP, реалізація програмних SNMP-агентів для імітації роботи мережевих пристроїв, розроблення Custom MIB для опису спеціалізованих параметрів обладнання, а також аналіз альтернативних підходів до вирішення задачі верифікації систем

моніторингу та безпечного моделювання мережевих інцидентів у контрольованому середовищі.

5 . Перелік ілюстративного матеріалу: Use case діаграми, скріншоти веб інтерфейсу

6. Дата видачі завдання «31» жовтня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	31.10.2025	Виконано
2	Дослідження предметної області	01.11.2025 – 31.11.2025	Виконано
3	Постановка вимог до проєктування системи	01.12.2025 – 31.12.2025	Виконано
4	Дослідження існуючих рішень	01.01.2026 – 31.01.2026	Виконано
5	Розробка програмного продукту	01.02.2026 – 03.05.2026	Виконано
6	Тестування	04.05.2026 – 10.05.2026	Виконано
7	Захист програмного продукту	11.05.2026 – 15.05.2026	Виконано
8	Оформлення дипломної роботи	18.05.2026 – 31.05.2026	Виконано
9	Передзахист	01.06.2026 – 05.06.2026	
10	Захист	15.06.2026 – 19.06.2026	

Студент

(підпис)

Салюк Владислав

(ім'я, ПРІЗВИЩЕ)

Керівник роботи

(підпис)

Вадим Шпурик

(ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота за темою «Організація моніторингу локальної мережі за допомогою протоколу SNMP» виконана студентом кафедри інженерії програмного забезпечення в енергетиці НН ІАТЕ Салюком Владиславом Юрійовичем зі спеціальності 121 «Інженерія програмного забезпечення» за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці» і складається зі вступу; 5 розділів: «Аналіз предметної області та постановка задачі», «Проектування програмного комплексу», «Реалізація програмного комплексу», «Тестування та верифікація», «Аналіз результатів та перспективи розвитку»; висновків; списку використаних джерел, який налічує 20 найменувань; 7 ілюстрацій та 1 додатка. Загальний обсяг роботи становить 64 сторінки.

Актуальність теми зумовлена необхідністю забезпечення стабільної роботи локальних мереж, своєчасного виявлення відмов, перевантажень, помилок передавання даних та інших інцидентів, що можуть впливати на доступність інформаційних сервісів. У сучасних корпоративних, навчальних та спеціалізованих мережах важливу роль відіграють системи моніторингу, які дають змогу централізовано отримувати інформацію про стан мережевих вузлів і реагувати на зміни їхніх параметрів. Одним із поширених протоколів для розв'язання таких задач є SNMP, який підтримується великою кількістю мережевого обладнання та систем адміністрування.

Метою роботи є розроблення програмного комплексу для організації моніторингу локальної мережі за допомогою протоколу SNMP, який забезпечує симуляцію роботи мережевих пристроїв, збір телеметричних показників, генерацію подій та інтеграцію із системою моніторингу Zabbix.

Для досягнення поставленої мети у роботі проаналізовано принципи роботи протоколу SNMP, архітектуру взаємодії SNMP Manager та SNMP Agent,

структуру MIB і використання OID; досліджено наявні засоби симуляції SNMP-агентів; сформульовано вимоги до програмного комплексу; спроектовано компонентну модель системи; реалізовано SNMP-агент на основі бібліотеки SNMP4J; розроблено Custom MIB у просторі Enterprise OID; реалізовано механізм надсилання Trap-повідомлень; створено русій симуляційних сценаріїв; підготовлено контейнеризоване середовище з використанням Docker Compose; виконано інтеграцію із Zabbix та проведено тестування розробленої системи.

Практичне значення одержаних результатів полягає у створенні програмного комплексу, який може використовуватися як навчальний, тестовий або демонстраційний стенд для перевірки роботи систем мережевого моніторингу без потреби у фізичному мережевому обладнанні. Розроблена система дає змогу безпечно моделювати аварійні ситуації, перевіряти налаштування елементів моніторингу, тригерів і сповіщень, а також досліджувати поведінку NMS під час зміни стану мережевих вузлів.

Ключові слова: SNMP, MIB, OID, Zabbix, SNMP4J, Docker Compose, Trap, Polling, локальна мережа, мережевий моніторинг, SNMP-агент, симуляція, кіберінцидент, Java, JUnit, JaCoCo.

ABSTRACT

The diploma thesis on the topic “Organization of Local Network Monitoring Using the SNMP Protocol” was completed by Vladyslav Saliuk, a student of the Department of Software Engineering in Energy at the Educational and Scientific Institute of Atomic and Thermal Energy, specialty 121 “Software Engineering”, educational and professional program “Software Engineering of Intelligent Cyber-Physical Systems in Energy”. The thesis consists of an introduction; 5 chapters: “Analysis of the Subject Area and Problem Statement”, “Design of the Software System”, “Implementation of the Software System”, “Testing and Verification”, “Analysis of Results and Development Prospects”; conclusions; a list of references containing 20 sources; 7 figures and 1 appendix. The total volume of the thesis is 64 pages.

The relevance of the topic is determined by the need to ensure stable operation of local networks, timely detection of failures, overloads, data transmission errors and other incidents that may affect the availability of information services. In modern corporate, educational and specialized networks, monitoring systems play an important role by enabling centralized collection of information about the state of network nodes and response to changes in their parameters. One of the widely used protocols for solving such tasks is SNMP, which is supported by a large number of network devices and administration systems.

The purpose of the thesis is to develop a software system for organizing local network monitoring using the SNMP protocol. The system provides simulation of network device operation, collection of telemetry indicators, generation of events and integration with the Zabbix monitoring system.

To achieve this goal, the thesis analyzes the principles of the SNMP protocol, the interaction architecture of SNMP Manager and SNMP Agent, the MIB structure and the use of OID; examines existing SNMP agent simulation tools; defines requirements

for the software system; designs the component model of the system; implements an SNMP agent based on the SNMP4J library; develops a Custom MIB in the Enterprise OID space; implements the mechanism for sending Trap messages; creates a simulation scenario engine; prepares a containerized environment using Docker Compose; integrates the system with Zabbix and performs testing of the developed software.

The practical value of the obtained results lies in the creation of a software system that can be used as an educational, testing or demonstration environment for verifying network monitoring systems without the need for physical network equipment. The developed system makes it possible to safely simulate emergency situations, test monitoring items, triggers and notifications, and study the behavior of an NMS when the state of network nodes changes.

Keywords: SNMP, MIB, OID, Zabbix, SNMP4J, Docker Compose, Trap, Polling, local network, network monitoring, SNMP agent, simulation, cyber incident, Java, JUnit, JaCoCo.

КОРОТКИЙ ОПИС ДОСЛІДЖЕННЯ

Дане дослідження показує, як можна організувати моніторинг локальної мережі за допомогою протоколу SNMP та програмної симуляції мережевих пристроїв. У межах роботи розроблено програмний комплекс, який імітує роботу SNMP-агентів, надає доступ до змодельованих параметрів через MIB/OID, генерує Trap-повідомлення та дозволяє відтворювати контрольовані сценарії мережевих інцидентів. Для розгортання середовища використано Docker Compose, а для перевірки роботи моніторингу виконано інтеграцію із системою Zabbix.

БІБЛІОГРАФІЧНЕ ПОСИЛАННЯ

Салюк, В. Ю. Організація моніторингу локальної мережі за допомогою протоколу SNMP : дипломна робота : 121 Інженерія програмного забезпечення / Салюк Владислав Юрійович. – Київ, 2026.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ	10
ВСТУП.....	11
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ.....	14
1.1 Роль моніторингу локальних мереж.....	14
1.2 Огляд протоколу SNMP	16
1.3 Архітектура MIB та ідентифікатори OID.....	19
1.4 Аналіз існуючих засобів симуляції SNMP-агентів.....	22
1.5 Постановка задачі та вимоги до системи	24
2 ПРОЄКТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ	27
2.1 Обґрунтування архітектурного підходу	27
2.2 Компонентна модель системи.....	29
2.3 Проєктування Custom MIB	32
2.4 Модель симуляції подій та інцидентів	34
3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ.....	37
3.1 Реалізація SNMP-агента	37
3.2 Реалізація механізму Trap-повідомлень	39
3.3 Контейнеризація та мережеве середовище	41
3.4 Інтеграція із Zabbix.....	45
4 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ	48
4.1 Стратегія тестування	48
4.2 Автоматизоване тестування	50
4.3 Інтеграційне тестування.....	53
5 АНАЛІЗ РЕЗУЛЬТАТІВ ТА ПЕРСПЕКТИВИ РОЗВИТКУ	57
ВИСНОВКИ	61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API — Application Programming Interface, інтерфейс програмування застосунків.

АСУ ТП — автоматизована система управління технологічними процесами.

CMDB — Configuration Management Database, база даних управління конфігураціями.

DDoS — Distributed Denial of Service, розподілена атака на відмову.

HTTP — Hypertext Transfer Protocol, протокол передачі гіпертексту.

IED — Intelligent Electronic Device, інтелектуальний електронний пристрій.

IETF — Internet Engineering Task Force, організація з розроблення інтернет-стандартів.

JaCoCo — Java Code Coverage, інструмент вимірювання покриття коду тестами.

JVM — Java Virtual Machine, віртуальна машина Java.

MIB — Management Information Base, база керуючої інформації SNMP.

NMS — Network Management System, система мережевого управління.

OID — Object Identifier, об'єктний ідентифікатор у дереві MIB.

PDU — Protocol Data Unit, блок даних протоколу.

PEN — Private Enterprise Number, приватний номер підприємства у дереві OID.

RFC — Request for Comments, документ зі стандартами та рекомендаціями IETF.

SCADA — Supervisory Control and Data Acquisition, диспетчерське управління та збір даних.

SNMP — Simple Network Management Protocol, простий протокол мережевого управління.

UDP — User Datagram Protocol, протокол датаграм користувача.

Trap — асинхронне SNMP-повідомлення від агента до менеджера про подію.

Polling — періодичне опитування агентів менеджером для збору показників.

ВСТУП

Сучасні локальні мережі є основою інформаційної інфраструктури підприємств, навчальних закладів, державних організацій та об'єктів критичної інфраструктури. Через мережеві канали передаються службові дані, результати вимірювань, команди керування, журнали подій, облікова інформація та інші відомості, від доступності яких залежить безперервність бізнес-процесів. Зростання кількості вузлів, віртуалізація сервісів, використання контейнерних середовищ і підключення промислових контролерів ускладнюють адміністрування мережі та підвищують вимоги до якості моніторингу [17, 18].

Ефективний моніторинг локальної мережі передбачає не лише перевірку доступності вузлів, а й регулярний збір кількісних показників, виявлення аномалій, фіксацію змін стану обладнання та швидке сповіщення адміністратора про інциденти. У типовій інфраструктурі необхідно контролювати завантаження інтерфейсів, кількість помилок передавання, доступність сервісів, стан процесорних і пам'ятевих ресурсів, температуру пристроїв, а також специфічні параметри, характерні для серверів, маршрутизаторів, комутаторів або промислових пристроїв [19, 20].

Одним із найпоширеніших протоколів для збору такої інформації є SNMP. Його перевага полягає у простоті моделі взаємодії, підтримці великою кількістю обладнання, наявності стандартизованих MIB-модулів і можливості розширення за допомогою приватних гілок OID. SNMP дає змогу будувати централізовану систему спостереження, у якій NMS виконує регулярне опитування агентів, отримує асинхронні повідомлення про події та формує на їх основі алерти, графіки, історичні дані й звіти [3, 4].

Разом із тим тестування систем моніторингу у реальних мережах часто є складним і ризикованим. Для перевірки тригерів, сценаріїв аварій, логіки сповіщень і правил обробки подій потрібні ситуації, які небажано або неможливо

штучно створювати на продуктивному обладнанні. Наприклад, симуляція перевантаження каналу, відмови інтерфейсу, різкого зростання помилок або підозрілої активності може призвести до порушення роботи сервісів. Тому актуальним є створення ізольованого програмного середовища, здатного імітувати поведінку мережевих пристроїв [17, 18].

Метою дипломної роботи є розроблення програмного комплексу для організації моніторингу локальної мережі за допомогою протоколу SNMP, який забезпечує симуляцію типових мережевих вузлів, генерацію контрольованих телеметричних показників, надсилання Trap-повідомлень і інтеграцію із системою моніторингу Zabbix. Такий комплекс може використовуватися для навчання, демонстрації, верифікації конфігурацій NMS та відпрацювання сценаріїв реагування на інциденти [2, 9].

Для досягнення поставленої мети необхідно проаналізувати предметну область мережевого моніторингу, дослідити архітектуру SNMP, розглянути структуру MIB і принципи побудови OID, оцінити існуючі засоби симуляції SNMP-агентів, сформулювати функціональні та нефункціональні вимоги, спроектувати архітектуру програмного комплексу, реалізувати основні компоненти системи, підготувати контейнеризоване середовище розгортання та виконати тестування розробленого рішення [1].

Об'єктом дослідження є процес моніторингу локальної мережі. Предметом дослідження є методи програмної симуляції SNMP-агентів, формування телеметричних показників і інтеграції з NMS. Практичним результатом роботи є програмний комплекс, що містить SNMP-агент, Custom MIB, механізм генерації Trap-повідомлень, рушій сценаріїв кіберінцидентів, HTTP API для керування симуляціями та конфігурацію Docker Compose для розгортання ізольованої мережі [2, 9].

Структура пояснювальної записки відповідає логіці виконання дипломної роботи. У першому розділі наведено аналіз предметної області та постановку

задачі. У другому розділі описано архітектуру системи та проєктування Custom MIB. Третій розділ присвячено реалізації програмного комплексу. У четвертому розділі розглянуто тестування та верифікацію. У п'ятому розділі наведено аналіз результатів і перспективи розвитку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Роль моніторингу локальних мереж

Локальна мережа підприємства складається з багатьох взаємопов'язаних компонентів: маршрутизаторів, комутаторів, серверів, робочих станцій, віртуальних машин, контейнерів, контролерів і допоміжних сервісів. Кожен із цих елементів має власний життєвий цикл, набір параметрів і характерні режими відмови. Без централізованого моніторингу адміністратор змушений реагувати на проблеми постфактум, коли користувачі вже помітили недоступність сервісу або погіршення якості роботи [17, 18].

Моніторинг дає змогу перейти від реактивного адміністрування до проактивного керування інфраструктурою. Якщо система своєчасно фіксує зростання навантаження на інтерфейс, збільшення кількості помилок або нестачу пам'яті, адміністратор може вжити заходів до виникнення критичної відмови. У цьому контексті важливими є історичні дані, адже вони дозволяють аналізувати тренди, планувати розширення інфраструктури та оцінювати ефективність внесених змін.

Окреме значення має моніторинг мереж спеціального призначення, зокрема сегментів, що взаємодіють з АСУ ТП, SCADA або IED-пристроями. У таких середовищах відмова мережевого вузла може впливати не лише на інформаційні сервіси, а й на технологічні процеси. Тому до системи спостереження висуваються вимоги передбачуваності, ізоляції, мінімального впливу на обладнання та наявності чітких механізмів сповіщення.

Типова система моніторингу складається з серверної частини, агентів або протокольних адаптерів, бази даних часових рядів, механізму тригерів, інтерфейсу візуалізації та підсистеми повідомлень. У випадку SNMP агент часто

уже вбудований у пристрій, а роль системи моніторингу полягає у регулярному опитуванні потрібних OID і правильній інтерпретації отриманих значень [17, 18].

Узагальнену структуру системи моніторингу локальної мережі наведено на рисунку 1.1.

Узагальнена структура моніторингу локальної мережі

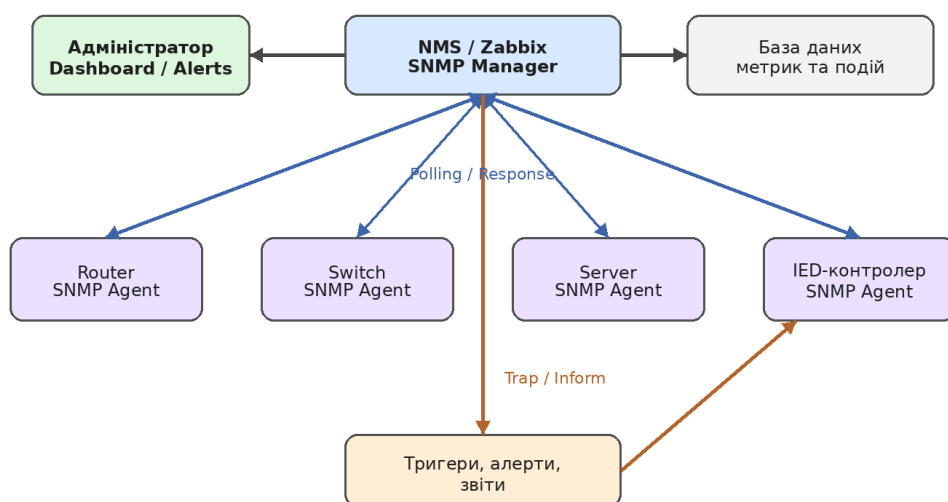


Рисунок 1.1 – Узагальнена структура моніторингу локальної мережі

Під час проєктування моніторингу важливо відокремити показники доступності від показників продуктивності та діагностичних даних. Доступність вказує, чи може система взаємодіяти з вузлом. Продуктивність характеризує поточний рівень використання ресурсів. Діагностичні дані пояснюють причину змін, наприклад помилки на інтерфейсі, перезапуск пристрою або зміну статусу порту. Саме комбінація цих груп показників забезпечує повноцінну картину стану мережі [17, 18].

Недоліком багатьох впроваджень моніторингу є відсутність належного тестового середовища. Конфігурація тригерів часто перевіряється лише в момент реальної аварії, коли виправлення помилок потребує швидкої реакції. Симуляційний стенд дозволяє заздалегідь перевірити, чи правильно NMS отримує дані, чи коректно працюють пороги спрацювання, чи надсилаються

повідомлення відповідальним особам та чи не створюються надлишкові хибні спрацювання [17, 18].

Під час практичного використання системи важливо документувати кожен OID, який передається до NMS. Документація повинна містити числовий ідентифікатор, коротку назву, тип даних, допустимі значення, одиницю вимірювання та приклад інтерпретації. Такий опис зменшує ризик помилок під час створення `item` і `trigger`, а також полегшує підтримку системи іншими користувачами.

Окремою перевагою симуляційного підходу є можливість безпечно відтворювати рідкісні або небезпечні події. У реальній мережі адміністратор не завжди може навмисно вимкнути інтерфейс або створити перевантаження, особливо якщо обладнання обслуговує критичні сервіси. У контейнерному середовищі такі дії не впливають на продуктивну інфраструктуру.

При виборі періодичності опитування необхідно знаходити баланс між актуальністю даних і навантаженням на мережу. Занадто часті запити можуть створювати зайвий трафік і навантаження на агент, а занадто рідкісні — призводити до запізненого виявлення проблем. Для демонстраційного стенду періоди можна скоротити, щоб швидше показати зміну стану.

1.2 Огляд протоколу SNMP

SNMP є прикладним протоколом, призначеним для керування та моніторингу мережевих пристроїв. Його модель побудована навколо взаємодії менеджера та агента. Менеджер розміщується у системі мережевого управління, а агент працює на контрольованому пристрої або у програмному середовищі, що імітує такий пристрій. Менеджер надсилає запити, агент повертає значення об'єктів керування, а в окремих випадках агент сам ініціює повідомлення про подію [5].

Найпоширенішими операціями SNMP є Get, GetNext, GetBulk і Set. Операція Get використовується для отримання значення конкретного OID. GetNext дозволяє послідовно проходити дерево MIB, що корисно для таблиць. GetBulk зменшує кількість мережових обмінів під час отримання великих наборів даних. Set застосовується для зміни керованих параметрів, хоча у багатьох середовищах її обмежують або вимикають з міркувань безпеки [3, 4].

Окрім синхронного опитування, SNMP підтримує асинхронні повідомлення Trap та Inform. Trap надсилається агентом без попереднього запиту менеджера й повідомляє про подію, наприклад зміну стану інтерфейсу або перевищення порогу. Inform є підтверджуваним варіантом повідомлення, коли агент очікує відповідь від менеджера. У дипломній роботі акцент зроблено на Trap-повідомленнях, оскільки вони добре підходять для демонстрації інцидентів у тестовому середовищі [2, 9].

Типову модель обміну між SNMP-менеджером і агентом наведено на рисунку 1.2.

Модель SNMP-взаємодії Manager-Agent

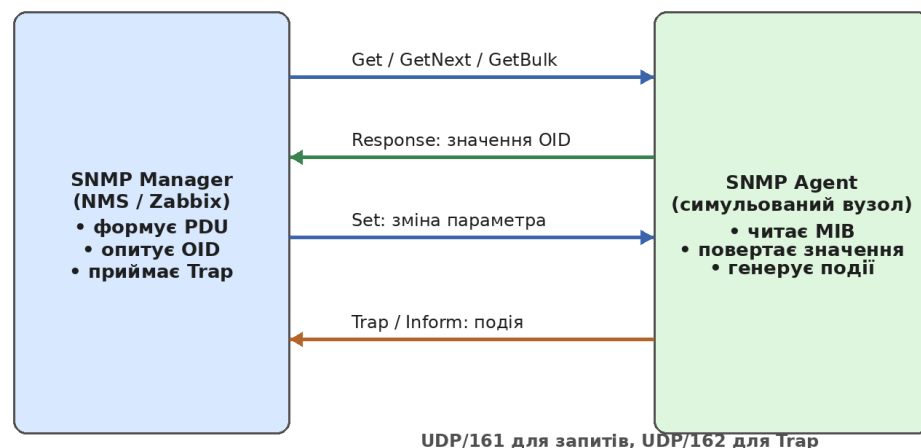


Рисунок 1.2 – Модель SNMP-взаємодії Manager-Agent

Різні версії SNMP мають різний рівень функціональності та безпеки. SNMPv1 є найпростішим і підтримує базовий набір операцій. SNMPv2c додає

ефективніші механізми отримання даних, зокрема GetBulk, але зберігає community string як простий механізм доступу. SNMPv3 вводить автентифікацію, шифрування й модель безпеки користувачів, що робить його кращим вибором для продуктивних середовищ. У навчальних і демонстраційних стендах часто використовують SNMPv2c через простоту налаштування [6, 7, 8].

Передавання SNMP-повідомлень зазвичай здійснюється поверх UDP. Такий вибір зменшує накладні витрати та відповідає природі моніторингу, де окремо втрачений пакет не завжди є критичним, якщо наступний цикл опитування відновить актуальні дані. Водночас використання UDP вимагає уважного налаштування таймаутів, повторних спроб і періодичності опитування, щоб система не створювала надмірного навантаження на мережу [17; 18].

Для програмної реалізації SNMP-агента у роботі використовується бібліотека SNMP4J. Вона надає засоби для формування PDU, оброблення запитів, роботи з типами даних SNMP, надсилання Trap-повідомлень і побудови агентської логіки у Java-застосунках. Це дозволяє зосередитися на моделі поведінки пристрою та сценаріях симуляції, не реалізуючи низькорівневий протокольний обмін вручну [2, 9].

Під час практичного використання системи важливо документувати кожен OID, який передається до NMS. Документація повинна містити числовий ідентифікатор, коротку назву, тип даних, допустимі значення, одиницю вимірювання та приклад інтерпретації. Такий опис зменшує ризик помилок під час створення item і trigger, а також полегшує підтримку системи іншими користувачами.

Окремою перевагою симуляційного підходу є можливість безпечно відтворювати рідкісні або небезпечні події. У реальній мережі адміністратор не завжди може навмисно вимкнути інтерфейс або створити перевантаження, особливо якщо обладнання обслуговує критичні сервіси. У контейнерному середовищі такі дії не впливають на продуктивну інфраструктуру.

Таблиця 1.1 – Порівняння основних версій SNMP

Версія	Особливості	Доцільність використання
SNMPv1	Базові операції Get, GetNext, Set, Trap; простий community-доступ.	Навчальні та застарілі середовища.
SNMPv2c	Підтримка GetBulk, покращена продуктивність, community-модель безпеки.	Тестові стенди та прості локальні мережі.
SNMPv3	Автентифікація, шифрування, модель користувачів.	Продуктивні середовища з вимогами безпеки.

При виборі періодичності опитування необхідно знаходити баланс між актуальністю даних і навантаженням на мережу. Занадто часті запити можуть створювати зайвий трафік і навантаження на агент, а занадто рідкісні — призводити до запізненого виявлення проблем. Для демонстраційного стенду періоди можна скоротити, щоб швидше показати зміну стану.

1.3 Архітектура MIB та ідентифікатори OID

MIB є логічною моделлю даних, які можуть бути прочитані або змінені за допомогою SNMP. Вона організована у вигляді ієрархічного дерева, де кожен вузол має числовий ідентифікатор OID. Завдяки такій структурі різні виробники можуть розширювати стандартний простір об'єктів, не конфліктуючи між собою. Стандартизовані гілки описують загальні параметри системи, інтерфейсів та протоколів, а приватні гілки використовуються для специфічних показників [3, 4].

Класичним прикладом стандартної MIB є MIB-II, у якій визначено об'єкти для опису системної інформації, мережеских інтерфейсів, IP, TCP, UDP та інших

базових параметрів. Для моніторингу локальної мережі особливо важливими є системна група та група інтерфейсів, оскільки вони дозволяють отримати ім'я пристрою, час роботи, опис інтерфейсів, адміністративний і операційний статус, лічильники байтів, пакетів та помилок [3].

У дипломному проєкті використовується Custom MIB у просторі Enterprise OID 1.3.6.1.4.1.99999. Такий підхід дозволяє описати власні симуляційні параметри, які не обмежуються стандартними показниками реального обладнання. Наприклад, можна додати індикатор активного сценарію, умовний рівень ризику, кількість згенерованих подій, статус симульованого пристрою або параметри, що характеризують кіберінцидент [3, 4].

Місце власної гілки Enterprise OID у загальному дереві MIB показано на рисунку 1.3.

Фрагмент дерева MIB та простір Enterprise OID

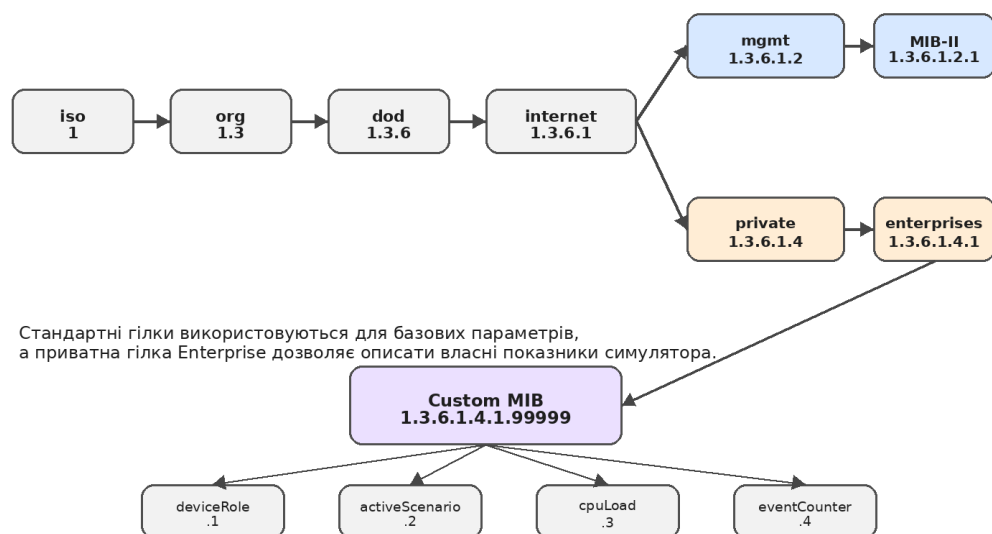


Рисунок 1.3 – Фрагмент дерева MIB та простір Enterprise OID

Проектування Custom MIB потребує узгодженості імен, типів даних і семантики. Кожен OID повинен мати зрозуміле призначення, стабільний тип значення та опис одиниць вимірювання, якщо вони застосовуються. Невдала структура MIB ускладнює інтеграцію із Zabbix, призводить до дублювання

показників і підвищує ризик неправильного трактування даних адміністратором [3, 4].

У симуляційному середовищі MIB виконує також роль контракту між агентом і системою моніторингу. Агент гарантує, що на певний OID буде повернуто значення очікуваного типу, а NMS використовує цей OID для створення `item`, `trigger` і `dashboard`. Якщо контракт змінюється, необхідно оновлювати не лише код агента, а й конфігурацію моніторингу. Тому MIB варто розглядати як окремий артефакт проєктування [3, 4].

Для зручності подальшого розвитку доцільно групувати об'єкти за логічними піддеревами: базова інформація про пристрій, показники ресурсів, мережеві інтерфейси, події безпеки, керування сценаріями та службові метрики. Така структура полегшує навігацію, дозволяє поступово додавати нові об'єкти та спрощує документування системи для користувачів і розробників.

Під час практичного використання системи важливо документувати кожен OID, який передається до NMS. Документація повинна містити числовий ідентифікатор, коротку назву, тип даних, допустимі значення, одиницю вимірювання та приклад інтерпретації. Такий опис зменшує ризик помилок під час створення `item` і `trigger`, а також полегшує підтримку системи іншими користувачами.

Окремою перевагою симуляційного підходу є можливість безпечно відтворювати рідкісні або небезпечні події. У реальній мережі адміністратор не завжди може навмисно вимкнути інтерфейс або створити перевантаження, особливо якщо обладнання обслуговує критичні сервіси. У контейнерному середовищі такі дії не впливають на продуктивну інфраструктуру.

При виборі періодичності опитування необхідно знаходити баланс між актуальністю даних і навантаженням на мережу. Занадто часті запити можуть створювати зайвий трафік і навантаження на агент, а занадто рідкісні —

призводити до запізненого виявлення проблем. Для демонстраційного стенду періоди можна скоротити, щоб швидше показати зміну стану.

1.4 Аналіз існуючих засобів симуляції SNMP-агентів

На ринку існують спеціалізовані засоби для симуляції SNMP-пристроїв, серед яких можна виділити MIMIC та MG-SOFT. Вони орієнтовані на професійне тестування систем мережевого управління й дозволяють моделювати значну кількість пристроїв, імпортувати MIB, налаштовувати поведінку агентів і відтворювати сценарії зміни показників. Такі продукти корисні для великих лабораторій, виробників обладнання та інтеграторів [15, 16].

Перевагою комерційних симуляторів є зрілість, масштабованість, підтримка багатьох протоколів і наявність інструментів для автоматизації. Проте вони можуть бути надмірними для навчальної або дипломної роботи, де важливо продемонструвати принципи побудови агента, інтеграцію із системою моніторингу та керовану генерацію подій. Крім того, закритий характер таких продуктів обмежує можливість детального аналізу внутрішньої логіки [17, 18].

Альтернативою є створення власного симуляційного середовища з використанням відкритих бібліотек. Такий підхід потребує більше розробницьких зусиль, але забезпечує повний контроль над моделлю даних, сценаріями, архітектурою та способом інтеграції. Для дипломної роботи це є доцільним, оскільки дозволяє продемонструвати не лише використання готового інструмента, а й розуміння протоколу, структури MIB та процесу тестування.

Власний симулятор також простіше адаптувати до конкретного навчального сценарію. Наприклад, можна створити чотири логічні пристрої: маршрутизатор, комутатор, сервер і IED-контролер. Для кожного пристрою можна визначити набір показників, що відповідає його ролі у мережі, а потім

запускати сценарії, які змінюють ці показники або генерують Trap-повідомлення. Це робить стенд зрозумілим і керованим [2, 9].

Під час аналізу існуючих засобів важливо враховувати не лише функціональність, а й вартість володіння, складність розгортання, прозорість конфігурації, можливість інтеграції з CI/CD, підтримку контейнеризації та доступність для студентського середовища. У цьому контексті власне рішення на Java, SNMP4J і Docker Compose має перевагу простоти відтворення та відкритості архітектури [1].

Таким чином, розроблення власного програмного комплексу є обґрунтованим для задачі дипломного проєктування. Воно дозволяє сфокусуватися на ключових аспектах: моделюванні SNMP-агента, описі Custom MIB, інтеграції із Zabbix, симуляції інцидентів і тестуванні. При цьому отримане рішення може бути розширене у майбутньому, якщо виникне потреба у масштабуванні або підтримці нових типів пристроїв [11].

Під час практичного використання системи важливо документувати кожен OID, який передається до NMS. Документація повинна містити числовий ідентифікатор, коротку назву, тип даних, допустимі значення, одиницю вимірювання та приклад інтерпретації. Такий опис зменшує ризик помилок під час створення item і trigger, а також полегшує підтримку системи іншими користувачами.

Окремою перевагою симуляційного підходу є можливість безпечно відтворювати рідкісні або небезпечні події. У реальній мережі адміністратор не завжди може навмисно вимкнути інтерфейс або створити перевантаження, особливо якщо обладнання обслуговує критичні сервіси. У контейнерному середовищі такі дії не впливають на продуктивну інфраструктуру.

При виборі періодичності опитування необхідно знаходити баланс між актуальністю даних і навантаженням на мережу. Занадто часті запити можуть створювати зайвий трафік і навантаження на агент, а занадто рідкісні —

призводити до запізненого виявлення проблем. Для демонстраційного стенду періоди можна скоротити, щоб швидше показати зміну стану.

1.5 Постановка задачі та вимоги до системи

Постановка задачі полягає у створенні програмного комплексу, який імітує роботу локальної мережі та надає дані для системи моніторингу через SNMP. Комплекс має дозволяти перевіряти конфігурацію NMS, відпрацьовувати сценарії інцидентів, демонструвати реакцію тригерів і аналізувати отримані показники. При цьому рішення повинно бути достатньо простим для запуску на типовому робочому комп'ютері без використання фізичного мережевого обладнання [19, 20].

До функціональних вимог належать: реалізація SNMP-агента, підтримка читання значень за визначеними OID, формування Custom MIB, генерація Trap-повідомлень, моделювання щонайменше трьох сценаріїв інцидентів, розгортання кількох логічних вузлів у контейнерній мережі, інтеграція із Zabbix, наявність HTTP API для керування сценаріями та можливість автоматизованого тестування ключових компонентів [3, 4].

Нефункціональні вимоги охоплюють відтворюваність розгортання, зрозумілість конфігурації, розширюваність, ізоляцію середовища, мінімальні системні вимоги та підтримуваність коду. Важливо, щоб запуск стенду не залежав від складної ручної інсталяції. Саме тому доцільно використовувати Docker Compose, який дозволяє описати мережу, контейнери, адреси та залежності у декларативному вигляді [12].

Система має підтримувати кілька ролей пристроїв. Маршрутизатор може характеризуватися показниками проходження трафіку, станом інтерфейсів і кількістю маршрутів. Комутатор може демонструвати зміну статусу портів і помилки кадрів. Сервер може надавати дані про CPU, пам'ять і доступність

сервісу. IED-контролер може мати специфічні показники, пов'язані з технологічним сегментом і підвищеними вимогами до надійності.

Окремою вимогою є можливість контролюваного запуску інцидентів. Симуляція повинна бути детермінованою настільки, щоб користувач міг повторити сценарій і перевірити очікувану реакцію системи моніторингу. Водночас значення показників мають виглядати правдоподібно, тобто змінюватися поступово або відповідно до логіки події, а не бути випадковим набором чисел без зв'язку з поведінкою пристрою [17, 18].

Результатом виконання задачі має стати цілісний програмний комплекс і набір супровідних матеріалів: опис архітектури, опис MIB, інструкція з розгортання, приклади конфігурації Zabbix, сценарії тестування та результати верифікації. Така форма подання дозволяє оцінити не лише працездатність коду, а й готовність рішення до практичного використання у навчальному або тестовому процесі [11].

Таблиця 1.2 – Основні вимоги до програмного комплексу

Група вимог	Зміст вимоги
Функціональні	SNMP-агент, Custom MIB, Trap, сценарії інцидентів, HTTP API, інтеграція із Zabbix.
Нефункціональні	Відтворюваність, контейнеризація, розширюваність, простота запуску, тестованість.
Експлуатаційні	Ізольована мережа, кілька логічних пристроїв, керований запуск сценаріїв.

Під час практичного використання системи важливо документувати кожен OID, який передається до NMS. Документація повинна містити числовий ідентифікатор, коротку назву, тип даних, допустимі значення, одиницю вимірювання та приклад інтерпретації. Такий опис зменшує ризик помилок під

час створення item і trigger, а також полегшує підтримку системи іншими користувачами.

Окремою перевагою симуляційного підходу є можливість безпечно відтворювати рідкісні або небезпечні події. У реальній мережі адміністратор не завжди може навмисно вимкнути інтерфейс або створити перевантаження, особливо якщо обладнання обслуговує критичні сервіси. У контейнерному середовищі такі дії не впливають на продуктивну інфраструктуру.

При виборі періодичності опитування необхідно знаходити баланс між актуальністю даних і навантаженням на мережу. Занадто часті запити можуть створювати зайвий трафік і навантаження на агент, а занадто рідкісні — призводити до запізненого виявлення проблем. Для демонстраційного стенду періоди можна скоротити, щоб швидше показати зміну стану.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Обґрунтування архітектурного підходу

Архітектура програмного комплексу побудована за модульним принципом. Кожен компонент відповідає за окрему частину функціональності: оброблення SNMP-запитів, зберігання поточного стану симульованого пристрою, генерацію Trap-повідомлень, виконання сценаріїв, взаємодію через HTTP API та конфігурацію розгортання. Такий підхід спрощує розуміння системи, тестування та подальше розширення [1].

Модульність є особливо важливою для симуляційного середовища, оскільки поведінка пристроїв може змінюватися залежно від сценарію. Якщо логіка SNMP-агента буде жорстко пов'язана з конкретними значеннями показників, додавання нового сценарію вимагатиме зміни протокольного шару. Натомість відокремлення агента від SimulationEngine дозволяє оновлювати модель стану незалежно від механізму оброблення SNMP-запитів.

У ролі основної мови програмування обрано Java. Вона має розвинену екосистему, підтримку багатопотоковості, зрілі інструменти тестування та стабільні бібліотеки для роботи з мережевими протоколами. Для реалізації SNMP використовується SNMP4J, а для тестування — JUnit 5 і JaCoCo. Такий стек є достатньо надійним для дипломного проєкту та водночас доступним для локального запуску [10].

Контейнеризація винесена на рівень розгортання. Кожен логічний пристрій може запускатися як окремий контейнер із власною IP-адресою в ізольованій мережі 192.168.100.0/24. Це дозволяє моделювати топологію локальної мережі та перевіряти поведінку Zabbix так, ніби він взаємодіє з окремими вузлами. Контейнерний підхід також спрощує очищення середовища після експериментів [11].

Архітектура не ставить за мету повністю відтворити поведінку конкретного фізичного обладнання. Натомість вона моделює достатній набір параметрів, необхідний для перевірки принципів моніторингу. Такий рівень абстракції є виправданим, адже головною метою є не емуляція апаратної платформи, а верифікація логіки збору даних, сповіщень і реакції на інциденти [17, 18].

Важливою архітектурною вимогою є повторюваність результатів. Користувач повинен мати можливість запуснути сценарій, отримати передбачувані зміни показників, побачити відповідний алерт у Zabbix і повторити експеримент. Для цього стан симуляції має бути керованим, а початкові значення — визначеними. Це відрізняє навчальний симулятор від випадкового генератора метрик [11].

Надійність моніторингу значною мірою залежить від правильно підібраних порогів. Якщо пороги занадто низькі, система створює багато хибних спрацювань. Якщо занадто високі, реальна проблема може залишатися непоміченою. Симуляційний стенд допомагає експериментувати з порогамі без ризику для продуктивної мережі [17, 18].

У процесі проєктування важливо розглядати моніторинг як частину ширшого циклу керування інцидентами. Дані SNMP повинні не лише відображатися на графіках, а й приводити до зрозумілих дій: повідомлення відповідальної особи, створення запису в журналі, запуск діагностики або оновлення статусу у системі обліку інцидентів.

З точки зору супроводу, код системи має бути розділений на невеликі класи з чіткими відповідальностями. Це спрощує тестування, дозволяє швидко знаходити помилки та зменшує ризик побічних ефектів під час внесення змін. Такий підхід відповідає загальним принципам інженерії програмного забезпечення.

Використання Docker Compose робить стенд зручним для передачі між користувачами. Якщо всі залежності описані у файлах проєкту, інший студент

або викладач може відтворити середовище без складної інструкції. Це особливо важливо для дипломної роботи, де результат має бути перевірним і демонстраційним [12].

2.2 Компонентна модель системи

Компонентна модель системи включає кілька ключових класів і служб. Центральним елементом є SnmpAgent, який приймає SNMP-запити, визначає потрібний OID і повертає значення з поточного стану пристрою. Він реалізує протокольну взаємодію, але не повинен містити складної бізнес-логіки симуляції. Це дозволяє тестувати оброблення запитів незалежно від сценаріїв.

Взаємозв'язок основних компонентів розробленого програмного комплексу подано на рисунку 2.1.

Компонентна модель розробленого програмного комплексу

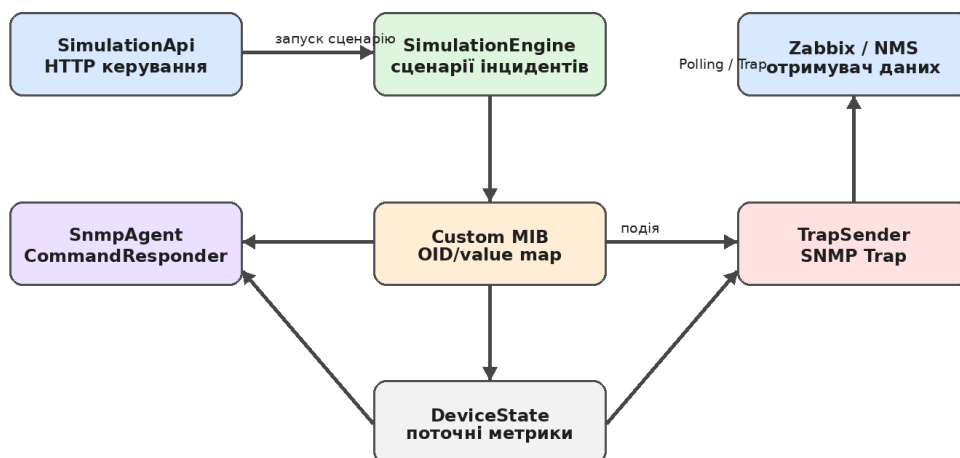


Рисунок 2.1 – Компонентна модель розробленого програмного комплексу

SimulationEngine відповідає за зміну внутрішнього стану пристрою відповідно до обраного сценарію. Він може підвищувати навантаження, змінювати статус інтерфейсу, збільшувати кількість помилок, імітувати недоступність сервісу або формувати ознаки підозрілої активності. Рушій

сценаріїв є місцем, де описується поведінка системи у часі та взаємозв'язок між окремими показниками.

TrapSender інкапсулює логіку формування і надсилання SNMP Trap-повідомлень. Його завдання полягає у перетворенні внутрішньої події на протокольне повідомлення, яке може бути прийняте Zabbix або іншою NMS. Відокремлення цього компонента дозволяє змінювати формат або адресата Trap без впливу на код сценаріїв і агента [2, 9].

SimulationApi забезпечує HTTP-інтерфейс для керування сценаріями. Через API користувач або тестовий скрипт може запустити інцидент, зупинити його, перевірити поточний стан або повернути систему до базового режиму. Такий інтерфейс зручний для демонстрації та автоматизації, оскільки не потребує ручного втручання всередину контейнера.

Окремим логічним компонентом є сховище стану пристрою. Воно містить значення OID, службову інформацію, часові мітки та параметри активного сценарію. У простій реалізації це може бути in-memory модель, що зберігається у межах процесу. Для дипломного стенду такого підходу достатньо, оскільки після перезапуску контейнера середовище може повертатися до початкового стану.

Компонентна модель підтримує розширення через додавання нових типів пристроїв і нових сценаріїв. Наприклад, якщо потрібно змодельовати бездротову точку доступу, достатньо описати її набір OID, базові показники та сценарії зміни стану. При цьому загальні компоненти SNMP-обміну, TrapSender і HTTP API можуть залишатися незмінними.

Надійність моніторингу значною мірою залежить від правильно підібраних порогів. Якщо пороги занадто низькі, система створює багато хибних спрацювань. Якщо занадто високі, реальна проблема може залишатися непоміченою. Симуляційний стенд допомагає експериментувати з порогоми без ризику для продуктивної мережі [17, 18].

У процесі проєктування важливо розглядати моніторинг як частину ширшого циклу керування інцидентами. Дані SNMP повинні не лише відображатися на графіках, а й приводити до зрозумілих дій: повідомлення відповідальної особи, створення запису в журналі, запуск діагностики або оновлення статусу у системі обліку інцидентів.

З точки зору супроводу, код системи має бути розділений на невеликі класи з чіткими відповідальностями. Це спрощує тестування, дозволяє швидко знаходити помилки та зменшує ризик побічних ефектів під час внесення змін. Такий підхід відповідає загальним принципам інженерії програмного забезпечення.

Таблиця 2.1 – Компоненти програмного комплексу

Компонент	Призначення
SnmpAgent	Оброблення SNMP-запитів і повернення значень OID.
SimulationEngine	Керування сценаріями та зміна стану симульованого пристрою.
TrapSender	Формування та надсилання SNMP Trap-повідомлень.
SimulationApi	HTTP-інтерфейс для запуску й зупинки сценаріїв.
DeviceState	Зберігання поточних значень показників та службової інформації.

Використання Docker Compose робить стенд зручним для передачі між користувачами. Якщо всі залежності описані у файлах проєкту, інший студент або викладач може відтворити середовище без складної інструкції. Це особливо важливо для дипломної роботи, де результат має бути перевірним і демонстраційним [12].

2.3 Проектування Custom MIB

Custom MIB у роботі призначена для представлення специфічних показників симуляційного середовища. Її коренем обрано Enterprise OID 1.3.6.1.4.1.99999, у межах якого створюються підгілки для інформації про пристрій, ресурсних показників, мережевих інтерфейсів, сценаріїв і подій безпеки. Така організація дозволяє відокремити власні об'єкти від стандартних MIB і уникнути неоднозначності [3, 4].

Під час проектування MIB визначаються не лише числові OID, а й семантика кожного параметра. Наприклад, об'єкт `deviceRole` може містити роль вузла, `activeScenario` — назву поточного сценарію, `incidentSeverity` — числовий рівень критичності, `simulatedCpuLoad` — відсоток завантаження процесора, `interfaceErrorRate` — умовну частоту помилок інтерфейсу. Для кожного параметра потрібно вказати тип і допустимий діапазон [3, 4].

Особливу увагу слід приділити стабільності OID. Після інтеграції з Zabbix зміна числового ідентифікатора призведе до втрати зв'язку між `item` і агентом. Тому MIB має розвиватися еволюційно: нові параметри додаються у вільні гілки, а існуючі не перейменовуються і не змінюють тип без вагомої причини. Це відповідає загальним принципам підтримки API [3, 4].

Custom MIB також корисна для документування системи. Вона слугує формальним описом того, які дані доступні для моніторингу. Для користувача Zabbix це спрощує створення шаблону, оскільки кожен `item` може бути пов'язаний з конкретним OID і поясненням. Для розробника MIB є орієнтиром під час реалізації мапінгу між станом пристрою та SNMP-відповідями [3, 4].

У межах дипломного проєкту доцільно поєднувати стандартні об'єкти MIB-II з власними об'єктами. Стандартні OID дозволяють NMS сприймати симульований вузол як типовий мережевий пристрій, а власні OID додають

доменну інформацію про сценарії та інциденти. Саме така комбінація робить стенд реалістичним і водночас керованим [3, 4].

Для перевірки правильності MIB можна використовувати SNMP-клієнти, такі як `snmpget` або `snmpwalk`, а також вбудовані засоби Zabbix. Перевірка повинна підтвердити, що агент повертає значення потрібних типів, таблиці проходяться послідовно, недоступні OID обробляються коректно, а зміна сценарію відображається у відповідних параметрах без перезапуску системи [3, 4].

Надійність моніторингу значною мірою залежить від правильно підібраних порогів. Якщо пороги занадто низькі, система створює багато хибних спрацювань. Якщо занадто високі, реальна проблема може залишатися непоміченою. Симуляційний стенд допомагає експериментувати з порогоми без ризику для продуктивної мережі [17, 18].

У процесі проєктування важливо розглядати моніторинг як частину ширшого циклу керування інцидентами. Дані SNMP повинні не лише відображатися на графіках, а й приводити до зрозумілих дій: повідомлення відповідальної особи, створення запису в журналі, запуск діагностики або оновлення статусу у системі обліку інцидентів.

З точки зору супроводу, код системи має бути розділений на невеликі класи з чіткими відповідальностями. Це спрощує тестування, дозволяє швидко знаходити помилки та зменшує ризик побічних ефектів під час внесення змін. Такий підхід відповідає загальним принципам інженерії програмного забезпечення.

Використання Docker Compose робить стенд зручним для передачі між користувачами. Якщо всі залежності описані у файлах проєкту, інший студент або викладач може відтворити середовище без складної інструкції. Це особливо важливо для дипломної роботи, де результат має бути перевірим і демонстраційним [12].

2.4 Модель симуляції подій та інцидентів

Модель симуляції подій описує, як змінюється стан пристрою у відповідь на запуск сценарію. На відміну від простого встановлення фіксованого значення, симуляція повинна враховувати часову динаміку. Наприклад, під час сценарію перевантаження трафік може зростати поступово, потім певний час залишатися на високому рівні, а після завершення сценарію повертатися до нормального діапазону.

У роботі передбачено три базові сценарії кіберінцидентів. Перший сценарій імітує перевантаження мережевого каналу або DDoS-подібну активність, що проявляється у зростанні трафіку та навантаження. Другий сценарій моделює деградацію інтерфейсу, коли збільшується кількість помилок і можливі зміни операційного статусу. Третій сценарій пов'язаний із підозрілою активністю або відмовою сервісу, що впливає на доступність вузла та генерує подію безпеки [20].

Послідовність оброблення інциденту від запуску сценарію до спрацювання триггера у Zabbix показано на рисунку 2.2.

Послідовність оброблення симуляційного інциденту

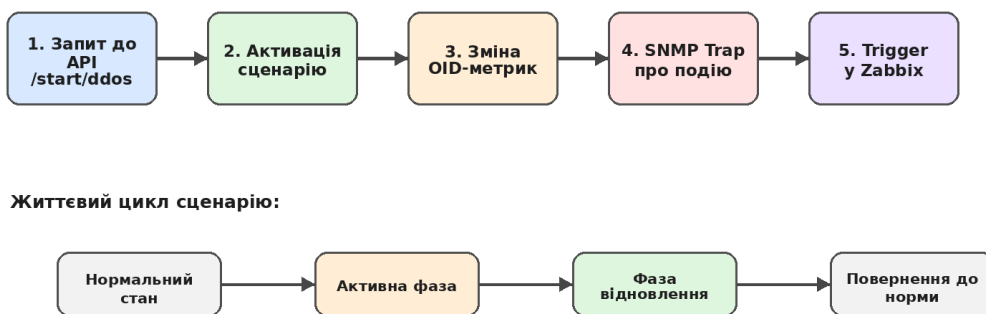


Рисунок 2.2 – Послідовність оброблення симуляційного інциденту

Кожен сценарій має початковий стан, активну фазу та фазу відновлення. Початковий стан відповідає нормальній роботі пристрою. Активна фаза змінює

ключові метрики та може ініціювати Trap-повідомлення. Фаза відновлення поступово повертає значення до базових. Така структура робить сценарії зрозумілими для демонстрації й дозволяє перевірити, чи NMS правильно реагує не лише на виникнення, а й на завершення проблеми [2, 9].

Для уникнення хаотичної поведінки сценарії повинні мати параметри: тривалість, інтенсивність, список змінюваних OID, пороги для надсилання Trap і правило завершення. Наприклад, сценарій перевантаження може мати інтенсивність 80%, що визначає максимальне значення використання інтерфейсу. У майбутньому такі параметри можна винести у конфігураційний файл або передавати через API.

Модель інцидентів також повинна враховувати взаємозв'язок показників. Якщо зростає обсяг трафіку, може збільшуватися навантаження CPU, кількість оброблених пакетів і затримка. Якщо інтерфейс переходить у стан down, лічильники трафіку можуть припинити зростати, але має з'явитися подія зміни статусу. Такі залежності підвищують правдоподібність симуляції.

Результатом моделювання є послідовність значень, яку отримує система моніторингу. Саме на цій послідовності перевіряються item, trigger, dashboard і notification. Якщо сценарій сформовано коректно, адміністратор бачить у Zabbix логічну картину: нормальний стан, поступове погіршення, спрацювання тригера, повідомлення про інцидент і повернення до норми після завершення сценарію [11].

Надійність моніторингу значною мірою залежить від правильно підібраних порогів. Якщо пороги занадто низькі, система створює багато хибних спрацювань. Якщо занадто високі, реальна проблема може залишатися непоміченою. Симуляційний стенд допомагає експериментувати з порогоми без ризику для продуктивної мережі [17, 18].

У процесі проектування важливо розглядати моніторинг як частину ширшого циклу керування інцидентами. Дані SNMP повинні не лише

відображатися на графіках, а й приводити до зрозумілих дій: повідомлення відповідальної особи, створення запису в журналі, запуск діагностики або оновлення статусу у системі обліку інцидентів.

З точки зору супроводу, код системи має бути розділений на невеликі класи з чіткими відповідальностями. Це спрощує тестування, дозволяє швидко знаходити помилки та зменшує ризик побічних ефектів під час внесення змін. Такий підхід відповідає загальним принципам інженерії програмного забезпечення.

Таблиця 2.2 – Приклади сценаріїв симуляції

Сценарій	Ознаки у моніторингу	Очікувана реакція Zabbix
Перевантаження каналу	Зростання трафіку, CPU, incidentSeverity.	Спрацювання тригера високого навантаження.
Деградація інтерфейсу	Зростання помилок, можливий стан down.	Алерт щодо проблеми інтерфейсу.
Відмова сервісу	Зміна доступності, Тгар про подію.	Створення події та повідомлення адміністратору.

Використання Docker Compose робить стенд зручним для передачі між користувачами. Якщо всі залежності описані у файлах проекту, інший студент або викладач може відтворити середовище без складної інструкції. Це особливо важливо для дипломної роботи, де результат має бути перевірним і демонстраційним [12].

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Реалізація SNMP-агента

Реалізація SNMP-агента є центральною частиною програмного комплексу. Агент повинен приймати UDP-пакети з SNMP-запитами, розпізнавати тип операції, знаходити потрібний OID у внутрішній моделі та повертати відповідне значення. Для цього використовується бібліотека SNMP4J, яка надає готові класи для роботи з транспортом, PDU, адресами, типами даних і обробниками запитів [10].

Клас `SnmAgent` реалізує інтерфейс `CommandResponder`. Це дозволяє отримувати повідомлення від SNMP4J і формувати відповідь на основі внутрішньої логіки. У найпростішому випадку обробник читає OID з PDU, звертається до мапи значень, перетворює результат у відповідний SNMP-тип і додає його до response PDU. Якщо OID невідомий, агент має повернути коректну помилку або ознаку відсутності об'єкта [10].

Важливою частиною реалізації є типізація значень. SNMP використовує власні типи, зокрема `Integer32`, `OctetString`, `Gauge32`, `Counter32`, `Counter64`, `TimeTicks` та `OID`. Неправильний вибір типу може призвести до помилок у Zabbix або некоректного відображення графіків. Наприклад, лічильники трафіку бажано представляти як `Counter64`, а поточне завантаження — як `Gauge32` або `Integer32` у відсотках [11].

Для підтримки кількох ролей пристроїв агент може отримувати конфігурацію під час запуску контейнера. Змінні середовища або конфігураційний файл визначають ім'я вузла, роль, IP-адресу, початкові значення та набір активних OID. Це дозволяє використовувати один і той самий образ контейнера для Router, Switch, Server та IED-контролера, змінюючи лише параметри запуску.

Особливу увагу приділено потокобезпечності. SNMP-запити можуть надходити паралельно, а SimulationEngine у цей самий час може змінювати значення показників. Тому доступ до стану пристрою має бути організований так, щоб уникати не консистентних значень або помилок читання. Для дипломного стенду можна використовувати потоко безпечні структури даних або синхронізований доступ до спільної моделі.

Коректність агента перевіряється за допомогою модульних тестів і ручного опитування через SNMP-клієнти. Тести мають покривати відомі OID, невідомі OID, різні типи значень, зміну стану під час активного сценарію та базові помилки. Така перевірка гарантує, що агент є надійною основою для подальшої інтеграції із Zabbix [11].

Використання Docker Compose робить стенд зручним для передачі між користувачами. Якщо всі залежності описані у файлах проекту, інший студент або викладач може відтворити середовище без складної інструкції. Це особливо важливо для дипломної роботи, де результат має бути перевірним і демонстраційним [12].

Таблиця 3.1 – Приклад групування OID у Custom MIB

Група	Приклади параметрів
deviceInfo	deviceName, deviceRole, deviceUptime
resourceMetrics	simulatedCpuLoad, simulatedMemoryUsage
networkMetrics	interfaceStatus, trafficIn, trafficOut, interfaceErrorRate
incidentMetrics	activeScenario, incidentSeverity, generatedEvents

Під час інтеграції із Zabbix бажано створювати шаблони, а не налаштовувати кожен host вручну. Шаблон дозволяє повторно використовувати items, triggers і графіки для однотипних пристроїв. У симуляційному стенді це

корисно, якщо потрібно додати новий контейнер із тією самою роллю або порівняти поведінку кількох вузлів [11].

Для підвищення реалістичності симуляції можна використовувати не лише статичні значення, а й функції зміни у часі. Наприклад, навантаження може коливатися у межах нормального діапазону, а під час інциденту зростати за заданою кривою. Це дозволяє отримати графіки, схожі на дані реальних систем моніторингу [17, 18].

У дипломній роботі важливо показати зв'язок між теоретичною частиною та практичною реалізацією. Аналіз SNMP, MIB і засобів симуляції має переходити у конкретні архітектурні рішення, а опис коду — у результати тестування та демонстрацію в Zabbix. Така логіка робить пояснювальну записку цілісною [1].

3.2 Реалізація механізму Trap-повідомлень

Trap-повідомлення використовуються для асинхронного інформування NMS про події. На відміну від polling, коли менеджер сам ініціює запит, Trap надсилається агентом у момент виникнення ситуації, яку потрібно зафіксувати. У контексті симуляційного стенду це особливо корисно, оскільки запуск сценарію може миттєво створювати повідомлення про початок інциденту [2, 9].

Клас TrapSender відповідає за формування PDU, встановлення адреси отримувача, додавання varbind-ів і надсилання повідомлення через SNMP4J. Varbind-и можуть містити OID події, рівень критичності, назву пристрою, назву сценарію, часову мітку та поточні значення ключових показників. Такий набір даних дозволяє системі моніторингу не лише зафіксувати факт події, а й відобразити її контекст [10].

Під час реалізації Trap важливо узгодити формат повідомлень із конфігурацією Zabbix. Якщо NMS очікує певні OID або текстові шаблони, агент

повинен формувати повідомлення у відповідному вигляді. У протилежному випадку Trap може бути отриманий, але не оброблений триггером. Тому тестування Trap-повідомлень має виконуватися разом із перевіркою правил обробки у Zabbix [2, 9].

Механізм Trap не повинен замінювати регулярне опитування. Найкращий результат досягається, коли Trap повідомляє про факт події, а polling підтверджує поточний стан показників. Наприклад, агент надсилає Trap про деградацію інтерфейсу, після чого Zabbix через SNMP items бачить зростання лічильника помилок. Така комбінація зменшує ризик втрати інформації через недоставлений UDP-пакет [11].

У системі передбачено можливість надсилання Trap під час початку, зміни та завершення сценарію. Це дозволяє будувати повний життєвий цикл інциденту. Для критичних сценаріїв можна надсилати повторні повідомлення або додаткові Trap на кожному етапі, але у дипломному стенді достатньо базової логіки, яка демонструє принцип асинхронного сповіщення.

Перевірка TrapSender виконується як модульно, так і інтеграційно. На модульному рівні можна перевіряти правильність створення PDU та заповнення varbind-ів. На інтеграційному рівні необхідно переконатися, що Zabbix отримує повідомлення, коректно його парсить і відображає у журналі подій. Це підтверджує готовність механізму до практичного використання [11].

Використання Docker Compose робить стенд зручним для передачі між користувачами. Якщо всі залежності описані у файлах проекту, інший студент або викладач може відтворити середовище без складної інструкції. Це особливо важливо для дипломної роботи, де результат має бути перевірим і демонстраційним [12].

Під час інтеграції із Zabbix бажано створювати шаблони, а не налаштовувати кожен host вручну. Шаблон дозволяє повторно використовувати items, triggers і графіки для однотипних пристроїв. У симуляційному стенді це

корисно, якщо потрібно додати новий контейнер із тією самою роллю або порівняти поведінку кількох вузлів [11].

Для підвищення реалістичності симуляції можна використовувати не лише статичні значення, а й функції зміни у часі. Наприклад, навантаження може коливатися у межах нормального діапазону, а під час інциденту зростати за заданою кривою. Це дозволяє отримати графіки, схожі на дані реальних систем моніторингу [17, 18].

У дипломній роботі важливо показати зв'язок між теоретичною частиною та практичною реалізацією. Аналіз SNMP, MIB і засобів симуляції має переходити у конкретні архітектурні рішення, а опис коду — у результати тестування та демонстрацію в Zabbix. Така логіка робить пояснювальну записку цілісною [1].

3.3 Контейнеризація та мережеве середовище

Контейнеризація є ключовим засобом забезпечення відтворюваності симуляційного стенду. Docker Compose дозволяє описати кілька контейнерів, їхні образи, змінні середовища, мережу, порти та залежності в одному файлі. Завдяки цьому користувач може розгорнути весь комплекс однією командою, не налаштовуючи кожен вузол вручну [12].

У роботі передбачено ізольовану мережу з підмережею 192.168.100.0/24. У ній розміщуються логічні пристрої Router, Switch, Server та IED-контролер. Кожен контейнер має власну адресу, що дозволяє Zabbix опитувати його як окремий мережевий вузол. Така схема наближує стенд до реальної локальної мережі, але залишається безпечною для запуску на одному комп'ютері [11].

Схему контейнеризованого стенду та взаємозв'язок його вузлів наведено на рисунку 3.1.

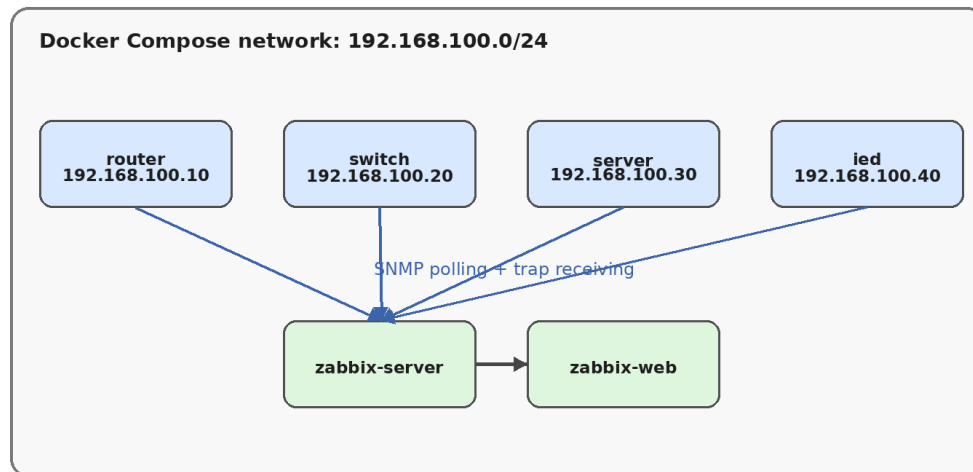


Рисунок 3.1 – Контейнеризоване середовище симуляційного стенду

Використання контейнерів спрощує експерименти зі сценаріями. Якщо під час тестування стан системи став некоректним, середовище можна швидко перезапустити та повернути до початкової конфігурації. Це особливо корисно для дипломної демонстрації, де потрібно мати стабільний і передбачуваний стенд незалежно від попередніх запусків.

Контейнерний підхід також дозволяє ізолювати залежності. Java-застосунок, SNMP-порти, HTTP API та конфігураційні файли знаходяться всередині образу або передаються через volume. Користувачу не потрібно встановлювати Java, бібліотеки чи додаткові пакети на основну систему, якщо образ підготовлено правильно. Це зменшує кількість помилок, пов'язаних із середовищем.

Для Zabbix можливі два підходи: використання зовнішнього вже встановленого сервера або додавання Zabbix-компонентів до docker-compose. У дипломному проєкті важливо описати інтеграцію так, щоб вона була зрозумілою для обох варіантів. Основними параметрами є IP-адреси агентів, SNMP community, список OID, періодичність опитування та правила приймання Trap [11].

Контейнеризація не усуває потреби у правильній мережевій діагностиці. Під час налаштування необхідно перевіряти доступність UDP-портів, маршрутизацію між контейнерами, відповідність адрес у Zabbix і правильність binding SNMP-агента. Для цього використовуються базові інструменти на кшталт ping, netstat, tcpdump, snmpget і snmpwalk [11].

Використання Docker Compose робить стенд зручним для передачі між користувачами. Якщо всі залежності описані у файлах проекту, інший студент або викладач може відтворити середовище без складної інструкції. Це особливо важливо для дипломної роботи, де результат має бути перевірним і демонстраційним [12].

Під час інтеграції із Zabbix бажано створювати шаблони, а не налаштовувати кожен host вручну. Шаблон дозволяє повторно використовувати items, triggers і графіки для однотипних пристроїв. У симуляційному стенді це корисно, якщо потрібно додати новий контейнер із тією самою роллю або порівняти поведінку кількох вузлів [11].

Таблиця 3.2 – Логічні вузли стенду

Вузол	Роль	Приклад IP-адреси
Router	Маршрутизація та трафік між сегментами	192.168.100.10
Switch	Порти, статуси інтерфейсів, помилки кадрів	192.168.100.20
Server	CPU, пам'ять, доступність сервісу	192.168.100.30
IED	Спеціалізований контрольований пристрій	192.168.100.40

Для підвищення реалістичності симуляції можна використовувати не лише статичні значення, а й функції зміни у часі. Наприклад, навантаження може коливатися у межах нормального діапазону, а під час інциденту зростати за

заданою кривою. Це дозволяє отримати графіки, схожі на дані реальних систем моніторингу [17, 18].

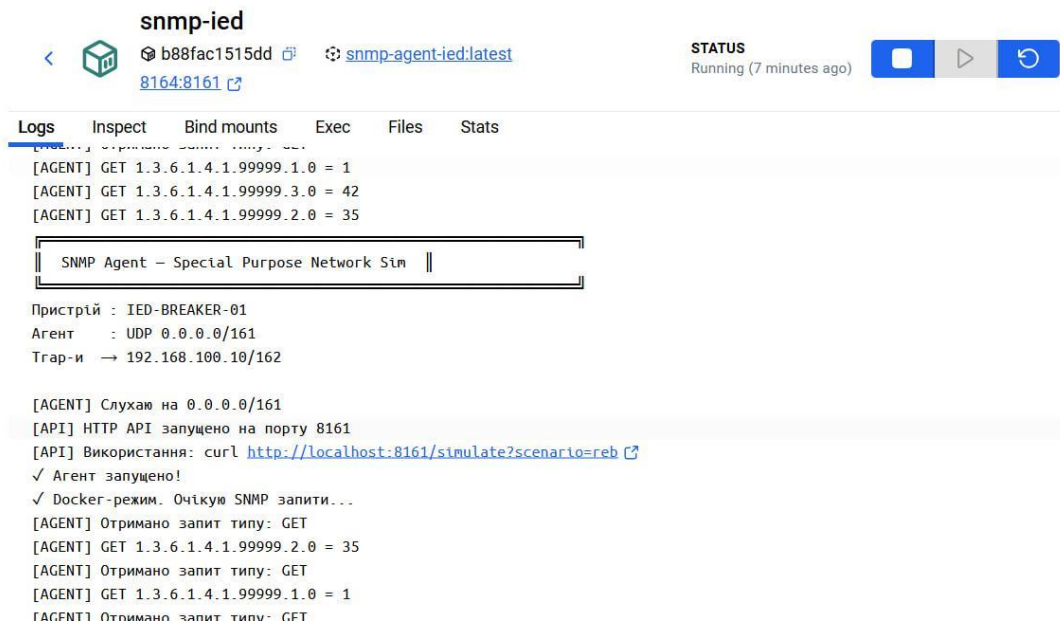


Рисунок 3.2 –SNMP-агент в імітованому пристрої

У дипломній роботі важливо показати зв'язок між теоретичною частиною та практичною реалізацією. Аналіз SNMP, MIB і засобів симуляції має переходити у конкретні архітектурні рішення, а опис коду — у результати тестування та демонстрацію в Zabbix. Така логіка робить пояснювальну записку цілісною [1].

3.4 Інтеграція із Zabbix

Zabbix використовується як система мережевого моніторингу, що отримує показники від симульованих SNMP-агентів. Для кожного логічного пристрою створюється host із відповідною IP-адресою, SNMP-інтерфейсом і шаблоном item. Items визначають, які OID потрібно опитувати, з якою періодичністю та у якому форматі зберігати отримані значення [11].

Першим етапом інтеграції є перевірка доступності агентів. Якщо Zabbix не може отримати значення за базовим OID, необхідно перевірити community string, IP-адресу, порт, правила мережі та роботу контейнера. Лише після підтвердження базового зв'язку доцільно створювати повний набір item і trigger [11].

Items у Zabbix можна розділити на базові та сценарні. Базові відображають постійні характеристики пристрою: ім'я, роль, uptime, стан інтерфейсів, поточне завантаження. Сценарні пов'язані з Custom MIB і показують активний інцидент, рівень критичності, кількість подій або спеціальні індикатори. Такий поділ спрощує побудову dashboard і аналіз проблем [11].

Triggers визначають умови, за яких Zabbix вважає ситуацію проблемною. Наприклад, якщо завантаження інтерфейсу перевищує заданий поріг протягом кількох циклів опитування, створюється подія. Якщо статус інтерфейсу змінюється на down, генерується алерт високої важливості. Для сценарних OID можна створити тригер, що реагує на не нульовий рівень incidentSeverity [11].

Інтеграція Trap-повідомлень потребує налаштування приймання SNMP traps у Zabbix. Після отримання повідомлення система може створювати подію або зберігати текст Trap у відповідному item. У дипломному стенді Trap доцільно використовувати для миттєвої фіксації початку сценарію, а регулярні SNMP items — для підтвердження зміни числових показників [2, 9].

У результаті інтеграції користувач отримує повний цикл моніторингу: агент надає дані, Zabbix регулярно їх опитує, dashboard відображає графіки,

triggers визначають проблеми, а Trap-повідомлення створюють асинхронні події. Це підтверджує, що розроблений програмний комплекс виконує практичну функцію, а не обмежується лише прототипом на рівні коду [2, 9].

Використання Docker Compose робить стенд зручним для передачі між користувачами. Якщо всі залежності описані у файлах проекту, інший студент або викладач може відтворити середовище без складної інструкції. Це особливо важливо для дипломної роботи, де результат має бути перевірним і демонстраційним [12].

Під час інтеграції із Zabbix бажано створювати шаблони, а не налаштовувати кожен host вручну. Шаблон дозволяє повторно використовувати items, triggers і графіки для однотипних пристроїв. У симуляційному стенді це корисно, якщо потрібно додати новий контейнер із тією самою роллю або порівняти поведінку кількох вузлів [11].

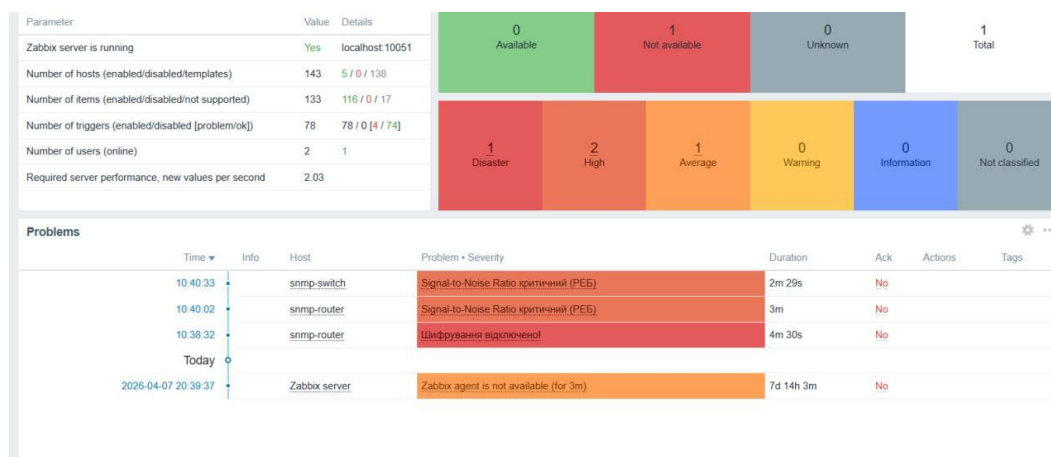


Рисунок 3.2 – Трапи в середовищі Zabbix

Для підвищення реалістичності симуляції можна використовувати не лише статичні значення, а й функції зміни у часі. Наприклад, навантаження може коливатися у межах нормального діапазону, а під час інциденту зростати за

заданою кривою. Це дозволяє отримати графіки, схожі на дані реальних систем моніторингу [17, 18].

У дипломній роботі важливо показати зв'язок між теоретичною частиною та практичною реалізацією. Аналіз SNMP, MIB і засобів симуляції має переходити у конкретні архітектурні рішення, а опис коду — у результати тестування та демонстрацію в Zabbix. Така логіка робить пояснювальну записку цілісною [1].

4 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ

4.1 Стратегія тестування

Тестування програмного комплексу має підтвердити правильність роботи як окремих компонентів, так і системи в цілому. Оскільки рішення включає протокольну взаємодію, симуляцію стану, HTTP API, контейнеризацію та інтеграцію із Zabbix, необхідно застосовувати кілька рівнів перевірки. Модульні тести перевіряють класи ізольовано, інтеграційні тести перевіряють взаємодію компонентів, а ручна верифікація у Zabbix підтверджує практичну придатність [11].

Узагальнену стратегію перевірки програмного комплексу наведено на рисунку 4.1.

Стратегія тестування програмного комплексу

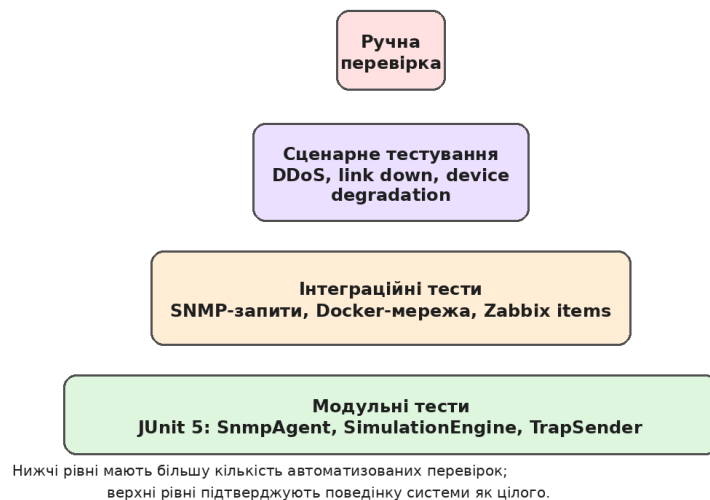


Рисунок 4.1 – Стратегія тестування програмного комплексу

Стратегія тестування базується на принципі поступового нарощування складності. Спочатку перевіряються прості функції: отримання значення за OID, зміна стану, формування Trap. Далі перевіряється робота сценаріїв, які змінюють

кілька показників. Після цього виконується розгортання у Docker Compose і перевірка доступності кожного контейнера. Завершальним етапом є інтеграційна перевірка з NMS [12].

Для модульного тестування використовується JUnit 5. Тести мають бути швидкими, повторюваними та незалежними від зовнішньої інфраструктури. Якщо компонент залежить від мережі або часу, доцільно використовувати підстановки, мок-об'єкти або керовані джерела часу. Це дозволяє стабільно перевіряти логіку без випадкових збоїв, пов'язаних із середовищем [13].

Покриття коду вимірюється за допомогою JaCoCo. Досягнення високого покриття саме по собі не гарантує відсутності помилок, але показує, що основні гілки логіки виконуються тестами. У роботі зазначено досягнення 98% покриття, що є високим результатом для дипломного проекту. Водночас важливо аналізувати не лише відсоток, а й якість перевірок [14].

Інтеграційне тестування має враховувати особливості SNMP як мережевого протоколу. Необхідно перевіряти таймаути, повторні запити, поведінку при недоступності контейнера, правильність типів даних і реакцію Zabbix на зміну значень. Частина таких перевірок може виконуватися автоматично, але остаточна демонстрація часто включає перегляд графіків, подій і тригерів у веб-інтерфейсі [11].

Критерієм успішного тестування є не лише проходження тестів, а й відповідність поведінки системи очікуваному сценарію. Якщо при запуску інциденту Zabbix отримує дані, тригер спрацьовує у правильний момент, Trap з'являється у журналі, а після завершення сценарію стан повертається до норми, можна вважати, що основна функціональність підтверджена [11].

Під час практичного використання системи важливо документувати кожен OID, який передається до NMS. Документація повинна містити числовий ідентифікатор, коротку назву, тип даних, допустимі значення, одиницю вимірювання та приклад інтерпретації. Такий опис зменшує ризик помилок під

час створення item і trigger, а також полегшує підтримку системи іншими користувачами.

Окремою перевагою симуляційного підходу є можливість безпечно відтворювати рідкісні або небезпечні події. У реальній мережі адміністратор не завжди може навмисно вимкнути інтерфейс або створити перевантаження, особливо якщо обладнання обслуговує критичні сервіси. У контейнерному середовищі такі дії не впливають на продуктивну інфраструктуру.

При виборі періодичності опитування необхідно знаходити баланс між актуальністю даних і навантаженням на мережу. Занадто часті запити можуть створювати зайвий трафік і навантаження на агент, а занадто рідкісні — призводити до запізненого виявлення проблем. Для демонстраційного стенду періоди можна скоротити, щоб швидше показати зміну стану.

Надійність моніторингу значною мірою залежить від правильно підібраних порогів. Якщо пороги занадто низькі, система створює багато хибних спрацювань. Якщо занадто високі, реальна проблема може залишатися непоміченою. Симуляційний стенд допомагає експериментувати з порогоми без ризику для продуктивної мережі [17, 18].

У процесі проектування важливо розглядати моніторинг як частину ширшого циклу керування інцидентами. Дані SNMP повинні не лише відображатися на графіках, а й приводити до зрозумілих дій: повідомлення відповідальної особи, створення запису в журналі, запуск діагностики або оновлення статусу у системі обліку інцидентів.

4.2 Автоматизоване тестування

Автоматизоване тестування охоплює класи `SnmpAgentTest`, `SimulationEngineTest`, `TrapSenderTest` і `SnmpTesterTest`. Такий набір дозволяє перевірити протокольний шар, логіку сценаріїв, формування повідомлень і

допоміжні засоби перевірки SNMP-зв'язку. Загальна кількість тестів становить 40, що дозволяє покрити основні варіанти використання системи.

`SnmpAgentTest` перевіряє, чи агент правильно відповідає на запити до відомих OID, чи обробляє невідомі OID, чи повертає значення правильного типу та чи враховує зміни стану, внесені рушієм симуляції. Особливу увагу приділено тому, щоб відповідь агента була придатною для оброблення системою моніторингу, а не лише формально створювала PDU [17, 18].

`SimulationEngineTest` перевіряє переходи між станами сценаріїв. Тести мають підтвердити, що сценарій запускається з нормального стану, коректно змінює показники, не допускає конфліктів між одночасними сценаріями та повертає систему до базових значень після завершення. Для таких тестів важливо контролювати час або використовувати спрощену модель кроків.

`TrapSenderTest` фокусується на правильності формування Trap-повідомлень. Перевіряється наявність потрібних varbind-ів, коректність OID події, значення рівня критичності та ідентифікація пристрою. Якщо надсилання виконується через мережу, у тестах може використовуватися локальний приймач або мок транспортного шару, щоб уникнути залежності від зовнішнього Zabbix [2, 9].

`SnmpTesterTest` перевіряє допоміжний код, який використовується для діагностики агента. Такий компонент може виконувати тестові SNMP-запити, перевіряти доступність і повертати зрозумілий результат для користувача. Наявність такого інструмента спрощує виявлення проблем під час розгортання, оскільки дозволяє відокремити помилки агента від помилок конфігурації Zabbix [11].

Результати автоматизованого тестування свідчать про готовність основних компонентів до інтеграції. Досягнуте покриття JaCoCo на рівні 98% демонструє, що більшість коду виконується тестами. Проте у висновках тестування варто

зазначити, що остаточна якість системи підтверджується не лише coverage, а й інтеграційними перевітками у контейнерному середовищі [14].

Під час практичного використання системи важливо документувати кожен OID, який передається до NMS. Документація повинна містити числовий ідентифікатор, коротку назву, тип даних, допустимі значення, одиницю вимірювання та приклад інтерпретації. Такий опис зменшує ризик помилок під час створення item і trigger, а також полегшує підтримку системи іншими користувачами.

Окремою перевагою симуляційного підходу є можливість безпечно відтворювати рідкісні або небезпечні події. У реальній мережі адміністратор не завжди може навмисно вимкнути інтерфейс або створити перевантаження, особливо якщо обладнання обслуговує критичні сервіси. У контейнерному середовищі такі дії не впливають на продуктивну інфраструктуру.

При виборі періодичності опитування необхідно знаходити баланс між актуальністю даних і навантаженням на мережу. Занадто часті запити можуть створювати зайвий трафік і навантаження на агент, а занадто рідкісні — призводити до запізненого виявлення проблем. Для демонстраційного стенду періоди можна скоротити, щоб швидше показати зміну стану.

Таблиця 4.1 – Набір автоматизованих тестів

Тестовий клас	Що перевіряється
SnmpAgentTest	Оброблення OID, типи значень, відповіді агента.
SimulationEngineTest	Переходи сценаріїв, зміна показників, відновлення стану.
TrapSenderTest	Формування Trap і заповнення varbind-ів.
SnmpTesterTest	Діагностичні SNMP-запити та оброблення результатів.

Надійність моніторингу значною мірою залежить від правильно підібраних порогів. Якщо пороги занадто низькі, система створює багато хибних спрацювань. Якщо занадто високі, реальна проблема може залишатися непоміченою. Симуляційний стенд допомагає експериментувати з порогамі без ризику для продуктивної мережі [17, 18].

У процесі проектування важливо розглядати моніторинг як частину ширшого циклу керування інцидентами. Дані SNMP повинні не лише відображатися на графіках, а й приводити до зрозумілих дій: повідомлення відповідальної особи, створення запису в журналі, запуск діагностики або оновлення статусу у системі обліку інцидентів.

4.3 Інтеграційне тестування

Інтеграційне тестування проводиться після розгортання контейнерного середовища. Його мета полягає у перевірці взаємодії між симульованими агентами, мережею Docker, Zabbix і механізмом керування сценаріями. На цьому етапі важливо переконатися, що всі контейнери доступні за очікуваними IP-адресами, SNMP-порти відкриті, а Zabbix може виконувати опитування [11].

Перший тестовий сценарій полягає у базовому SNMP-опитуванні. Для кожного контейнера виконуються запити до системних OID і до об'єктів Custom MIB. Якщо значення повертаються коректно, можна переходити до створення items у Zabbix. Якщо виникають помилки таймауту або неправильного community, проблема вирішується до запуску складніших сценаріїв [3, 4].

Другий сценарій перевіряє зміну показників у часі. Через SimulationApi запускається інцидент, після чого Zabbix протягом кількох циклів опитування має отримати змінені значення. На графіках повинно бути видно перехід від нормального стану до аварійного. Якщо тригер налаштований правильно, він

створює проблему лише після досягнення заданої умови, а не одразу після короткого стрибка [11].

Третій сценарій перевіряє оброблення Trap-повідомлень. Під час запуску інциденту агент надсилає Trap на адресу Zabbix. У журналі подій або відповідному item має з'явитися запис із назвою сценарію, пристроєм і рівнем критичності. Ця перевірка підтверджує, що асинхронний канал сповіщень працює паралельно з регулярним опитуванням [2, 9].

Окремо перевіряється відновлення після завершення сценарію. Система повинна повернути показники до нормальних значень, а тригери мають перейти у стан ОК. Якщо після завершення інциденту проблема залишається активною, це може свідчити про неправильну формулу тригера, занадто довгий період оцінювання або помилку в SimulationEngine.

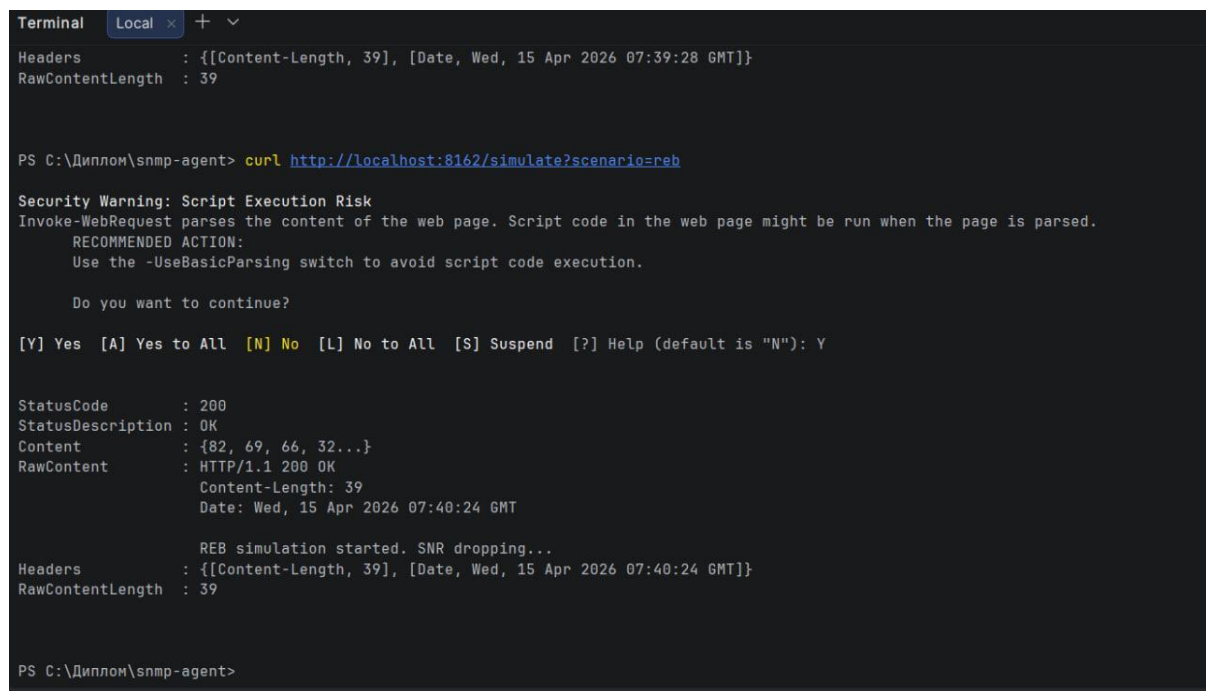
Результати інтеграційного тестування показують, що розроблений комплекс може використовуватися як стенд для перевірки систем моніторингу. Він дозволяє демонструвати повний цикл: запуск сценарію, зміну SNMP-показників, отримання Trap, спрацювання тригера, відображення проблеми та повернення до нормального стану. Це підтверджує практичну цінність роботи [17, 18].

Під час практичного використання системи важливо документувати кожен OID, який передається до NMS. Документація повинна містити числовий ідентифікатор, коротку назву, тип даних, допустимі значення, одиницю вимірювання та приклад інтерпретації. Такий опис зменшує ризик помилок під час створення item і trigger, а також полегшує підтримку системи іншими користувачами.

Окремою перевагою симуляційного підходу є можливість безпечно відтворювати рідкісні або небезпечні події. У реальній мережі адміністратор не завжди може навмисно вимкнути інтерфейс або створити перевантаження,

особливо якщо обладнання обслуговує критичні сервіси. У контейнерному середовищі такі дії не впливають на продуктивну інфраструктуру.

При виборі періодичності опитування необхідно знаходити баланс між актуальністю даних і навантаженням на мережу. Занадто часті запити можуть створювати зайвий трафік і навантаження на агент, а занадто рідкісні — призводити до запізненого виявлення проблем. Для демонстраційного стенду періоди можна скоротити, щоб швидше показати зміну стану.



```
Terminal Local x + v
Headers      : {[Content-Length, 39], [Date, Wed, 15 Apr 2026 07:39:28 GMT]}
RawContentLength : 39

PS C:\Диплом\snmp-agent> curl http://localhost:8162/simulate?scenario=reb

Security Warning: Script Execution Risk
Invoke-WebRequest parses the content of the web page. Script code in the web page might be run when the page is parsed.
RECOMMENDED ACTION:
  Use the -UseBasicParsing switch to avoid script code execution.

Do you want to continue?

[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): Y

StatusCode      : 200
StatusDescription : OK
Content         : {82, 69, 66, 32...}
RawContent      : HTTP/1.1 200 OK
                  Content-Length: 39
                  Date: Wed, 15 Apr 2026 07:40:24 GMT

                  REB simulation started. SNR dropping...
Headers        : {[Content-Length, 39], [Date, Wed, 15 Apr 2026 07:40:24 GMT]}
RawContentLength : 39

PS C:\Диплом\snmp-agent>
```

Рисунок 4.2 – Робота SimulationEngine

Надійність моніторингу значною мірою залежить від правильно підібраних порогів. Якщо пороги занадто низькі, система створює багато хибних спрацювань. Якщо занадто високі, реальна проблема може залишатися непоміченою. Симуляційний стенд допомагає експериментувати з порогамі без ризику для продуктивної мережі [17, 18].

У процесі проектування важливо розглядати моніторинг як частину ширшого циклу керування інцидентами. Дані SNMP повинні не лише відображатися на графіках, а й приводити до зрозумілих дій: повідомлення

відповідальної особи, створення запису в журналі, запуск діагностики або оновлення статусу у системі обліку інцидентів.

5 АНАЛІЗ РЕЗУЛЬТАТІВ ТА ПЕРСПЕКТИВИ РОЗВИТКУ

У результаті виконання дипломної роботи створено програмний комплекс для організації моніторингу локальної мережі за допомогою протоколу SNMP. Комплекс включає SNMP-агент, Custom MIB, механізм Trap-повідомлень, рушій симуляції інцидентів, HTTP API та контейнеризоване середовище розгортання. Його основне призначення полягає у створенні безпечного й відтворюваного стенду для перевірки систем мережевого моніторингу [2, 9].

Практична цінність рішення полягає в тому, що воно дозволяє тестувати моніторинг без фізичного обладнання. Це особливо корисно для навчання студентів, підготовки демонстрацій, перевірки шаблонів Zabbix і відпрацювання реакції на інциденти. Адміністратор або викладач може швидко запустити стенд, активувати сценарій і показати, як NMS реагує на зміну стану мережі [11].

Розроблений комплекс демонструє переваги поєднання SNMP polling і Trap-повідомлень. Регулярне опитування забезпечує історичні графіки та стабільний збір метрик, а Trap дозволяє швидко повідомити про подію. У реальних системах така комбінована модель є більш надійною, ніж використання лише одного механізму, оскільки вона поєднує періодичний контроль і асинхронне сповіщення [2, 9].

Серед обмежень поточної реалізації можна виділити спрощену модель поведінки пристроїв. Вона достатня для дипломного стенду, але не повністю відтворює особливості конкретного виробника обладнання. Також система орієнтована на невелику кількість контейнерів, тому для масштабування до сотень або тисяч агентів потрібно додатково оптимізувати конфігурацію, ресурси та генерацію даних.

Перспективним напрямом розвитку є підтримка SNMPv3 з автентифікацією та шифруванням. Це наблизить стенд до продуктивних

середовищ, де безпека доступу до даних моніторингу має важливе значення. Також можна додати веб-інтерфейс для запуску сценаріїв, редактор Custom MIB, імпорт конфігурацій Zabbix і сценарії, що описуються у YAML або JSON [6, 7, 8].

Ще одним напрямом розвитку є розширення бібліотеки сценаріїв. До базових інцидентів можна додати втрату пакетів, нестабільність інтерфейсу, повільне вичерпання пам'яті, перезапуск пристрою, аномалії у технологічному сегменті, підозрілу зміну конфігурації або каскадну відмову кількох вузлів. Це зробить стенд більш корисним для навчання та практичних тренувань.

У майбутньому комплекс може бути інтегрований у CI/CD процеси для автоматичної перевірки шаблонів моніторингу. Наприклад, після зміни конфігурації Zabbix можна запускати симуляційне середовище, активувати набір сценаріїв і перевіряти, чи створюються потрібні події. Такий підхід дозволить підвищити якість моніторингу та зменшити кількість помилок у продуктивному середовищі [11].

Загалом отримані результати підтверджують доцільність використання програмної симуляції для задач мережевого моніторингу. Рішення є достатньо простим для запуску, відкритим для модифікації та демонструє ключові принципи роботи SNMP, MIB, Zabbix і сценаріїв інцидентів. Це робить його корисним як для дипломного проекту, так і для подальших навчальних або дослідницьких робіт [11].

Під час практичного використання системи важливо документувати кожен OID, який передається до NMS. Документація повинна містити числовий ідентифікатор, коротку назву, тип даних, допустимі значення, одиницю вимірювання та приклад інтерпретації. Такий опис зменшує ризик помилок під час створення `item` і `trigger`, а також полегшує підтримку системи іншими користувачами.

Окремою перевагою симуляційного підходу є можливість безпечно відтворювати рідкісні або небезпечні події. У реальній мережі адміністратор не завжди може навмисно вимкнути інтерфейс або створити перевантаження, особливо якщо обладнання обслуговує критичні сервіси. У контейнерному середовищі такі дії не впливають на продуктивну інфраструктуру.

При виборі періодичності опитування необхідно знаходити баланс між актуальністю даних і навантаженням на мережу. Занадто часті запити можуть створювати зайвий трафік і навантаження на агент, а занадто рідкісні — призводити до запізненого виявлення проблем. Для демонстраційного стенду періоди можна скоротити, щоб швидше показати зміну стану.

Надійність моніторингу значною мірою залежить від правильно підібраних порогів. Якщо пороги занадто низькі, система створює багато хибних спрацювань. Якщо занадто високі, реальна проблема може залишатися непоміченою. Симуляційний стенд допомагає експериментувати з порогоми без ризику для продуктивної мережі [17, 18].

У процесі проектування важливо розглядати моніторинг як частину ширшого циклу керування інцидентами. Дані SNMP повинні не лише відображатися на графіках, а й приводити до зрозумілих дій: повідомлення відповідальної особи, створення запису в журналі, запуск діагностики або оновлення статусу у системі обліку інцидентів.

З точки зору супроводу, код системи має бути розділений на невеликі класи з чіткими відповідальностями. Це спрощує тестування, дозволяє швидко знаходити помилки та зменшує ризик побічних ефектів під час внесення змін. Такий підхід відповідає загальним принципам інженерії програмного забезпечення.

Використання Docker Compose робить стенд зручним для передачі між користувачами. Якщо всі залежності описані у файлах проекту, інший студент або викладач може відтворити середовище без складної інструкції. Це особливо

важливо для дипломної роботи, де результат має бути перевірним і демонстраційним [12].

Під час інтеграції із Zabbix бажано створювати шаблони, а не налаштовувати кожен host вручну. Шаблон дозволяє повторно використовувати items, triggers і графіки для однотипних пристроїв. У симуляційному стенді це корисно, якщо потрібно додати новий контейнер із тією самою роллю або порівняти поведінку кількох вузлів [11].

Для підвищення реалістичності симуляції можна використовувати не лише статичні значення, а й функції зміни у часі. Наприклад, навантаження може коливатися у межах нормального діапазону, а під час інциденту зростати за заданою кривою. Це дозволяє отримати графіки, схожі на дані реальних систем моніторингу [17, 18].

У дипломній роботі важливо показати зв'язок між теоретичною частиною та практичною реалізацією. Аналіз SNMP, MIB і засобів симуляції має переходити у конкретні архітектурні рішення, а опис коду — у результати тестування та демонстрацію в Zabbix. Така логіка робить пояснювальну записку цілісною [1].

ВИСНОВКИ

У дипломній роботі розглянуто задачу організації моніторингу локальної мережі за допомогою протоколу SNMP та розроблено програмний комплекс для симуляції SNMP-агентів у контейнеризованому середовищі. Проведений аналіз показав, що SNMP залишається актуальним інструментом мережевого управління завдяки простоті, підтримці багатьма пристроями та можливості інтеграції з системами NMS [17, 18].

У межах роботи досліджено архітектуру SNMP, структуру MIB, механізми polling і Trap/Inform, а також існуючі засоби симуляції агентів. На основі цього сформовано вимоги до програмного комплексу, який повинен бути відтворюваним, розширюваним, придатним для тестування та безпечним для використання без фізичного обладнання [1].

Спроектовано модульну архітектуру системи, що включає SnmpAgent, SimulationEngine, TrapSender, SimulationApi та модель стану пристрою. Розроблено Custom MIB у просторі Enterprise OID 1.3.6.1.4.1.99999, що дозволяє представляти як базові параметри пристроїв, так і спеціалізовані показники сценаріїв кібер інцидентів [1].

Реалізовано контейнеризоване середовище на базі Docker Compose з ізольованою мережею 192.168.100.0/24 та логічними вузлами Router, Switch, Server і IED-контролер. Забезпечено інтеграцію із Zabbix, створення SNMP items, triggers, алертів і оброблення Trap-повідомлень. Це дозволяє демонструвати повний цикл моніторингу від зміни стану агента до появи події у NMS [2, 9].

Проведено тестування програмного комплексу з використанням JUnit 5 і JaCoCo. Автоматизовані тести перевіряють роботу SNMP-агента, рушія сценаріїв, механізму Trap-повідомлень і допоміжних діагностичних компонентів. Інтеграційна перевірка через Zabbix підтвердила працездатність розробленого рішення та його практичну цінність [2, 9].

Подальший розвиток роботи може включати підтримку SNMPv3, розширення бібліотеки сценаріїв, створення веб-інтерфейсу керування, масштабування кількості агентів, імпорт MIB-файлів і використання стенду у процесах автоматизованої перевірки конфігурацій моніторингу. Отже, поставлену мету дипломної роботи досягнуто, а розроблений комплекс може застосовуватися у навчальних, демонстраційних і тестових задачах [6, 7, 8].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Harrington D., Presuhn R., Wijnen B. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks : RFC 3411. IETF, 2002. URL: <https://www.rfc-editor.org/rfc/rfc3411> (дата звернення: 26.05.2026).
2. Presuhn R. Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP) : RFC 3416. IETF, 2002. URL: <https://www.rfc-editor.org/rfc/rfc3416> (дата звернення: 26.05.2026).
3. McCloghrie K., Rose M. Management Information Base for Network Management of TCP/IP-based internets: MIB-II : RFC 1213. IETF, 1991. URL: <https://www.rfc-editor.org/rfc/rfc1213> (дата звернення: 26.05.2026).
4. McCloghrie K., Rose M. Structure and Identification of Management Information for TCP/IP-based internets : RFC 1155. IETF, 1990. URL: <https://www.rfc-editor.org/rfc/rfc1155> (дата звернення: 26.05.2026).
5. Case J., Fedor M., Schoffstall M., Davin J. Simple Network Management Protocol : RFC 1157. IETF, 1990. URL: <https://www.rfc-editor.org/rfc/rfc1157> (дата звернення: 26.05.2026).
6. Blumenthal U., Wijnen B. User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) : RFC 3414. IETF, 2002. URL: <https://www.rfc-editor.org/rfc/rfc3414> (дата звернення: 26.05.2026).
7. Wijnen B., Presuhn R., McCloghrie K. View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) : RFC 3415. IETF, 2002. URL: <https://www.rfc-editor.org/rfc/rfc3415> (дата звернення: 26.05.2026).
8. Levi D., Meyer P., Stewart B. SNMP Applications : RFC 3413. IETF, 2002. URL: <https://www.rfc-editor.org/rfc/rfc3413> (дата звернення: 26.05.2026).
9. Presuhn R. Transport Mappings for the Simple Network Management Protocol (SNMP) : RFC 3417. IETF, 2002. URL: <https://www.rfc-editor.org/rfc/rfc3417> (дата звернення: 26.05.2026).

10. SNMP4J. SNMP4J Documentation. URL: <https://www.snmp4j.org/> (дата звернення: 26.05.2026).
11. Zabbix Documentation. SNMP monitoring and SNMP traps. URL: <https://www.zabbix.com/documentation/current/en/manual/config/items/itemtypes/snmp> (дата звернення: 26.05.2026).
12. Docker Documentation. Docker Compose overview. URL: <https://docs.docker.com/compose/> (дата звернення: 26.05.2026).
13. JUnit 5 User Guide. URL: <https://junit.org/junit5/docs/current/user-guide/> (дата звернення: 26.05.2026).
14. JaCoCo. Java Code Coverage Library. Documentation. URL: <https://www.jacoco.org/jacoco/trunk/doc/> (дата звернення: 26.05.2026).
15. Gambit Communications. MIMIC SNMP Simulator. URL: https://www.gambitcomm.com/site/products/mimic_simulator.shtml (дата звернення: 26.05.2026).
16. MG-SOFT. SNMP Agent Simulator. URL: <https://www.mg-soft.com/> (дата звернення: 26.05.2026).
17. Mauro D., Schmidt K. Essential SNMP. 2nd ed. Sebastopol : O'Reilly Media, 2005. 460 p.
18. Stallings W. SNMP, SNMPv2, SNMPv3, and RMON 1 and 2. 3rd ed. Boston : Addison-Wesley, 1999. 619 p.
19. Comer D. E. Internetworking with TCP/IP. Vol. 1: Principles, Protocols, and Architecture. 6th ed. Boston : Pearson, 2014. 672 p.
20. Tanenbaum A. S., Wetherall D. J. Computer Networks. 5th ed. Boston : Pearson, 2011. 960 p.