Міністерство освіти і науки України

Національний технічний університет
«Дніпровська політехніка»

Навчально-науковий
інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

здобувача Соловйов Олексій Дмитрович
(ПІБ)

академічної групи 123-22ск-1
(шифр)

спеціальності 123 Комп'ютерна інженерія
(код і назва спеціальності)

за освітньо-професійною програмою 123 Комп'ютерна інженерія
(офіційна назва)

на тему " Розподілений програмний додаток для аналізу та моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP"
(назва за наказом ректора)

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	доц. Каштан В.Ю.			
загального розділу	доц. Каштан В.Ю.			
спеціального розділу	доц. Каштан В.Ю.			
Рецензент				
Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро
2025

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)
_____ Гнатушенко В.В.
(підпис) (прізвище, ініціали)

"____" _____ 2025 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня бакалавр

здобувача Соловйов О.Д. **академічної групи** 123-22ск-1
(прізвище та ініціали) (шифр)

спеціальності 123 Комп'ютерна інженерія

за освітньо-професійною програмою Комп'ютерна інженерія
(офіційна назва)

на тему " Розподілений програмний додаток для аналізу та моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP "

затверджену наказом ректора НТУ «Дніпровська політехніка» від 05.05.2025 № 336-с

Розділ	Зміст	Термін виконання
Загальний розділ	На основі матеріалів виробничих практик, інших науково-технічних джерел показати актуальність завдання, сформулювати мету та задачі виконання кваліфікаційної роботи	10.02.2025
Спеціальний розділ	Сформулювати найменування й призначення програмного додатку для аналізу та моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP, висунути технічні вимоги до нього	15.03.2025
	Реалізувати програмний додаток, використовуючи відповідні інструменти програмування та враховуючи принципи розробки надійних та ефективних програмних систем, що є частиною комп'ютерної інженерії	07.05.2025
	Протестувати розроблений програмний додаток для аналізу та моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP на предмет її функціональності, продуктивності обміну даними в реальному часі та стійкості до можливих збоїв	31.05.2025

Завдання видано _____ доц. Каштан В.Ю.
(підпис керівника) (прізвище, ініціали)

Дата видачі 25.01.2025

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ Соловйов О.Д.
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 82 с., 29 рис., 5 табл., 1 додаток, 14 джерел.

Об'єктом дослідження є процеси аналізу та моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP у локальних комп'ютерних мережах.

Предметом дослідження є методи та програмні засоби для розробки розподіленого програмного додатку, що забезпечує ефективний збір, обробку та візуалізацію даних, отриманих за допомогою протоколів ICMP, Traceroute та IPv6 NDP, з метою покращення розуміння стану мережі, виявлення проблем та оптимізації її функціонування.

Метою даної кваліфікаційної роботи є розробка розподіленого програмного додатку для ефективного аналізу та моніторингу ключових мережевих протоколів – ICMP, Traceroute та IPv6 NDP – в локальних комп'ютерних мережах. Розроблений додаток повинен забезпечити збір, обробку та візуалізацію даних, отриманих за допомогою цих протоколів, у розподіленому середовищі, що сприятиме покращенню розуміння стану мережі, швидкому виявленню проблем та оптимізації її функціонування, зокрема в умовах зростання використання протоколу IPv6.

Завдання:

1. Провести аналіз існуючих підходів та засобів моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP.

2. Дослідити особливості структури та функціонування протоколів ICMP (у тому числі з фрагментацією), Traceroute та IPv6 Neighbor Discovery Protocol.

3. Розробити архітектуру розподіленого програмного додатка для збору, обробки та аналізу відповідного мережевого трафіку.

4. Реалізувати програмний засіб з підтримкою: виявлення та класифікації ICMP-пакетів за типами та кодами; аналізу маршрутів передачі пакетів за допомогою Traceroute; виявлення IPv6 NDP-повідомлень та візуалізації зібраної статистики в зручному інтерфейсі.

5. Провести тестування та оцінювання ефективності роботи програмного додатка на основі реального або згенерованого мережевого трафіку.

Ключові слова: мережевий моніторинг, ICMP, Traceroute, IPv6 NDP.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ ТА ТЕРМІНІВ

PCAP	–	Packet Capture;
DPI	–	Deep Packet Inspection;
SNMP	–	Simple Network Management Protocol;
APM	–	Application Performance Monitoring;
ПАМ	–	підсистема агентів моніторингу (;
ПЦУВ	–	підсистема центрального управління та візуалізації;
СКБД	–	системи керування базами даних;
ICMP	–	Internet Control Message Protocol;
NDP	–	Neighbor Discovery Protocol.

ВСТУП

У сучасних умовах стрімкого зростання кількості мережевих пристроїв і складності інфраструктурних рішень питання ефективного управління комп'ютерними мережами набуває особливої актуальності. Управління мережею – це процес адміністрування компонентів мережевої інфраструктури з метою забезпечення стабільної, безпечної та продуктивної роботи інформаційних систем. Воно включає моніторинг, оптимізацію, виявлення збоїв та їх усунення, а також адаптацію до змін у середовищі функціонування.

Розвиток великих і динамічних мереж унеможливлює централізоване управління, що призводить до потреби у розподілених рішеннях. Такі підходи дозволяють знизити навантаження на окремі вузли, підвищити стійкість до відмов та покращити адаптивність системи в цілому. Проте із зростанням масштабів мереж зростає ймовірність виникнення зловмисних дій, наявності шкідливих вузлів або критичних збоїв. У відповідь на ці виклики з'явилася концепція автономних мереж, що поєднують самоналаштування, самовідновлення, самооптимізацію та самозахист. У 2001 році компанія IBM запропонувала автономні обчислювальні системи як аналог вегетативної нервової системи людини — системи, здатної адаптуватися до змін без зовнішнього втручання.

Сучасні підходи до побудови автономних мереж, зокрема у рамках ініціативи Autonomic Networking Integrated Model and Approach) запропонованої IETF/IRTF, передбачають розробку стандартів і технологій для самокерованих мережевих доменів. Одним із ключових елементів таких систем є виявлення топології, що забезпечує збір інформації про структуру мережі, сусідні вузли та їхні взаємозв'язки. У цьому контексті надзвичайно важливим є ефективний аналіз мережевих протоколів нижчих рівнів, таких як ICMP (Internet Control Message Protocol), Traceroute, що використовується для трасування маршрутів, та IPv6 NDP (Neighbor Discovery Protocol). Ці протоколи є джерелом критично важливої інформації для аналізу топології, моніторингу з'єднань, виявлення несправностей та діагностики мережі.

У даній роботі запропоновано розподілений програмний додаток для аналізу

та моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP з метою забезпечення ефективного виявлення проблем мережевої взаємодії, аналізу маршрутів передавання даних.

Для реалізації мети потрібно вирішити завдання:

– провести аналіз існуючих підходів і засобів моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP;

– дослідити особливості структури та функціонування протоколів ICMP (у тому числі з фрагментацією), Traceroute та IPv6 Neighbor Discovery Protocol;

– розробити архітектуру розподіленого програмного додатка для збору, обробки та аналізу відповідного мережевого трафіку;

– реалізувати програмний засіб з підтримкою:виявлення та класифікації ICMP-пакетів за типами та кодами; аналізу маршрутів передачі пакетів за допомогою Traceroute; виявлення IPv6 NDP-повідомлень та візуалізації зібраної статистики в зручному інтерфейсі;

– провести тестування та оцінювання ефективності роботи програмного додатка на основі реального або згенерованого мережевого трафіку.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Актуальність вирішуваної задачі

Збільшення кількості мережевих пристроїв, впровадження нових технологій та зростання обсягів переданих даних зумовлюють підвищені вимоги до їхньої продуктивності, надійності та керованості. В умовах такої складності ефективний моніторинг та своєчасна діагностика проблем стають критично важливими для забезпечення безперебійної роботи мережевої інфраструктури.

Протоколи ICMP, Traceroute та IPv6 NDP відіграють ключову роль у забезпеченні функціонування мереж. ICMP є основним інструментом для діагностики мережевих з'єднань та виявлення помилок передачі даних. Traceroute дозволяє візуалізувати маршрути проходження трафіку, що є незамінним при локалізації проблем на певних ділянках мережі та аналізі ефективності маршрутизації. Зі зростанням популярності протоколу IPv6, моніторинг та аналіз NDP стає особливо актуальним, оскільки цей протокол відповідає за виявлення сусідніх вузлів, автоматичну конфігурацію адрес та інші важливі функції в мережах нового покоління.

Однак, традиційні підходи до моніторингу та аналізу цих протоколів часто є централізованими та можуть бути неефективними у великих та розподілених мережах. Збір та обробка даних з великої кількості мережевих пристроїв на одному центральному сервері може призводити до перевантаження, затримок та обмеженої масштабованості. У цьому контексті розробка розподіленого програмного забезпечення для аналізу та моніторингу ICMP, Traceroute та IPv6 NDP є актуальною та своєчасною задачею.

Розподілена архітектура пропонованого рішення дозволить збирати та аналізувати дані про стан мережі з різних її сегментів одночасно, забезпечуючи більш повну, оперативну та детальну картину її функціонування. Це, в свою чергу, сприятиме швидшому виявленню та локалізації несправностей, оптимізації маршрутизації трафіку та підвищенню загальної надійності мережі. Крім того, розроблений додаток може стати важливим елементом у побудові інтелектуальних

та самокерованих мереж, надаючи необхідні дані для систем автоматичного прийняття рішень щодо оптимізації та відновлення мережевої інфраструктури в рамках концепції автономних мереж. Таким чином, вирішення задачі розробки розподіленого програмного забезпечення для аналізу ключових мережевих протоколів є важливим кроком на шляху до створення більш ефективних, надійних та керованих комп'ютерних мереж майбутнього [1].

1.2 Принципи та типи моніторингу комп'ютерних мереж

Моніторинг комп'ютерних мереж є невід'ємною складовою ефективного управління та підтримки їхньої працездатності. Він передбачає систематичне спостереження за різними аспектами функціонування мережі з метою виявлення проблем, аналізу продуктивності, забезпечення безпеки та планування подальшого розвитку. Існують два основні підходи до здійснення мережевого моніторингу: активний та пасивний моніторинг, кожен з яких має свої особливості, переваги та недоліки (рис.1.1) [2].

Рисунок 1.1 – Загальна архітектура мережевого моніторингу

Активний моніторинг передбачає ініціювання спеціальних запитів або введення так званих "зондів" у мережу з єдиною метою – отримати інформацію про її стан та продуктивність. Ці зонди можуть мати форму спеціально сформованих пакетів даних, що надсилаються до цільових пристроїв або сегментів мережі. Аналізуючи відповіді на ці запити (або їхню відсутність), система моніторингу може визначати такі параметри, як час затримки, втрата пакетів, доступність сервісів тощо.

Однією з основних переваг активного моніторингу є повний контроль над процесом вимірювання. Система моніторингу може самостійно визначати інтервал надсилання зондів, розмір пакетів та маршрути їхнього проходження. Це дозволяє цілеспрямовано досліджувати конкретні аспекти мережевої інфраструктури та отримувати дані, які не залежать від інтенсивності поточного мережевого трафіку. Вартість активного моніторингу полягає у внесенні додаткового трафіку в мережу. Однак, у більшості випадків, розмір цих службових пакетів є відносно невеликим порівняно із загальною пропускною здатністю мережі, тому вплив цього додаткового трафіку часто є мінімальним. Зменшення інтенсивності зондування може ще більше знизити накладні витрати, проте це може призвести до зниження детальності та оперативності отримуваних даних. Важливо також зазначити, що дані, отримані в результаті активного моніторингу, не є специфічними для будь-якої конкретної прикладної програми, а відображають загальний стан мережевої інфраструктури.

До недоліків належить [3]:

– додаткове навантаження на мережу;
– обмеження в застосуванні в умовах обмеженої пропускної здатності;
– потенційна фрагментація віртуальних каналів.

Пасивний моніторинг, на відміну від активного, полягає у спостереженні за існуючим мережевим трафіком та зборі інформації з нього без активного втручання у вигляді додаткових запитів. У цьому підході система моніторингу аналізує пакети даних, які передаються в мережі в рамках роботи різних прикладних програм. Для збору цих даних використовуються спеціальні програми-сніфери, які можуть бути розгорнуті на окремих хостах (джерелі або призначенні трафіку) або на проміжних мережевих пристроях (наприклад, за допомогою механізму port mirroring на комутаторах або TAP-пристроїв).

Основною перевагою пасивного моніторингу є відсутність накладних витрат, пов'язаних з генерацією додаткового трафіку. Крім того, він дозволяє аналізувати реальний трафік, що генерується користувачами та програмами, і отримувати дані про фактичну продуктивність мережі в умовах реального навантаження. Однак

пасивний моніторинг має і свої суттєві обмеження. Система моніторингу має обмежений або взагалі не має контролю над інтервалом спостереження, розміром пакетів або маршрутами їхнього проходження. Інформація для аналізу залежить від наявності відповідного трафіку в мережі. Зі збільшенням пропускної здатності мережевих каналів стає складніше і дорожче ідентифікувати та вимірювати пакети конкретного потоку. Крім того, пасивний моніторинг може викликати питання конфіденційності, оскільки передбачає перехоплення та аналіз даних користувачів.

Вибір між активним та пасивним моніторингом залежить від конкретних завдань та вимог. Наприклад, для оцінки продуктивності існуючих застосунків або для систем виявлення вторгнень пасивний підхід може бути більш прийнятним, оскільки він не впливає на роботу цих систем та дозволяє аналізувати реальні загрози. У той же час, для діагностики мережевої інфраструктури або для застосунків, що використовують механізми підвищення продуктивності, такі як балансування навантаження, активний моніторинг може надати більш контрольовані та передбачувані результати. Нерідко на практиці використовується комбінація обох підходів для досягнення більш повної та ефективної картини стану мережі. Наприклад, система моніторингу може використовувати пасивний режим для спостереження за загальним трафіком, а активний – для періодичної перевірки доступності критично важливих сервісів [3.

Для здійснення пасивного моніторингу мережевого трафіку необхідно мати можливість отримати копію даних, що передаються мережею, без активного втручання в процес комунікації. Існує кілька основних технік дублювання трафіку, серед яких найбільш поширеними є використання вбудованих можливостей мережевих пристроїв (дзеркалювання портів) та застосування спеціалізованих апаратних засобів (TAP-пристроїв).

Дзеркалювання портів (Port Mirroring) є вбудованою функцією багатьох сучасних комутаторів та маршрутизаторів, яка дозволяє скопіювати трафік, що проходить через один або кілька вихідних портів, та направити цю копію на інший порт, призначений для підключення аналізатора трафіку або системи моніторингу (рис.1.2). Ця функція може бути налаштована для копіювання вхідного, вихідного

або обох напрямків трафіку на вказаних портах [4].

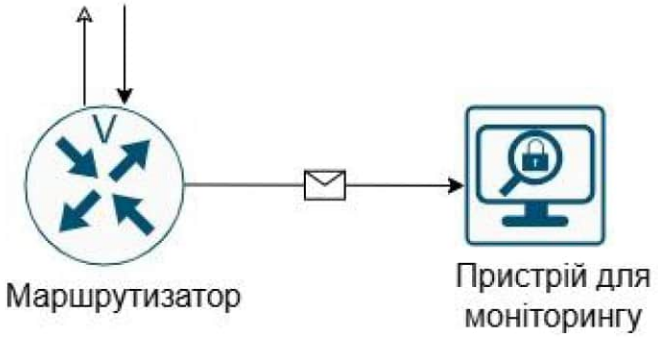

Рисунок 1.2 – Структурна схема реалізації Port Mirroring

Незважаючи на зручність та доступність, дзеркалювання портів має певні обмеження. Одним з основних недоліків є можливе перевантаження дзеркального порту. Якщо сумарна пропускна здатність трафіку, що копіюється з вихідних портів, перевищує пропускну здатність дзеркального порту, це може призвести до втрати частини пакетів. Особливо це актуально для повнодуплексного трафіку, де дані передаються одночасно в обох напрямках. При копіюванні трафіку з двох повнодуплексних портів на один дзеркальний порт, навантаження на останній може майже вдвічі перевищити його номінальну пропускну здатність. При копіюванні трафіку з більшої кількості портів ризик перевантаження зростає ще більше.

Іншим потенційним недоліком є те, що деякі комутатори можуть мати обмежену обчислювальну потужність, необхідну для одночасного виконання основних функцій комутації та дублювання трафіку. В умовах високого навантаження на мережу це може призвести до зниження продуктивності комутатора або некоректного виконання операції дзеркалювання.

TAP (Test Access Port) є спеціалізованим апаратним пристроєм, який встановлюється безпосередньо на фізичному з'єднанні між двома мережевими пристроями. TAP дозволяє отримати копію всього вхідного та вихідного трафіку з мережевого інтерфейсу без створення додаткового навантаження або затримок у мережі, що проходить через це з'єднання, без будь-якого впливу на основний потік даних (рис.1.3). TAP зазвичай має два порти для підключення до мережевого

сегменту, що досліджується, та один або кілька вихідних портів для підключення інструментів моніторингу [4].

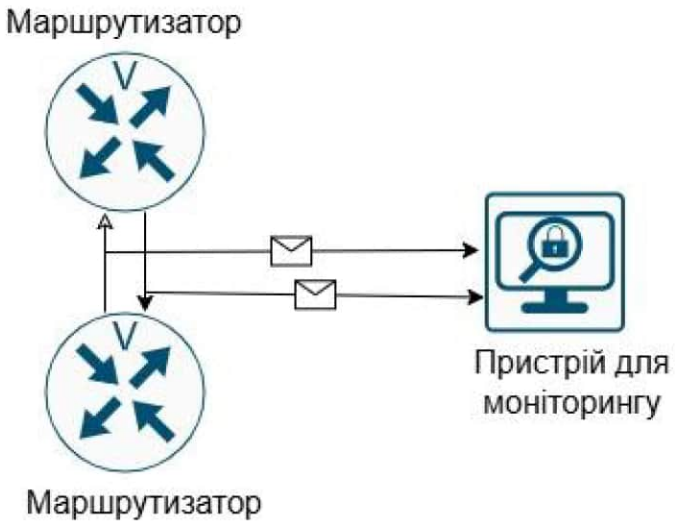

Рисунок 1.3 – Структурна схема реалізації Test Access Port

Основною перевагою TAP-пристроїв є їхня здатність забезпечувати повне дублювання трафіку на фізичному рівні без ризику втрати пакетів, навіть при високих навантаженнях та повнодуплексному обміні даними. TAP-пристрої є пасивними, вони не потребують налаштування на мережевих пристроях та не споживають їхні обчислювальні ресурси. Існують різні типи TAP-пристроїв, що підтримують різні типи фізичних інтерфейсів (Ethernet, Fiber Optic тощо) та швидкості передачі даних.

TAP-подібне налаштування з використанням обхідних мережевих карт (NICs) є програмним підходом до дублювання трафіку, який використовує можливості операційної системи та двох або більше мережевих інтерфейсів на сервері, що виконує функцію моніторингу. В цьому сценарії один мережевий інтерфейс використовується для отримання трафіку, а інший – для його аналізу або перенаправлення. Таке налаштування може бути реалізовано за допомогою спеціалізованого програмного забезпечення для обробки та дублювання мережевих пакетів.

Перевагою цього підходу є його гнучкість та потенційно нижча вартість порівняно з використанням апаратних TAP-пристроїв. Однак ефективність та надійність такого рішення можуть залежати від продуктивності сервера, можливостей операційної системи та якості програмного забезпечення. Також можуть виникати затримки при обробці та дублюванні трафіку, особливо при високих навантаженнях.

Вибір конкретної техніки дублювання трафіку залежить від багатьох факторів, включаючи вимоги до точності моніторингу, пропускної здатності мережевих з'єднань, наявного обладнання та бюджету проекту. У багатьох випадках оптимальним рішенням може бути комбінація різних підходів для забезпечення найбільш ефективного та надійного пасивного моніторингу мережі.

1.3 Перехоплення та методи глибокого аналізу пакетів

Процес моніторингу мережі часто включає етап перехоплення (захоплення) мережевого трафіку з подальшим його аналізом. Ці два етапи можуть бути розділені в часі та просторі, коли захоплені дані зберігаються для подальшого дослідження, або ж інтегровані в єдиний конвеєр обробки в реальному часі. Захоплені пакети часто зберігаються у форматі PCAP (Packet Capture), який є стандартним для багатьох інструментів мережевого аналізу. Ці файли PCAP можуть слугувати джерелом даних для глибокого аналізу пакетів (Deep Packet Inspection, DPI) [4].

Глибокий аналіз пакетів (DPI) є методом інспектування вмісту мережевих пакетів на рівнях, що перевищують заголовки IP та портів (тобто на рівнях L7 моделі OSI та вище). DPI дозволяє аналізувати протоколи прикладного рівня, виявляти специфічні патерни у даних, розпізнавати типи трафіку, виявляти шкідливу активність або порушення політик безпеки. Існує два основних типи аналізу на основі DPI: зіставлення патернів (Pattern Matching) та аналіз подій (Event Analysis) (рис.1.4).

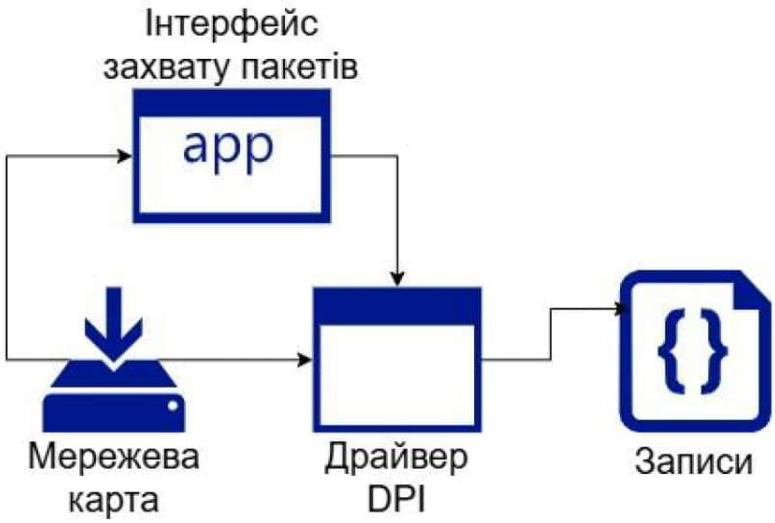

Рисунок 1.4 – Структурна схема захвату пакетів на основі DPI

Ідентифікація характерних структур (Pattern Matching) є одним з найбільш поширених методів DPI, який передбачає пошук у повному вмісті мережевих пакетів заздалегідь визначених послідовностей байтів або регулярних виразів (рис.1.5). Принцип роботи цього методу полягає у порівнянні вмісту пакетів з набором відомих сигнатур або шаблонів, які можуть вказувати на певну активність, наприклад, відомі атаки, використання певних протоколів або передачу конфіденційних даних. Пошук може бути обмежений конкретними частинами пакетів або застосовуватися лише до певних типів пакетів, що підвищує ефективність аналізу [3].

Перевагою ідентифікації характерних структур є його відносна простота реалізації та розуміння. Опис шаблонів для пошуку за допомогою послідовностей байтів або регулярних виразів часто є інтуїтивно зрозумілим. Існують численні алгоритми для ефективної ідентифікації характерних структур, а також готові програмні бібліотеки та інструменти, такі як Flex та MultiFast, які можуть бути інтегровані в системи моніторингу. Популярні системи виявлення та запобігання вторгненням (IDS/IPS), такі як Snort та Suricata, також значною мірою використовують зіставлення патернів. Утиліта командного рядка Ngrep є прикладом інструменту, що дозволяє здійснювати зіставлення патернів безпосередньо у файлах PCAP.

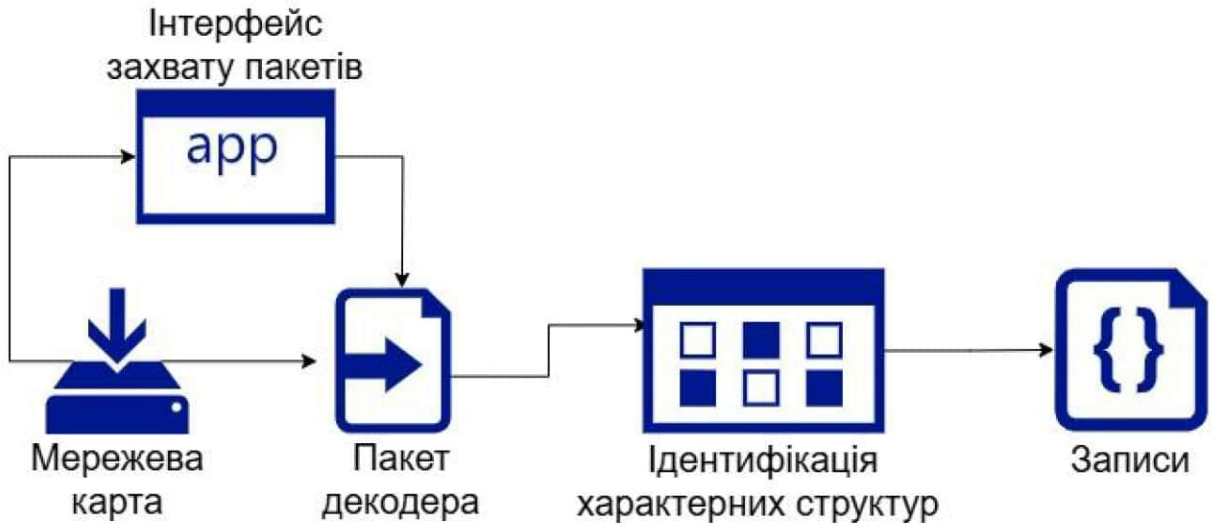

Рисунок 1.5 – Принцип роботи методу ідентифікації характерних структур

Однак, ідентифікація характерних структур має і свої обмеження. Ефективність цього підходу може знижуватися при необхідності пошуку складних шаблонів, які важко або неможливо описати за допомогою регулярних виразів. Наприклад, якщо дані в пакеті попередньо стиснуті або зашифровані, для успішного зіставлення патерну може знадобитися їхнє декодування. Якщо система моніторингу не має вбудованих механізмів для такого декодування, створення відповідного регулярного виразу може бути неможливим [5].

Крім того, ідентифікація характерних структур погано підходить для реалізації складної логіки прийняття рішень, яка вимагає врахування контексту або стану мережевих з'єднань. Наприклад, виявлення прострочених SSL-сертифікатів для HTTPS-з'єднань з певного списку IP-адрес є завданням, яке складно вирішити лише за допомогою регулярних виразів. Навіть якщо вдасться виявити дані сертифіката, перевірка його наявності у динамічно змінюваному списку або реалізація порогових правил (наприклад, сповіщення про хости, що генерують більше 10 помилок DNS на годину) стає практично неможливою за допомогою стандартних механізмів зіставлення патернів. Сучасні системи моніторингу, що використовують зіставлення патернів, часто включають в себе попереднє декодування найбільш поширених протоколів для підвищення ефективності аналізу.

Ще одним недоліком ідентифікації характерних структур є його відносна

швидкість обробки порівняно з методами, що базуються на аналізі мережевих потоків (flow-based analysis). Для досягнення високої пропускної здатності аналізу (наприклад, на швидкостях 10 Гбіт/с і вище) часто потрібне апаратне прискорення з використанням FPGA (Field-Programmable Gate Array). На противагу цьому, системи, що використовують аналіз потоків без апаратного прискорення, можуть обробляти трафік на значно вищих швидкостях (наприклад, 40 Гбіт/с). У порівнянні з підходом аналізу подій, зіставлення патернів також є більш спрощеним методом, який менш орієнтований на контекстний аналіз мережевої активності [6].

Аналіз подій (Event Analysis) є більш складним та контекстно-орієнтованим підходом до DPI порівняно зі зіставленням патернів (рис.1.6). Замість простого пошуку статичних сигнатур, аналіз подій передбачає відстеження послідовностей мережевих подій та встановлення кореляцій між ними для виявлення складних або аномальних поведінок. Цей метод часто включає в себе розбір протоколів прикладного рівня, відстеження стану з'єднань та аналіз послідовності операцій.

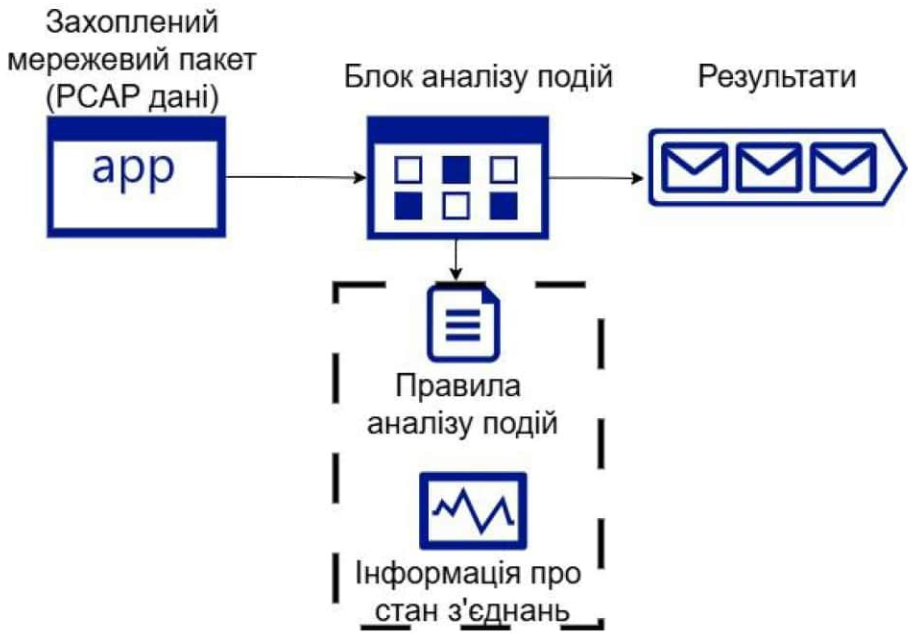

Рисунок 1.6 – Принцип роботи методу аналізу подій

Основна ідея аналізу подій полягає в тому, що окремі мережеві пакети самі по собі можуть не нести шкідливого навантаження або не вказувати на атаку, але певна

послідовність дій або нетипова взаємодія між вузлами може свідчити про спробу вторгнення, розповсюдження шкідливого програмного забезпечення або інші небажані активності.

Перевагою аналізу подій є його здатність виявляти складні та багатоетапні атаки, які можуть не бути помічені при простому зіставленні патернів. Оскільки цей метод враховує контекст мережевої взаємодії та стан з'єднань, він може бути більш ефективним у виявленні нових або модифікованих загроз, для яких ще не існують відомі сигнатури. Аналіз подій також краще підходить для реалізації складної логіки прийняття рішень, такої як виявлення аномальної кількості невдалих спроб авторизації, нетипових послідовностей команд у протоколах прикладного рівня або порушень встановлених шаблонів поведінки.

Для реалізації аналізу подій часто використовуються складніші алгоритми та структури даних, такі як скінченні автомати, машини станів та графові моделі. Системи, що використовують аналіз подій, зазвичай включають в себе механізми розбору протоколів, відстеження сесій та кореляції подій, що дозволяє їм розуміти контекст мережевої взаємодії на більш глибокому рівні.

Однак, складність аналізу подій також є його недоліком. Розробка та налаштування правил для виявлення складних послідовностей подій може бути значно складнішим завданням, ніж створення простих сигнатур для зіставлення патернів. Крім того, аналіз подій може вимагати значних обчислювальних ресурсів, особливо при обробці великих обсягів трафіку в реальному часі, оскільки необхідно відстежувати стан багатьох мережевих з'єднань та аналізувати послідовності пакетів [7].

1.4 Огляд програмних засобів моніторингу мережевої інфраструктури

Системи моніторингу LAN можна класифікувати за різними критеріями, включаючи їхню архітектуру, методи збору даних та основні функціональні можливості.

Системи на основі SNMP (Simple Network Management Protocol) – це один з найбільш поширених протоколів для управління та моніторингу мережевих пристроїв. Агенти SNMP, встановлені на керованих пристроях (маршрутизатори, комутатори, сервери, принтери тощо), надають інформацію про їхній стан, трафік, завантаження процесора, використання пам'яті та інші параметри у відповідь на запити від центральної системи моніторингу (SNMP-менеджера).

Системи аналізу мережевого трафіку – ці системи зосереджені на перехопленні та аналізі мережевого трафіку для отримання детальної інформації про комунікації між вузлами, використовувані протоколи, обсяги переданих даних, затримки та помилки. Вони можуть використовувати як пасивні (дзеркалювання портів, TAP-пристрої), так і активні (зондування) методи збору даних.

Системи моніторингу журналів (Log Management Systems) – ці системи збирають, централізують та аналізують журнали подій з різних пристроїв та операційних систем у локальній мережі. Аналіз журналів може допомогти у виявленні проблем з безпекою, діагностиці несправностей та аудиті дій користувачів [8].

Системи моніторингу продуктивності додатків (Application Performance Monitoring, APM) – моніторинг продуктивності програмного забезпечення, APM-системи також можуть надавати цінну інформацію про вплив мережі на роботу додатків, виявляти мережеві затримки, що впливають на користувацький досвід.

Інтегровані платформи моніторингу – ці системи поєднують у собі функціональність кількох вищезазначених типів, надаючи комплексну картину стану локальної мережі та її компонентів. Вони можуть включати можливості SNMP-моніторингу, аналізу трафіку, управління журналами, моніторингу продуктивності та візуалізації даних.

На ринку представлений широкий спектр як комерційних, так і open-source рішень для моніторингу локальних мереж [9].

Wireshark – це один із найвідоміших аналізаторів мережевого трафіку з відкритим вихідним кодом (рис.1.7). Він реалізує пасивний моніторинг шляхом захоплення пакетів із мережевого інтерфейсу, дозволяючи проводити глибокий

аналіз до рівня окремих байтів.

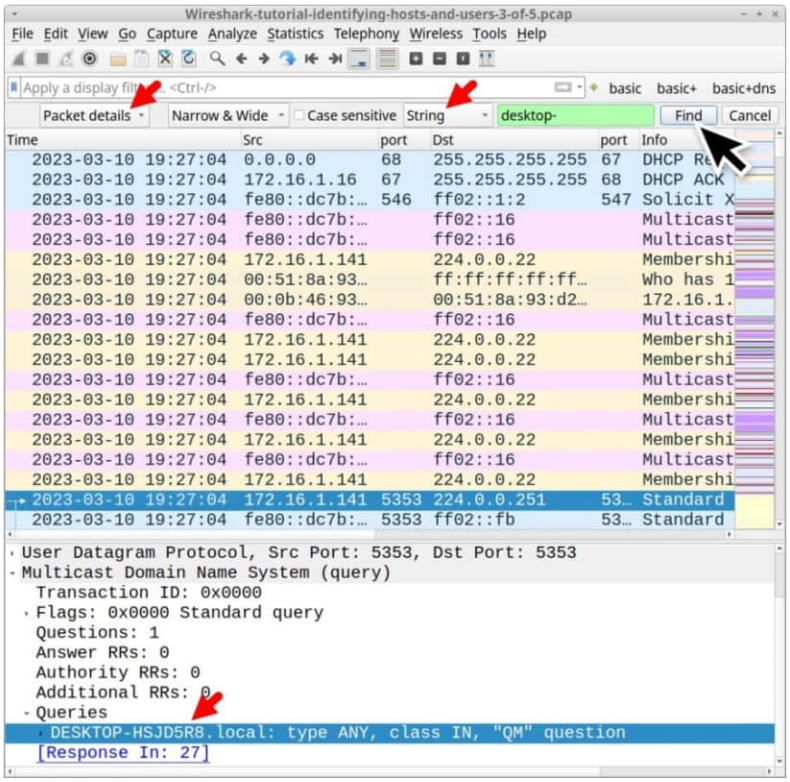

Рисунок 1.7 – Wireshark

PRTG – це комерційне рішення від компанії Paessler для комплексного моніторингу локальної та глобальної мережі (рис.1.8). Підтримує як активні, так і пасивні методи моніторингу.

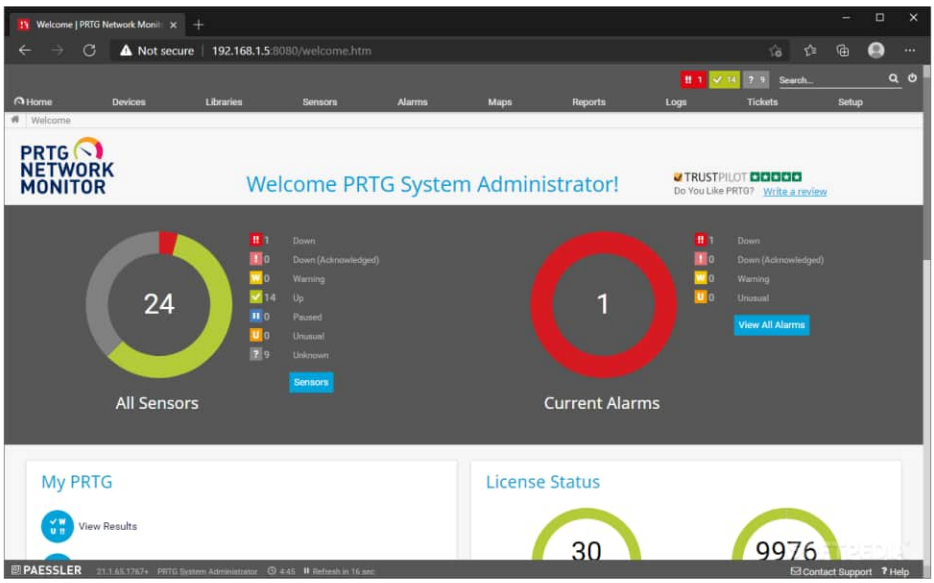

Рисунок 1.8 – PRTG

Zabbix – це безкоштовна система моніторингу з відкритим вихідним кодом, яка підтримує розширене налаштування і підходить для масштабованих інфраструктур (рис.1.9).

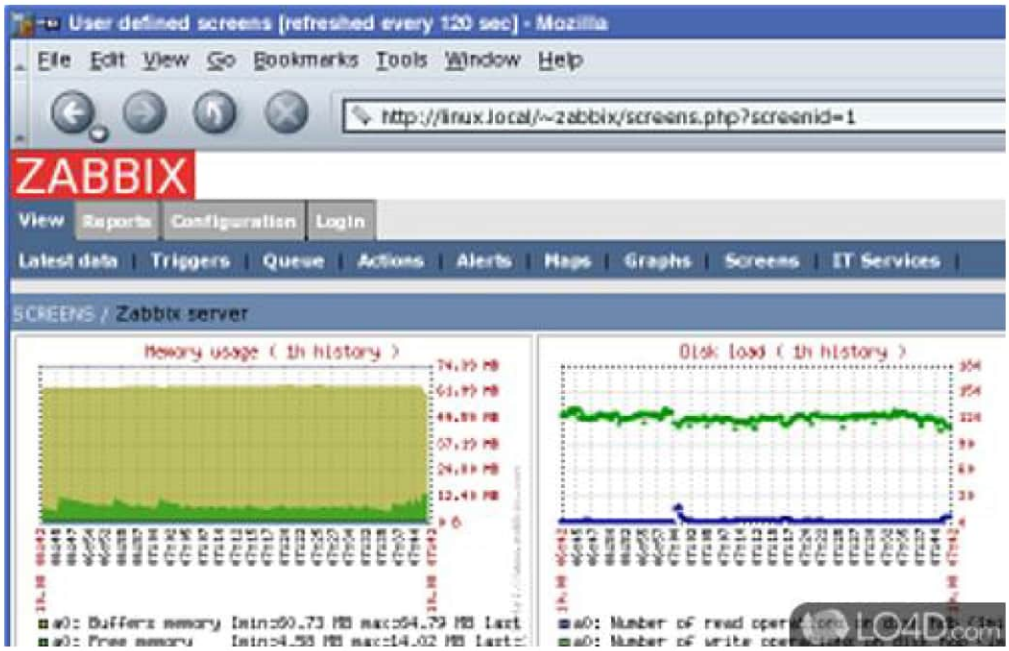

Рисунок 1.9 – PRTG

Nagios – потужне рішення для моніторингу серверів, служб, додатків і мереж. Підтримує як LAN, так і WAN середовища (рис.1.10).

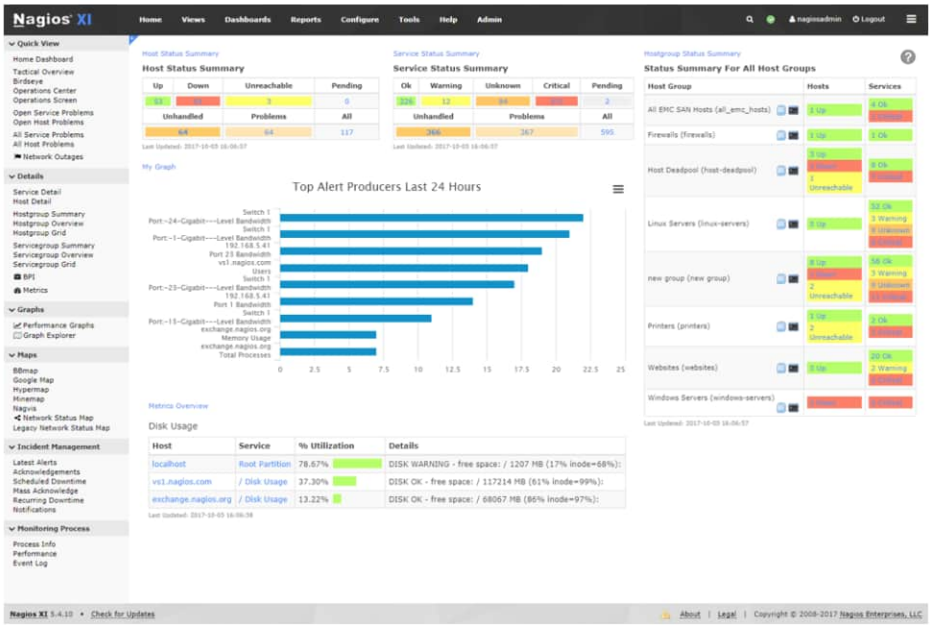

Рисунок 1.10 – Nagios

Порівняння поширених систем для моніторингу мереж наведено в табл. 1.1.

Таблиця 1.1 – Порівняння систем моніторингу локальної мережі

Система	Тип ліцензії	Підтримка ICMP/Traceroute /NDP	Активний/Пасивний моніторинг	Гнучкість	Інтерфейс	Масштабованість
Wireshark	Відкритий	✓/ ✓/ ✗	Пасивний	Висока	GUI	Обмежена
PRTG	Пропрієтарний	✓/ ✓/ ✗	Обидва	Висока	Web/Mobile	Середня
Zabbix	Відкритий	✓/ ✓/ ✗	Обидва	Висока	Web	Висока
Nagios	Відкритий	✓/ ✓/ ✗	Обидва	Середня	Web	Висока
Icinga	Відкритий	✓/ ✓/ ✗	Обидва	Висока	Web/API	Висока

Кожна з розглянутих систем має свої переваги й доцільність застосування в різних умовах. Для глибокого аналізу конкретних пакетів та протоколів (зокрема ICMP, Traceroute, NDP) найкраще підходить Wireshark. Для безперервного моніторингу великих інфраструктур — Zabbix, Nagios або Icinga. Комерційні рішення на кшталт PRTG вигідні для швидкого впровадження та зручності користування, проте поступаються у гнучкості розширення.

1.5 Обґрунтування напрямку вирішення задачі для аналізу та моніторингу мережевих протоколів ICMP, Traceroute та IPv6

У зв’язку з ускладненням архітектури комп’ютерних мереж, появою нових мережевих протоколів та збільшенням обсягів передаваних даних зростає потреба у побудові ефективних, масштабованих і гнучких рішень для аналізу та моніторингу мережевого трафіку. Зокрема, актуальним є створення інструментів, здатних здійснювати аналіз протоколів ICMP, Traceroute та NDP у розподіленому середовищі з можливістю реального часу отримувати аналітику про стан мережі, виявляти затримки, втрати пакетів, недоступні вузли та аномальні дії [10].

Запропонований напрям вирішення задачі полягає у розробці розподіленого програмного додатку, який буде побудований за принципами мікросервісної архітектури з можливістю масштабування, гнучкого розгортання та інтеграції з іншими інструментами мережевого аналізу. Такий підхід дозволяє ефективно

використовувати ресурси системи, розподіляючи навантаження між вузлами, та забезпечує високу надійність та доступність сервісу в цілому.

Основна ідея полягає у створенні окремих модулів, відповідальних за:

– збір даних з мережі за допомогою ICMP (ping), Traceroute та NDP-запитів;

– обробку результатів аналізу (затримки, втрати, топологія маршруту тощо);

– візуалізацію зібраної інформації в зручному веб-інтерфейсі;

– обмін повідомленнями між модулями через легкий протокол (наприклад, gRPC або REST API);

– зберігання журналів та статистики в централізованій базі даних.

Рішення дозволяє легко інтегруватися у локальні або корпоративні мережі для їх моніторингу з урахуванням як IPv4, так і IPv6-протоколів.

Розподілений програмний додаток має модульну архітектуру, яка представлена на рис. 1.11 та складається з кількох ключових компонентів, які взаємодіють між собою для виконання завдань моніторингу та аналізу мережі. Основними структурними елементами є агенти моніторингу, центральний модуль управління та візуалізації, а також підсистема конфігурації.

Агенти моніторингу є легко розгортаються програмними модулями, які встановлюються на окремих вузлах локальної мережі, що підлягають моніторингу. Кожен агент відповідає за збір даних, пов'язаних з визначеними мережевими протоколами, у своєму локальному сегменті. Основні функції агентів включають [10,11]:

– активне надсилання ICMP Echo Request (пінг) до цільових вузлів та аналіз отриманих ICMP Echo Reply для визначення доступності, часу затримки (round-trip time) та втрати пакетів;

– ініціювання серії запитів Traceroute для визначення маршруту проходження пакетів до заданих цілей, фіксація проміжних вузлів (маршрутизаторів) та часу затримки на кожному "стрибку";

– пасивне прослуховування та аналіз повідомлень Neighbor Discovery Protocol (Neighbor Solicitation, Neighbor Advertisement, Router Solicitation, Router Advertisement) для отримання інформації про сусідні IPv6 вузли, їхні MAC-адреси, префікси та стан;

– передача зібраних даних та результатів аналізу до центрального модуля управління та візуалізації. Для передачі даних може використовуватись стандартизований протокол (наприклад, HTTP, gRPC) або власний протокол обміну.

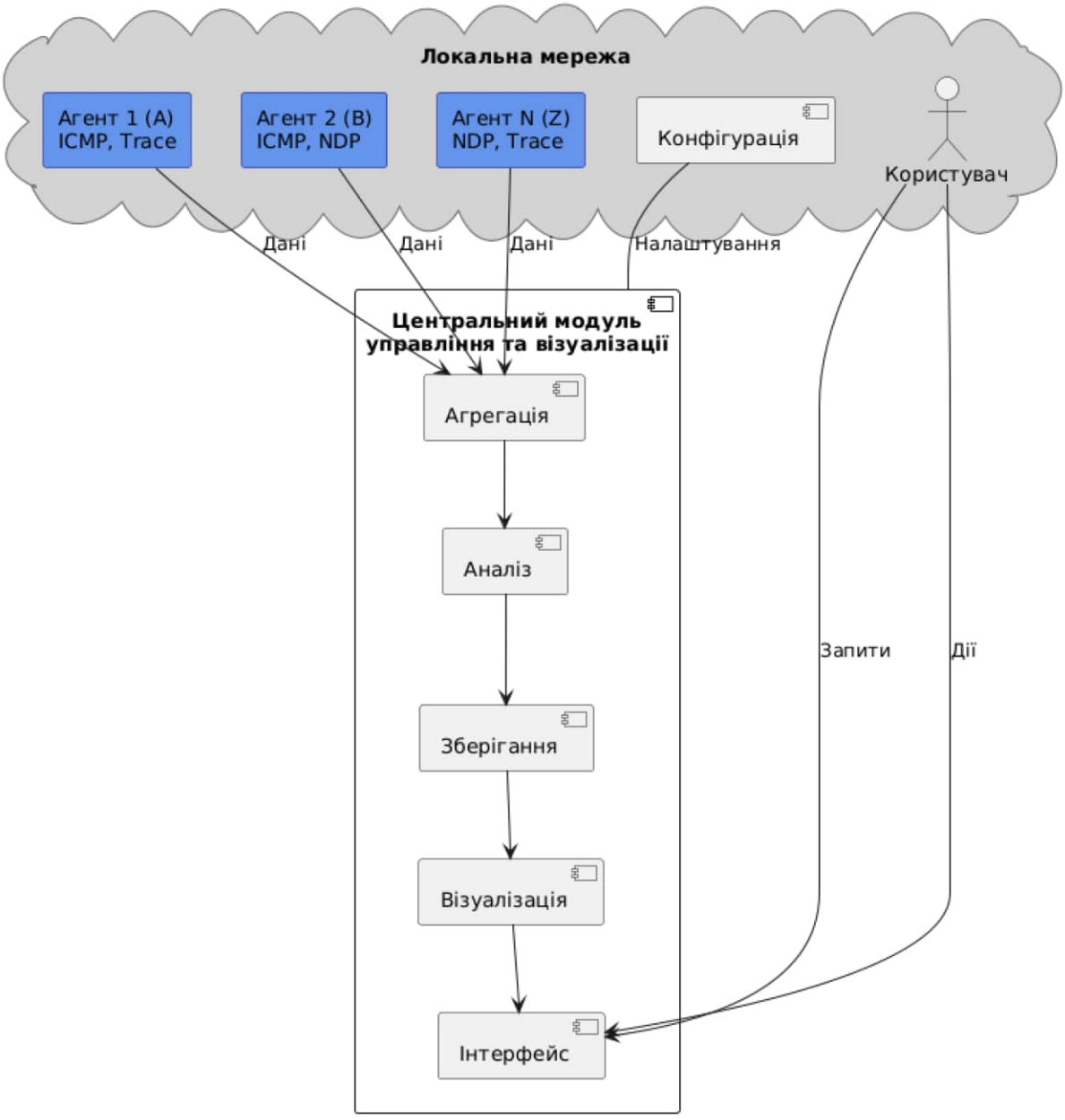

Рисунок 1.11 – Функціональна структура розподіленого програмного додатку

Центральний модуль є основним компонентом системи, який відповідає за агрегацію, аналіз, зберігання та візуалізацію даних, отриманих від агентів моніторингу. Він надає користувацький інтерфейс для налаштування моніторингу та перегляду результатів. Основні функції центрального модуля включають [12]:

– збір та об'єднання даних, отриманих від усіх активних агентів моніторингу;

– обробка отриманих даних для виявлення тенденцій, аномалій, перевищення заданих порогів та генерування сповіщень. Аналіз може включати статистичні розрахунки, порівняння з історичними даними тощо;

– персистентне зберігання зібраних даних та результатів аналізу у базі даних для подальшого використання, формування звітів та аналізу історичних тенденцій;

– представлення зібраної та проаналізованої інформації у зручному для користувача форматі, такому як графіки, діаграми, таблиці та карти мережі. Візуалізація допомагає швидко оцінити стан мережі та виявити проблемні ділянки;

– надання графічного (веб-інтерфейс) або командного інтерфейсу для взаємодії з системою. Користувач може використовувати інтерфейс для налаштування параметрів моніторингу (цільові вузли, інтервали опитування), перегляду поточних та історичних даних, налаштування сповіщень та керування агентами.

Підсистема конфігурації забезпечує механізми для налаштування параметрів роботи як агентів моніторингу, так і центрального модуля. Це може включати визначення цільових вузлів для моніторингу, інтервалів опитування, порогів для генерування сповіщень, параметрів підключення до бази даних та інших налаштувань системи. Конфігурація може здійснюватися через користувацький інтерфейс центрального модуля або за допомогою файлів конфігурації.

Агенти моніторингу періодично або за запитом виконують збір даних та надсилають їх до центрального модуля. Центральний модуль приймає ці дані, обробляє їх, зберігає та відображає через користувацький інтерфейс. Користувач взаємодіє з центральним модулем для налаштування системи та перегляду

результатів моніторингу.

UML-діаграма варіантів поведінки об'єкту вивчення, яким є "Агент моніторингу" представлена на рис.1.12 та у вигляді діаграми послідовностей (sequence diagram), де час плине зверху вниз, а стрілки показують обмін повідомленнями між різними учасниками процесу.

Першим учасником взаємодії є "Користувач" (actor), який ініціює процес моніторингу, надсилаючи "Запит на моніторинг вузла" до "Центрального модуля" (boundary). Центральний модуль, у свою чергу, передає команду агенту моніторингу, надсилаючи повідомлення "Надіслати команду моніторингу" до "Агента моніторингу" (entity).

Після отримання команди від центрального модуля, агент моніторингу ініціює збір даних за різними протоколами. Для збору даних ICMP агент взаємодіє з "Модулем збору даних ICMP" (control), надсилаючи йому команду "Ініціювати пінг". Модуль збору даних ICMP відправляє "ICMP Echo Request" до "Локальної мережі" (database), представляючи цільовий вузол. У відповідь локальна мережа повертає "ICMP Echo Reply" до модуля збору даних ICMP. Отримані результати ICMP ("Зберегти результати ICMP") передаються назад агенту моніторингу.

Аналогічний процес відбувається для протоколу Traceroute. Агент моніторингу надсилає команду "Ініціювати Traceroute" до "Модуля виконання Traceroute" (control). Модуль виконання Traceroute відправляє серію пакетів з поступово зростаючим значенням Time-to-Live ("Відправити серію пакетів TTL") до локальної мережі. У відповідь модуль отримує повідомлення "ICMP Time Exceeded/Reply" від проміжних маршрутизаторів або кінцевого вузла. Отримані результати Traceroute ("Зберегти результати Traceroute") передаються агенту моніторингу.

Для збору інформації за протоколом IPv6 NDP агент моніторингу надсилає "Запит інформації про сусідів NDP" до "Модуля збору даних NDP" (control). Модуль збору даних NDP відправляє "NDP Neighbor Solicitation" до локальної мережі для виявлення сусідніх IPv6 вузлів. У відповідь локальна мережа надсилає "NDP Neighbor Advertisement" з інформацією про сусідів. Отримані результати NDP

("Зберегти результати NDP") передаються агенту моніторингу.

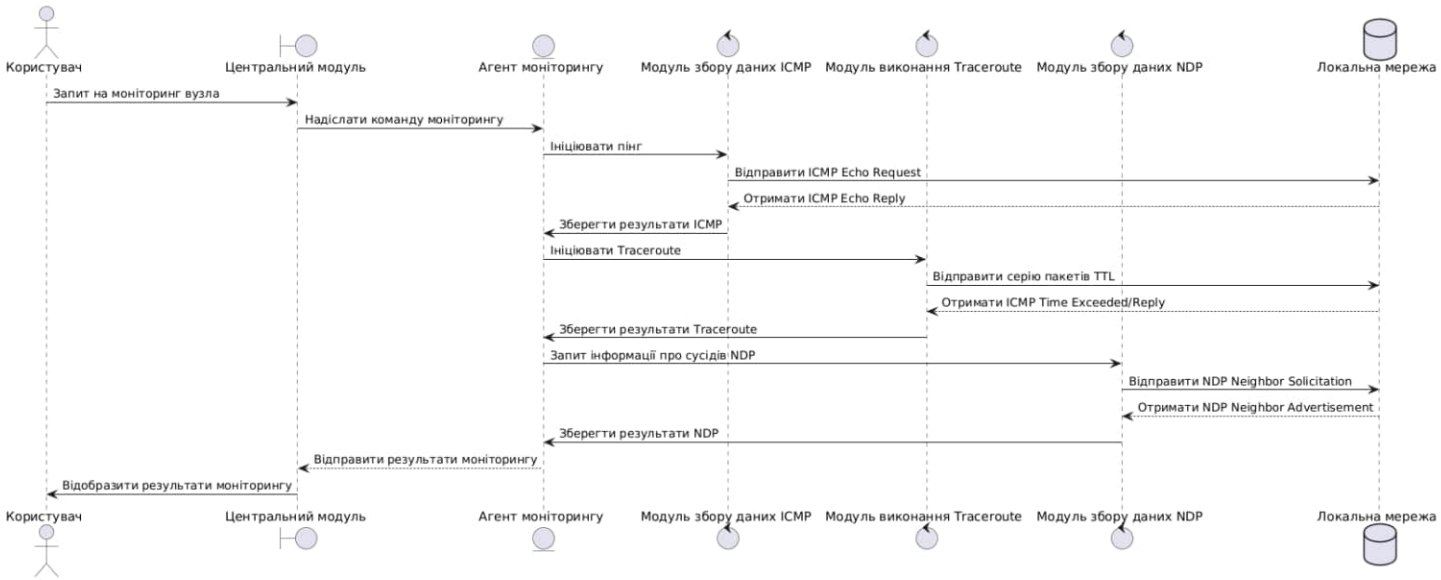

Рисунок 1.12 – UML-діаграма варіантів поведінки агента моніторингу

1.6 Мета і задачі і роботи

Метою даної кваліфікаційної роботи є розробка розподіленого програмного додатку для ефективного аналізу та моніторингу ключових мережевих протоколів – ICMP, Traceroute та IPv6 NDP – в локальних комп'ютерних мережах. Розроблений додаток повинен забезпечити збір, обробку та візуалізацію даних, отриманих за допомогою цих протоколів, у розподіленому середовищі, що сприятиме покращенню розуміння стану мережі, швидкому виявленню проблем та оптимізації її функціонування, зокрема в умовах зростання використання протоколу IPv6.

Для реалізації мети потрібно вирішити завдання:

– провести аналіз існуючих підходів і засобів моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP;

– дослідити особливості структури та функціонування протоколів ICMP (у тому числі з фрагментацією), Traceroute та IPv6 Neighbor Discovery Protocol;

– розробити архітектуру розподіленого програмного додатка для збору, обробки та аналізу відповідного мережевого трафіку;

– реалізувати програмний засіб з підтримкою: виявлення та класифікації ICMP-пакетів за типами та кодами; аналізу маршрутів передачі пакетів за допомогою Traceroute; виявлення IPv6 NDP-повідомлень та візуалізації зібраної статистики в зручному інтерфейсі;

– провести тестування та оцінювання ефективності роботи програмного додатка на основі реального або згенерованого мережевого трафіку.

2 СПЕЦІАЛЬНИЙ РОЗДІЛ

2.1 Технічні вимоги до програмного додатку для аналізу та моніторингу мережевих протоколів ICMP, Traceroute та IPv6-кодів

2.1.1 Найменування і призначення програмного додатку

Об’єктом професійної діяльності в рамках даної кваліфікаційної роботи є розподілений програмний додаток для аналізу та моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP. Призначення розробленого програмного додатку полягає у забезпеченні ефективного та гнучкого інструменту для адміністраторів та інженерів комп'ютерних мереж

2.1.2 Вимоги до структури і функціонування програмного додатку

Розподілений програмний додаток для аналізу та моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP складається з двох основних підсистем (рис.2.1) [4]:

– підсистема агентів моніторингу (ПАМ);

– підсистема центрального управління та візуалізації (ПЦУВ).

Підсистема агентів моніторингу (ПАМ) включає наступні функціональні складові:

– модуль збору даних ICMP: блок формування ICMP Echo Request; блок відправки ICMP Echo Request до заданих цільових вузлів; блок прийому ICMP Echo Reply; блок вимірювання часу затримки (RTT) та фіксації втрати пакетів.

– модуль виконання Traceroute: блок формування серії UDP або ICMP пакетів з послідовно зростаючим значенням TTL; блок відправки пакетів Traceroute до заданих цільових вузлів; блок прийому ICMP Time Exceeded та ICMP Echo Reply повідомлень; блок ідентифікації проміжних вузлів та вимірювання часу затримки на кожному "стрибку".

– модуль збору даних IPv6 NDP: блок прослуховування мережевого трафіку на предмет NDP повідомлень (Neighbor Solicitation, Neighbor Advertisement,

Router Solicitation, Router Advertisement; блок парсингу та інтерпретації отриманих NDP повідомлень; блок вилучення інформації про сусідні IPv6 вузли (адреси, MAC-адреси, префікси, стан).

– блок керування конфігурацією агента: рийом та застосування налаштувань від центрального модуля (цільові вузли, інтервали моніторингу).

– блок передачі даних: форматування зібраних даних та результатів аналізу у визначений формат; встановлення з'єднання та передача даних до центрального модуля.

Підсистема центрального управління та візуалізації (ПЦУВ) включає наступні функціональні складові:

– блок прийому даних від агентів: отримання даних моніторингу від підключених агентів; декодування отриманих даних;

– блок агрегації даних: збір та об'єднання даних, отриманих від різних агентів; збереження отриманих даних у базі даних;

– блок аналізу даних: аналіз зібраних даних ICMP (середній час затримки, відсоток втрат); аналіз результатів Traceroute (відображення маршрутів, виявлення проблемних ділянок); аналіз даних NDP (відображення сусідів, їхнього стану); виявлення аномалій та генерування сповіщень (за перевищенням заданих порогів).

– блок управління базою даних: зберігання історичних та поточних даних моніторингу; забезпечення доступу до даних для аналізу та візуалізації.

– блок візуалізації даних: відображення даних ICMP у вигляді графіків (часу затримки, втрат); відображення маршрутів Traceroute у текстовому або графічному форматі; відображення інформації про сусідів IPv6 у табличному вигляді або у вигляді карти мережі; відображення сповіщень та статусів мережевих вузлів.

– блок інтерфейсу користувача: надання інтуїтивно зрозумілого графічного інтерфейсу для налаштування параметрів моніторингу (додавання/видалення вузлів, налаштування інтервалів; відображення результатів моніторингу та аналізу даних; керування агентами моніторингу (додавання,

видалення, перегляд статусу).

– блок керування конфігурацією системи: налаштування глобальних параметрів системи, порогів сповіщень; керування параметрами підключення агентів.

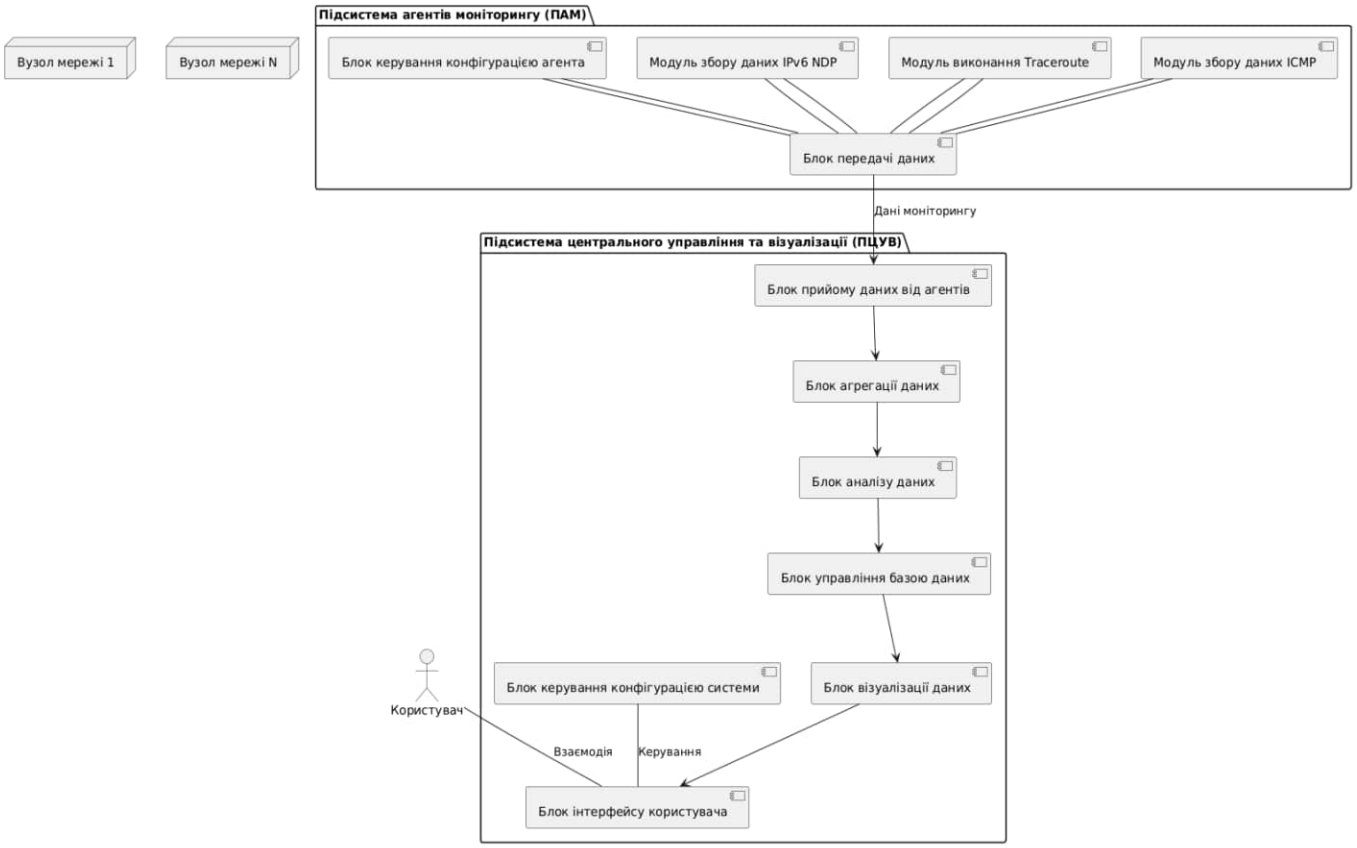

Рисунок 2.1 – Структурна схема підсистем програмного додатку

2.1.3 Вимоги до показників призначення програмного додатку

Розподілений програмний додаток для аналізу та моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP та його функціональні складові повинні забезпечувати наступні показники призначення:

Моніторинг доступності ICMP:

– періодичність надсилання ICMP Echo Request: не рідше ніж 1 раз на 5 секунд для критично важливих вузлів, з можливістю налаштування користувачем;

– час очікування ICMP Echo Reply: не більше 2 секунд з можливістю налаштування користувачем;

– відображення стану доступності вузла: "доступний", "недоступний", "часткова втрата пакетів";

– відображення відсотка втрачених пакетів та середнього часу затримки (RTT).

Трасування маршруту (Traceroute):

– можливість ініціювання трасування маршруту до заданого цільового вузла за запитом користувача;

– відображення повного маршруту до цільового вузла з зазначенням IP-адрес та імен хостів (за наявності);

– відображення часу затримки (RTT) для кожного "стрибка" маршруту;

– можливість налаштування максимальної кількості "стрибків".

Моніторинг сусідів IPv6 (NDP):

– періодичність опитування сусідів (Neighbor Solicitation): не рідше ніж 1 раз на 1 хвилину для активних вузлів, з можливістю налаштування користувачем;

– відображення списку виявлених сусідніх IPv6 вузлів;

– відображення MAC-адрес сусідніх вузлів;

– відображення стану сусідніх вузлів (reachable, stale, delay, probe);

– фіксація змін у стані сусідніх вузлів.

Розподілена архітектура:

– підтримка підключення до центрального модуля до 32 агентів моніторингу в одному сегменті мережі;

– можливість масштабування системи шляхом додавання нових агентів;

– відсутність критичної залежності функціонування всієї системи від одного агента (у випадку виходу з ладу одного агента, інші продовжують роботу).

Централізоване управління та візуалізація:

– надання єдиного інтерфейсу для налаштування моніторингу для всіх підключених агентів;

– відображення агрегованої інформації про стан мережі з усіх агентів на одній панелі керування;

– можливість фільтрації та сортування даних моніторингу за різними критеріями (вузол, протокол, час).

Зберігання даних:

– зберігання історичних даних моніторингу (доступність, час затримки, маршрути, інформація про сусідів) протягом щонайменше 7 календарних днів з можливістю налаштування терміну зберігання користувачем;

– можливість експорту даних у стандартних форматах (CSV, JSON).

Інтерфейс користувача:

– інтуїтивно зрозумілий графічний веб-інтерфейс;

– адаптивний дизайн для відображення на різних пристроях (ПК, планшети);

– можливість налаштування панелей моніторингу з відображенням ключових показників.

Сповіщення (планується в майбутніх версіях):

– можливість налаштування сповіщень про виявлення проблем (недоступність вузла, значне збільшення часу затримки, зміни в таблиці сусідів NDP);

– підтримка різних каналів сповіщень (електронна пошта, системні повідомлення).

2.2 Розробка апаратної частини

2.2.1 Апаратна інфраструктура

Розроблений програмний додаток призначений для функціонування на стандартних апаратних засобах, які зазвичай присутні в локальній мережі (рис.2.2):

– сервери або персональні комп'ютери для розгортання центрального модуля управління та візуалізації, який потребує обчислювальних ресурсів для обробки та зберігання даних, а також для надання користувацького інтерфейсу. Технічні вимоги до цих систем визначаються очікуваним навантаженням та обсягом даних моніторингу;

– мережеві вузли (кінцеві пристрої, сервери, IoT-пристрої) для розгортання агентів моніторингу. Агенти розробляються з урахуванням мінімальних вимог до ресурсів, щоб їх можна було ефективно встановлювати на різноманітних пристроях, що мають мережеве підключення та підтримують необхідні програмні бібліотеки для виконання мережевих операцій;

– мережеве обладнання (маршрутизатори, комутатори) використовується існуюча інфраструктура для передачі даних між агентами та центральним модулем, а також для здійснення мережевих запитів (ICMP, Traceroute, NDP) відповідно до функціональності додатку.

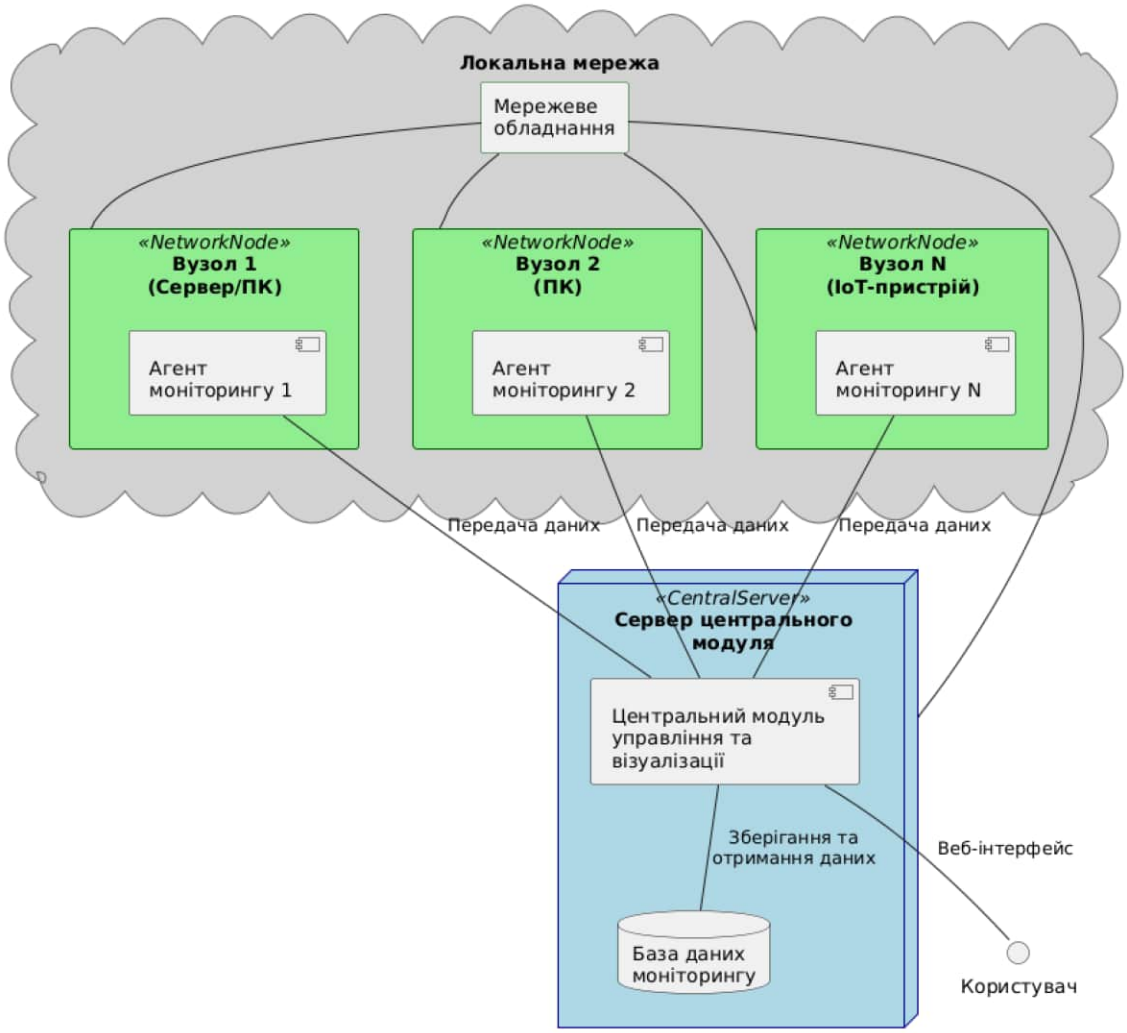

Рисунок 2.2 – Структурна схема апаратної інфраструктури

Використання існуючої апаратної інфраструктури є оптимальним рішенням з кількох причин:

– відсутність необхідності розробки та впровадження спеціалізованого обладнання значно знижує вартість та складність впровадження розробленого програмного забезпечення;

– програмний додаток може бути легко розгорнутий у більшості сучасних локальних мереж без потреби в додаткових апаратних інвестиціях;

– розподілена архітектура дозволяє масштабувати систему моніторингу шляхом додавання нових агентів на існуючі вузли мережі без необхідності придбання спеціалізованого обладнання для кожного нового пристрою, що контролюється.

Для агентів моніторингу, які розгортаються на вузлах локальної мережі, що підлягають контролю, вимоги до апаратного забезпечення є відносно невисокими. Мінімально достатнім буде будь-який сучасний процесор з тактовою частотою 1 ГГц або вище, 512 МБ оперативної пам'яті та 100 МБ вільного дискового простору для встановлення програмного забезпечення та тимчасових файлів. Обов'язковою є наявність активного підключення до локальної мережі через Ethernet або Wi-Fi, а також сучасна операційна система з підтримкою Python 3.x, така як Windows, Linux або macOS. Для більш комфортної роботи та моніторингу більшої кількості вузлів рекомендується використовувати багатоядерний процесор з тактовою частотою 2 ГГц або вище, 1 ГБ або більше оперативної пам'яті та 500 МБ вільного дискового простору для встановлення та ведення логів. Також бажаним є стабільне та швидке підключення до локальної мережі через Gigabit Ethernet, та оптимізована серверна операційна система з підтримкою Python 3.x, сучасні дистрибутиви Linux. Важливо зазначити, що фактичні вимоги можуть варіюватися залежно від специфіки мережі та інтенсивності моніторингу.

Сервер основного модуля управління та візуалізації, який відповідає за прийом даних від усіх агентів, їхню обробку, зберігання в базі даних та надання користувацького інтерфейсу, потребує більш потужного апаратного забезпечення. Мінімально необхідним є двоядерний процесор з тактовою частотою 2 ГГц або

вище, 4 ГБ оперативної пам'яті та 50 ГБ вільного дискового простору для встановлення програмного забезпечення, бази даних та логів. Рекомендується використання SSD для підвищення швидкодії. Обов'язковою є наявність стабільного підключення до локальної мережі через Ethernet та серверна операційна система (наприклад, Linux Server або Windows Server) з підтримкою Python 3.x та обраної бази даних (PostgreSQL, TimescaleDB). Для забезпечення належної продуктивності при великій кількості підключених агентів та значному обсязі даних моніторингу рекомендується використовувати чотириядерний або восьмиядерний процесор з тактовою частотою 2.5 ГГц або вище, 8 ГБ або більше оперативної пам'яті (рекомендується 16 ГБ для великих мереж) та 200 ГБ або більше вільного місця на швидкому SSD для операційної системи, програмного забезпечення та бази даних. Розмір дискового простору слід планувати з урахуванням очікуваного обсягу даних та періоду їхнього зберігання.

Для доступу до веб-інтерфейсу центрального модуля клієнтські робочі станції не потребують спеціалізованого апаратного забезпечення. Будь-яка сучасна комп'ютерна система зі встановленим сучасним веб-браузером (Google Chrome, Mozilla Firefox, Microsoft Edge або Safari) та активним підключенням до локальної мережі або Інтернету буде достатньою.

Щодо мережевого обладнання, розроблений програмний додаток використовує існуючу інфраструктуру, таку як комутатори та маршрутизатори, для забезпечення зв'язку між агентами та центральним модулем, а також для здійснення мережевих запитів. Додаткових специфічних вимог до мережевого обладнання не висувається, за умови його належного функціонування та підтримки базових мережевих протоколів TCP/IP.

2.2.2 Вибір елементної бази програмного додатку

Розробка складного програмного забезпечення, такого як розподілений додаток для аналізу та моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP, вимагає ретельного вибору компонентів, які забезпечать необхідну

функціональність, продуктивність, надійність та зручність розробки. У контексті програмної частини системи, "елементна база" охоплює мови програмування, бібліотеки, фреймворки, бази даних та інструменти, що використовуються на кожному етапі створення та експлуатації додатку.

Основною мовою програмування для розробки як агентів моніторингу, так і центрального модуля управління та візуалізації було обрано Python. Це рішення зумовлене низкою вагомих переваг, які роблять Python одним з найпопулярніших та найефективніших інструментів для розробки мережевих застосунків та систем обробки даних.

Python має надзвичайно багату стандартну бібліотеку, яка включає модулі для роботи з мережевими сокетами (socket), обробки текстових даних, виконання системних викликів та багато іншого. Це дозволяє виконувати базові мережеві операції без необхідності встановлення сторонніх бібліотек.

Для специфічних завдань, таких як робота з мережевими протоколами, аналіз трафіку, розробка веб-інтерфейсів та візуалізація даних, існує безліч високоякісних та добре підтримуваних сторонніх бібліотек. Це значно прискорює процес розробки та дозволяє використовувати вже готові, перевірені рішення замість написання коду з нуля.

Синтаксис Python є відносно простим та інтуїтивно зрозумілим, що полегшує вивчення мови новими розробниками та робить код більш читабельним і

Завдяки великій кількості готових інструментів та високому рівню абстракції, Python дозволяє розробникам писати менше коду для виконання складних завдань, що значно скорочує час розробки.

Python є кросплатформною мовою, що дозволяє запускати розроблений додаток на різних операційних системах (Windows, Linux, macOS) без значних змін коду. Це є важливим для розподіленої системи, де агенти можуть бути розгорнуті на різноманітних пристроях.

Велика та активна спільнота розробників Python забезпечує постійну підтримку, оновлення бібліотек, виявлення та виправлення помилок, а також велику кількість документації та навчальних матеріалів.

Для ефективної роботи з протоколами ICMP, Traceroute та IPv6 NDP було обрано наступні ключові бібліотеки Python:

– на рівні агентів моніторингу, стандартна бібліотека socket використовується для низькорівневої взаємодії з мережевими сокетами. Це необхідно для надсилання спеціально сформованих пакетів ICMP Echo Request (пінг) та для обробки отриманих відповідей. Також socket може бути використаний для надсилання UDP-пакетів, які часто застосовуються в Traceroute;

– бібліотека ping3значно спрощує процес надсилання ICMP-запитів та отримання відповідей. Вона надає зручні функції для вимірювання часу затримки (Round-Trip Time - RTT) та виявлення втрати пакетів з мінімальною кількістю коду. ping3 є більш високорівневим інструментом порівняно з безпосередньою роботою з сокетами для ICMP;

– бібліотека scapy є надзвичайно потужним інструментом для створення, відправки, перехоплення та аналізу мережевих пакетів на різних рівнях моделі OSI. Для реалізації функціональності Traceroute, scapy дозволяє легко формувати серію IP-пакетів з послідовно зростаючим значенням Time-To-Live (TTL) та аналізувати отримані ICMP Time Exceeded повідомлення від проміжних маршрутизаторів. Крім того, scapy надає можливості для пасивного прослуховування мережевого трафіку та аналізу протоколу IPv6 NDP, дозволяючи виділяти інформацію про сусідні вузли, їхні адреси, MAC-адреси та стан. Гнучкість та широкі можливості scapy роблять її незамінною для глибокого аналізу мережевих протоколів.

Для розробки центрального модуля управління та візуалізації, який включає веб-інтерфейс для користувача та API для взаємодії з агентами використано Python-фреймворк Flask, який надає базову функціональність для створення веб-застосунків, залишаючи розробнику свободу вибору додаткових компонентів (ORM, шаблонізатор тощо). Flask є легким, гнучким та добре підходить для розробки невеликих та середніх за складністю веб-застосунків та API. Його простота може прискорити початковий етап розробки.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Програмування додатку для аналізу та моніторингу мережевих протоколів

3.1.1 Призначення програмного додатку

Розроблений програмний додаток призначений для автоматизованого аналізу та моніторингу стану локальних комп'ютерних мереж шляхом збору та обробки даних, отриманих за допомогою ключових мережевих протоколів: ICMP, Traceroute та IPv6 NDP. Програмний додаток надає адміністраторам мережі інструменти для:

– оперативного відстеження доступності мережевих вузлів за допомогою протоколу ICMP, виявляючи непрацюючі пристрої та проблеми зі з'єднанням;

– діагностики мережевих проблем шляхом аналізу маршрутів проходження пакетів (Traceroute) та виявлення проблемних ділянок з високою затримкою або втратою пакетів;

– моніторингу мережевого оточення IPv6 через аналіз протоколу NDP, отримуючи інформацію про сусідні IPv6 вузли, їхні MAC-адреси та стан, що є критично важливим для розуміння та підтримки мереж нового покоління;

– централізованого збору та аналізу даних з різних вузлів мережі за допомогою розподіленої архітектури, що забезпечує гнучкість та масштабованість рішення;

– візуалізації зібраних та проаналізованих даних у зручному для користувача форматі (графіки, таблиці), що полегшує розуміння стану мережі та виявлення аномалій.

Програма складається з двох основних компонентів: агентів моніторингу, які розгортаються на окремих вузлах мережі, та центрального модуля управління та візуалізації, який агрегує, обробляє та відображає отримані дані.

Розроблений програмний додаток для аналізу та моніторингу мережевих протоколів функціонує за розподіленою архітектурою, що включає окремих агентів моніторингу, розгорнутих на вузлах мережі, та центральний модуль управління та візуалізації, який агрегує, обробляє та відображає зібрані дані. Алгоритм роботи програми представлено на рис.3.1 та чітко розділяється на процеси, що виконуються

на рівні агента, та процеси центрального модуля, забезпечуючи ефективний та скоординований моніторинг мережевого стану.

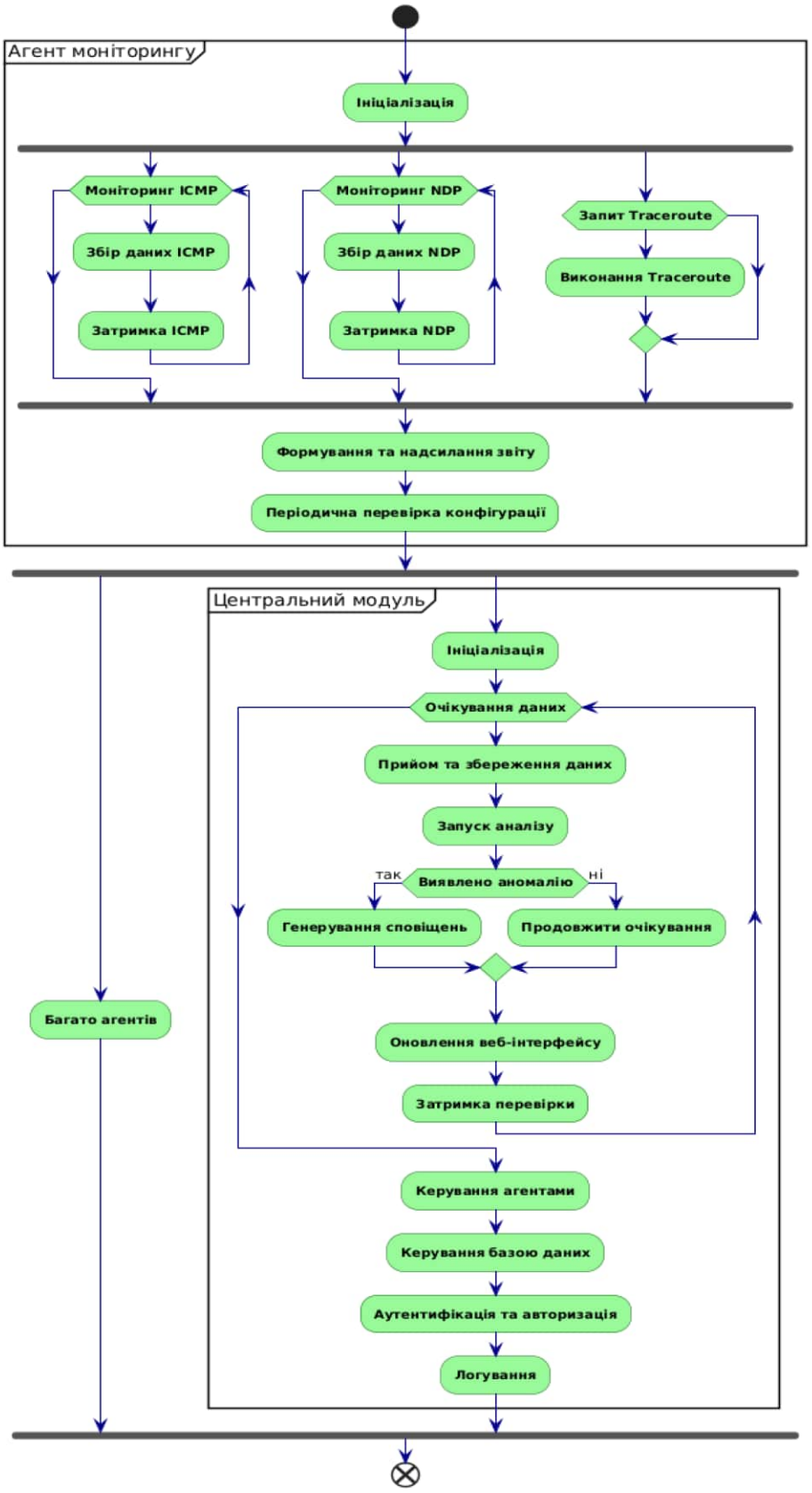

Рисунок 3.1 – Структурна схема алгоритму роботи програмного додатку

На рівні агента моніторингу першим кроком є ініціалізація, під час якої агент встановлює необхідні параметри та готується до виконання своїх функцій. Після ініціалізації агент переходить до паралельного виконання кількох основних задач моніторингу. Одна з гілок цього паралельного виконання відповідає за безперервний моніторинг протоколу ICMP. У циклі, що триває протягом часу роботи агента, відбувається періодичний збір даних ICMP, що включає формування та надсилання ICMP Echo Request до цільових вузлів та аналіз отриманих відповідей для визначення доступності вузла та часу затримки. Між кожними ітераціями збору даних ICMP передбачена певна затримка, що визначає частоту опитування.

Друга паралельна гілка відповідає за безперервний моніторинг протоколу IPv6 NDP. Аналогічно до моніторингу ICMP, агент у безкінечному циклі прослуховує мережевий трафік на предмет NDP повідомлень, аналізує їхній вміст та оновлює локальну інформацію про сусідні IPv6 вузли. Між кожними ітераціями прослуховування також існує певна затримка, що регулює частоту оновлення інформації про сусідів.

Третя паралельна гілка відповідає за виконання Traceroute. Цей процес є умовним і запускається лише за наявності відповідного запиту від центрального модуля або за певним розкладом. При отриманні запиту агент ініціює процедуру Traceroute до вказаного цільового вузла, відстежуючи маршрут проходження пакетів та фіксуючи проміжні вузли та час затримки до них.

Після завершення (або паралельно з виконанням) процесів моніторингу агент здійснює формування та надсилання звіту до центрального модуля. Звіт містить зібрані дані щодо стану ICMP, інформацію про сусідів NDP та результати Traceroute (за наявності). Крім того, агент періодично перевіряє наявність оновлень конфігурації від центрального модуля та застосовує їх для адаптації до змін у вимогах моніторингу.

На рівні основного модуля управління та візуалізації першим кроком також є ініціалізація, під час якої запускаються необхідні сервіси, встановлюється з'єднання з базою даних та готується веб-інтерфейс для користувачів. Основний процес роботи центрального модуля полягає в безперервному очікуванні даних від агентів.

У циклі, що триває протягом роботи модуля, відбувається прийом даних, надісланих агентами, їхня обробка та збереження в базі даних для подальшого аналізу та відображення.

Після збереження отриманих даних запускається процес аналізу даних, під час якого відбувається агрегація, порівняння з попередніми значеннями або заданими порогами для виявлення потенційних проблем або аномалій у мережі. У випадку виявлення аномалії генеруються відповідні сповіщення для адміністраторів. Результати аналізу та поточний стан мережі відображаються через веб-інтерфейс, який періодично оновлюється для надання актуальної інформації користувачам.

Центральний модуль також забезпечує функціональність для керування агентами, що включає можливість додавання нових агентів, перегляду їхнього статусу та налаштування параметрів їхньої роботи. Важливим аспектом є керування базою даних, що включає операції зі збереження, резервного копіювання та очищення даних моніторингу. Для забезпечення безпеки доступу до системи реалізовані механізми аутентифікації та авторизації користувачів. Всі важливі події та дії в роботі центрального модуля фіксуються в системних журналах за допомогою процесу логування.

3.1.2 Модуль агента

Програмна реалізація агента моніторингу мережевих протоколів являє собою багатопотоковий застосунок на мові Python, призначений для періодичного збору та передачі ключових метрик мережевого стану до центрального модуля управління. Архітектура агента розроблена з метою забезпечення одночасного моніторингу різних мережевих протоколів, асинхронної обробки даних та ефективної передачі зібраної інформації [13].

Основу функціональності агента складають модулі для моніторингу трьох ключових протоколів: ICMP, IPv6 NDP та Traceroute. Кожен з цих модулів відповідає за специфічний метод збору даних та їхнє первинне форматування. Для забезпечення паралельного виконання завдань моніторингу, зокрема безперервного

прослуховування NDP трафіку, використовується механізм багатопоточності.

Процес моніторингу ICMP реалізовано за допомогою бібліотеки ping3, яка дозволяє надсилати ICMP Echo Request до заданих цільових хостів та аналізувати отримані відповіді. Агент періодично опитує список цільових IP-адрес або доменних імен, вимірюючи час затримки (RTT) та визначаючи статус доступності кожного хоста. Зібрані статистичні дані (цільовий хост, RTT, статус) зберігаються у структурованому форматі для подальшої передачі (рис.3.2).

```
import time
import platform
import requests
import json
import subprocess
from ping3 import ping
from scapy.all import IP, UDP, ICMP, srl, sniff, Ether, IPv6, ICMPv6, ICMPv6NDOp
tDstLLAddr

# --- Налаштування агента ---
CENTRAL_MODULE_URL = "http://127.0.0.1:5000/data"  # URL центрального модуля
AGENT_ID = f"net_agent_{platform.node()}"  # Унікальний ідентифікатор агента
MONITORING_INTERVAL = 10  # Інтервал між повними циклами моніторингу у секундах
TARGET_HOSTS_ICMP = ["8.8.8.8", "192.168.1.1"]  # Цільові хости для ICMP монітор
ингу (може отримуватися з конфігурації)
TARGET_HOSTS_TRACEROUTE = ["google.com"]  # Цільові хости для Traceroute (може о
тримуватися з конфігурації)
NDP_INTERFACE = "eth0"  # Назва мережевого інтерфейсу для NDP моніторингу (може
отримуватися з конфігурації)
MAX_TRACEROUTE_HOPS = 30

# --- Функції збору даних ---
def get_icmp_stats(target_host):
    """Виконує ICMP пінг та повертає статистику."""
    delay = ping(target_host, timeout=1)
    if delay is not None:
        return {"target": target_host, "rtt": delay * 1000, "status": "up"}
    else:
        return {"target": target_host, "status": "down"}
```

Рисунок 3.2 – Фрагмент реалізації моніторингу ICMP

Моніторинг протоколу IPv6 Neighbor Discovery Protocol (NDP) здійснюється з використанням бібліотеки scapy. Для забезпечення безперервного збору інформації про сусідів IPv6, агент запускає окремий потік, в якому відбувається прослуховування вказаного мережевого інтерфейсу на предмет NDP пакетів (Neighbor Solicitation та Neighbor Advertisement). Функція обробки пакетів process_ndp_packet аналізує отримані NDP повідомлення, витягуючи інформацію

про IPv6 адреси сусідів та їхні відповідні MAC-адреси, яка зберігається у глобальному словнику NDP_NEIGHBORS. Періодично, основний процес агента отримує копію цього словника для подальшої відправки на центральний модуль (рис.3.3).

```
NDP_NEIGHBORS = {}
def process_ndp_packet(packet):
    """Обробляє NDP пакети та оновлює інформацію про сусідів."""
    if IPv6 in packet and ICMPv6NDOptDstLLAddr in packet:
        target_ip = packet[IPv6].dst
        target_mac = packet[ICMPv6NDOptDstLLAddr].lladdr
        NDP_NEIGHBORS[target_ip] = target_mac
```

Рисунок 3.3 – Фрагмент реалізації моніторингу протоколу IPv6 Neighbor Discovery Protocol

Функціональність Traceroute реалізована також з використанням бібліотеки scapy. За запитом (у поточній версії - періодично для заданих цільових хостів), агент формує та надсилає серію UDP пакетів з послідовно зростаючим значенням Time-To-Live (TTL) до цільового IP-адреси. Аналізуючи отримані у відповідь ICMP Time Exceeded повідомлення, агент визначає маршрут проходження пакетів, фіксуючи IP-адреси проміжних вузлів та час затримки до них. Результати трасування для кожного цільового хоста зберігаються у вигляді списку пройдених вузлів (рис.3.4).

```
def traceroute(target_ip):
    """Виконує трасування маршруту до вказаного IP."""
    results = []
    for ttl in range(1, MAX_TRACEROUTE_HOPS + 1):
        packet = IP(dst=target_ip, ttl=ttl) / UDP(dport=33434)
        reply = sr1(packet, timeout=1, verbose=0)
        if reply is None:
            results.append({"hop": ttl, "ip": "*", "rtt": None})
        elif reply.type == 3:  # ICMP Destination Unreachable
            results.append({"hop": ttl, "ip": reply.src, "rtt": (reply.time - pa
cket.sent_time) * 1000})
            break
        else:
            results.append({"hop": ttl, "ip": reply.src, "rtt": (reply.time - pa
cket.sent_time) * 1000})
    return results
```

Рисунок 3.4 – Фрагмент реалізації моніторингу протоколу Traceroute

Зібрані дані за всіма протоколами (статистика ICMP, інформація про сусідів NDP, результати Traceroute) агрегуються у єдиний словник report, де кожен тип

даних представлений під відповідним ключем. Для передачі цих даних на центральний модуль використовується бібліотека requests, яка здійснює HTTP POST запит на визначений URL-адресу (CENTRAL_MODULE_URL). Дані серіалізуються у формат JSON перед відправкою, забезпечуючи стандартизований спосіб обміну інформацією між агентом та центральним модулем. У заголовках запиту вказується тип вмісту application/json. У випадку успішної відправки або виникнення помилки під час передачі, відповідні повідомлення виводяться у консоль агента для локального моніторингу.

```
# --- Функція для надсилання даних на центральний модуль ---
def send_data_to_central(data):
    """Надсилає зібрані дані у форматі JSON на центральний модуль."""
    try:
        headers = {'Content-Type': 'application/json'}
        payload = {"agent_id": AGENT_ID, "timestamp": time.time(), "data": data}
        response = requests.post(CENTRAL_MODULE_URL, headers=headers, json=paylo
ad)
        response.raise_for_status()
        print(f"[{AGENT_ID}] Data sent successfully: {response.status_code}")
    except requests.exceptions.RequestException as e:
        print(f"[{AGENT_ID}] Error sending data: {e}")
```

Рисунок 3.5 – Фрагмент реалізації статистики

Основний цикл роботи агента полягає у періодичному виконанні описаних вище кроків. Агент ініціалізується, запускає потік для прослуховування NDP, а потім в безкінечному циклі, з інтервалом, визначеним змінною MONITORING_INTERVAL, збирає дані за протоколами ICMP, NDP та Traceroute, формує звіт та надсилає його на центральний модуль. Такий підхід забезпечує постійний моніторинг мережевого стану та своєчасне інформування центрального модуля про зміни та потенційні проблеми.

```
# --- Основна функція агента ---
def main():
    """Основний цикл роботи агента моніторингу."""
    print(f"[{AGENT_ID}] Network Agent started.")

    # Запуск прослуховування NDP у окремому потоці (або асинхронно)
    def ndp_listener():
        sniff(iface=NDP_INTERFACE, filter="icmp6 and ip6[40] == 135 or icmp6 and
 ip6[40] == 136", prn=process_ndp_packet, store=0)

    import threading
    ndp_thread = threading.Thread(target=ndp_listener, daemon=True)
    ndp_thread.start()

    while True:
        report = {}

        # Збір статистики ICMP
        icmp_stats = [get_icmp_stats(target) for target in TARGET_HOSTS_ICMP]
        report["icmp_stats"] = icmp_stats

        # Збір інформації про сусідів NDP
        report["ndp_neighbors"] = get_ndp_neighbors()

        # Виконання Traceroute
        traceroute_results = {}
        for target in TARGET_HOSTS_TRACEROUTE:
            traceroute_results[target] = traceroute(target)
        report["traceroute_results"] = traceroute_results

        # Надсилання зібраних даних
        send_data_to_central(report)

        time.sleep(MONITORING_INTERVAL)
```

Рисунок 3.6 – Фрагмент реалізації основного циклу роботи агента

3.1.3 Основний модуль

Основний модуль управління та візуалізації є ключовим компонентом розподіленої системи моніторингу мережевих протоколів. Він відповідає за прийом даних від численних агентів моніторингу, їхнє зберігання, агрегацію, аналіз та надання користувачам зручного інтерфейсу для перегляду стану мережі та управління системою. Програмна реалізація центрального модуля виконана на мові Python з використанням фреймворку Flask для створення RESTful API та забезпечення веб-інтерфейсу.

Основна функціональність центрального модуля реалізована через набір API-ендпоінтів та внутрішніх функцій. Для забезпечення можливості прийому даних від агентів, розроблено POST-ендпоінт /data. Коли агент надсилає дані моніторингу,

центральний модуль приймає HTTP POST запит, перевіряє його вміст на відповідність формату JSON та витягує ідентифікатор агента, час відправлення та самі метрики. Отримані дані зберігаються у структурованому вигляді. У поточній реалізації для зберігання використовується словник data_store в оперативній пам'яті, де ключем є ідентифікатор агента, а значенням - список отриманих від нього метрик з часовими мітками. У промисловому середовищі замість словника слід використовувати повноцінну базу даних (PostgreSQL) для забезпечення стійкості та масштабованості зберігання даних.

Для надання інформації про підключені агенти та їхні дані, розроблено GET-ендпоінти /agents та /agent/<agent_id>. Запит на /agents повертає список ідентифікаторів усіх агентів, від яких були отримані дані. Запит на /agent/<agent_id> повертає JSON-відповідь, що містить список усіх даних, отриманих від конкретного агента з вказаним ідентифікатором. Ці ендпоінти забезпечують базову можливість перегляду активних агентів та їхніх сирих даних.

Для надання користувачам можливості візуалізації зібраних та проаналізованих даних, передбачається розробка веб-інтерфейсу. У поточному коді закладено лише базову структуру Flask-сервера, але для реального веб-інтерфейсу знадобиться використання шаблонізатора (Jinja2) для динамічного формування HTML-сторінок, а також бібліотек для побудови графіків (Matplotlib, Chart.js) та інтерактивних елементів. Веб-інтерфейс повинен відображати поточний стан мережі, історичні дані у вигляді графіків та таблиць, а також надавати інструменти для фільтрації, сортування та експорту даних (рис.3.7).

Керування базою даних є важливим аспектом центрального модуля. У поточній спрощеній реалізації використовується словник в пам'яті, що не є стійким рішенням. Для забезпечення надійного зберігання даних, їхньої цілісності та можливості масштабування, необхідно інтегрувати повноцінну систему керування базами даних (СКБД). Вибір конкретної СКБД залежить від вимог до продуктивності, масштабованості та складності даних. Після інтеграції СКБД необхідно реалізувати функції для збереження, отримання, оновлення та видалення даних моніторингу, а також механізми для архівування та очищення застарілих

даних відповідно до політик зберігання.

Аутентифікація та авторизація є важливими для забезпечення безпеки доступу до центрального модуля та його функціональності. У наданому коді ці механізми не реалізовані. У майбутньому необхідно додати систему облікових записів користувачів, процедури аутентифікації (перевірки ідентичності) та авторизації (перевірки прав доступу) для захисту даних та запобігання несанкціонованому доступу до функцій управління.

Логування є невід'ємною частиною будь-якої серверної програми. Центральний модуль повинен вести докладні журнали своєї роботи, фіксуючи події прийому даних, обробки запитів, виникнення помилок та інші важливі системні події. Це полегшить діагностику проблем, відстеження активності та аудит безпеки. Для реалізації логування можна використовувати вбудовані засоби Python або спеціалізовані бібліотеки для логування (наприклад, logging).

```
from flask import Flask, request, jsonify
import time

app = Flask(__name__)
data_store = {}

@app.route('/data', methods=['POST'])
def receive_data():
    if request.is_json:
        data = request.get_json()
        agent_id = data.get('agent_id')
        timestamp = data.get('timestamp')
        metrics = data.get('data')

        if agent_id:
            if agent_id not in data_store:
                data_store[agent_id] = []
            data_store[agent_id].append({'timestamp': timestamp, 'metrics': metr
ics})
            print(f"[Central Module] Received data from {agent_id} at {time.strf
time('%Y-%m-%d %H:%M:%S', time.localtime(timestamp))}: {metrics}")
            return jsonify({"status": "success"}), 200
        else:
            return jsonify({"error": "Agent ID not provided"}), 400
    else:
        return jsonify({"error": "Request must be JSON"}), 400

@app.route('/agents', methods=['GET'])
def get_agents():
    return jsonify(list(data_store.keys())), 200

@app.route('/agent/<agent_id>', methods=['GET'])
def get_agent_data(agent_id):
    if agent_id in data_store:
        return jsonify(data_store[agent_id]), 200
    else:
        return jsonify({"error": f"Agent {agent_id} not found"}), 404

if __name__ == '__main__':
    print("[Central Module] Starting...")
    app.run(debug=True, host='0.0.0.0')
```

Рисунок 3.7 – Фрагмент реалізації основного модуля

3.1.4 Розробка бази даних PostgreSQL для центрального модуля

Для ефективного зберігання, обробки та аналізу даних, отриманих від агентів моніторингу, необхідна надійна та структурована база даних. В якості такої системи керування базами даних (СКБД) обрано PostgreSQL, яка відрізняється своєю стабільністю, розширюваністю та підтримкою складних запитів.

Розроблена схема бази даних включає кілька взаємопов'язаних таблиць, призначених для зберігання інформації про агентів, результатів моніторингу ICMP, даних NDP та результатів Traceroute. Така структура дозволяє логічно організувати дані, спростити їхнє отримання та аналіз, а також забезпечити цілісність інформації.

Таблиця agents представлена в табл.3.1 призначена для зберігання інформації про зареєстрованих агентів моніторингу. Кожен запис у таблиці відповідає одному агенту.

Таблиця 3.1 – Таблиця agents

Назва стовпця	Тип даних	Обмеження	Опис
agent_id	VARCHAR(255)	PRIMARY KEY	Унікальний ідентифікатор агента (наприклад, hostname)
first_seen	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Час першої реєстрації агента
last_seen	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Час останнього отримання даних від агента
description	TEXT		Додатковий опис агента (за потреби)

Таблиця icmp_stats (табл. 3.2) зберігає статистику моніторингу протоколу ICMP, отриману від агентів.

Таблиця ndp_neighbors (табл. 3.3) зберігає інформацію про сусідів IPv6, виявлених агентами.

Таблиця 3.2 – Таблиця icmp_stats

Назва стовпця	Тип даних	Обмеження	Опис
id	BIGSERIAL	PRIMARY KEY	Унікальний ідентифікатор запису
agent_id	VARCHAR(255)	NOT NULL REFERENCES agents(agent_id)	Ідентифікатор агента, що надіслав дані
timestamp	TIMESTAMP	NOT NULL	Час збору даних агентом
target_host	VARCHAR(255)	NOT NULL	IP-адреса або доменне ім'я цільового хоста
rtt_ms	DOUBLE PRECISION		Час затримки (Round-Trip Time) у мілісекундах
status	VARCHAR(10)	NOT NULL	Статус цільового хоста ("up" або "down")

Таблиця 3.3 – Таблиця ndp_neighbors

Назва стовпця	Тип даних	Обмеження	Опис
id	BIGSERIAL	PRIMARY KEY	Унікальний ідентифікатор запису
agent_id	VARCHAR(255)	NOT NULL REFERENCES agents(agent_id)	Ідентифікатор агента, що надіслав дані
timestamp	TIMESTAMP	NOT NULL	Час виявлення сусіда агентом
neighbor_ip	INET	NOT NULL	IPv6 адреса сусіднього вузла
neighbor_mac	MACADDR	NOT NULL	MAC адреса сусіднього вузла

Таблиця traceroute_results (табл. 3.4) зберігає результати виконання трасування маршруту агентами.

Таблиця 3.4 – Таблиця traceroute_results

Назва стовпця	Тип даних	Обмеження	Опис
id	BIGSERIAL	PRIMARY KEY	Унікальний ідентифікатор запису
agent_id	VARCHAR(255)	NOT NULL REFERENCES agents(agent_id)	Ідентифікатор агента, що виконав трасування
timestamp	TIMESTAMP	NOT NULL	Час виконання трасування агентом
target_host	VARCHAR(255)	NOT NULL	IP-адреса або доменне ім'я цільового хоста
hop	INTEGER	NOT NULL	Порядковий номер стрибка
hop_ip	INET		IP-адреса проміжного вузла (або NULL, якщо не знайдено)

Розроблена схема бази даних для центрального модуля системи моніторингу мережевих протоколів ґрунтується на реляційній моделі та передбачає використання PostgreSQL як основної СКБД. Структура бази даних організована навколо чотирьох таблиць, кожна з яких відповідає за зберігання певного типу інформації, отриманої від агентів моніторингу. Зв'язки між цими таблицями забезпечують цілісність даних та ефективність виконання запитів (рис.3.8).

Центральною таблицею схеми є agents. Вона виконує роль довідника, що містить інформацію про кожен зареєстрований в системі агент моніторингу. Первинним ключем цієї таблиці є стовпець agent_id, який має тип VARCHAR(255) і призначений для зберігання унікального ідентифікатора кожного агента. Цей ідентифікатор може відповідати, наприклад, імені хоста, IP-адресі або іншому унікальному значенню, що дозволяє однозначно ідентифікувати джерело даних.

Для зберігання результатів моніторингу протоколу ICMP призначена таблиця icmp_stats. Кожен запис у цій таблиці відображає статистичні дані, зібрані одним з

агентів у певний момент часу для конкретного цільового хоста. Таблиця має первинний ключ id типу BIGSERIAL, що забезпечує автоматичне генерування унікальних ідентифікаторів для кожного запису. Ключовим елементом, що пов'язує ці дані з конкретним агентом, є стовпець agent_id типу VARCHAR(255), який є зовнішнім ключем та посилається на первинний ключ (agent_id) таблиці agents. Цей зв'язок є типу "один-до-багатьох", оскільки один агент може надсилати безліч записів статистики ICMP протягом свого життєвого циклу. Зв'язок між таблицями agents та icmp_stats семантично описується як "has", підкреслюючи, що агент "має" ці статистичні дані.

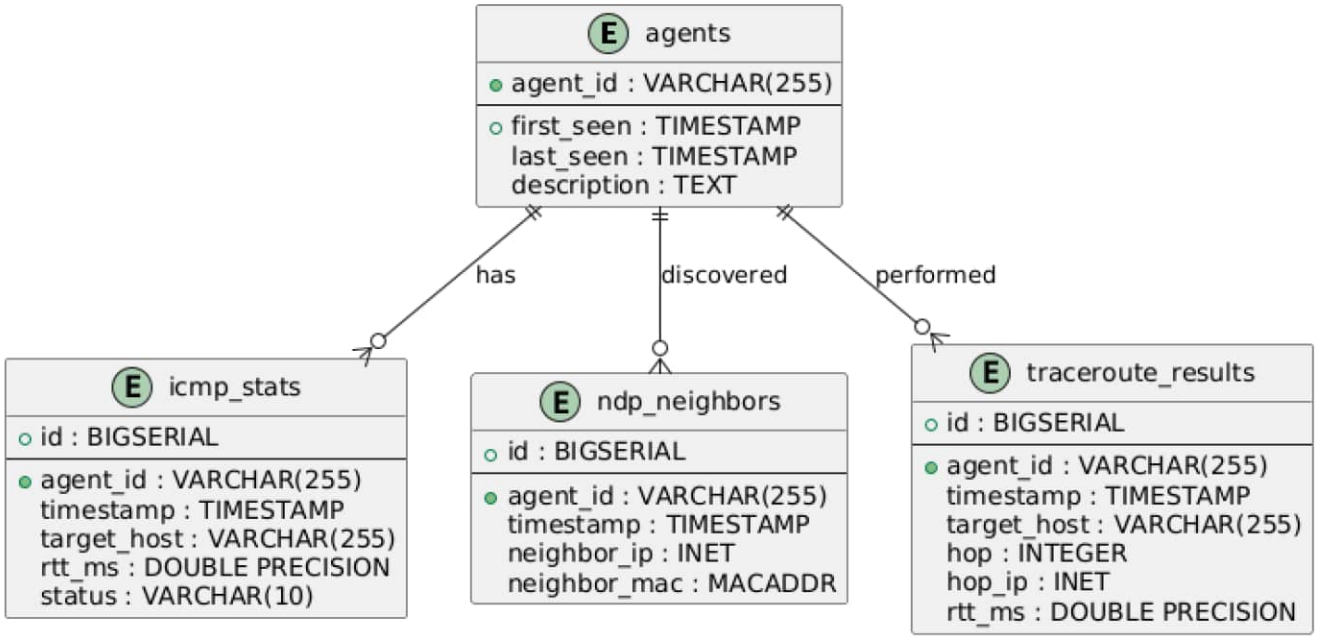

Рисунок 3.8 – Схема бази даних програмного додатку для аналізу та моніторингу мережевих протоколів

Інформація про сусідів IPv6, виявлених агентами за допомогою протоколу NDP, зберігається в таблиці ndp_neighbors. Аналогічно до таблиці icmp_stats, вона має первинний ключ id типу BIGSERIAL для унікальної ідентифікації записів. Зв'язок з агентом, що надав дані, встановлюється через зовнішній ключ agent_id, який посилається на таблицю agents. Таким чином, один агент може виявляти та повідомляти про безліч сусідів NDP, що також відображає зв'язок "один-до-багатьох". Семантично цей зв'язок описується як "discovered", вказуючи, що агент

"виявив" цих сусідів.

Результати виконання трасування маршруту (Traceroute), ініційованого агентами, зберігаються в таблиці traceroute_results. Ця таблиця також має первинний ключ id типу BIGSERIAL. Зовнішній ключ agent_id забезпечує зв'язок з агентом, який виконав трасування. Оскільки один агент може виконувати трасування до різних цільових хостів і отримувати кілька проміжних результатів для кожного трасування, між таблицями agents та traceroute_results також існує зв'язок "один-до-багатьох". Характер цього зв'язку описується як "performed", оскільки агент "виконав" трасування.

Для забезпечення унікальності кожного запису в таблицях icmp_stats, ndp_neighbors та traceroute_results використовується первинний ключ id типу BIGSERIAL. Тип BIGSERIAL є зручним способом автоматичного генерування унікальних зростаючих цілих чисел у PostgreSQL, що є ідеальним для первинних ключів [14].

При розробці схеми бази даних особливу увагу було приділено вибору відповідних типів даних для кожного стовпця. Це необхідно для ефективного зберігання інформації та забезпечення її коректної обробки. Наприклад, для зберігання IP-адрес використано тип INET, який оптимізований для зберігання та порівняння IPv4 та IPv6 адрес. Для MAC-адрес використано спеціалізований тип MACADDR, що забезпечує валідацію формату MAC-адрес. Для числових значень, таких як час затримки (RTT), використано тип DOUBLE PRECISION, що дозволяє зберігати числа з плаваючою комою високої точності. Для текстових даних використовуються типи VARCHAR та TEXT з відповідними обмеженнями на довжину.

Для стовпців, які є обов'язковими для заповнення, встановлено обмеження NOT NULL. Це гарантує, що при додаванні нового запису відповідні поля не можуть бути порожніми, що сприяє підтримці цілісності даних. Наприклад, час збору даних (timestamp), цільовий хост (target_host), статус (status) в таблиці icmp_stats є обов'язковими.

Для стовпців, що відображають час створення запису (наприклад, first_seen та

last_seen в таблиці agents, timestamp в таблицях статистики та результатів), встановлено значення за замовчуванням CURRENT_TIMESTAMP. Це означає, що якщо при вставці нового запису значення для цих стовпців не буде вказано явно, PostgreSQL автоматично встановить поточний час. Це спрощує процес додавання нових даних та забезпечує точну фіксацію часу подій.

3.1.5 Розробка графічного інтерфейсу

Функціональність GUI реалізована у функції create_gui(). Першим кроком є створення головного вікна застосунку за допомогою конструктора tk.Tk(). Для головного вікна встановлюється заголовок "Network Analysis Tool", який відображається у верхній частині вікна.

Далі у вікні розміщуються чотири основні кнопки, кожна з яких відповідає за запуск певного інструменту аналізу. Кнопка з текстом "Run ICMP Analysis" призначена для запуску скрипту ICMP.py. При натисканні на цю кнопку викликається функція run_icmp(). Аналогічно, кнопка "Run Traceroute MPLS" запускає скрипт traceroute_MPLS.py через виклик функції run_traceroute_mpls(), а кнопка "Run IPv6 NDP Analysis" ініціює виконання скрипту ipv6_ndp.py за допомогою функції run_ipv6_ndp(). Четверта кнопка з текстом "Analysis" відповідає за запуск загального аналізу, представленого скриптом Analyze.py, через функцію run_analyze().

Кожна кнопка створюється за допомогою класу tk.Button(). При створенні кожної кнопки визначаються її батьківське вікно (window), текст, що відображається на кнопці, ширина кнопки (25 символів) та команда, яка буде виконана при натисканні на кнопку. Команда передається через параметр command і пов'язується з відповідною функцією запуску скрипту. Для забезпечення візуального розділення між кнопками та кращого розташування елементів у вікні використовується метод pack() з параметром pady=10, який додає вертикальні відступи у 10 пікселів зверху та знизу кожної кнопки.

Останнім елементом GUI є кнопка "Exit", яка призначена для закриття всього

застосунку. При натисканні на цю кнопку викликається вбудована функція window.quit(), яка завершує головний цикл обробки подій Tkinter і закриває вікно програми. Для цієї кнопки також застосовується метод pack() з параметром pady=20 для більшого вертикального відступу від попередніх елементів.

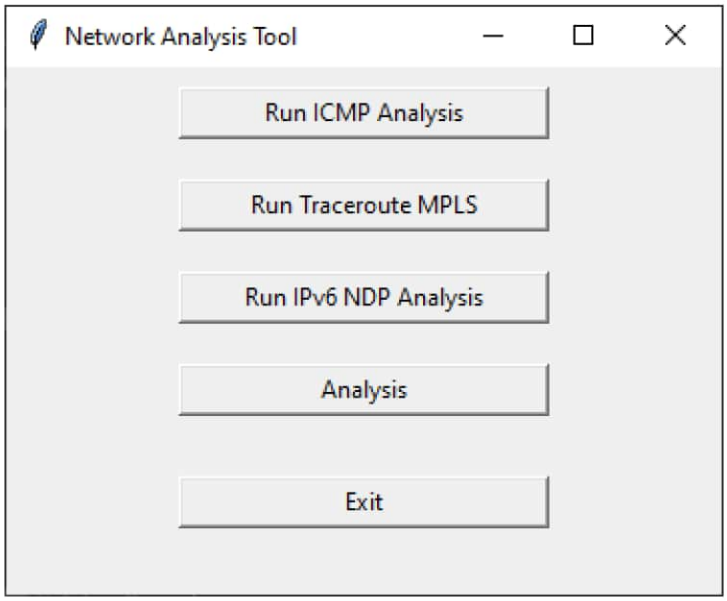

Рисунок 3.9 – Графічний інтерфейс програмного додатку для аналізу та моніторингу мережевих протоколів

Функції run_icmp(), run_traceroute_mpls(), run_ipv6_ndp() та run_analyze() відповідають за фактичний запуск відповідних Python-скриптів як окремих підпроцесів. Для цього використовується модуль subprocess та його функція subprocess.run(). При запуску кожного скрипту вказується команда для виконання (['python', 'назва_скрипту.py']) та параметр check=True, який забезпечує перевірку коду повернення підпроцесу. У випадку успішного виконання скрипту, на екран виводиться інформаційне повідомлення за допомогою messagebox.showinfo(). Якщо під час виконання підпроцесу виникає помилка (код повернення не дорівнює нулю), спрацьовує блок except subprocess.CalledProcessError as e, і на екран виводиться повідомлення про помилку з детальною інформацією за допомогою messagebox.showerror().

Після створення та розміщення всіх елементів інтерфейсу викликається метод window.mainloop(), який запускає головний цикл обробки подій Tkinter. Цей цикл

відповідає за відображення вікна програми, обробку дій користувача (наприклад, натискання кнопок) та оновлення інтерфейсу.

Блок if __name__ == "__main__": гарантує, що функція create_gui() буде викликана лише при безпосередньому запуску скрипту central_module.py. Це запобігає випадковому створенню GUI при імпорті цього скрипту як модуля в іншому файлі.

3.1.6 Тестування роботи програмного додатку

Програмний додаток розроблено для надання користувачам простого та інтуїтивно зрозумілого графічного інтерфейсу, що дозволяє ініціювати різні види аналізу мережевих протоколів. За допомогою набору кнопок, розміщених у головному вікні програми, ви можете легко запускати аналіз протоколу ICMP, трасування маршруту з підтримкою MPLS (багатопротокольної комутації за мітками), аналіз протоколу IPv6 NDP (Neighbor Discovery Protocol), а також ініціювати виконання комплексного загального аналізу мережі.

Після успішного запуску програми на вашому екрані з'явиться головне вікно (рис.3.9). У верхній частині вікна буде відображено заголовок "Network Analysis Tool", що ідентифікує призначення програми. Основну частину вікна займають п'ять великих кнопок, розташованих одна під одною в центрі інтерфейсу. Кожна кнопка має чітке текстове позначення, що вказує на функцію

Run ICMP Analysis – ця кнопка є вашим основним інструментом для запуску процесу аналізу протоколу ICMP. Натискання цієї кнопки ініціює збір та обробку інформації, пов'язаної з обміном ICMP-повідомленнями в мережі.

Якщо необхідно простежити маршрут проходження мережевих пакетів до певної цілі, враховуючи особливості технології MPLS, вам слід скористатися кнопкою Run Traceroute MPLS. Її натискання запустить відповідний інструмент трасування.

Run IPv6 NDP Analysis – для аналізу взаємодії сусідніх вузлів у мережах IPv6, зокрема обміну повідомленнями протоколу NDP, використовуйте цю кнопку. Вона

ініціює процес збору та відображення відповідної інформації.

Analysis – кнопка відповідає за запуск комплексного, загального аналізу мережі. Функціональність, що виконується при її натисканні, може включати одночасний або послідовний запуск кількох видів аналізу або виконання спеціалізованих діагностичних процедур.

Коли ви завершите роботу з інструментом аналізу мережі, ви можете закрити програму, натиснувши на кнопку Exit. Вона відповідає за коректне завершення всіх процесів та закриття головного вікна програми.

Для того щоб ініціювати будь-який з доступних видів аналізу, вам необхідно просто натиснути лівою кнопкою миші на відповідну кнопку, що відображає бажану функцію. Наприклад, якщо вашою метою є аналіз протоколу ICMP, знайдіть кнопку з текстом "Run ICMP Analysis" та натисніть її. Після того як ви натиснете кнопку запуску аналізу, програма розпочне виконання відповідного інструменту у фоновому режимі. Важливо розуміти, що під час активної роботи інструменту аналізу головне вікно програми може тимчасово залишатися неактивним або візуально не змінюватися. Це є нормальним процесом, оскільки основна робота відбувається у окремому процесі. Після того як процес аналізу буде завершено, програма повідомить вас про це, відобразивши на екрані стандартне діалогове вікно з інформаційним повідомленням про успішне завершення операції. Однак, слід мати на увазі, що детальні результати самого аналізу не завжди відображаються безпосередньо в головному вікні цієї програми. Для їхнього перегляду вам може знадобитися звернутися до вікна терміналу (консолі), з якого ви запустили програму, або ж результати можуть бути виведені в окремому вікні, створеному самим інструментом аналізу. Це залежить від того, як саме реалізовані внутрішні скрипти ICMP.py, traceroute_MPLS.py, ipv6_ndp.py та Analyze.py. У випадку, якщо під час виконання будь-якого з обраних видів аналізу виникне непередбачена помилка, програма повідомить вас про це, відобразивши діалогове вікно з повідомленням про помилку. Це вікно також може містити додаткову інформацію про характер помилки та її можливі причини. Уважно ознайомтеся з текстом помилки, оскільки він може надати корисні підказки для діагностики та усунення

проблеми. Після того як ви завершите всі необхідні види аналізу та переглянете їхні результати, ви можете безпечно закрити програму. Для цього знайдіть у головному вікні кнопку з текстом "Exit" та натисніть її. Програма завершить свою роботу та закриє всі відкриті вікна.

На рис.3.10 відображено результат виконання функції "Run ICMP Fragmentation Analysis", яка була запущена через графічний інтерфейс інструменту "Network Analysis Tool". Після натискання відповідної кнопки та успішного завершення аналізу, на екрані з'явилося інформаційне діалогове вікно з заголовком "Info".

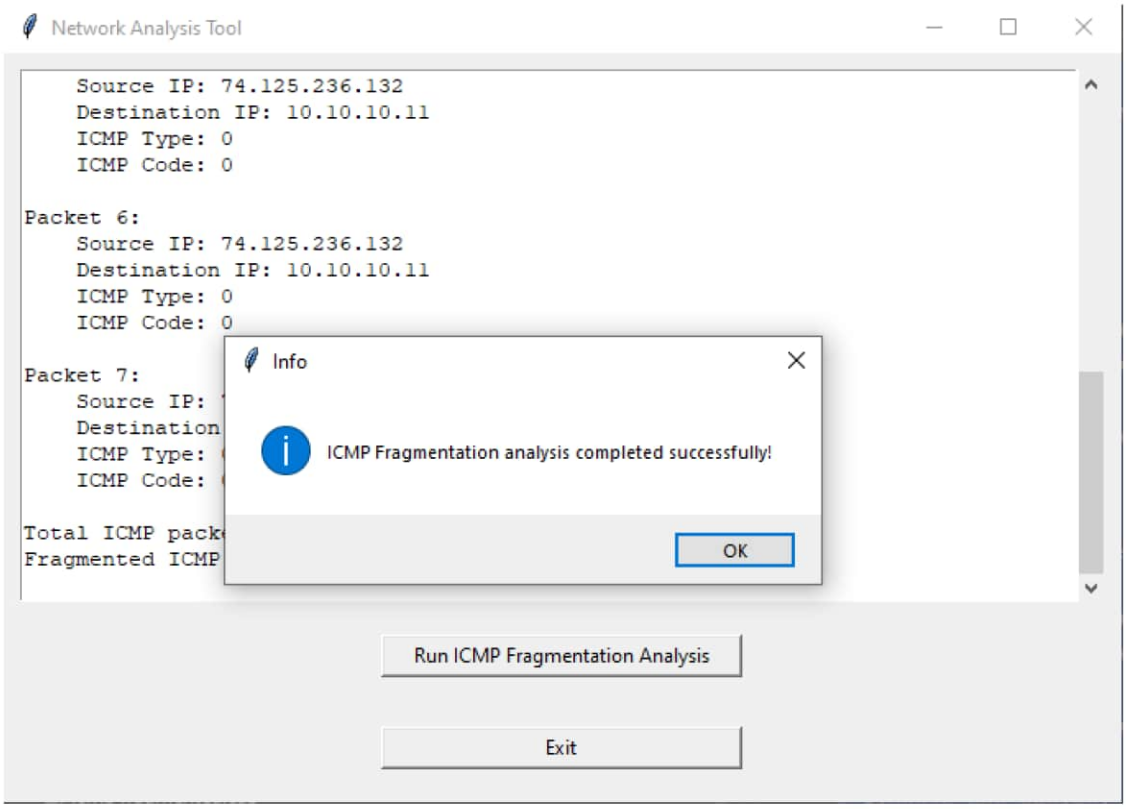

Рисунок 3.10 – Результат виконання функції "Run ICMP Fragmentation Analysis"

У тілі діалогового вікна міститься повідомлення: "ICMP Fragmentation analysis completed successfully!" (Аналіз фрагментації ICMP успішно завершено!). Це повідомлення інформує користувача про те, що процес аналізу ICMP-трафіку на предмет фрагментації завершився без виявлених критичних помилок на рівні виконання програми.

Під головним вікном програми, у текстовому полі з можливістю прокрутки, відображаються деталі проаналізованих ICMP-пакетів. Кожен пакет ідентифікується порядковим номером ("Packet 6:", "Packet 7:", і так далі). Для кожного пакета виводиться наступна інформація:

– Source IP – IP-адреса відправника пакета (наприклад, "74.125.236.132");

– Destination IP – IP-адреса отримувача пакета (наприклад, "10.10.10.11");

– ICMP Type – тип ICMP-повідомлення (наприклад, "0"). Тип 0 зазвичай відповідає ICMP Echo Reply (відповідь на пінг);

– ICMP Code – код ICMP-повідомлення (наприклад, "0"). Код 0 для Echo Reply вказує на успішну відповідь.

Крім детальної інформації про окремі пакети, в кінці текстового поля наведена загальна статистика:

– Total ICMP packets – загальна кількість проаналізованих ICMP-пакетів;

– Fragmented ICMP – кількість виявлених фрагментованих ICMP-пакетів. (На зображенні ця частина тексту частково прихована).

На рис.3.11 відображено частину результату виконання функції "Run Traceroute MPLS" у вікні "Network Analysis Tool". Після натискання відповідної кнопки, програма, ймовірно, запустила процес трасування маршруту до вказаної цілі з урахуванням особливостей MPLS.

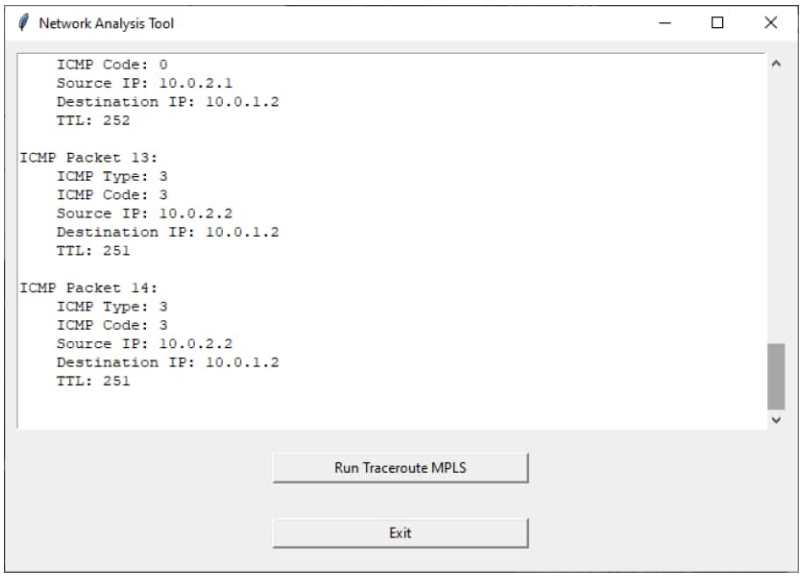

Рисунок 3.11 – Результат виконання функції " Run Traceroute MPLS"

У текстовому полі з можливістю прокрутки відображаються деталі проаналізованих ICMP-пакетів, які є типовою складовою частиною процесу Traceroute. Кожен пакет ідентифікується порядковим номером ("ICMP Packet 12:", "ICMP Packet 13:", "ICMP Packet 14:"). Для кожного пакета виводиться наступна інформація:

– ICMP Type – тип ICMP-повідомлення;

– для "ICMP Packet 12:" – тип не вказано на цій частині зображення. Однак, присутні "ICMP Code: 0", "Source IP: 10.0.2.1", "Destination IP: 10.0.1.2" та "TTL: 252";

– для "ICMP Packet 13:" та "ICMP Packet 14:" вказано "ICMP Type: 3". Тип 3 відповідає ICMP Destination Unreachable (неможливо досягти ціль);

– ICMP Code – код ICMP-повідомлення;

– для "ICMP Packet 12:" вказано "ICMP Code: 0";

– для "ICMP Packet 13:" та "ICMP Packet 14:" вказано "ICMP Code: 3". Код 3 для Destination Unreachable зазвичай вказує на "Port Unreachable" (порт недоступний).

– Source IP – IP-адреса відправника пакета;

– для "ICMP Packet 12:" вказано "Source IP: 10.0.2.1";

– для "ICMP Packet 13:" та "ICMP Packet 14:" вказано "Source IP: 10.0.2.2".

– Destination IP: IP-адреса отримувача пакета. Для всіх відображених пакетів вказано "Destination IP: 10.0.1.2";

– TTL – значення Time-To-Live (час життя) IP-пакета;

– для "ICMP Packet 12:" вказано "TTL: 252";

– для "ICMP Packet 13:" та "ICMP Packet 14:" вказано "TTL: 251".

Інтерпретація результату:

– Відображені ICMP-пакети є частиною результату виконання команди Traceroute до цільової IP-адреси 10.0.1.2.

– ICMP Packet 12 – цей пакет, з вихідною IP-адресою 10.0.2.1 та TTL 252, міг бути одним із зондувальних пакетів, відправлених інструментом Traceroute.

Відсутність явного типу може означати, що це був вихідний UDP або ICMP Echo запит;

– ICMP Packet 13 та ICMP Packet 14 – ці пакети є ICMP Destination Unreachable (Type 3, Code 3 - Port Unreachable) повідомленнями, які були надіслані з IP-адреси 10.0.2.2 у відповідь на зондувальні пакети, надіслані до 10.0.1.2. Отримання "Port Unreachable" зазвичай вказує на те, що трасування досягло цільового хоста, оскільки цільовий хост відповідає таким ICMP-повідомленням, коли отримує UDP-пакет на закритий порт, який зазвичай використовується Traceroute. Зменшення значення TTL від 252 до 251 свідчить про проходження одного мережевого вузла.

На рис.3.12 представлено результат успішного виконання функції "Run IPv6 NDP Analysis" в рамках інструменту "Network Analysis Tool". Після ініціювання аналізу протоколу IPv6 Neighbor Discovery Protocol (NDP) та його завершення без виявлених помилок, користувачеві відображається інформаційне вікно "Info" з підтвердженням "IPv6 NDP analysis completed successfully!".

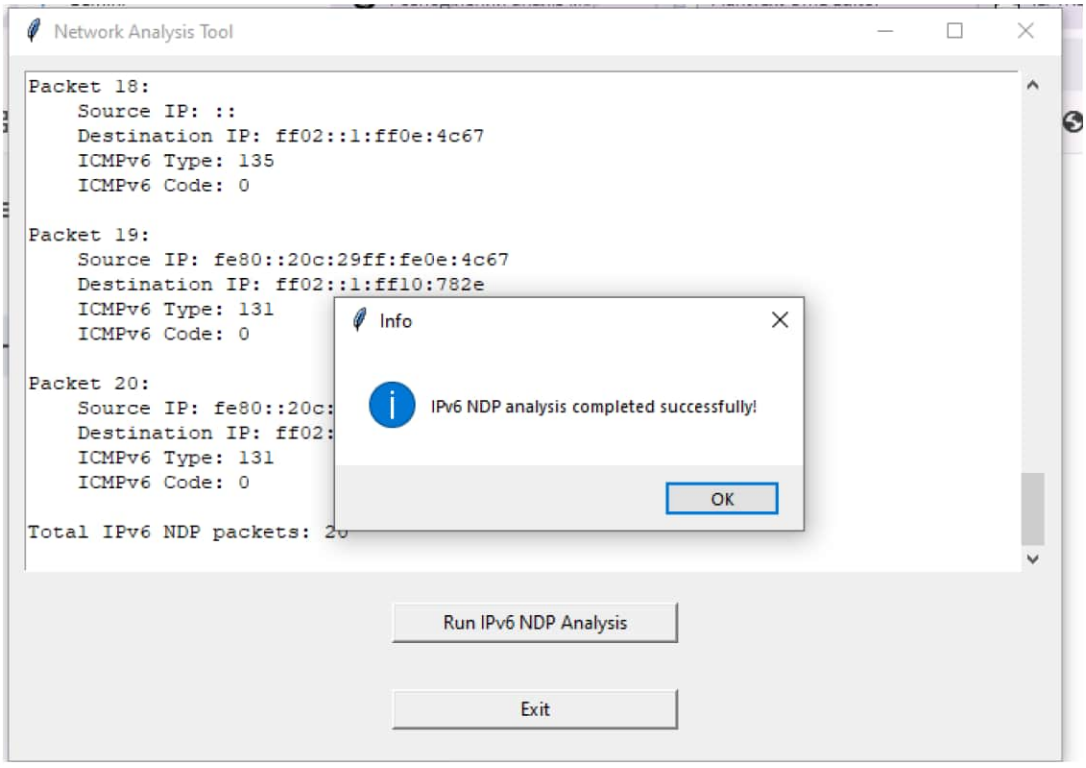

Рисунок 3.12 – Результат виконання функції "Run IPv6 NDP Analysis "

Під цим повідомленням, у головному вікні програми, в текстовому полі з вертикальною прокруткою, відображаються деталі захоплених та проаналізованих NDP-пакетів. Кожен пакет ідентифікується порядковим номером, починаючи з "Packet 18:". Для кожного зафіксованого пакету надається наступна ключова інформація:

– Source IP – IPv6-адреса вузла, що відправив даний NDP-пакет. Прикладами є "::" (невизначена адреса) та локальна link-local адреса "fe80::20c:29ff:fe0e:4c67";

– Destination IP – IPv6-адреса вузла або групи вузлів, яким призначений даний NDP-пакет. Прикладами є multicast-адреси "ff02::1:ff0e:4c67" та "ff02::1:ff10:782e", а також "ff02::1:ff0e:4c67". Варто зазначити, що адреси, які починаються з префікса "ff02::", є multicast-адресами, що використовуються для широкомовних розсилок у локальному сегменті мережі IPv6;

– ICMPv6 Type – числовий код, що вказує на тип ICMPv6-повідомлення. Для "Packet 18:" зафіксовано тип "135", який відповідає Neighbor Solicitation (запит сусіда). Для "Packet 19:" та "Packet 20:" відображається тип "131", що означає Router Solicitation (запит маршрутизатора);

– ICMPv6 Code – додатковий код, що надає більш детальну інформацію про конкретне ICMPv6-повідомлення. У всіх відображених пакетах значення цього поля становить "0".

У нижній частині текстового виводу наведена загальна статистика щодо кількості оброблених NDP-пакетів: "Total IPv6 NDP packets:". На зображенні видно, що загальна кількість проаналізованих пакетів становить "2".

Аналізуючи представлені дані, можна зробити наступні висновки: зафіксовані пакети є типовими для протоколу IPv6 NDP, який відіграє важливу роль у виявленні сусідніх вузлів та маршрутизаторів у локальній мережі IPv6. "Packet 18" являє собою запит Neighbor Solicitation, надісланий для визначення MAC-адреси пристрою з певною IPv6-адресою, що закінчується на ":ff0e:4c67". "Packet 19" та "Packet 20" є запитами Router Solicitation, які хости надсилають на multicast-адресу всіх маршрутизаторів ("ff02::2") з метою виявлення доступних маршрутизаторів у своєму

сегменті мережі.

Таким чином, результат виконання "Run IPv6 NDP Analysis" надає користувачеві цінну інформацію про обмін NDP-повідомленнями в досліджуваній мережі IPv6, включаючи типи пакетів, їхніх відправників та отримувачів. Ці дані можуть бути використані для розуміння топології мережі IPv6, діагностики проблем зі зв'язком між вузлами та виявлення некоректних конфігурацій сусідів або маршрутизаторів. Загальна кількість зафіксованих NDP-пакетів дає уявлення про рівень активності цього протоколу в мережі на момент збору даних.

На рис.3.13 представлено графік, який є результатом виконання функції "Analysis" у вікні "Network Analysis Tool". Графік відображається в окремому вікні під назвою "Figure 1" та має заголовок "Distribution of ICMP Types" (Розподіл типів ICMP).

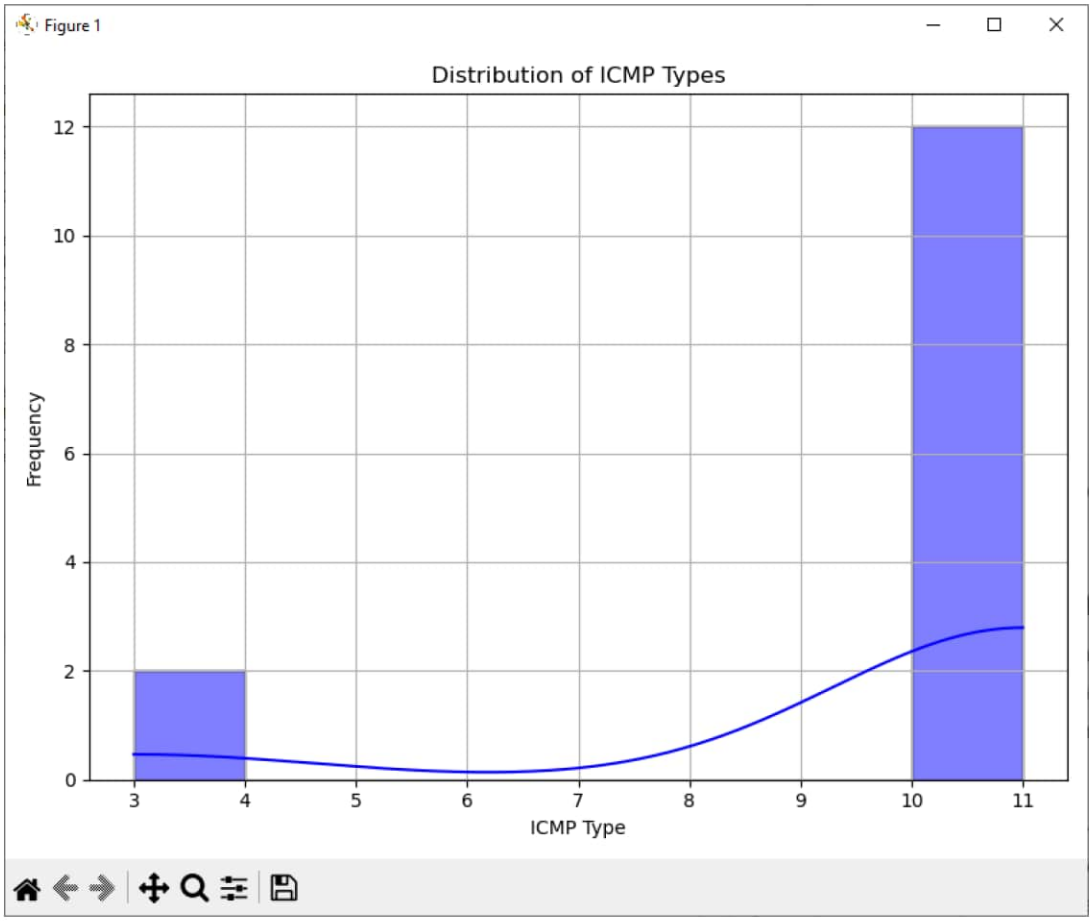

Рисунок 3.13 – Графік розподілу типів ICMP

Графік є гістограмою, де по горизонтальній осі (ICMP Type) відкладено різні

типи ICMP-повідомлень, а по вертикальній осі (Frequency) - частота їхньої появи у проаналізованому мережевому трафіку.

На гістограмі чітко видно два основних стовпці (бара): стовпець у районі значення 3-4 по осі ICMP Type: Цей стовпець досягає висоти приблизно 2 по осі Frequency. Це означає, що ICMP-повідомлення з типом 3 (Destination Unreachable - призначення недосяжне) зустрічалися у проаналізованому трафіку 2 рази. Стовпець у районі значення 10-11 по осі ICMP Type: Цей стовпець є значно вищим і досягає висоти 12 по осі Frequency. Це вказує на те, що ICMP-повідомлення з типом 11 (Time Exceeded - час життя пакета вичерпано) були зафіксовані у проаналізованому трафіку 12 разів.

Крім гістограми, на графіку також присутня синя крива, яка плавно проходить через основу стовпців. Ця крива є лінією щільності ймовірності (probability density function - PDF), яка намагається змоделювати розподіл частот ICMP-типів. Вона показує загальну тенденцію розподілу, навіть між дискретними значеннями типів.

Рисунок 3.14 демонструє ще один графік, який є результатом виконання функції "Analysis" у вікні "Network Analysis Tool". Цей графік також відображається в окремому вікні під назвою "Figure 1", але цього разу має заголовок "Distribution of TTL Values" (Розподіл значень TTL).

Графік є гістограмою, де по горизонтальній осі (TTL) відкладено різні значення Time-To-Live (час життя) IP-пакетів, а по вертикальній осі (Frequency) - частота їхньої появи у проаналізованому мережевому трафіку.

На гістограмі відображено кілька стовпців (барів) червоного кольору, кожен з яких відповідає певному діапазону значень TTL:

– стовпець у районі значення 248-250 по осі TTL: Цей стовпець досягає висоти приблизно 3.0 по осі Frequency. Це означає, що пакети зі значеннями TTL у цьому діапазоні зустрічалися у проаналізованому трафіку 3 рази;

– стовпець у районі значення 251-252 по осі TTL: Цей стовпець досягає висоти приблизно 2.0 по осі Frequency. Це вказує на те, що пакети зі значеннями TTL у цьому діапазоні були зафіксовані у проаналізованому трафіку 2 рази;

– стовпець у районі значення 254-255 по осі TTL: Цей стовпець також

досягає висоти приблизно 3.0 по осі Frequency. Це означає, що пакети зі значеннями TTL у цьому діапазоні зустрічалися у проаналізованому трафіку 3 рази.

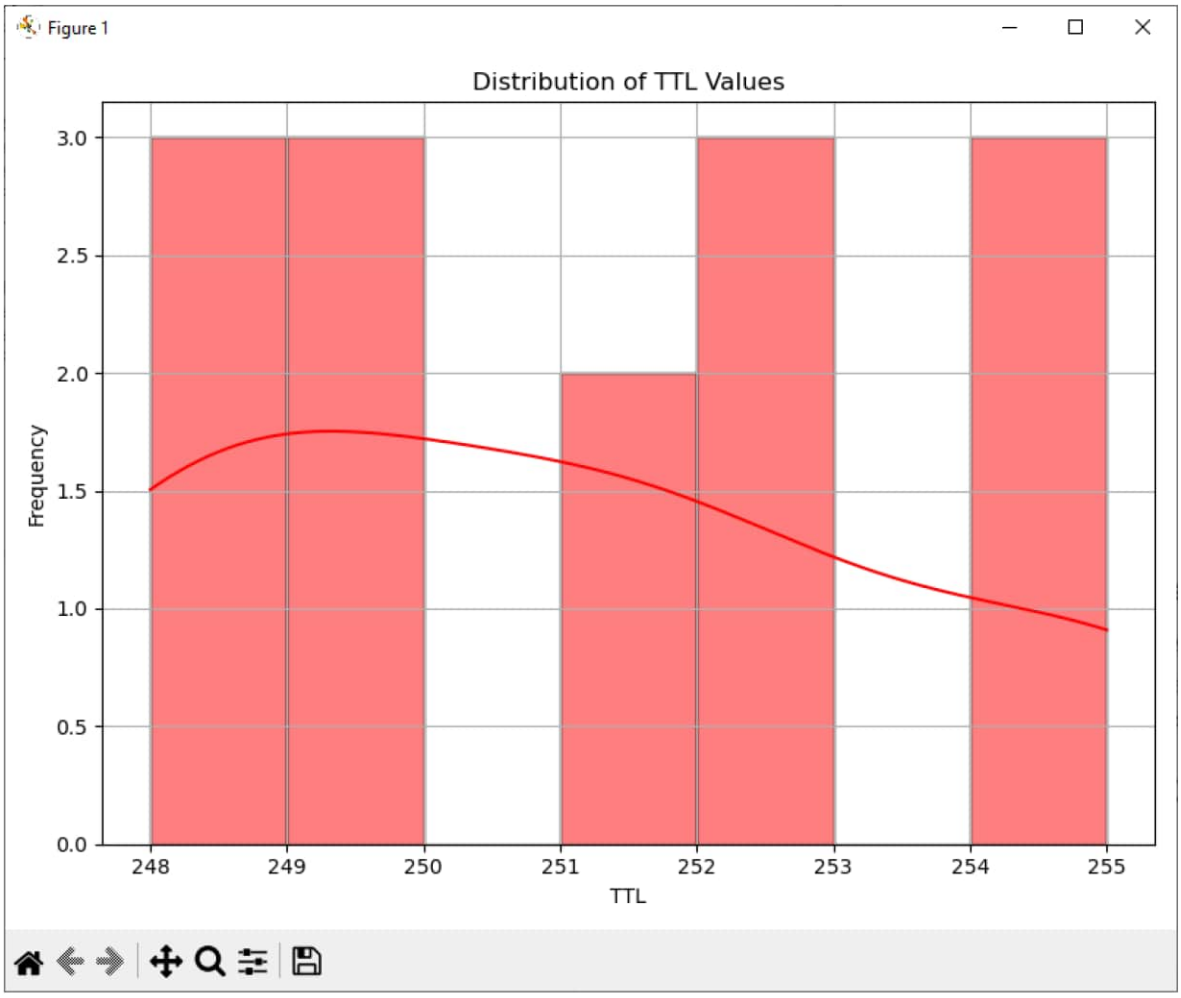

Рисунок 3.14 – Графік розподілу значень TTL

Крім гістограми, на графіку присутня червона крива, яка плавно проходить через верхню частину стовпців. Ця крива, ймовірно, є лінією щільності ймовірності (probability density function - PDF), яка намагається змоделювати розподіл частот значень TTL. Вона показує загальну тенденцію розподілу значень TTL у проаналізованому трафіку.

Рис.3.15 представляє ще один графік, отриманий в результаті виконання функції "Analysis" у вікні "Network Analysis Tool". Цей графік, відображений у вікні "Figure 1", має заголовок "Distribution of ICMP Codes" (Розподіл кодів ICMP).

Графік є гістограмою, де по горизонтальній осі (ICMP Code) відкладено різні коди ICMP-повідомлень, а по вертикальній осі (Frequency) - частота їхньої появи у проаналізованому мережевому трафіку.

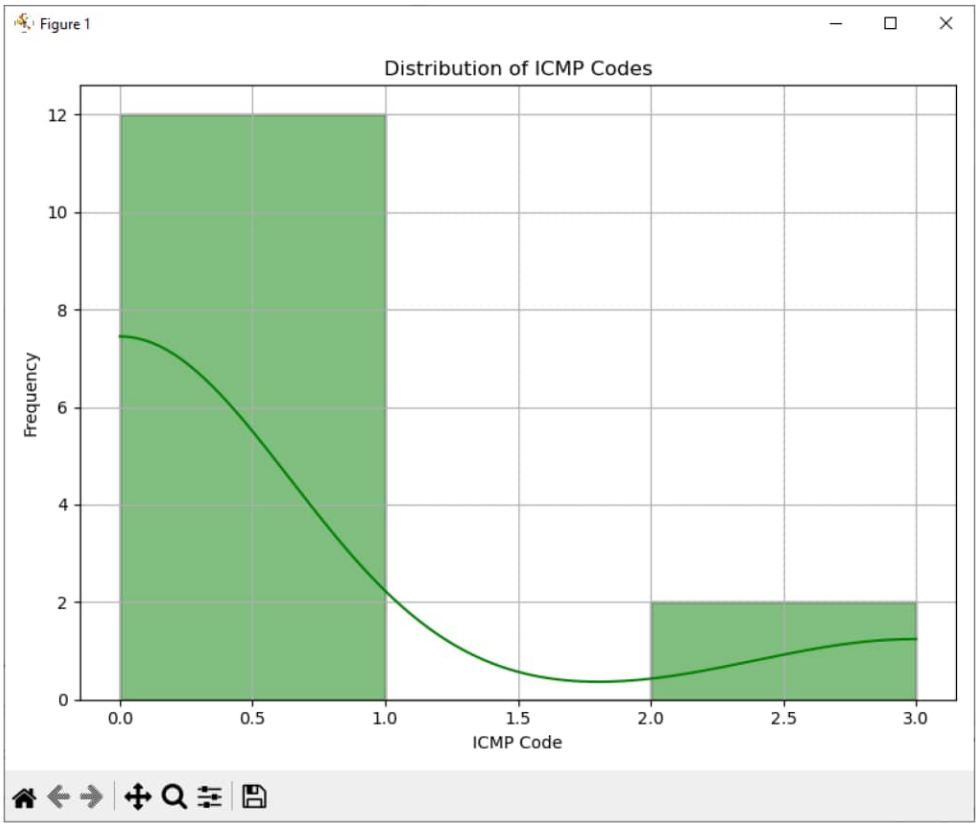

Рисунок 3.15 – Графік розподілу кодів ICMP

На гістограмі чітко видно два основних стовпці (бари) зеленого кольору: стовпець у районі значення 0 по осі ICMP Code: Цей стовпець є найвищим і досягає висоти приблизно 12.0 по осі Frequency. Це означає, що ICMP-повідомлення з кодом 0 були зафіксовані у проаналізованому трафіку 12 разів. Стовпець у районі значення 2-3 по осі ICMP Code: Цей стовпець значно нижчий і досягає висоти приблизно 2.0 по осі Frequency. Це вказує на те, що ICMP-повідомлення з кодом у діапазоні 2-3 зустрічалися у проаналізованому трафіку 2 рази.

Крім гістограми, на графіку присутня зелена крива, яка плавно проходить через верхню частину стовпців. Ця крива є лінією щільності ймовірності (probability density function - PDF), що моделює розподіл частот кодів ICMP. Вона показує загальну тенденцію розподілу, навіть між дискретними значеннями кодів.

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено розподілений програмний додаток, призначений для аналізу та моніторингу ключових мережевих протоколів: ICMP, Traceroute та IPv6 NDP. Метою роботи було створення функціональної системи, яка б дозволяла здійснювати збір даних про стан мережі за допомогою агентів моніторингу, їхню централізовану обробку та візуалізацію для надання користувачеві інформації про працездатність та топологію мережі.

Розроблена архітектура включає в себе розподілених агентів моніторингу, відповідальних за періодичний збір даних за визначеними протоколами, та центральний модуль, який приймає, зберігає та надає ці дані для аналізу та візуалізації. Агенти моніторингу, реалізовані на мові Python, здатні виконувати пінг-тестування (ICMP), трасування маршруту (Traceroute) та аналіз сусідів IPv6 (NDP). Для забезпечення ефективної передачі даних між агентами та центральним модулем було використано протокол HTTP з форматом даних JSON.

Основний модуль, також розроблений на базі фреймворку Flask на мові Python, забезпечує RESTful API для прийому даних від агентів та надання інформації про зареєстровані агенти та зібрані ними метрики. Для зберігання отриманих даних було спроектовано реляційну базу даних PostgreSQL зі структурованими таблицями для кожного типу моніторингових даних та інформації про агентів. Схема бази даних забезпечує цілісність даних та ефективність виконання запитів.

Для забезпечення базової взаємодії з користувачем було розроблено простий графічний інтерфейс на основі бібліотеки Tkinter. Цей інтерфейс дозволяє ініціювати локальний аналіз PCAP-файлів за протоколами ICMP, Traceroute та IPv6 NDP, надаючи користувачеві можливість переглядати деталі пакетів та основні статистичні дані. Хоча розроблений GUI є базовим, він демонструє потенціал для подальшого розвитку у повноцінний інтерфейс центрального модуля з візуалізацією даних, отриманих від розподілених агентів.

Проведене тестування окремих компонентів розробленої системи підтвердило їхню працездатність. Агенти моніторингу успішно збирають дані за визначеними протоколами, а основний модуль коректно приймає та зберігає цю інформацію. Базовий графічний інтерфейс дозволяє запускати локальний аналіз мережевих даних.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ijomah, Tochukwu & Eyo-Udo, Nsisong & Anjorin, Kikelomo. (2024). The role of big data analytics in customer relationship management: Strategies for improving customer engagement and retention. World Journal of Advanced Science and Technology.

2. Bock L. Learn Wireshark: A definitive guide to expertly analyzing protocols and troubleshooting networks using Wireshark, Packt Publishing, 2022, 606 p.

3. S. Ali, A. Ashraf, S. B. Qaisar, M. Kamran Afridi, H. Saeed, S. Rashid, E. A. Felemban, and A. A. Sheikh, “Simplimote: A wireless sensor network monitoring platform for oil and gas pipelines,” IEEE Systems Journal, vol. 12, no. 1, pp. 778–789, 2018.

4. Observium documentation. [Електронний ресурс]. – Режим доступу: docs.observium.org.

5. A. Azzouni, R. Boutaba, N. T. M. Trang, and G. Pujolle, “sOFTDP: Secure and efficient openflow topology discovery protocol,” in Proceedings of the IEEE/IFIP Network Operations and Management Symposium, 2018, pp. 1–7..

6. M. H. Behringer, B. E. Carpenter, T. Eckert, L. Ciavaglia, and J. C. Nobre, “A Reference Model for Autonomic Networking,” no. 8993, 2021.

7. C. Bormann, B. E. Carpenter, and B. Liu, “GeneRic Autonomic Signaling Protocol (GRASP),” RFC 8990, 2021.

8. ISO. Online browsing platform [Електронний ресурс]: – Режим доступу: https://www.iso.org/obp/ui/#iso:std:iso:31000:ed-2:v1:en

9. Огляд сучасних інструментів аналізу мережевого трафіку capture the packets [Електронний ресурс] - Режим доступу: https://www.ispras.ru/preprints/docs

10. B. E. Carpenter, M. C. Richardson, F. Toerless Eckert, J. C. Nobre, S. Jiang, Y. Li, and C. Bormann, “Autonomic networking gets serious,” in The Internet Protocol Journal, vol. 24, no. 3. IPJ, 2021, pp. 2–19.

11. L. Ochoa-Aday, C. Cervello-Pastor, and A. Fernandez-Fernandez, “Self-healing topology discovery protocol for software-defined networks,” IEEE Communications Letters, vol. 22, no. 5, pp. 1070–1073, 2018.

12. A. Zacharis, S. V. Margariti, E. Stergiou, and C. Angelis, “Performance evaluation of topology discovery protocols in software defined networks,” Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 135–140, 2021.

13. Widman J. The Most Popular Programming Languages of 2019 [Електронний ресурс] / Jake Widman // New Relic. – 2019. – Режим доступу до ресурсу: https://blog.newrelic.com/technology/most-popular-programminglanguages-of-2019.

14. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org.

ДОДАТОК А

Текст програми розподіленого програмного додатку для аналізу та моніторингу мережевих протоколів ICMP, Traceroute та IPv6 NDP

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

РОЗПОДІЛЕНИЙ ПРОГРАМНИЙ ДОДАТОК ДЛЯ АНАЛІЗУ ТА МОНІТОРИНГУ МЕРЕЖЕВИХ ПРОТОКОЛІВ ICMP, TRACEROUTE ТА IPV6 NDP

Текст програми

804.02070743.25018-01 12 01

Листів 10

2025

АНОТАЦІЯ

Дана програма представляє собою програмний модуль "Network Analysis Tool", що забезпечує комплексний аналіз та моніторинг ключових мережевих протоколів: ICMP, Traceroute та IPv6 NDP.

Програма призначена для збору, обробки та візуалізації даних, отриманих за допомогою зазначених протоколів. Вона дозволяє користувачам запускати окремі модулі для аналізу ICMP-пакетів, дослідження маршрутів за допомогою Traceroute (включаючи MPLS), а також виявлення та аналізу IPv6 NDP-повідомлень. Це сприяє покращенню розуміння стану мережі, швидкому виявленню проблем та оптимізації її функціонування, зокрема в умовах зростання використання протоколу IPv6.

Програма розроблена як десктопний додаток на Python з використанням бібліотеки Tkinter, що забезпечує її автономну роботу та стабільність. Вона може бути використана для навчальних цілей, досліджень у галузі мережевого адміністрування та безпеки, а також для діагностики та оптимізації роботи комп'ютерних мереж.

ЗМІСТ

Analyze.py

```
import pyshark
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from tkinter import filedialog, messagebox, scrolledtext
import tkinter as tk


# Функція для витягування інформації про ICMP пакет
def extract_icmp_info(packet):
    icmp_info = {}
    icmp_info['Type'] = int(packet.icmp.type)
    icmp_info['Code'] = int(packet.icmp.code)
    icmp_info['Source IP'] = packet.ip.src
    icmp_info['Destination IP'] = packet.ip.dst
    icmp_info['TTL'] = int(packet.ip.ttl) if 'IP' in packet else None
    return icmp_info


# Функція для виведення інформації про ICMP пакет
def print_icmp_info(icmp_info, result_text_widget):
    result_text_widget.insert(tk.END, f"    ICMP Type: {icmp_info['Type']}\n")
    result_text_widget.insert(tk.END, f"    ICMP Code: {icmp_info['Code']}\n")
    result_text_widget.insert(tk.END, f"    Source IP: {icmp_info['Source IP']}\n")
    result_text_widget.insert(tk.END,  f"      Destination IP: {icmp_info['Destination
IP']}\n")
    if icmp_info['TTL'] is not None:
        result_text_widget.insert(tk.END, f"    TTL: {icmp_info['TTL']}\n")
```

```
# Функція для аналізу ICMP пакетів та збору статистики
def analyze_icmp(capture_file, result_text_widget):
    capture = pyshark.FileCapture(capture_file)

    icmp_packets = 0
    icmp_info_list = []
    source_ips = []
    destination_ips = []

    result_text_widget.insert(tk.END, "ICMP Packets Details:\n")
    result_text_widget.insert(tk.END, "----------------------\n")

    for packet in capture:
        if "ICMP" in packet:
            icmp_packets += 1
            # Витягування інформації про ICMP пакет
            icmp_info = extract_icmp_info(packet)
            icmp_info_list.append(icmp_info)  # Додавання інформації до списку
            source_ips.append(icmp_info['Source IP'])
            destination_ips.append(icmp_info['Destination IP'])
            result_text_widget.insert(tk.END, f"ICMP Packet {icmp_packets}:\n")
            print_icmp_info(icmp_info, result_text_widget)
            result_text_widget.insert(tk.END, "\n")

    # Закриття файлу захоплення
    capture.close()

    # Підготовка статистики
    icmp_types = [info['Type'] for info in icmp_info_list]
```

```
        icmp_codes = [info['Code'] for info in icmp_info_list]
        ttl_values = [info['TTL'] for info in icmp_info_list if info['TTL'] is not None]

        icmp_df = pd.DataFrame({
            'Type': icmp_types,
            'Code': icmp_codes,
            'TTL': ttl_values,
            'Source IP': source_ips,
            'Destination IP': destination_ips
        })

        # Виведення статистики
        result_text_widget.insert(tk.END, "\nSummary Statistics:\n")
        result_text_widget.insert(tk.END, f"Total ICMP packets: {icmp_packets}\n")
        result_text_widget.insert(tk.END, f"Unique source IPs: {len(set(source_ips))}\n")
        result_text_widget.insert(tk.END, f"Unique destination IPs: {len(set(destination_ips))}\n")
        result_text_widget.insert(tk.END, "\n")

        # Генерація візуалізацій
        # Побудова гістограми для типів ICMP
        plt.figure(figsize=(8, 6))
        sns.histplot(icmp_types, kde=True, color='blue', bins=range(min(icmp_types), max(icmp_types) + 1))
        plt.title("Distribution of ICMP Types")
        plt.xlabel("ICMP Type")
        plt.ylabel("Frequency")
        plt.grid(True)
        plt.tight_layout()
        plt.show()
```

```
        # Побудова гістограми для кодів ICMP
        plt.figure(figsize=(8, 6))
        sns.histplot(icmp_codes, kde=True, color='green', bins=range(min(icmp_codes), max(icmp_codes) + 1))
        plt.title("Distribution of ICMP Codes")
        plt.xlabel('ICMP Code')
        plt.ylabel('Frequency')
        plt.grid(True)
        plt.tight_layout()
        plt.show()

        # Побудова гістограми для TTL значень
        plt.figure(figsize=(8, 6))
        sns.histplot(ttl_values, kde=True, color='red', bins=range(min(ttl_values), max(ttl_values) + 1))
        plt.title('Distribution of TTL Values')
        plt.xlabel('TTL')
        plt.ylabel('Frequency')
        plt.grid(True)
        plt.tight_layout()
        plt.show()

        # Зауваження про протокол
        result_text_widget.insert(tk.END, "Зауваження щодо поведінки протоколу:\n")
        result_text_widget.insert(tk.END, "1. Пакети ICMP були успішно захоплені.\n")
        result_text_widget.insert(tk.END,
                                  "2. Значення типу та коду ICMP змінюються, що вказує на різні типи ICMP повідомлень (наприклад, Echo Request, Echo Reply).\n")
```

```
        result_text_widget.insert(tk.END, "3. Значення TTL дають уявлення про мережеві хопи та переміщення пакетів.\n")
        result_text_widget.insert(tk.END,
                                  "4. Рекомендується моніторити високу частоту Echo Request для виявлення потенційних мережевих проблем чи атак.\n")
        result_text_widget.insert(tk.END, "\n")


    # Функція для запуску аналізу ICMP
    def run_icmp_analysis(result_text_widget):
        try:
            # Вибір файлу для аналізу
            file_path = filedialog.askopenfilename(title="Select Capture File",
                                                   filetypes=[("PCAP files", "*.cap"), ("All files", "*.*")])
            if file_path:
                analyze_icmp(file_path, result_text_widget)
                messagebox.showinfo("Info", "ICMP analysis completed successfully!")
            else:
                messagebox.showwarning("Warning", "No file selected!")
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred while running ICMP analysis: {e}")


    # Графічний інтерфейс
    def create_gui():
        window = tk.Tk()
        window.title("Network Analysis Tool")
```

ICMP.py

```
import pyshark
import tkinter as tk
from tkinter import messagebox, scrolledtext, filedialog

# функція для аналізу фрагментованих icmp пакетів
def analyze_icmp_fragments(capture_file, result_text_widget):
    capture = pyshark.filecapture(capture_file)

    total_icmp_packets = 0
    fragmented_icmp_packets = 0

    result_text_widget.insert(tk.end, "icmp packets details:\n")
    result_text_widget.insert(tk.end, "----------------------\n")

    for packet in capture:
        # перевірка чи є пакет icmp
        if "icmp" in packet:
            total_icmp_packets += 1
            icmp_type = packet.icmp.type
            icmp_code = packet.icmp.code
            src_ip = packet.ip.src
            dst_ip = packet.ip.dst
            result_text_widget.insert(tk.end, f"packet {total_icmp_packets}:\n")
            result_text_widget.insert(tk.end, f"    source ip: {src_ip}\n")
            result_text_widget.insert(tk.end, f"    destination ip: {dst_ip}\n")
            result_text_widget.insert(tk.end, f"    icmp type: {icmp_type}\n")
            result_text_widget.insert(tk.end, f"    icmp code: {icmp_code}\n")
            # перевірка на фрагментацію пакету
            try:
```

```
                if packet.icmp.fragment_count:
                    fragmented_icmp_packets += 1
                    result_text_widget.insert(tk.end, "    fragmented: yes\n")
            except attributeerror:
                pass
            result_text_widget.insert(tk.end, "\n")

        # закриття файлу захоплення
        capture.close()

        # виведення статистики
        result_text_widget.insert(tk.end, f"total icmp packets: {total_icmp_packets}\n")
        result_text_widget.insert(tk.end, f"fragmented icmp packets: {fragmented_icmp_packets}\n")
        result_text_widget.yview(tk.end)  # прокрутка до кінця

    # функція для запуску аналізу icmp фрагментів
    def run_icmp_analysis(result_text_widget):
        try:
            # вибір файлу для аналізу
            file_path = filedialog.askopenfilename(title="select capture file", filetypes=[("pcap files", "*.cap"), ("all files", "*.*")])
            if file_path:
                analyze_icmp_fragments(file_path, result_text_widget)
                messagebox.showinfo("info", "icmp fragmentation analysis completed successfully!")
            else:
                messagebox.showwarning("warning", "no file selected!")
        except exception as e:
            messagebox.showerror("error", f"an error occurred while running icmp
```

```
analysis: {e}")

    # графічний інтерфейс
    def create_gui():
        window = tk.tk()
        window.title("network analysis tool")

        # текстове поле для виведення результатів аналізу
        result_text_widget = scrolledtext.scrolledtext(window, width=80, height=20, wrap=tk.word)
        result_text_widget.pack(padx=10, pady=10)

        # налаштування кнопки для запуску аналізу icmp фрагментації
        btn_icmp_analysis = tk.button(window, text="run icmp fragmentation analysis", width=30, command=lambda: run_icmp_analysis(result_text_widget))
        btn_icmp_analysis.pack(pady=10)
```