Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Система моніторингу та аналізу мережевого трафіку»

08-54.БКР.026.00.000 ПЗ

Виконав: студент 4 курсу, групи 2СП-21б

спеціальності 123 — «Комп'ютерна інженерія»

освітня програма — «Системне програмування»

Данчук Д. О.

Керівник: к.т.н., доц. каф. ОТ

Тарновський М. Г.

Рецензент к.т.н., доц. каф. ПЗ

Романюк О. В.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.

«18» 06 2025 р.

Вінниця ВНТУ — 2025 рік

Вінницький національний технічний університет
Факультет Інформаційних технологій та комп'ютерної інженерії
Кафедра Обчислювальної техніки
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 - Інформаційні технології
Спеціальність 123 - «Комп'ютерна інженерія»
Освітня програма «Системне програмування»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Данчуку Дмитру Олександовичу

1 Тема роботи: «Система моніторингу та аналізу мережевого трафіку», керівник роботи Тарновський Микола Геннадійович, к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу від «20» березня 2025 року №97.

2 Термін подання студентом роботи 13.05.2025.

3. Вихідні дані до роботи: спіцифікація з протоколів сімества TCP/IP; операційна система — Windows, технології розробки — мова програмування C#, платформа .NET Framework, графічна підсистема WPF, шаблон MVVM, бібліотека для побудови графіків та діаграм LiveCharts2, середовище розробки — Visual Studio Community.

4 Зміст розрахунково-пояснювальної записки вступ, аналіз предметної області, аналіз існуючих рішень, вибір методу перехоплення та аналізу пакетів даних, розробка системи моніторингу та аналізу висновки.

5 Перелік графічного матеріалу: блок-схема архітектури програми, блок-схема алгоритму захоплення пакетів, інтерфейс головного вікна.

6 Консультанти розділів роботи приведений в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	к.т.н., доц. каф. ОТ Тарновський М. Г.	21.03.25	13.05.25
Нормоконтроль	Швець С. І.	14.05.25	30.05.25

7. Дата видачі завдання 21.03.2025 р.

8. Календарний план приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання	Підпис
1	Постановка задачі роботи	21.03.25	
2	Пошук матеріалів та інформаційних джерел	21.03.25— 30.03.25	
3	Аналіз існуючих прототипів та визначення вимог до створюваної системи	21.03.25— 30.03.25	
4	Обґрунтування структури та принципів роботи системи	31.03.25— 13.04.25	
5	Розробка й тестування апаратно-програмної системи	14.04.25— 27.04.25	
6	Оформлення пояснювальної записки та ілюстративного матеріалу	28.04.25— 11.05.25	
7	Перевірка якості виконання бакалаврської кваліфікаційної роботи та усунення недоліків	13.05.25	

Студент Дмитро ДАНЧУК

Керівник роботи к.т.н., доц. Микола ТАРНОВСЬКИЙ

АНОТАЦІЯ

Данчук Д.О. апаратно-програмні засоби системи «Моніторингу та аналізу мережевого трафіку». Бакалаврська кваліфікаційна робота зі спеціальності 123 — Системне програмування, Вінниця: ВНТУ, 2025. 73 с.

На укр. мові. Бібліогр.: 19 назв, рис.: 30, табл. 3.

В бакалаврській роботі розроблено апаратні та програмні засоби системи «Моніторингу та аналізу мережевого трафіку». На основі обґрунтування доцільності цієї теми здійснений аналіз існуючих рішень, їх переваги та недоліки, розглянуто поширені мережеві стандарти передавання інформації. Відповідно до поставленої мети спроектовано систему «Моніторингу та аналізу мережевого трафіку», шляхом модернізації методів захоплення та аналізу інтернет пакетів, використання апаратних та програмних засобів.

Ключові слова: моніторинг, аналіз, рівні мережевого трафіку, пакет, у реальному часі.

ABSTRACT

Danchuk, D.O. Hardware and Software Tools of the “Network Traffic Monitoring and Analysis” System / Bachelor’s Thesis in Specialty 123 – Systems Programming. Vinnytsia : VNTU, 2025. 73 p.

In Ukrainian. References: 19 sources, Figures: 30, Tables: 3.

In the bachelor’s thesis, the hardware and software tools of the “Network Traffic Monitoring and Analysis” system were developed. Based on a justification of the topic’s relevance, an analysis of existing solutions—including their advantages and disadvantages—was conducted, and common network information transmission standards were reviewed. In line with the stated objective, the «Network Traffic Monitoring and Analysis» system was designed by modernizing methods for capturing and analyzing Internet packets and by employing both hardware and software tools.

Keywords: monitoring, analysis, network traffic levels, packet, real-time.

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та зростання обсягів переданих даних ефективне відстеження й аналіз мережевого трафіку набувають критичного значення з позиції забезпечення надійності, безпеки та оптимізації роботи комп'ютерних мереж. З кожним роком мережеві інфраструктури зазнають постійного навантаження: від масових хмарних сервісів і «інтернету речей» до віртуалізованих робочих середовищ. Наявність інструментів, які дозволяють у реальному часі фіксувати характер трафіку, виявляти аномалії та прогнозувати навантаження, стає запорукою якісного функціонування інформаційних систем на підприємствах будь-якого рівня.

Актуальність теми роботи обумовлена тим, що системи моніторингу та аналізу мережевого трафіку відіграють ключеву роль у забезпечені безпеки та стабільності сучасних мереж. Постійне ускладнення протоколів передачі даних і поява нових сценаріїв обміну інформацією вимагають гнучких і масштабованих інструментів для збору, обробки та візуалізації статистики. Зростання кількості кібератак та витоків даних, своєчасне виявлення аномалій у мережевому трафіку набуває критично важливого значення для попередження інцидентів безпеки та оперативного реагування на них. Ефективне управління пропускною здатністю мережі гарантує підвищення якості обслуговування кінцевих користувачів, зниження затримок і оптимізацію використання мережевих ресурсів.

Існуючі рішення мають обмежену адаптивність до умов експлуатації, складні в налаштуванні або недостатньо інформативні в частині аналізу глибинних мережевих метрик. Це створює необхідність дослідження та розробки комплексної системи, здатної в режимі реального часу збирати детальні дані про пакети, автоматизувати виявлення аномалій і надавати зручні засоби для їхнього візуального аналізу. Реалізація такої системи сприятиме підвищенню надійності мережевої інфраструктури, оптимізації процесів їхнього адміністрування та зміцненню загального рівня інформаційної безпеки організації.

Таким чином, розробка системи моніторингу та аналізу мережевого трафіку є своєчасною й необхідною задачею, яка поєднує в собі практичну значущість для IT-інфраструктури будь-якого масштабу та науковий інтерес щодо удосконалення методів обробки й інтерпретації мережевих даних.

Метою роботи є вдосконалення засобів моніторингу та аналізу мережевого трафіку.

Для досягнення поставленої мети у робочі вирішуються такі **задачі**:

— аналіз наукових джерел і досліджень, пов’язаних з захопленням мережевого трафіку та структури інтернет пакетів, для ефективного їх аналізу.

— аналіз існуючих методів і засобів реалізації системи «Моніторингу та аналізу мережевого трафіку»;

— розробка алгоритму аналізу інтернет пакетів на основі вибраних методів і засобів;

— програмна реалізація системи.

Об’єктом досліджень є мережевий трафік, що проходить через локальні та корпоративні комп’ютерні мережі, сукупність пакетів даних, протоколів обміну й каналів зв’язку.

Предметом досліджень є методи, алгоритми й програмно-апаратні засоби збору, обробки та аналізу мережевого трафіку.

Апробація результатів наукової роботи проведено на науковій конференції: «Молодь у науці: дослідження, проблеми, перспективи (МН — 2025)», доповідь на тему «Система моніторингу та аналізу мережевого трафіку», [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25547.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Мережеві технології та моніторинг

Комп'ютерна мережа — це з'єднання декількох пристроїв, які можуть спілкуватися та обмінюватися даними між собою. Зв'язок може бути як провідним, так і безпровідним. У цій статті ми розглянемо, як можна здійснювати моніторинг і захоплення пакетів.

Моніторинг мережі — це процес постійного спостереження за мережею з різних причин: виявлення системних збоїв, уповільнення трафіку, проблем, пов'язаних із мережею, аналізу даних тощо. Це дуже важливий процес, який допомагає виявляти, відстежувати та контролювати мережу, а також пов'язані з нею пристрої та компоненти мережі, такі як комутатори, маршрутизатори, сервери, віртуальні машини тощо [1].

Захоплення пакетів – це процес аналізу, перехоплення та запису даних пакетів, що передаються або отримуються в мережі. Це важлива техніка для аналізу проблем із продуктивністю, відстеження втрат пакетів, управління трафіком тощо. Захоплення пакетів допомагає адміністраторам виявляти вразливості, атаки або спроби вторгнення, небажану поведінку мережі, перевантаження тощо.

1.2 Техніки моніторингу мережі

NetFlow — це техніка захоплює пакети для аналізу трафіку, що проходить через мережу. Вона збирає дані трафіку та надсилає їх до інструмента моніторингу для аналізу. Цей аналіз визначає потік мережевого трафіку та обсяг даних для з'ясування того, як дані передаються через мережу. Такий підхід дозволяє аналізувати комунікацію між пристроями та забезпечує безперебійність передачі даних. Розроблений компанією Cisco Systems, NetFlow використовується для запису метаданих про потоки IP-трафіку, що проходять через мережевий пристрій, такий як маршрутизатор, комутатор або хост. Пристрій з підтримкою NetFlow генерує метадані на рівні інтерфейсу та

відправляє інформацію про потоки даних до колектора потоків, де ці записи зберігаються для забезпечення аналітики та управління мережевим трафіком. Мережевий оператор може використовувати дані NetFlow для визначення пропускної здатності мережі, втрат пакетів і перевантаження трафіку на конкретному рівні інтерфейсу. Дані NetFlow також підтримують інші сценарії моніторингу на мережевому рівні, такі як виявлення атак типу DDoS та аналіз BGP-пірінгу. NetFlow був спочатку розроблений компанією Cisco у 1995 році для моніторингу та запису всього мережевого трафіку, що проходить через їхні мережеві пристрої. З часом вони зрозуміли, що дані про мережевий потік надзвичайно корисні, і це призвело до виникнення цілого напряму моніторингу мережі, який пізніше був скопійований і перейменований іншими платформами. Сьогодні версія протоколів NetFlow стала галузевим стандартом для оптимізації продуктивності мережі. Цей постійний розвиток NetFlow призвів до появи декількох версій з різними функціями протягом років [2].

Моніторинг за допомогою Ping — цей метод надсилає пакет на пристрій і очікує на відповідь. Якщо відповідь отримана, вважається, що пристрій працює. Якщо відповіді немає, інструмент моніторингу надсилає декілька пінгів, а за відсутності відповіді повідомляє користувача про несправність пристрою. Пінг-моніторинг працює на автоматизованій основі, забезпечуючи регулярні перевірки (налаштування дозволяє запускати моніторинг від кожних 30 секунд до 24 годин). Можна встановити частоту перевірок кожні 30 секунд, хвилину, щогодини або безперервно 24/7 протягом усього року. Після налаштування він потребує мінімального обслуговування чекати на повідомлення. достатньо налаштувати сповіщення та чекати на повідомлення. Налаштування пінг-моніторингу за допомогою UptimeRobot є простим і швидким, зазвичай займає лише 30 секунд. Достатньо кількох кліків, щоб безкоштовно отримати можливість моніторингу на все життя, використовуючи лише свою електронну пошту без жодних додаткових вимог (наприклад, без введення даних кредитної картки). Цей інструмент моніторингу дозволяє тестувати підключення з декількох глобальних кінцевих точок. Ця функція є важливою для виявлення

проблем, які характерні для окремих регіонів, на відміну від проблем, що зачіпають усіх користувачів. Це особливо актуально для сервісів, орієнтованих на міжнародну аудиторію. Ping-моніторинг дозволяє створювати спеціалізовані монітори для конкретних компонентів вашої інфраструктури, таких як бази даних, поштові клієнти або вебсайти. Це дає можливість здійснювати цільовий моніторинг кожного елемента, замість загального підходу для всієї системи. Це найстаріший метод моніторингу, але все ще залишається найкращим способом перевірки працездатності пристроїв [3].

Моніторинг SNMP (Simple Network Management Protocol) — цей протокол використовується для моніторингу мережевих пристроїв. Інструмент SNMP збирає дані з пристроїв і використовує їх для моніторингу продуктивності мережі. Він працює на основі спільної мови для комунікації між пристроями. Система використовує агенти всередині пристрою для надання інформації інструментам моніторингу або менедж м мережі. Інструмент SNMP надсилає опитування для отримання поточного стану, а пристрій може надіслати сигнал при виникненні важливих подій. Моніторинг SNMP базується на взаємодії між програмними компонентами менеджером і агентом. Така клієнт-серверна архітектура забезпечує ефективний збір даних з усіх пристроїв у вашому мережевому інвентарі. Менеджер SNMP відправляє запити до агента на мережевому пристрої для отримання інформації про його стан і продуктивність. Агент збирає ці дані та надсилає відповіді менеджеру. Також агент може самостійно надсилати сповіщення (трап-сигнали) у разі виникнення важливих подій. Таким чином, моніторинг SNMP дозволяє централізовано контролювати стан мережевих пристроїв, виявляти проблеми та здійснювати оптимізацію роботи інфраструктури. Він один із найтриваліших стандартів у мережевих технологіях, який слугує основою для забезпечення видимості інфраструктури з 1988 року. На відміну від пропрієтарних рішень моніторингу, SNMP створив універсальну мову, що дозволяє IT-командам розуміти робочий стан мережевого обладнання незалежно від виробника. У своїй основі SNMP вирішує критичну проблему розширення мереж: здатність постійно моніторити гетерогенні

середовища. Формалізований Інженерною інтернет-групою (IETF) у RFC 1157, цей протокол проклав новий шлях, встановивши стандартизовані методи запиту статистичних даних пристроїв, збору показників продуктивності та отримання сповіщень про події в мультивендорних середовищах [4].

Моніторинг SQL — це процес контролю продуктивності SQL-запитів, що виконуються в базі даних. SQL (Structured Query Language) — це мова програмування, яка використовується для управління та обробки даних у реляційних базах даних. Моніторинг SQL включає вимірювання та аналіз різних показників продуктивності, таких як час виконання запитів, використання процесора (CPU) та пам'яті, з метою виявлення та усунення проблем продуктивності. Основна мета моніторингу SQL — оптимізація продуктивності запитів та покращення загальної ефективності бази даних. Цей метод використовується для баз даних, підключених до мережі. Вони запитують базу для отримання інформації про кількість запитів, передачі даних тощо. Якщо моніторинговий інструмент виявляє повільну роботу бази даних, він може сповістити адміністрацію мережі [5].

Існує кілька інструментів та методів для моніторингу SQL, серед яких [5]:

— інструменти профілювання бази даних, дозволяють отримати детальні відомості про виконання запитів;

— SQL-трасування, забезпечує відстеження всіх операцій у базі даних для виявлення проблемних запитів;

— плани виконання запитів, допомагають аналізувати, як база даних обробляє SQL-запити, що дозволяє виявити вузькі місця в продуктивності.

Ці інструменти допомагають ідентифікувати повільні або неефективні запити, а також інші проблеми продуктивності, такі як взаємне блокування та конфлікти доступу.

1.3 Техніки захоплення пакетів

Портове дзеркалювання (SPAN) — метод налаштовує комутатор на копіювання трафіку з одного або декількох портів на моніторинговий порт. До

нього підключається пристрій для захоплення пакетів, наприклад, ноутбук або сервер. Це найпоширеніший метод захоплення трафіку в корпоративних мережах, який легко налаштувати. Концепція портового дзеркалювання досить проста. Під час налаштування комутатора ви резервуєте один порт. Потім налаштовуєте комутатор таким чином, щоб він "дзеркалив" увесь трафік, що проходить через нього, на цей зарезервований порт. Коли комутатор обробляє пакет, він створює його копію та надсилає її на пристрій, підключений до згаданого порту. Зазвичай це буде спеціалізована система, налаштована для моніторингу трафіку на цьому комутаторі [6].

Якщо ваша мережна топологія охоплює декілька комутаторів, SPAN здатний працювати з розподіленими середовищами. Існують два варіанти SPAN, які ефективно підтримують такі конфігурації [6]:

— RSPAN (Remote SPAN) — дозволяє передавати дзеркалений трафік через декілька комутаторів;

— ERSPAN (Encapsulated Remote SPAN) — забезпечує віддалений моніторинг через IP-мережі, інкапсулюючи трафік у GRE-тунелі.

Таким чином, навіть якщо ваша топологія охоплює кілька комутаторів, SPAN може забезпечити моніторинг у таких складних розподілених середовищах.

TAP (Точка доступу для тестування) — це апаратний пристрій, який розміщується між мережевими пристроями та дозволяє інструменту моніторингу захоплювати копію трафіку. TAP часто використовується в високопродуктивних мережах, де втрата пакетів неприпустима, оскільки забезпечує безперервне захоплення. Проте TAP може бути дорогим та потребувати додаткових налаштувань. Мережеві TAP є важливими інструментами для управління мережею та забезпечення безпеки. Завдяки наданню незмінного вигляду мережевого трафіку, TAP допомагають адміністраторам здійснювати моніторинг продуктивності, виявляти аномалії та усувати неполадки в режимі реального часу. На відміну від традиційних методів моніторингу, які можуть використовувати дзеркальні порти, TAP забезпечують більш надійний та точний

метод захоплення мережевих даних. Коли дані проходять через Мережевий порт 1 до Мережевого порту 2, TAP одночасно надсилає копію даних на Моніторинговий порт. Такий процес гарантує, що інструмент моніторингу отримує точну копію мережевого трафіку без впливу на передачу даних між двома мережевими пристроями. Таким чином, TAP забезпечує неперервний та точний моніторинг трафіку, зберігаючи при цьому стабільність та безпеку мережевого з'єднання. Мережевий TAP зазвичай розгортається шляхом фізичної вставки в мережеве з'єднання [7].

TAP складається з трьох основних портів [7]:

— мережевий порт 1 (вхід), підключається до вхідного мережевого пристрою;

— мережевий порт 2 (вихід), підключається до вихідного мережевого пристрою;

— моніторинговий порт, підключається до інструмента моніторингу або пристрою.

Захоплення пакетів це інструмент, який допомагає аналізувати мережевий трафік та усувати проблеми з мережею. Інструмент захоплення пакетів збирає дані в режимі реального часу, що передаються мережею, для моніторингу та журналювання. Пакети захоплюються у вигляді бінарних даних без змін. Ви можете переглянути інформацію про пакети в режимі офлайн за допомогою аналізаторів пакетів, таких як Wireshark або tcpdump. Якщо вам потрібно швидко захопити пакети, що надходять на маршрутизуючий двигун або виходять з нього, і проаналізувати їх онлайн, можна скористатися діагностичним інструментом захоплення пакетів J-Web. Деякі мережеві пристрої, такі як маршрутизатори або комутатори, мають вбудовані можливості захоплення пакетів. Це дозволяє здійснювати захоплення без додаткового обладнання, але може бути неефективним для великих обсягів трафіку або захоплення на конкретних портах [8].

Захоплення на ноутбуці або настільному комп'ютер, метод передбачає використання інструментів, таких як Wireshark або tcpdump, на комп'ютері,

підключеному до мережі. Інструмент захоплює пакети з мережевого інтерфейсу комп'ютера, дозволяючи аналізувати отриманий трафік. Це зручно для невеликих мереж або захоплення трафіку на конкретних пристроях.

У деяких випадках потрібно захоплювати трафік з мобільного пристрою, наприклад, смартфона або планшета. Для цього існують інструменти, які дозволяють захоплювати та аналізувати мережевий трафік у мобільних додатках. Це корисно для усунення неполадок з підключенням або аналізу трафіку мобільних додатків.

1.4 Передача пакетів у мережах

У комп'ютерних мережах дані передаються у вигляді пакетів, цей метод називається — метод пакетної комутації. Щоб швидко та ефективно передати файл мережею і мінімізувати затримку передачі, дані розбиваються на невеликі блоки змінної довжини, які називаються пакетами. На приймальному боці всі ці дрібні частини (пакети), що належать одному файлу, повинні бути зібрані заново. Пакет складається з корисного навантаження й різної керуючої інформації. Не потрібно попередньо налаштовувати чи резервувати ресурси [9].

Пакетна комутація використовує техніку «зберігання та пересилання»: під час комутації пакетів на кожному вузлі спочатку відбувається зберігання пакета, а потім його пересилання далі. Ця техніка є дуже корисною, тому що пакети можуть бути відкинуті на будь-якому вузлі з різних причин. Між відправником і одержувачем може існувати більше одного маршруту. Кожен пакет містить у собі адреси відправника та одержувача, завдяки чому вони рухаються мережею незалежно один від одного. Іншими словами, пакети одного файлу можуть передаватися різними шляхами. Якщо на якомусь маршруті виникає затор, пакети можуть обирати інші доступні шляхи в існуючій мережі [9].

При пакетній комутації дані поділяються на дрібні пакети, що забезпечує швидше пересилання інформації. Кожен пакет складається з двох частин: заголовка та корисного навантаження. У заголовку кожного пакета міститься

службова інформація. Нижче наведено схему того, як працює пакетна комутація [9].

Приклад пакетної комутації наведено на рис. 1.1.

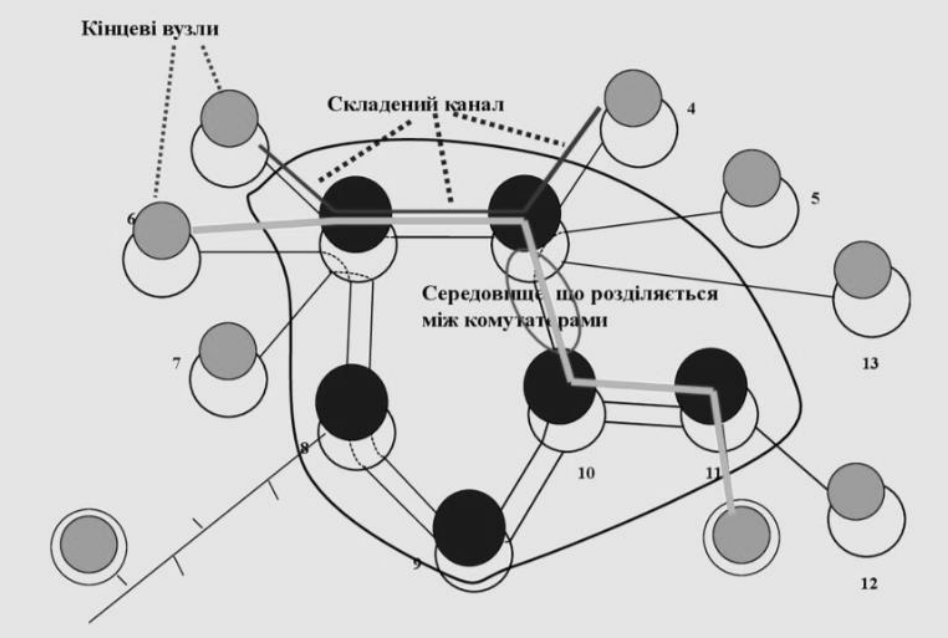

Рисунок 1.1 — Приклад пакетної комутації

Мультиплексуваня — це операція об'єднання декількох окремих інформаційних потоків даних в один загальний агрегований потік, який можна передавати по каналу зв'язку.

Демультиплексування — розділення сумарного агрегованого потоку на декілька інформаційних потоків.

Методи мультиплексуваня наведені на рис. 1.2.

Типи затримок у пакетній комутації:

— затримка передачі (Transmission Delay): час, необхідний відправному вузлу для того, щоб передати дані в канал;

— затримка поширення (Propagation Delay): час поширення даних по каналу від одного вузла до іншого;

— чергова затримка (Queueing Delay): час, який пакет проводить у черзі на вузлі призначення;

— обчислювальна затримка (Processing Delay): час обробки пакетів на

вузлі призначення.

Рисунок 1.2 — Методи мультиплексуваня

Є два типи пакетної комутації: орієнтована на з’єднання та безз’єднана.

У орієнтованій на з'єднання пакетній комутації (Virtual Circuit) перед початком передачі встановлюється логічний канал (віртуальне з'єднання) між відправником і одержувачем за допомогою протоколу сигналізації. Усі пакети цього потоку слідують заданим маршрутом. Комутатори чи маршрутизатори призначають ідентифікатор віртуального каналу (VC ID), який унікально ідентифікує цей шлях. Дані розбиваються на невеликі сегменти, кожен із яких маркується порядковим номером, тому пакети прибувають у правильному порядку [10].

У цій схемі виділяють три фази:

— установлення з'єднання (setup);

— передача даних (data transfer);

— завершення з'єднання (tear-down).

Уся інформація про адреси передається лише під час фази встановлення з'єднання. Як тільки маршрут до пункту призначення виявлено, запис про нього додається до таблиці комутації кожного проміжного вузла. Під час передачі даних заголовок пакета (локальний заголовок) може містити таку інформацію,

як довжина, часовий штамп, порядковий номер тощо. Орієнтована на з'єднання комутація дуже корисна в комутованих WAN. Деякі з популярних протоколів, що використовують підхід віртуальної комутації каналів, — X.25, Frame Relay, ATM та MPLS (Multi-Protocol Label Switching) [10].

Безз'єднана пакетна комутація (датаграма) на відміну від орієнтованої на з'єднання пакетної комутації, у безз'єднаній пакетній комутації кожен пакет містить усю необхідну інформацію про адресацію адресу відправника, адресу одержувача, номери портів тощо. Пакети одного потоку можуть слідувати різними маршрутами, оскільки рішення про маршрутизацію приймаються динамічно, тому пакети, що прибули до пункту призначення, можуть надходити не в тому порядку. У цій схемі немає фаз встановлення та розриву з'єднання, як у віртуальних каналах. Доставка пакетів у безз'єднаній пакетній комутації не гарантується, тому надійність мають забезпечувати кінцеві системи за допомогою додаткових протоколів. Оскільки мережа працює за принципом «зберігання та пересилання» (Store and Forward), під час відправлення пакета з A до B виникають затримки [11].

1.5 Аналіз існуючих рішень

Wireshark — це безплатний програмний аналізатор пакетів з відкритим вихідним кодом. Його використовують для діагностики мережі, аналізу, розробки програмного забезпечення та комунікаційних протоколів, а також у навчальних цілях. Він є багатоплатформеним додатком і використовує набір віджетів Qt у сучасних версіях для реалізації графічного інтерфейсу, а для захоплення пакетів — бібліотеку pcap. Програма працює на Linux, macOS, BSD, Solaris, деяких інших Unix-подібних операційних системах, а також Microsoft Windows. Існує також версія без графічного інтерфейсу для терміналу під назвою TShark. Wireshark і пов’язані з ним програми, зокрема TShark, є вільним програмним забезпеченням, яке розповсюджується на умовах GNU General Public License версії 2 або будь-якої пізнішої.

Tcpdump — це утиліта командного рядка, яка дозволяє захоплювати та аналізувати мережевий трафік, що проходить через вашу систему. Він часто використовується для усунення несправностей у мережі, а також для забезпечення безпеки. Типова історія - збираємо трафік, що приходить на потрібний інтерфейс і потім уже аналізуємо його Wireshark. Підхід практичний, адже Wireshark Дійсно дуже потужний та корисний інструмент і про нього ми напишемо ще не одну статтю, але сьогодні йтиметься про Tcpdump. Не секрет, що утиліта Tcpdump не інтерпретує протоколи прикладного рівня, обмежуючись роботою з транспортним рівнем. Однак у цій статті ми розглянемо різні варіанти використання утиліти Tcpdump для більш глибокої фільтрації трафіку.

ntopng — це засіб аналізу мережевого трафіку, який забезпечує повну 360°-оглядовість мережі завдяки здатності збирати інформацію про трафік з копій трафіку (mirror/SPAN портів), експортерів NetFlow, пристроїв SNMP, журналів міжмережевих екранів (firewall), а також систем виявлення вторгнень (IDS). ntopng розроблений у переносимому вигляді, щоб мати можливість працювати практично на будь-якій Unix-платформі, включаючи Linux і FreeBSD (зокрема pfSense та OPNsense), macOS, а також на Windows. Він захоплює трафік з портів SPAN/дзеркальних (mirror) або пристроїв TAP за допомогою libpcap або PF_RING (на Linux) для максимальної продуктивності. Крім того, його можна використовувати разом із nProbe для збору NetFlow/sFlow з маршрутизаторів і комутаторів, або з nProbe Cento для аналізу каналів зі швидкістю 100 Гбіт/с на повному навантаженні. ntopng — так, усе з маленької літери — надає інтуїтивно зрозумілий, захищений (зашифрований) веб-інтерфейс для перегляду інформації про трафік у режимі реального часу та історичних даних.

2 ВИБІР МЕТОДУ ДЛЯ АНАЛІЗУ ІНТЕРНЕТ ПАКЕТІВ

2.1 Етапи формування та передачі інтернет пакету

Найпоширенішою моделлю для пояснення взаємодії комп'ютерних систем у мережі є модель OSI (Open Systems Interconnection). Модель OSI має сім рівнів, кожен із яких використовує власні протоколи для виконання конкретних завдань мережевої взаємодії. Мережевий протокол – це набір правил, що регулюють обмін даними між різними пристроями у мережі. Він визначає, що, як і коли передається, та дозволяє підключеним пристроям взаємодіяти, незважаючи на їхні внутрішні чи структурні відмінності [12].

Рівні моделі OSI [12]:

— фізичний — це найнижчий рівень моделі OSI, який визначає механічні та електричні характеристики фізичного з'єднання. На цьому рівні відбувається передача бітів через канали зв'язку (кабелі, оптоволокно, бездротові сигнали);

— канальний — на цьому рівні дані групуються в кадри (фрейми), які містять заголовок, дані та контрольну суму, він забезпечує виявлення та виправлення помилок передачі, керування доступом до середовища (наприклад, через протоколи Ethernet або Wi-Fi);

— мережевий — відповідає за маршрутизацію пакетів даних через різні мережі, він використовує IP-адреси для визначення маршруту та включає такі протоколи, як IP;

— транспортний — відповідає за надійність передачі даних між кінцевими точками, основні протоколи: TCP — надійний, з підтвердженням доставки, і UDP — ненадійний, без встановлення з'єднання;

— сенсовий — забезпечує управління з'єднанням між двома додатками. Він встановлює, підтримує та завершує сеанс зв'язку, наприклад, сеанс віртуальної машини або підключення до бази даних;

— рівень представлення — займається перетворенням даних у форму, зручну для прикладного рівня, наприклад, шифрування та декодування даних,

форматування (JPEG, MP3);

— прикладний — взаємодіє безпосередньо з користувачем, програми, такі як веб-браузери та електронна пошта, працюють саме на цьому рівні.

Моделі OSI — це абстрактна модель, яка детально структурує процеси передачі даних, вона часто використовується як еталон для пояснення та розробки нових мережевих технологій, адже вона більш детально структурує процеси передачі даних, на практиці в реальних комп'ютерних мережах використовується TCP/IP. Модель TCP/IP була створена як частина розробки мережі ARPANET і стала основою для інтернету. Вона складається з чотирьох рівнів, що групують функції більш узагальнено, ніж OSI. TCP/IP - це набір протоколів, який забезпечує роботу інтернету та локальних мереж. Він зручний через свою простоту, ефективність та підтримку широкого спектра сервісів. TCP/IP орієнтований на практичність, тому став домінуючим у світі інтернету [13].

Рівні моделі TCP/IP [13]:

— канальний — поєднує функції фізичного та канального рівнів OSI, відповідає за фізичне підключення до мережі та забезпечує передачу даних через фізичне середовище;

— мережевий — забезпечує маршрутизацію пакетів через інтернет, основний протокол IP, а також ICMP для діагностики та ARP для визначення MAC-адрес;

— транспортний — рівень зв'язку між додатками, включає протоколи TCP та UDP;

— прикладний — поєднує прикладний, сеансовий та представницький рівні моделі OSI.

На рис. 2.1 наведено схему інкапсуляції даних моделі TCP/IP.

Передача інтернет-пакета це складний процес, що включає розбиття даних на пакети, маршрутизацію через інтернет та збирання на кінцевому пристрої. Завдяки використанню протоколів TCP/IP забезпечується надійність та ефективність передачі даних. Коли користувач надсилає запит (наприклад,

відкриває сайт), програма (наприклад, браузер) генерує дані, ці дані формуються у форматі прикладного протоколу, наприклад HTTP. Браузер відправляє запит на сервер з інформацією про потрібний вебресурс [13].

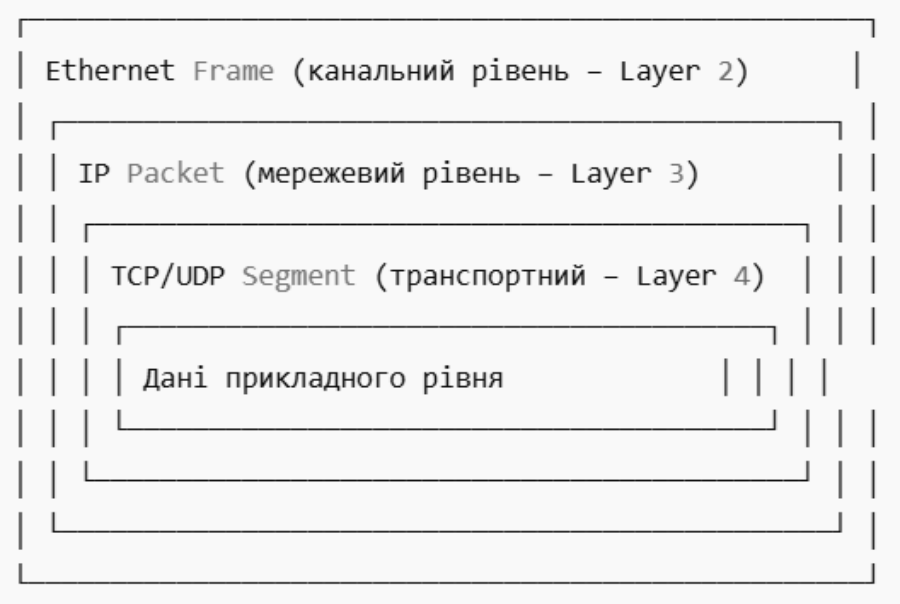

Рисунок 2.1 — Схема інкапсуляція даних моделі TCP/IP

Дані розбиваються на сегменти (TCP) або датаграми (UDP), кожен сегмент містить [13]:

— порт джерела та призначення (наприклад, 80 для HTTP);

— порядковий номер, для збору на іншій стороні;

— контрольна сума, для перевірки цілісності;

У випадку TCP додається також номер підтвердження. Потім кожен сегмент інкапсулюється в IP-пакет, що містить [13]:

— IP-адресу джерела (комп'ютера відправника);

— IP-адресу призначення (сервера або іншого комп'ютера);

— TTL (Time to Live) скільки маршрутизаторів може пройти пакет;

IP-пакет також містить поле Protocol, яке вказує, який транспортний протокол використовується (TCP або UDP). Далі IP-пакет інкапсулюється в кадр канального рівня (наприклад, Ethernet), кадр містить: MAC-адресу джерела (фізична адреса комп'ютера), MAC-адресу призначення (наприклад, маршрутизатора або кінцевого пристрою), затім пакет передається через

локальну мережу до маршрутизатора. Маршрутизатор отримує пакет, перевіряє IP-адресу призначення і вирішує, куди його переслати. Після аналізу пакет передається далі по маршруту через інші маршрутизатори. Кожен маршрутизатор зменшує поле TTL на 1, і якщо воно стає нулем -пакет знищується. Коли пакет досягає кінцевого маршрутизатора, він передається у локальну мережу, кінцевий пристрій приймає пакет, оскільки MAC-адреса збігається з його власною. Пакет розбирається з рівня канального до транспортного, якщо використовувався TCP, пакети збираються у правильному порядку, у разі втрати якогось пакета, TCP ініціює повторну відправку. Якщо використовувався UDP, порядок може не зберігатись, і втрата пакетів не компенсується. Коли всі пакети зібрані та перевірені на цілісність, дані передаються додатку (наприклад, веб-браузеру). Користувач бачить результат [13].

Під час передачі даних можуть відбутися помилки, наприклад пакет пошкоджений або загублений, то TCP ініціює повторне відправлення, у разі вичерпання TTL, маршрутизатор надсилає ICMP-повідомлення про недоступність (часто у вигляді помилки "Tite Exceeded"). Якщо невідповідності контрольної суми пакет просто відкидається.

2.2 Типи протоколів та їх призначення

Комунікаційні протоколи є фундаментальними для функціонування будь-якої мережі. Вони формалізують правила й формати передавання даних, відповідають за синтаксис, семантику, перевірку помилок, синхронізацію та автентифікацію.

Типи комунікаційних протоколів [13, 14]:

— протокол передачі гіпертексту (HTTP) — це протокол прикладного (7-го) рівня для передачі гіпертексту за клієнт-серверною моделлю, більшість веб-трафіку здійснюється саме через HTTP;

— протокол керування передачею (TCP) — це орієнтований на з'єднання протокол, що забезпечує надійну передачу даних із підтвердженням

доставки, використовується в електронній пошті, FTP, потоковому медіа тощо;

— протокол дейтаграм користувача (UDP) — це безз'єднаний протокол із базовим, але ненадійним сервісом передачі даних, він не реалізовує керування потоком, відновлення помилок чи підтвердження доставки, цей протокол підійде там, де потрібна висока швидкість (мультикаст, трансляції);

— протокол прикордонного шлюзу (BGP) — це протокол маршрутизації між автономними системами, який контролює, як пакети рухаються через маршрутизатори великої мережі, що об'єднує сину або кілька мереж, керованих єдиною організацією;

— протокол розпізнавання адрес (ARP) — цей протокол відповідає за відображення IP-адрес у фізичні MAC-адреси в локальній мережі, відповідності зберігаються в ARP-кеші;

— інтернет-протокол (IP) — це базовий протокол мережевого рівня для адресації та маршрутизації пакетів, що визначає їхній шлях від відправника до одержувача;

— протокол динамічної конфігурації хоста (DHCP) — протокол керування мережею для автоматичного призначення IP-адрес і параметрів (DNS, NTP тощо) пристроям у мережі.

Протоколи управління забезпечують моніторинг, налагодження й адміністрування мережевих пристроїв та служб.

Типи протоколів управління [13, 14]:

— протокол керування інтернет-повідомленнями (ICMP) — це протокол мережевого рівня для передачі повідомлень про помилки та діагностики (наприклад, утиліта ping);

— простий протокол керування мережею (SNMP) — це протокол прикладного рівня для моніторингу та управління мережевими пристроями через агента й менеджера SNMP;

— gopher — це старий протокол для пошуку та отримання файлів із ієрархічно організованого сервера, нині майже не використовуваний;

— протокол передачі файлів (FTP) — це клієнт-серверний протокол для передачі файлів між хостами;

— протокол поштового відділення (POP3) — це протокол для завантаження електронної пошти з поштового сервера на клієнт через TCP/IP;

— telnet – це потокол для віддаленого текстового доступу командного рядка на віддаленому комп'ютері.

Протоколи безпеки мережі забезпечують конфіденційність, цілісність і автентичність даних у мережі.

Типи протоколів безпеки мережі [13, 14]:

— рівень захищених сокетів (SSL) — це протокол для захисту чутливої інформації шляхом шифрування з'єднань;

— захищений протокол передачі гіпертексту (HTTPS) — це захищена версія HTTP із шифруванням трафіку за допомогою SSL/TLS;

— безпека транспортного рівня (TLS) — це сучасний протокол безпеки для шифрування й перевірки цілісності даних у мережевих з'єднаннях.

Деякі інші протоколи [14]:

— протокол доступу до повідомлень Інтернету (IMAP) — цей протокол використовується для отримання повідомлень із поштового сервера., користувач може переглядати й керувати поштою на своєму пристрої;

— протокол ініціації сесії (SIP) це протокол ініціації, керування і завершення сеансів у відео-, голосових та текстових застосунках;

— транспортний протокол реального часу (RTP) — цей протокол використовується для передачі аудіо- та відеоданих у реальному часі, часто разом із SIP;

— протокол доступу до маршруту (RAP) — це протокол керування мережею для доступу до найближчого маршрутизатора; менш ефективний, ніж SNMP;

— протокол тунелювання точка-точка (PPTP) — це протокол для реалізації VPN: інкапсулює PPP-кадри в IP-датаграми;

— тривіальний протокол передачі файлів (TFTP) — це спрощена версія FTP, призначена для передачі файлів без автентифікації;

— протокол розташування ресурсів (RLP) — цей протокол використовується для знаходження ресурсів (сервісів, принтерів тощо) у мережі за допомогою широкомовних запитів.

2.3 Визначення методу аналізу інтернет пакетів

Відповідно тому, як кожен рівень моделі TCP/IP виконує конкретну функцію у передачі пакету, для кожного рівня передається різний і відповідний набір даних, необхідний для виконання цієї функції. Тому для аналізу пакета необхідно проаналізувати його на кожному рівні TCP/IP. Залежно від протоколів, які використовуються в пакетах, пакети будуть мати різну структуру та набір даних, тому аналізувати пакет потрібно поетапно, етапи відповідатимуть моделі TCP/IP. Для цього потрібно розібратися у структурі пакетів на всіх рівнях [13, 14].

На канальному рівні («Ethernet» Layer 2) кадр має таку загальну структуру [13, 14]:

— Preamble (7 байт) — синхронізація такт-генераторів; послідовність 0:55;

— SFD (Start Frame Delimiter, 1 байт) — мітка 0xD5 - кінець преамбули і початок MAC-заголовка;

— Destination MAC (6 байт) — MAC-адреса отримувача (наприклад FF:FF:FF:FF:FF:FF для широкомовлення);

— Source MAC (6 байт) — MAC-адреса відправника (наприклад 00:1A:2B:3C:4D:5E);

— Ethertype (2 байти) — якщо ≥ 1536 (0x0600), вказує наступний протокол: 0x0800 (IPv4), 0x0806 (ARP), 0x86DD (IPv6), якщо менше, то визначає довжину поля Payload (IEEE 802.3);

— Payload (46...1500 байт) — дані вмісту (IP-пакет, ARP-повідомлення тощо), мінімум 46 байт (доповнюється нуль-байтами, якщо менше);

— FCS (Frame Check Sequence, 4 байти) — CRC-32 для перевірки цілісності кадру.

На рис. 2.2 наведено приклад Ethernet кадру.

Dst MAC	Src MAC	Ethertype	Payload (IP header + data)	FCS
FF FF FF FF FF FF	00 1A 2B 3C 4D 5E	08 00	45 00 00 54 … (60 bytes IP) …	DE AD BE EF

Рисунок 2.2 — Приклад Ethernet кадру

Якщо кадр має Ethertype — 0x0806, то це ARP пакет, у нього не має Payload, він використовується для визначення MAC-адреси пристрою в локальній мережі за його IP-адресою, тому його аналіз закінчується на цьому моменті. IPv4 та IPv6 потрібно продовжувати аналізувати на вищих рівнях [13, 14].

IP-пакет мережевого рівеня (Layer 3) завжди лежить всередині Ethernet-кадру (або іншого канального формату). Він починається з IP-заголовка (мінімум 20 байт) — структура фіксована, з опціями за потреби. Далі вже йде payload (TCP-сегмент, UDP-датаграма, ICMP-повідомлення тощо). Парсинг виконується шляхом зчитування відповідних бітів і байтів за фіксованими зміщеннями [13, 14].

Загальна структура IPv4-пакету [13, 14]:

— Version (4 біт) — версія IP (4 для IPv4);

— IHL (4 біт) — довжина заголовка в 32-бітних словах (мінімум 5- 20 байт);

— DSCP (6 біт) + ECN (2 біт) — поле сервісу (Quality of Service);

— Total Length (16 біт) загальна довжина (IP-заголовок + дані) в байтах;

— Identification (16 біт) — ідентифікатор пакету (для фрагментації);

— Flags (3 біт) — контроль фрагментації: Bit 0 (Reserved) — завжди 0, Bit 1 (DF) не фрагментується, Bit 2 (MF) більше фрагментів;

— Fragment Offset (13 біт) — зсув фрагмента в 8-байтових блоках;

— TTL (8 біт) — час життя (зменшується на 1 при кожному маршрутизаторі);

— Protocol (8 біт) — вказує, що йде в поля "Payload": 1 — ICMP, 2 — IGMP, 6 — TCP, 17 — UDP, тощо;

— Header Checksum (16 біт) — перевірка цілісності тільки заголовка;

— Source / Destination IP (32 6іт) — IPv4-адреси відправника/одержувача;

— Options + Padding — необов'язкові поля, вирівнюють заголовок до 32-бітного слова;

— Payload — дані прикладного рівня (HTTP, FTP, SSH тощо).

Приклад сирих байт IP-заголовка: 45 00 00 3C 1C 46 40 00 40 06 A6 EC C0 A8 01 64 C0 A8 01 C8. В таблиці 2.1 наведено аналіз цього заголовка.

Таблиця 2.1 — Приклад аналізу IP-заголовка

Поле	Байт(и)	Значення	Опис
Version + IHL	45	Version=4, IHL=5=20 байт	Версія IP
DSCP+ECN	00	DSCP=0, ECN=0	Стандартний сервіс
Total Length	00 3C	0x003C = 60	Загальна довжина пакету
Identification	1C 46	0x1c46 = 7238	Ідентифікатор
Flags+Frag Offset	40 00	DF=1, MF=0, Offset=0	Не фрагментувати
TTL	40	64	Час життя
Protocol	06	6 = TCP	TCP-сегмент у payload
Header Checksum	A6 EC	0xA6EC	Контрольна сума заголовка
Source IP	C0 A8 01 64	192.168.1.100	Адреса відправника
Destination IP	C0 A8 01 C8	192.168.1.200	Адреса отримувача

ICMP-повідомлення передаються в полі Payload IP-пакету, коли Protocol = 1 у IPv4. На рис. 2.3 наведено структуру ICMP-повідомлення.

Структура ICMP-повідомлення:

— Type — визначає підтип повідомлення (наприклад: 8 = Echo Request, 0 = Echo Reply, 3 = Destination Unreachable, 11 = Time Exceeded та ін.) [13, 14];

— Code – уточнює причину (для Type = 3: 0 = Net Unreachable, 1 = Host Unreachable тощо) [13, 14];

— Checksum — контрольна сума самого ICMP-заголовка + даних [13, 14].

Rest of Header – поля, специфічні для кожного Type [13, 14].

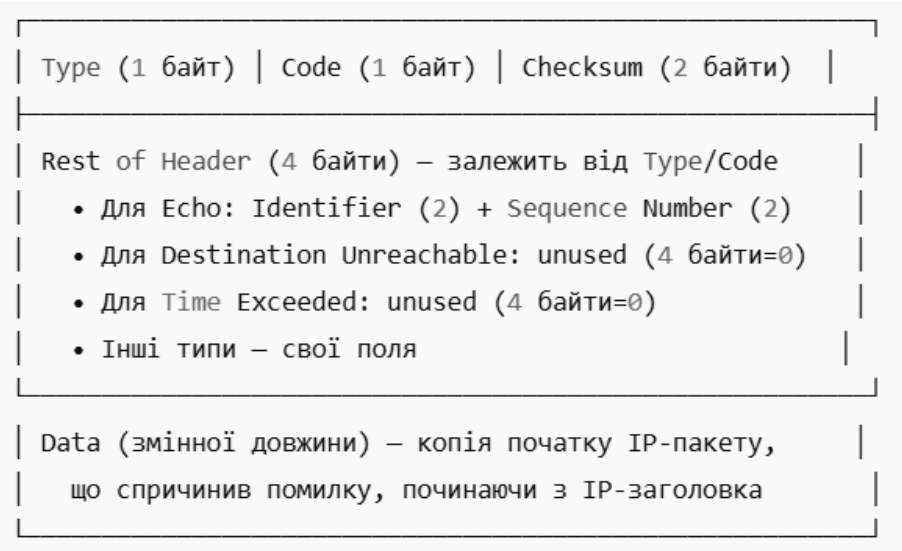

Рисунок 2.3 — Структура ICMP-повідомлення

На рис. 2.4 наведено приклад ICMP-повідомлення

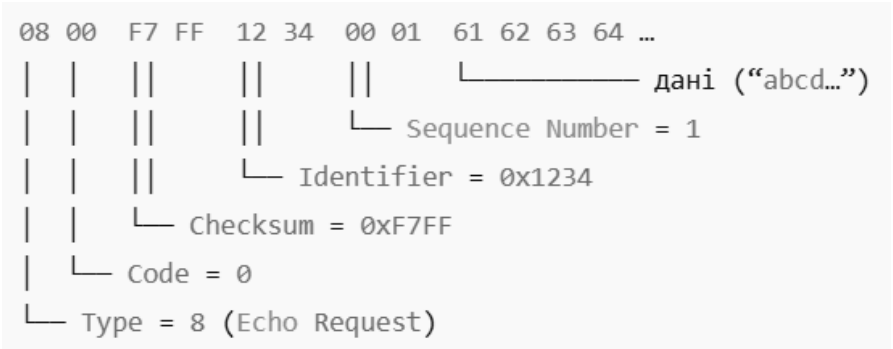

Рисунок 2.4 — Приклад ICMP-повідомлення

IGMP-протокол використовується для управління мультикаст-групами та передається коли Protocol = 2 у IPv4, теж в Payload IP-пакету. На рис. 2.5 наведено структуру IGMP-пакету [13, 14].

Рисунок 2.5 — Структура IGMP-пакету

Структура IGMP-пакету:

— Type — 0x11 = Membership Query, 0x16 = Version 2 Leave Group, 0x17 = Version 1 Report, 0x22 = Version 2 Report, тощо [13, 14];

— Max Resp Time — тільки в Membership Query, максимальна затримка відповіді (в1/10 с) [13, 14];

— Checksum — контрольна сума самого IGMP -заголовка + даних [13, 14];

— Group address — IP-адреса мультігрупи (0.0.0.0 для певних запитів) [13, 14].

На рис. 2.6 наведено приклад IGMP Membership Query.

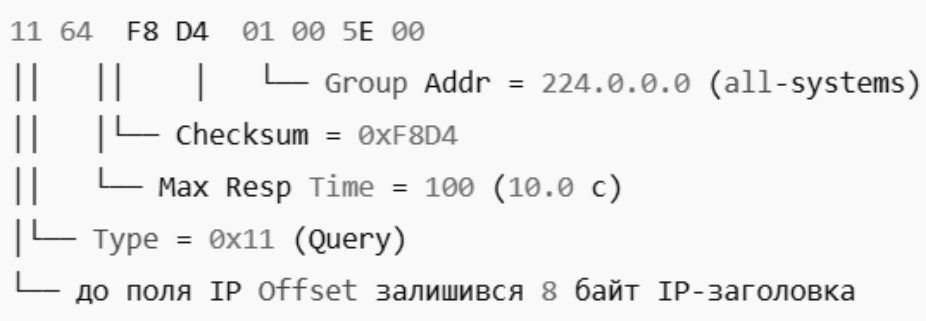

Рисунок 2.6 — Структура IGMP Membership Query

На транспортному рівні (Layer 4:) використовуються TCP та UDP протоколи. TCP-сегмент передається тоді коли в Protocol = 6, структура TCP-заголовка (мінімум 20 байт) наведена в таблиці 2.2 [13, 14]:

Таблиця 2.2 — Структура TCP-заголовка

Offset відкриття	Поле	Розмір/Біти	Опис
0	Source Port	16 біт	Порт відправника
2	Destination Port	16 біт	Порт призначення
4	Sequence Number	32 біт	Початковий порядковий номер байта
8	Acknowledgment Number	32 біт	Номер підтвердження (якщо ACK=1)
12	Data Offset	3 біт	Довжина TCP-заголовка в 32-бітних словах (×4 = байти заголовка)
	Reserved	4 біт	Має бути нуль
	Flags	6 біт	URG, ACK, PSH, RST, SYN, FIN
14	Window Size	16 біт	Розмір приймального вікна
16	Checksum	16 біт	Контрольна сума IP+TCP-заголовка+дані
18	Urgent Pointer	16 біт	Якщо URG=1, вказує на кінець термінових даних
20	Options + Padding	0…40 байт	Опції (MSS, Window Scale, SACK Permitted, Timestamp тощо)
20+opt_len	Payload	змінна	Дані прикладного рівня (HTTP, FTP, SSH тощо)

Приклад TCP-заголовка з HTTP GET наведено на рис. 2.7.

```
C0 A8 01 64   C0 A8 01 C8  // IP-адреси (вже всередині IP)
00 50         1F 90        // SrcPort=80, DstPort=8080
00 00 00 01                // SeqNum=1
00 00 00 00                // AckNum=0
50 02                       // DataOffset=5 (20 байт), Flags=SYN
72 10                       // Window=29200
E6 32                       // Checksum
00 00                       // UrgentPointer
// Ніяких опцій
```

Рисунок 2.7 — Приклад TCP-заголовка

Розбір:

— SourcePort = 0x0050 = 80;

— DestPort = 0x1F90 = 8080;

— SeqNum = 1;

— AckNum = 0 (SYN пакет);

— DataOffset = 5×4 = 20 байт заголовка;

— Flags = 0x02 → SYN=1;

— Window = 0x7210 = 29200;

— Checksum = 0xE632;

— UrgPtr = 0.

UDP-датаграма передається тоді коли в Protocol = 17. Структура UDP-заголовка (8 байт) наведена в таблиці 2.3.

Таблиця 2.3 — Структура UDP-заголовка

Offset	Розмір	Поле	Опис
0	2	Source Port	Порт відправника
2	2	Destination Port	Порт призначення
4	2	Length	Загальна довжина UDP-датаграми (header + payload)
6	2	Checksum	Загальна довжина UDP-датаграми (header + payload)
8	…	Payload	Дані протоколу вищого рівня (DNS, SSDP, …)

Приклад UDP-датаграми (коли Destination Port = 53, DNS) зображено на рис. 2.8.

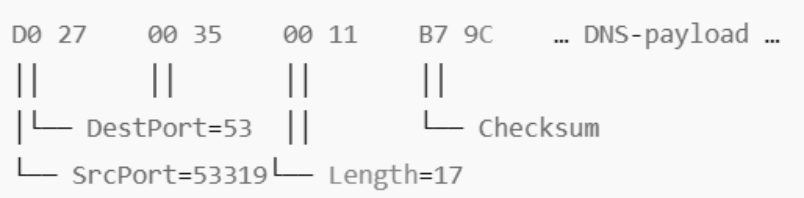

Рисунок 2.8 — Приклад UDP-датаграми

Після розбору ICMP/IGMP — аналізуєте поля Type/Code і за потреби звертаєтеся до даних, UDP — перевіряєте порти (53 → DNS, 1900 → SSDP, 5353 → mDNS, 123 → NTP тощо) та передаєте payload у відповідний парсер прикладного рівня. Таким чином, на мережевому рівні ми відокремлюємо IP-

заголовок, а транспортний рівень для Protocol=1,2,17 розбирає відповідні заголовки ICMP, IGMP або UDP, щоб передати свій payload на аналіз вище [13, 14].

На прикладному рівні (Application Layer) TCP/IP-моделі перебувають протоколи, які працюють безпосередньо із «корисними» даними: HTTP, FTP, DNS, SMTP, SSH, SNMP тощо. На цьому рівні ми вже не дивимося на бітові поля заголовків TCP/UDP чи IP — тут аналізуємо семантику та синтаксис конкретного протоколу. На прикладному рівні ми працюємо з буфером payload, який наша TCP/UDP-логіка скопіювала з транспортного рівня. Наприклад, для HTTP це текстове повідомлення у форматі ASCII/UTF-8, для DNS — бінарний формат у стилі DNS Message Format (RFC 1035) [13, 14].

Структура HTTP-запиту наведена на рис. 2. 9.

```
<Method> <Request-URI> HTTP/<Version>\r\n
Header-Name: Header-Value\r\n
… (багато заголовків) …\r\n
\r\n
[optional message body]
```

Рисунок 2.9 — Структура HTTP- запиту

Request-Line [13, 14]:

— Method: GET, POST, PUT, DELETE, HEAD, OPTIONS, PATCH;

— Request-URI: шлях або повний URL;

— HTTP/<major>.<minor> (наприклад, HTTP/1.1).

Заголовки (Header Fields): Host, User-Agent, Accept, Content-Type, Content-Length тощо.

Порожній рядок (\r\n) відокремлює заголовки від тіла.

Тіло (Message Body) — тільки для методів, які передають дані (POST, PUT).

Приклад HTTP GET-запиту наведено на рис. 2.10.

```
GET /index.html?user=123 HTTP/1.1\r\n
Host: example.com\r\n
User-Agent: MySniffer/1.0\r\n
Accept: text/html\r\n
\r\n
```

Рисунок 2.10 — Приклад HTTP GET-запиту

Структура HTTP-відповіді наведено на рис. 2.11.

```
HTTP/<Version> <StatusCode> <ReasonPhrase>\r\n
Header-Name: Header-Value\r\n
…\r\n
\r\n
[optional message body]
```

Рисунок 2.11 — Структура HTTP-відповіді

Приклад HTTP-відповіді наведено на рис. 2.12.

```
HTTP/1.1 200 OK\r\n
Content-Type: text/html; charset=UTF-8\r\n
Content-Length: 1254\r\n
\r\n
<html>…</html>
```

Рисунок 2.12 — Приклад HTTP-відповіді

FTP — це текстовий протокол на TCP/21 (команди) та TCP/20 (дані), побудований на серії рядкових команд <Command> [arguments]\r\n. У коді читаємо PayloadText як у HTTP, розбиваємо по \r\n, парсимо кожен рядок, розділяємо на команду і аргументи [13, 14].

Структура FTP:

— USER <username> — вхід;

— PASS <password> — пароль;

— PWD — показ поточного каталогу;

— CWD <directory> — зміна каталогу;

— LIST — отримання списку файлів;

— RETR <filename> — завантаження файлу;

— STOR <filename> — закачка файлу.

Приклад діалогу FTP наведено на рис. 2.13.

```
220 Welcome to FTP server\r\n
USER anonymous\r\n
331 Please specify the password.\r\n
PASS guest@\r\n
230 Login successful.\r\n
PWD\r\n
257 "/" is the current directory\r\n
QUIT\r\n
221 Goodbye.\r\n
```

Рисунок 2.13 — Приклад діалогу FTP

DNS-повідомлення — бінарний формат (RFC 1035), його структура зображена на рис. 2.14.

Header [13, 14]:

— ID (2 байти);

— Flags (2 байти) — QR, Opcode, AA, TC, RD, RA, Z, RCODE;

— QDCOUNT, ANCOUNT, NSCOUNT, ARCOUNT (по 2 байти).

Вміст DNS-повідомлення:

— QNAME — послідовність міток: [len][label]…0;

— QTYPE — A, AAAA, MX тощо;

— QCLASS — IN, CH, … .

Приклад простого DNS-запиту наведено на рис. 2.15.

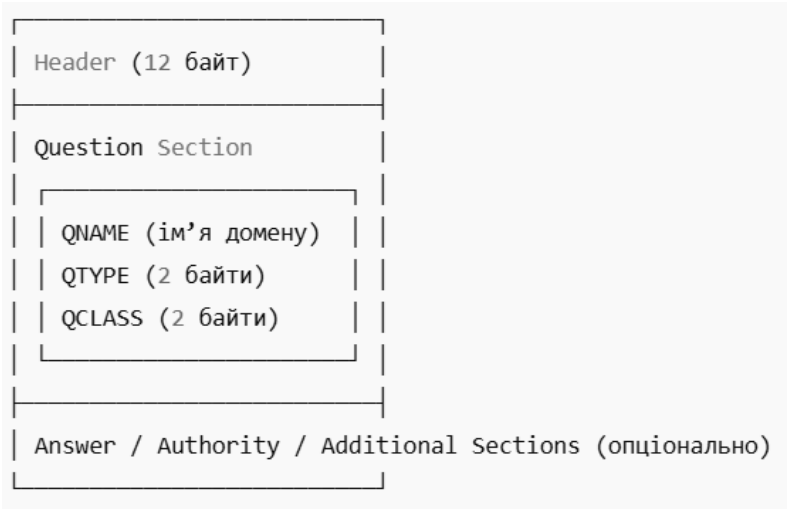

Рисунок 2.14 — Структура DNS-повідомлення

```
12 34  // ID = 0x1234
01 00  // Flags = standard query, RD=1
00 01  // QDCOUNT = 1
00 00  // ANCOUNT = 0
00 00  // NSCOUNT = 0
00 00  // ARCOUNT = 0
07 65 78 61 6D 70 6C 65  // "example"
03 63 6F 6D             // "com"
00                       // end of QNAME
00 01  // QTYPE = A
00 01  // QCLASS = IN
```

Рисунок 2.15 — Простого DNS-запит

3 РОЗРОБКА СИСТЕМИ МОНІТОРИНГУ ТА АНАЛІЗУ МЕРЕЖЕВОГО ТРАФІКУ

3.1 Вибір засобів для реалізації завдання

Для вирішення поставленої задачі я використав мову програмування C#. Це сучасна, об’єктно-орієнтована мова програмування, з широким і розвинутим набором інструментів, бібліотек та технологій, завдяки інтеграції в платформу .NET. Стандартний набір бібліотек (.NET Standard/.NET Core/.NET Framework), дає велику кількість готових класів для роботи з файлами, мережею, потоками, XML/JSON, колекціями тощо. Завдяки .NET можна писати кросплатформені додатки для Windows, macOS, Linux. C# має Garbage Collector — це збирач мусора, є автоматизованим механізмом керування пам'яттю [15].

Для розробки інтерфейсу я використав графічну підсистему WPF, вона вбудована в .NET Framework, має широкий набір готових класів для швидкої розробки інтерфейсу. Для цього використовується мова розмітки XAML, яка була розроблена Microsoft спеціально для WPF. XAML дуже схожа на HTML, оскільки при її розробці орієнтовувалися саме на HTML. Додатково з WPF, я використав шаблон MVVM, що дозволяє розділити інтерфейс (View) та логіку програми (Model) завдяки допоміжному класу (ViewModel), який їх «зв’язує» між собою. Розділення інтерфейсу та логіки необхідно для того, щоб можна було легко вносити зміни, наприклад в інтерфейс і при цьому не зачіпати логіку програми. ViewModel бере дані з Model, робить необхідні операції над ними та віддаває View, при цьому, ні View, ні Model нічого «не знають» один про одного. На рис. 3.1 наведено схему шаблону MVVM. [16, 17].

Для побудови графіків я використав бібліотеку LiveCharts2. Це відкрита бібліотека для .NET, яка спрощує побудову інтерактивних графіків і діаграм у десктопних додатках (WPF, WinForms) та UWP. Вона дозволяє візуалізувати дані в режимі реального часу (realtime) або статично, надаючи готові контролі для різних типів графіків (лінійні, стовпчикові, кругові, точкові та ін.). [18].

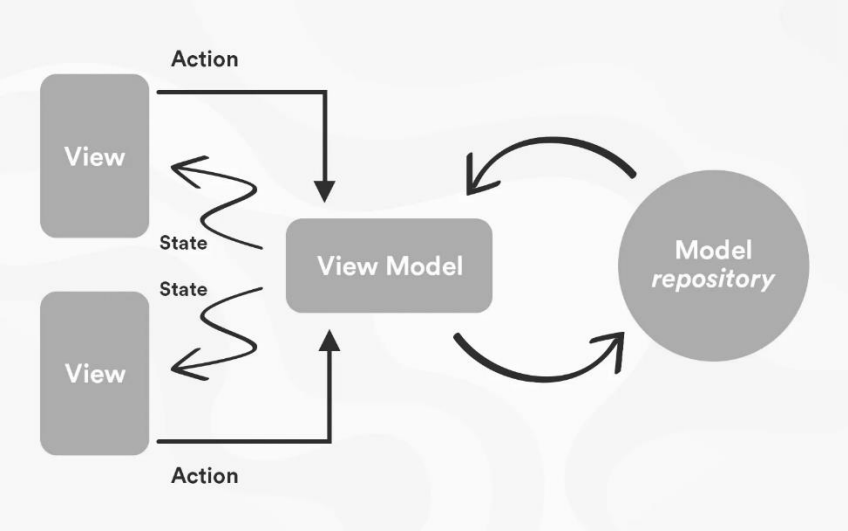

Рисунок 3.1 — Схема шаблону MVVM

Ці всі засоби дають значні переваги, що дозволять значно пришвидшити розробку програми та більше сконцентруватися на архітектурі й вирішені поставленої задачі. Блок-схема архітектури програми наведена, на рис. В.1.

3.2 Моніторинг мережевого трафіку

Для захоплення пакетів я використав клас Socket — це обгортка над нативними BSD-сокетами операційної системи. Він дозволяє, встановлювати TCP- та UDP-з’єднання, налаштовувати параметри передачі (таймаути, розмір буферів, NoDelay тощо), виконувати як синхронні (Send/Receive), так й асинхронні операції (BeginSend/EndSend, SendAsync, ReceiveAsync). Для захоплення всього трафіку необхідно використовувати Raw Socket, однак у Windows є обмеження, через яке при використанні Raw Socket у пакетів буде відсутній канальний рівень, а також система не дасть перехопити пакети на деяких портах, таких як: 80 (HTTP), 443 (HTTPS/QUIC), 21 (FTP), 22 (SSH), 53 (DNS), 5353 (mDNS), 1900 (SSDP). Ці обмеження обумовлені безпекою від злому та прослуховування. Для того щоб їх обійти необхідно буде використовувати сторонні бібліотеки та додаткові драйвера [19].

Параметри конструктора Socket:

— AddressFamily — визначає сімейство адрес (IPv4, IPv6);

— SocketType — Stream (потоковий, TCP), Dgram (датаграми, UDP), Raw (сировинні пакети);

— ProtocolType — конкретний протокол (TCP, UDP, ICMP тощо).

Клас PacketsWatcher служить для створення потоку у якому буде прогодити процес перехвату вхідних та вихідних пакетів. Блок-схема захоплення пакетів наведена на рис. В.2. Повний код класу PacketsWatcher наведено у лістингу Д.1.

Метод CreateSoket() створює та повертає Raw Socket, у лістингу 3.1 наведено код методу.

Лістинг 3.1 — Метод CreateSoket()

```
private static Socket CreateSoket()
{
   Socket socket = new(AddressFamily.InterNetwork, SocketType.Raw,
ProtocolType.IP);
   IPHostEntry host = Dns.GetHostEntry(Dns.GetHostName());
   IPAddress localIP = host.AddressList.FirstOrDefault(ip => ip.AddressFamily ==
AddressFamily.InterNetwork) ?? throw new InvalidOperationException("Не
знайдено локальну IPv4-адресу.");
   socket.Bind(new IPEndPoint(localIP, 0));
   socket.SetSocketOption(SocketOptionLevel.IP,
SocketOptionName.HeaderIncluded, true);
   byte[] optionInValue = [1, 0, 0, 0];
   byte[] optionOutValue = new byte[4];
   socket.IOControl(IOControlCode.ReceiveAll, optionInValue, optionOutValue);
   return socket;
}
```

У конструктор Socket передаються параметри:

— AddressFamily.InterNetwork — визначає сімейство адрес, в даному випадку IPv4;

— SocketType.Raw — «сировинний» дає доступ до необроблених IP-пакетів, включаючи заголовок IP;

— ProtocolType.IP — включає протокол, що буде використовуватися на мережевому рівні.

Метод Dns.GetHostName() повертає ім’я комп’ютера, а метод Dns.GetHostEntry() запитує у системи масив в якому знаходяться всі IP-адреси пов’язані з вказаним ім’ям комп’ютера.

host.AddressList.FirstOrDefault(ip => ip.AddressFamily == AddressFamily.InterNetwork) — цей фрагмент коду виконує пошук у масиві та повертає першу адресу яка задовільняє умову.

socket.Bind(new IPEndPoint(localIP, 0)) — виконує прив’язку до певного IP-інтерфейсу.

socket.SetSocketOption(SocketOptionLevel.IP, SocketOptionName.HeaderIncluded, true) — цей виклик встановлює опцію сокета, яка контролює, чи включати IPv4-заголовок у буфері, що ви отримуєте через Receive

socket.IOControl(IOControlCode.ReceiveAll, optionInValue, optionOutValue) – дає змогу виконати «низькорівневу» операцію введення/виведення (IOCTL) над сокетом, optionInValue масив byte, який передається в драйвер як аргумент команди, optionOutValue масив byte для отримання будь-яких результатів з драйвера.

Процес запуску моніторингу трафіку відбувається у методі StartMonitoringAsync(), щоб програма не зависала під час роботи цей метод запускається у іншому потоці, доки його не скасовують зовнішнім CancellationToken. Ось покроково, що проходить у середині:

— за допомогою CreateSoket() відкривається сірий (raw) сокет, обгорнутий у використанні, щоб автоматично вивільнити ресурси (закрити сокет) після виходу з блоку;

— виділяється масивний буфер розміром 65 535 байт — максимально можливий розмір IP-пакета, водночас у поле CaptureStartTime записується поточний час, від якого можна відрахувати тривалість моніторингу;

— цикл while (!token.IsCancellationRequested) працює доти, доки зовнішній токен не сигналізує про скасування, що дозволяє ззовні зупинити роботу моніторингу;

— рядок _pauseEvent.Wait(token) дозволяє тимчасово призупинити обробку (якщо, наприклад, користувач натиснув кнопку «Пауза»), метод чекає, поки подія (ManualResetEventSlim чи аналог) не перейде в сигнальний стан;

— властивість socket.Available повертає кількість байтів, які вже надійшли й чекають у внутрішньому буфері. Якщо вона більше за нуль, можна читати;

— виклик await socket.ReceiveAsync(buffer, SocketFlags.None) зчитує дані з мережевого інтерфейсу в масивний буфер і повертає реальну кількість байтів bytesRead;

— якщо буфер великий для одного пакета, копіюємо лише перші байти bytesRead у новий масив realData. Це полегшує подальший аналіз, не потрібно відсікати зайве;

— зчитуємо DateTime.Now у зміну captureTimePacket — цей час, коли пакет надійшов, і передаємо його в модель даних;

— визначення протоколу, в IPv4-заголовку 10-й байт (індекс 9) позначає тип протоколу — TCP, UDP, ICMP або інші. Приводимо цей байт до ProtocolType і далі розподіляємо пакети за типом;

— через комутатор створюється відповідний об’єкт (ICMPPacket, IGMPPacket, TCPPacket або UDPPacket) з даними байтів і часом захоплення, після чого накопичується AddPackeToPacketsList(...) для збереження в колекції;

— після кожної ітерації виконується await Task.Delay(1, token), щоб не отримати потік повністю завантаженим у циклі без затримок;

— operationCanceledException ловиться окремо й ігнорується, бо це нормальний шлях завершення під час моніторингу, усі інші виводяться в MessageBox, щоб повідомити користувача про несподівану помилку.

На початку метода створюється та ініціалізується сокет, за допомогою якого і будуть перехоплюватися пакети, для зберігання даних пакету

створюється буфер з довжиною 65535 байт, що являється максимальною довжиною будь якого пакету. У змінні CaptureStartTime зберігається час початку захоплення пакетів. Процес захоплення виконується у тілі циклу до тих пір поки у параметрі token не передасться сигнал закінчення або призупинення. Виконання можна при зупинити на деякий час завдяки _pauseEvent.Wait(token). Якщо вдалося прочитати дані захваченого пакету, то прочитані дані зберігаються у масиві realData, далі визначається протокол і залежно від нього створюється об’єкт класу відповідного пакету, при ініціалізації об’єкту в конструкторі, аналізуються дані з масиву realData, аналіз пакетів буде наведено далі. Створений об’єкти класу пакету завдяки методу AddPackeToPacketsList() додається у колекцію Packets, де зберігаються усі захопллені пакети.

await Task.Delay(1, token) — це асинхронна затримка ~1 мс, яка дотримується вашого CancellationToken і дозволяє безпечно «скидати» виконання методу на короткий час або негайно перерватися, якщо потрібно.

Для старту захоплення пакетів використовується метод Start (), код наведено у лістингу 3.2.

Лістинг 3.2 — Код методу Start()

```
public void Start()
{
   _cts?.Cancel();
   _cts = new CancellationTokenSource();
   _pauseEvent.Set();
   _ = Task.Run(() => StartMonitoringAsync(_cts.Token), _cts.Token);
   Packets.Clear();
}
```

Task.Run() поміщає вказаний лямбда-вираз у чергу виконання потоку з пулу потоків. Для контролю над Task є змінна _cts, через нього можна зупинити, або поставити на паузу виконання. Колекція Packets очищується на випадок якщо там були старі дані. За допомогою методів Resume(), Pause(), Stop() можна зупинити, поставити на паузу, або відновити виконання Task. Їхній код наведено у лістингу 3.3.

Лістинг 3.3 — Код методів Resume(), Pause(), Stop()

```
public void Resume()
{
   _pauseEvent.Set();
}
public void Pause()
{
   _pauseEvent.Reset();
}
public void Stop()
{
   _pauseEvent.Set();
   _cts?.Cancel();
}
```

Метод Dispose() використовується для коректного звільнення ресурсів перед тим як програма закінчить свою роботу.

3.3 Аналіз мережевих пакетів

Клас BasePacket служить базовим класом для всіх інших пакетів, він зберігає інформацію яка є у всіх пакетах незалежно від їхніх протоколів. В його конструктор передається масив з даними отриманого пакету, з нього беруться дані, конвертуються та зберігаються у відповідним їм властивостях. Так як у пакеті дані зберігаються структуровано і для кожного виділено своє місце, то їх можна легко прочитати звернувшись по відповідному індексу до масиву. Блок-схема аналізу пакетів класу BasePacket наведена у на рисунку В.3.

Властивості класу BasePacket:

— PacketData — зберігає дані пакету в «сирому вигляді»;

— CaptureTime — час коли пакет був захвачений;

— ProtocolType — протокол транспортного, або мережевого рівня;

— Version – версія IP;

— IPHeaderLength — довжина IP-заголовка;

— DSCP — класифікації пакетів за пріоритетом/класом обслуговування;

— ECN — механізм індикації перевантаження мережі без відкидання пакетів;

— TotalLength — загальна довжина всього пакету;

— TTL — час життя;

— SourceIP — IP-адреса відправника;

— DestinationIP — IP-адреса отримувача.

Клас ICMPPacket представляє пакет з протоколом ICMP, його властивості ініціалізуються подібно базовому класу BasePacket. У нього є такі властивості:

— Type — загальний тип повідомлення;

— Code — підтип повідомлення;

— Checksum — контрольна сума.

Конструктор класу наведено у лістингу 3.4.

Лістинг 3.4 — Конструктор класу ICMPPacket

```
public ICMPPacket(byte[] packetData, DateTime captureTime) : base(packetData, captureTime, ProtocolType.Icmp)
{
   Type = packetData[IPHeaderLength];
   Code = packetData[IPHeaderLength + 1];
   Checksum = BitConverter.ToUInt16(packetData, IPHeaderLength + 2);
}
```

На рис. 3.2 наведено блок-схему алгоритму аналізу ICMPPacket.

Клас IGMPPacket представляє пакет з протоколом IGMP, ініціалізація проходить подібно попередньому класові. Версія IGMP визначається залежно від того який тип, максимальний час очікування та довжини IGMP частини.

Властивості класу:

— Type — загальний тип повідомлення;

— MaxResponseTime — максимальний час очікування;

— Checksum — контрольна сума;

— GroupAddress — широкомовна адреса;

— IGMPVersion — версія IGMP.

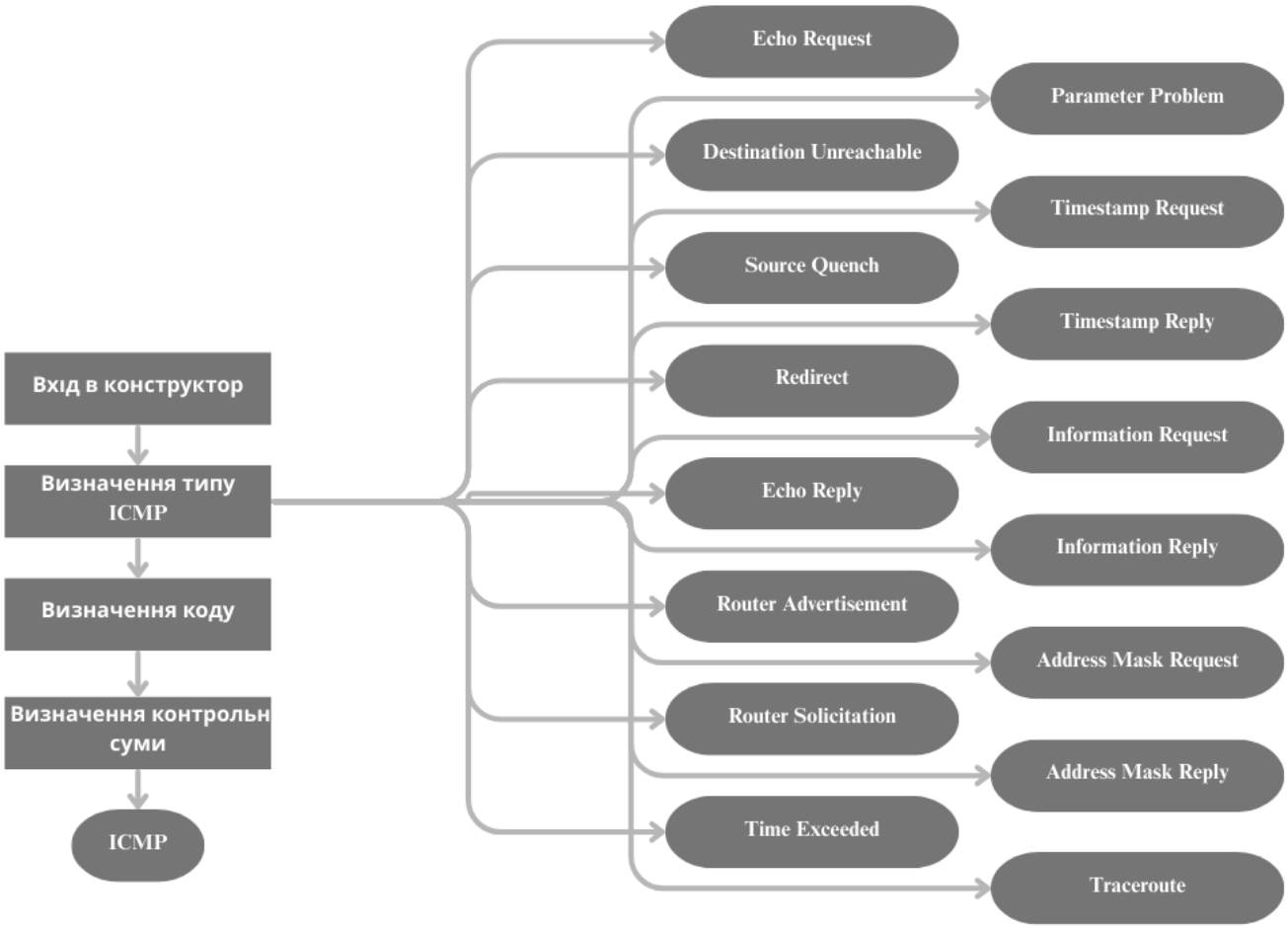

Рисунок 3.2 — Блок-схема алгоритму аналізу ICMPPacket

Конструктор створює об’єкт IGMPPacket на основі сірого масиву байтів пакета та його захоплення, і в ньому відбувається:

— Type = packetData[IPHeaderLength] — у IGMP-заголовку байт відразу після IP-заголовка (тобто за зміщенням IPHeaderLength) містить повідомлення типу (наприклад, “Membership Query” або “Membership Report”);

— MaxResponseTime = packetData[IPHeaderLength + 1] — це максимальний час відповіді (в одиницях часу, визначених стандартом) для деяких типів IGMP-повідомлень;

— Checksum = BitConverter.ToUInt16(packetData, IPHeaderLength + 2) — два байти, що йдуть за полем Max Response Time, вашу 16-бітну контрольну суму. BitConverter.ToUInt16 враховує порядок байтів платформи;

— GroupAddress = new([.. packetData.Skip(IPHeaderLength + 4).Take(4)]) – після контрольної суми іде 4-байтна IP-адреса мультикаст-групи. Тут ми

беремо участь у позиції IPHeaderLength + 4 та передаємо їх у конструктор якогось типу IPAddress або власної структури IPAddressLike;

— int msgLength = TotalLength - IPHeaderLength — загальна довжина IP-пакета (TotalLength), зменшена на довжину IP-заголовка, дає довжину чистого IGMP-блоку;

— визначення версії IGMP;

На рис. 3.3 наведено блок-схему алгоритму аналізу IGMPPacket.

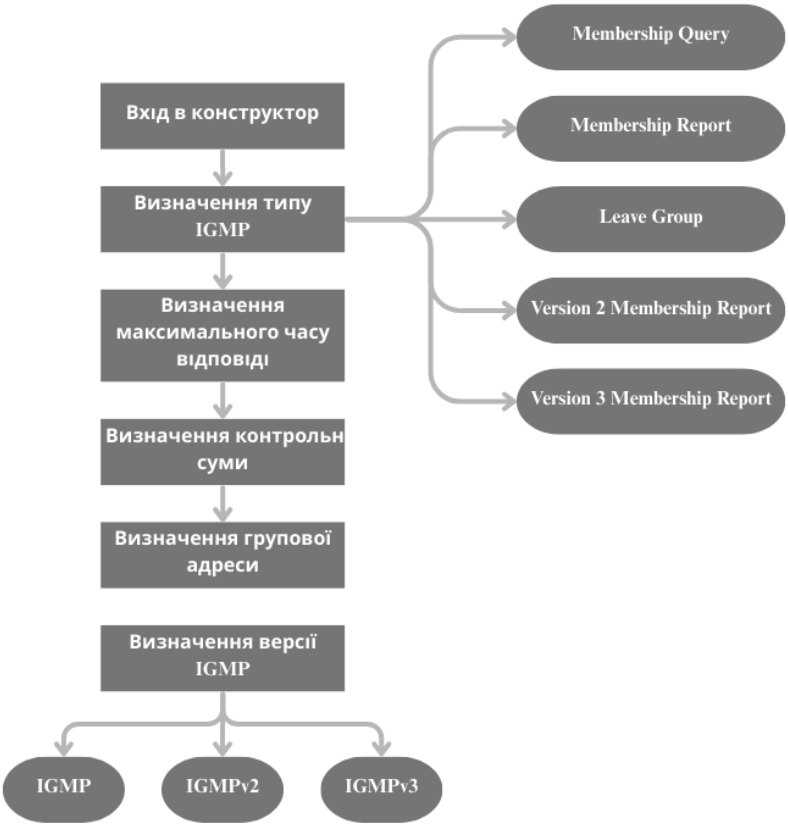

Рисунок 3.3 — Блок-схема алгоритму аналізу IGMPPacket

Клас TCPPacket представляє пакет, що передається по протоколу TCP. Поверх нього предаються інші протоколи, щоб їх визначити потрібно проаналізувати порти через які вони прийшли та, якщо можливо, яку несуть в собі інформацію. Зазвичай дані в пакеті шифруються, а тому прочитати їх не вдасться. Блок-схема аналізу пакетів класу TCPPacket наведена на рисунку 3.4.

Властивості класу:

— SourcePort — порт відправника;

— DestinationPort — порт отримувача;

— TCPHeaderLength — довжина TCP-заголовка;

— Flags — прапорці;

— Checksum — контрольна сума;

— PayloadData — дані що передаються;

— ApplicationProtocolType — протокол прикладного рівня.

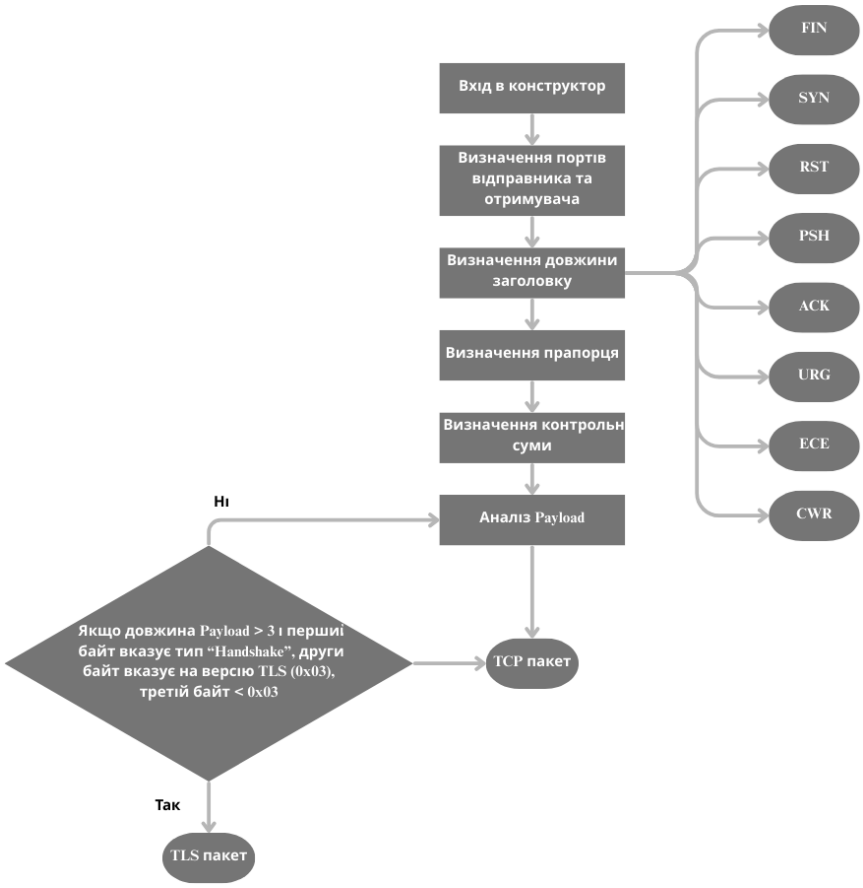

Рисунок 3.4 — Блок-схема аналізу пакетів класу TCPPacket

Розглнемо агоритм конструктору: SourcePort = BitConverter.ToUInt16(packetData, IPHeaderLength), DestinationPort = BitConverter.ToUInt16(packetData, IPHeaderLength + 2) – у TCP-заголовку перші два байти після IP-заголовка містять порт відправника, наступні два — порт призначення. BitConverter.ToUInt16 читає 16-бітне значення з байтів у правильному порядку.

TCPHeaderLength = (packetData[IPHeaderLength + 12] >> 4) * 4 — у 13-му байті TCP-заголовка (індекс 12 від початку TCP) старші чотири біти містять

“Data Offset” (кількість 32-бітних слів заголовка). Зсуваємо вправо на 4, множимо на 4 байти — отримуємо реальну довжину заголовка в байтах (мінімум 20 байт).

Flags = ProtocolDescriptions.TCPFLAGS.Where(kvp => (packetData[IPHeaderLength + 13] & kvp.Key) != 0).Select(kvp => kvp.Value) – у наступному байті (індекс 13) закодовані бітові прапорці (SYN, ACK, FIN тощо). Проходимо словник TCPFLAGS, відбираючи ті пари (маска→назва), де відповідний біт встановлений, й формуємо перелік активних прапорців.

Checksum = BitConverter.ToUInt16(packetData, IPHeaderLength + 16) – два байти, що йдуть після полів Sequence/ACK number та прапорців, утворюють TCP Checksum. Зчитуємо їх аналогічно до портів.

Наступним кроком обчислюємо зсув до початку даних, віднімаючи від загальної довжини IP-пакета суму довжин IP- і TCP-заголовків. Якщо дані є, копіюємо їх у масив PayloadData. Потім перевіряємо перші три байти завантаження: 0x16 (Handshake), 0x03 (major TLS version) та minor версію ≤ 3. Якщо збігається, відмічаємо, що в цьому TCP-потоці передається TLS-з’єднання.Д ля цього аналізуються перші три байти PayloadData. Для того щоб взнати де починається PayloadData, необхідно відсіяти IP-заголовок й TCP-заголовок.

Клас UDPPacket представляє пакет, датаграм UDP, поверх нього теж передаються деякі інші протоколи прикладного рівня, для їх визначення необхідно проаналізувати порти через які вони прийшли та, якщо можливо, їхні дані. У UDP пакті, дані при передачі, теж шифруються. Властивості класу UDPPacket:

— SourcePort — порт відправника;
— DestinationPort — порт отримувача;
— Checksum — контрольна сума;
— PayloadData — дані пакету;
— ApplicationProtocolType — протокол прикладного рівня.

У лістингу 3.5 наведено конструктор класу.

Лістинг 3.5 — Конструктор класу UDPPacket

```
public UDPPacket(byte[] packetData, DateTime captureTime, ProtocolType
protocolType = ProtocolType.Udp) : base(packetData, captureTime, protocolType)
{
   SourcePort = BitConverter.ToUInt16(packetData, IPHeaderLength);
   DestinationPort = BitConverter.ToUInt16(packetData, IPHeaderLength + 2);
   Checksum = BitConverter.ToUInt16(packetData, IPHeaderLength + 6);
   int payloadOffset = IPHeaderLength + 8;
   if (payloadOffset < TotalLength)
   {
      int payloadLen = packetData.Length - payloadOffset;
      PayloadData = new byte[payloadLen];
      Array.Copy(packetData, payloadOffset, PayloadData, 0, payloadLen);
   }
}
```

Клас GraphHelper служить для побудови графіків. В програмі реалізовано три графіка:

— кількість пакетів від часу захоплення;

— об’єм отриманих даних від часу захоплення;

— графік потоків.

Для кожного графіку визивається своя функція, в параметри якої передається колекція з пакетами. Пакети з переданої колекції групуються по протоколам та аналізуються. Після аналізу та отримання потрібних даних, дані запаковуються в об’єкт GraphDataObject, який потім повертає функція.

GetGraphLengthPacketsFromCaptureTime() — це метод, що потрібен для побудови графіку об’єму отриманих даних від часу захоплення. У цьому методі пакети групуються у словарі Dictionary<string, List<PacketViewModel>> packetsByProtocol, ключем підколекці є назва протоколу типу string. Для побудови гравіку беруться такі дані: час з моменту початку захоплення пакетів та довжина всього пакету. З метою оптимізації, було обмежено кількість точок на графіку, якщо кількість пакетів більше тисячі, то буде братися середнє значення тисячних частин загальної кількості пакетів однакового типу протоколу. Тобто якщо пакетів буде більше тисячі, то буде братися середнє

значення двох пакетів по осі X та Y, якщо більше двох тисяч то середнє значення трьох пакетів і так далі… . У методі також визначається кількість пакетів кожного типу протоколу, для кожного протоколу (TCP, UDP тощо) нараховується сумарний обсяг даних та зберігається найбільший розмір пакета, виявлений для цього протоколу, крім того визначаються середня довжина прийнятих пакетів, а також загальний об’єм всіх пакетів. По осі X відображається час у форматі hh:mm:ss або mm:ss залежно від тривалості, по осі Y — розмір пакета в байтах, у змінних деталях збирається текст із підсумковою статистикою: середній (тут береться максимальний поділ на кількість) і загальний обсяг для кожного протоколу. Таким чином, метод автоматизує підготовку всіх потрібних даних і метаданих для побудови інтерактивного графіка «розмір пакета проти часу від початку захоплення» з додатковою текстовою статистикою по кожному протоколу.

GetGraphCountPacketsFromCaptureTime — метод, що служить для побудови графіку кількості прийнятих пакетів від часу початку захоплення. У ньому пакети теж групуються за протоколами і потім з них беруться потрібні дані. Якщо кількість даних більше тисячі, то беруться середні значення тисячних частин від загальної кількості пакетів. Метод починає з прийому всіх пакетів і їхнього сортування за протоколами. Для цього він викликає вузькоспеціалізовану функцію GroupByKeys, яка повертає словник, де ключі — назви протоколів (TCP, UDP, ICMP, IGMP, TLS), а значення — списки відповідних об’єктів PacketViewModel. Завдяки цьому далі кожен протокол обробляється незалежно. Для кожного списку пакетів метод визначає розмір «порції» агрегації як приблизно одну тисячну від загальної кількості пакетів. Потім у циклі він проходить по всіх пакетах із кроком цієї порції: на кожному кроці збирає групу пакетів, обчислює суму їхніх часових відрізків (CaptureTimeSpan.TotalSeconds), а потім знаходить середній час надходження для цієї групи. Саму точку графіка він створює як пару (середній час, накопичена кількість пакетів до цього кроку). Кожна така серія точок додається до колекції series із відповідним кольором для протоколу. Після того, як для всіх протоколів

сформовано точки, метод готує конфігурацію осей: по горизонталі відкладається час від початку захоплення у форматі “xx:xx:xx” або “xx:xx”, залежно від тривалості, а по вертикалі — кількість прийнятих пакетів. Параметри форматування підписів задаються за допомогою лямбда-функцій, що перетворюють числові значення у рядкові представлення.Наприкінці метод створює текстове поле details, в яке лінійно записує загальну кількість пакетів для кожного протоколу. І нарешті, усі отримані дані — масив серій, налаштовані осі та текст із деталями — упаковуються в один об’єкт GraphDataObject, готовий до передачі компоненту, що малює графік у UI. Таким чином, без жодних списків чи надлишкових деталей ми отримуємо чіткий та ефективний спосіб продемонструвати, як зростає кількість пакетів у часі для кожного мережевого протоколу.

Метод public static GraphDataObject GetFlowsGraph() призначений для створення розсіювального графіку потоків. Потоки визначаються за п’ятіркою критеріїв, перші три – це однакові протоколи та IP-адреси отримувача й відправника,інші два для пакетів прикладного рівня, які наслідуються від інтерфейсу IApplicationLayer, є порти однакові порти відправника та отримувача, для ICMP-пакетів останні два – це однакові тип та код повідомлення, для IGMP-пакетів – це однакові тип повідомлення та групова IP-адреса. Для кожного потоку визначається час його життя та об’єм переданих даних. Крім того визначається загальна кількість переданих даних зі всіх потоків, максимальна та мінімальна кількість переданих даних одним потоком, максимальний і мінімальний час життя потоку, що складається мінімум з двох пакетів. Всі пакети з колекції allPacketsEnumerable зберігаються в списку allPackets, а ряд змінних ініціалізується для підрахунку мінімального та максимального часу життя потоку, мінімального та максимального обсягу та загальної суми переданих байт. Потім створити потоки словників, де для кожного протоколу (TCP, UDP, ICMP тощо) ініціалізується поріг списку точок. Після формування всіх потоків метод проходить по словнику потоків і для кожного протоколу створюється серія типу ScatterSeries, додаючи її до списку серій. Якщо задаються налаштування осі: вісь

X підписана як «Час життя потоку» та відформатовано в hh:mm:ss або mm:ss.ffffff залежно від тривалості, вісь Y — «Об’єм переданих даних (байт)». Наприкінці формується текстовий блок деталей, де виводяться знайдені мінімальні та максимальні показники часу життя потоків і обсягу переданих даних. І нарешті всі зібрані серії, осі та текст деталей упаковано в об’єкт GraphDataObject, який повертається для подальшої побудови розсіювального графіка.

3.4 Розробка графічного інтерфейсу

Під заголовком вікна находиться меню, у ньому можна вибрати який графік відобразити. Властивість Command прив’язуються до команд які знаходяться у MainWindowViewModel, що є ViewModel для головного вікна. Улістингу 3.6 показано XAML-розмітка меню.

Лістинг 3.6 — XAML-розмітка меню

```
<Menu Grid.Row="0">
   <MenuItem Header="Графік кількості пакетів від часу"
     Command="{Binding CreateGraphCountPacketsFromCaptureTimeGraph}"/>
   <MenuItem Header="Графік довжини пакетів від часу"
     Command="{Binding CreateGraphLengthPacketsFromCaptureTime}"/>
   <MenuItem Header="Графік потоків"
     Command="{Binding CreateGraphFlows}"/>
</Menu>
```

У верхній частині головного вікна програми знаходиться панель з трьома кнопками. Перша кнопка служить для запуску моніторингу трафіку, друга – для припинення, третя – для паузи. Кожна кнопка прив’язана до відповідної команди у ViewModel через команду влади. Наприклад, кнопка «Відтворити» захист StartPacketsCapture, «Stop» — StopPacketsCapture, а «Pause» — PausePacketsCapture. Завдяки цьому при натисканні на кнопку у фоновому коді виконується необхідна логіка запуску або зупинки моніторингу.

StackPanel – це елемент який розташовує в собі дочірні елементи горизонтально або вертикально. Стилі, що описують зовнішній вигляд всіх

елементів вікна, розташовані в окремих файлах і підключаються в файлі App.xaml. Властивість Command прив’язуються до команд які знаходяться у MainWindowViewModel, що є ViewModel для головного вікна. На рис. 3.5 наведено графічне зображення кнопок.

Нижче знаходиться TextBox, який використовується для фільтрування показу пакетів. Фільтрувати можна тільки по типу протоколу, для сього прото потрібно його ввести в TextBox. У лістингу 3.7 показано розмітку TextBox.

Рисунок 3.5 — Кнопки управління захватуванням трафіку

Лістинг 3.7 — Розмітка TextBox фільтрування

```
<TextBox Grid.Row="1"
        Style="{StaticResource FiltersTextBox}"
        Text="{Binding FiltersTextBlok, UpdateSourceTrigger=PropertyChanged}"/>
```

Захоплені пакети представлені у вигляді таблиці по середині вікна. Для цього був використаний елемент DataGrid. Таблиця має 6 колонок, номер пакеті у списку, час з моменту захоплення, джерело, призначення, протокол, довжина пакету. Дані в таблиці можна сортувати. Джерело даних прив’язується за допомогою властивості ItemsSource до колекції AllPackets, що знаходиться у MainWindowViewModel. У цій колекції находяться елементи класу PacketViewModel, що є ViewModel всіх пакетів, кожен PacketViewModel зберігає в собі силку на справжній пакет, таким чином через нього можна отримати дані про пакет, у зручному вигляді. Значення колонок таблиці прив’язуються до властивостей елемента класу PacketViewModel в колекції AllPackets. У лістингу 3.8 показано конструктор класа PacketViewModel. Таблиця гнучко розподіляє простір між колонками пропорційно, а всі прив’язки неможливо отримати UI синхронним із моделями даних без додаткового коду в code-behind.

Лістинг 3.8 — Конструктор класа PacketViewModel

```
public PacketViewModel(BasePacket packet, int number, DateTime
captureStartTime)
{
   Packet = packet;
   Number = number;
   CaptureTimeSpan = packet.CaptureTime - captureStartTime;
   PacketData = string.Join(" ", packet.PacketData.Select(b => b.ToString("X2")));
   (ProtocolType) = packet switch
   {
      IApplicationProtocolType appPT
         when appPT.ApplicationProtocolType != ApplicationProtocolType.Unknown
         => (
            appPT.ApplicationProtocolType.ToString().ToUpperInvariant()
           ),
      _ => (
         packet.ProtocolType.ToString().ToUpperInvariant()
        )
   };
   Details = GetDetails();
}
```

Властивість Packet зберігає силку на реаьний пакет, PacketData – зберігає «сирі» біти конвертовані у тип string. Властивість Details зберігає детальну інформацію про пакет. Метод GetDetails() бере інформацію з Packet та представляє її у зручному вигляді.

Спочатку він створює новий об’єкт StringBuilder і далі додає в нього базові поля IP-пакета: час фактичного захоплення, відлік часу від початку моніторингу, значення TTL, версію IP (як числове значення і в дужках — як перелік констант), довжину IP-заголовка, номер транспортного протоколу (також в обох форматах), значення DSCP та ECN (контрольований через словник розшифровок), загальну довжину. пакета, IP-адреси відправника й отримувача. Потім метод виконується після перевірки типу пакета через оператор. Якщо це ICMPPacket, він дістається з його поля Type та розшифровує його через словник ProtocolDescriptions.ICMPTYPES, додає кодове повідомлення та виводить перевірену суму в шістдесятковому форматі.

Якщо пакет — IGMPPacket, крім поля Type і розшифровки, додається максимальний час очікування відповіді, виявлена версія IGMP, контрольна сума й групова (multicast) адреса. У разі, якщо метод TCP-пакета доповнює рядок відправлених і цільових портів, зазначених прикладний протокол, довжину TCP-заголовка, перелік встановлених прапорців (якщо вони є) або слово «none», контрольну суму й довжину корисних даних. Для UDP-пакетів аналогічно вказуються порти, прикладний протокол, контрольна сума та довжина даних. Після того, як усі умови перевірки типів пройдені і відповідні рядки додані, метод повертає готовий текст із звітними відомостями про пакет. Такий підхід дозволяє в одному універсальному методі вивести деталі для будь-якого підкласу Packet, зберігаючи при цьому код стислим і зрозумілим.

За допомогою StringBuilder, вся інформація, формується у один рядок, таким чином її можна вивести в один TextBox. В головному вікні, у нижній частині, є панель, у якій є два елементи TextBox. З лівого боку виводиться інформація з Details, з правого боку — з PacketData. Розмітка цієї панелі наведена у лістингу 3.9.

Лістинг 3.9 — Розмітка панелі з детальною інформацією

```
<Grid Grid.Row="4">
   <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
   </Grid.ColumnDefinitions>
   <TextBox Grid.Column="0"
         Text="{Binding SelectedPacket.Details, Mode=OneWay,
FallbackValue='[ніщо не обрано]'}"
         Style="{StaticResource DataTextBox}"/>
   <TextBox Grid.Column="1"
         Text="{Binding SelectedPacket.PacketData, Mode=OneWay,
FallbackValue='[ніщо не обрано]'}"
         Style="{StaticResource DataTextBox}"/>
</Grid>
```

На рис. 3.6 наведено виведеня детальної інформації про пакет

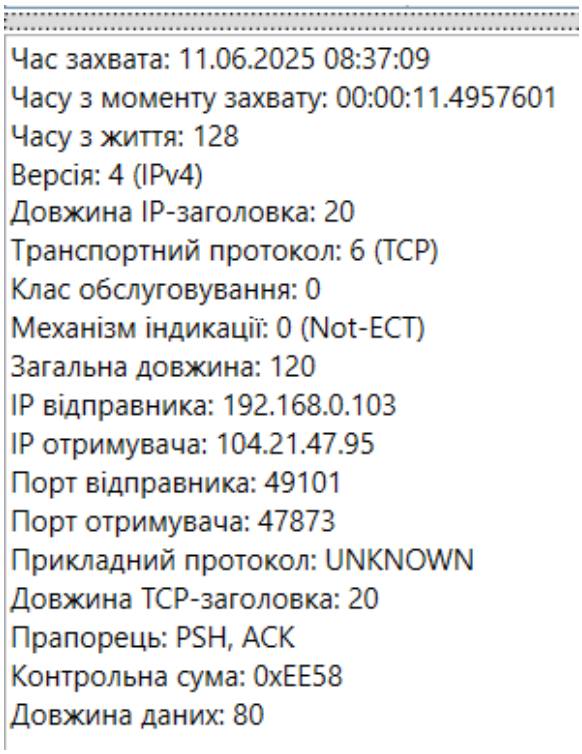

Рисунок 3.6 — Виведеня детальної інформації про пакет

У конструкторі MainWindowViewModel до колекції Packets із PacketsWatcher підписується подія Packets_CollectionChanged, таким чином коли у колекцію Packets додається новий елемент, то відбувається подія Packets_CollectionChanged у які створюється PacketViewModel для пакету із колекції Packets і додається до колекції AllPackets із MainWindowViewModel.

GraphWindow – призначений для відображення графіків. У конструкотр ViewModel вікна передається GraphDataObject, у якому запаковані всі необхідні дані. У ViewModel вікна є такі властивості:

— Series — масив графіків;

— XAxes — ось X;

— YAxes — ось Y;

— Details — додаткова інформація.

У лістингу 3.10 наведено код конструктора GraphWindowViewModel.

Лістинг 3.10 — Конструктор GraphWindowViewModel

```
public GraphWindowViewModel(GraphDataObject data)
{
```

```
    Series = data.Series;
    XAxes = data.XAxes;
    YAxes = data.YAxes;
    Details = data.Details;
}
```

Вікно для графіків складається з Grid, з’єднаній з ресурсом стилів MainGrid є CartesianChart. Він розташований у першому рядку (Grid.Row="0") і заповнює весь доступний простір завдяки Height="*". До самого елемента прив’язуються колекція серій даних (Series), а також налаштовані осі (XAxes і YAxes) — усі вони беруться із властивостей ViewModel. Параметр LegendPosition="Bottom" забезпечує вивід легенди під самою діаграмою. Далі йде GridSplitter у другому рядку (Grid.Row="1") із автоматичною висотою (Height="Auto"). Це «роздільник», який дає користувачу можливість перетягнути межу між діаграмою й текстовим вікном для зміни їхнього співвідношення по висоті. Нижній блок розташовано у третьому рядку (Grid.Row="2"), де відведено фіксовану висоту в 100 пікселів. Всередині нього лежить текстове поле TextBox, котре односторонньо прив’язане до властивості Details у ViewModel. Якщо даних ще немає або прив’язка не спрацює, показуватиметься значення "[Немає даних]". Сам контейнер і сам TextBox оформлені через стилі зі словника ресурсів (DataTextBox), що гарантує єдиний вигляд усіх блоків тексту в додатку.

На рисунку 3.7 показано графік залежності кількості пакетів від часу початку захоплення.

На рисунку 3.8 показано графік розміру пакетів від часу початку захоплення.

На рисунку 3.9 показано розсіювальний графік потоків.

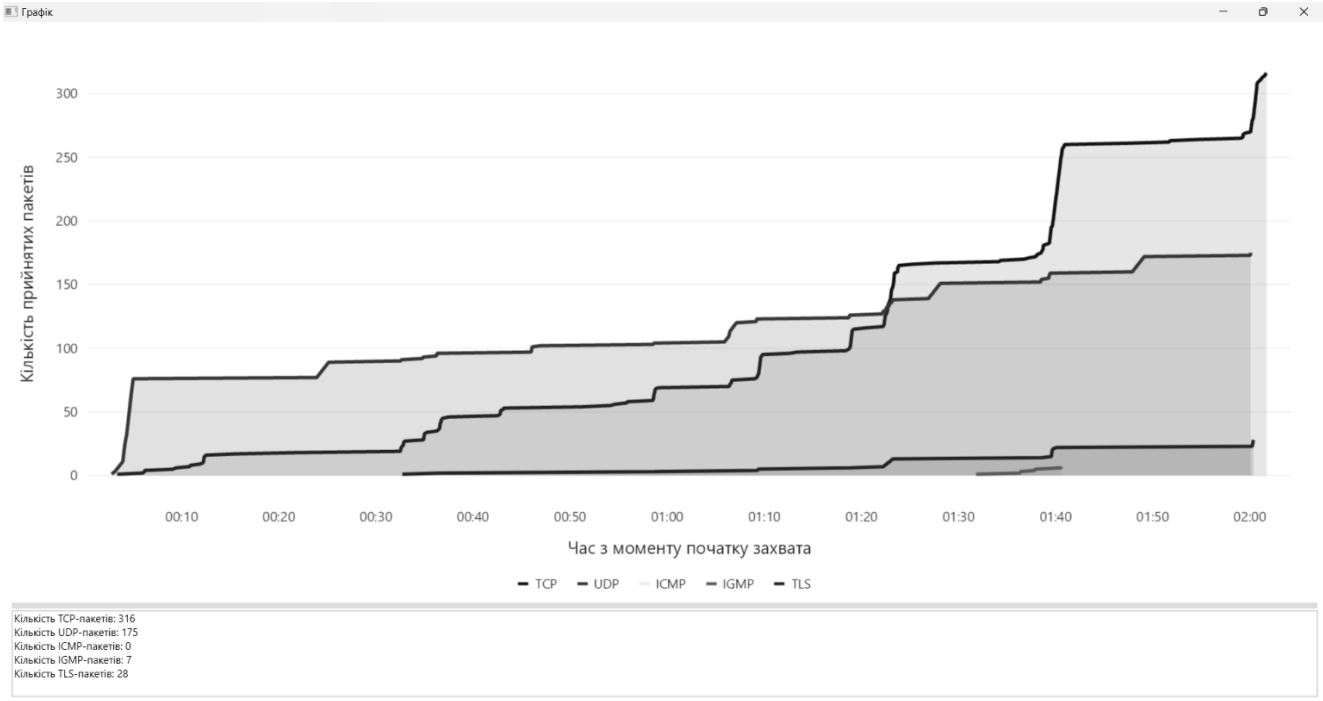

Рисунок 3.7 — Графік залежності кількості пакетів від часу початку захоплення

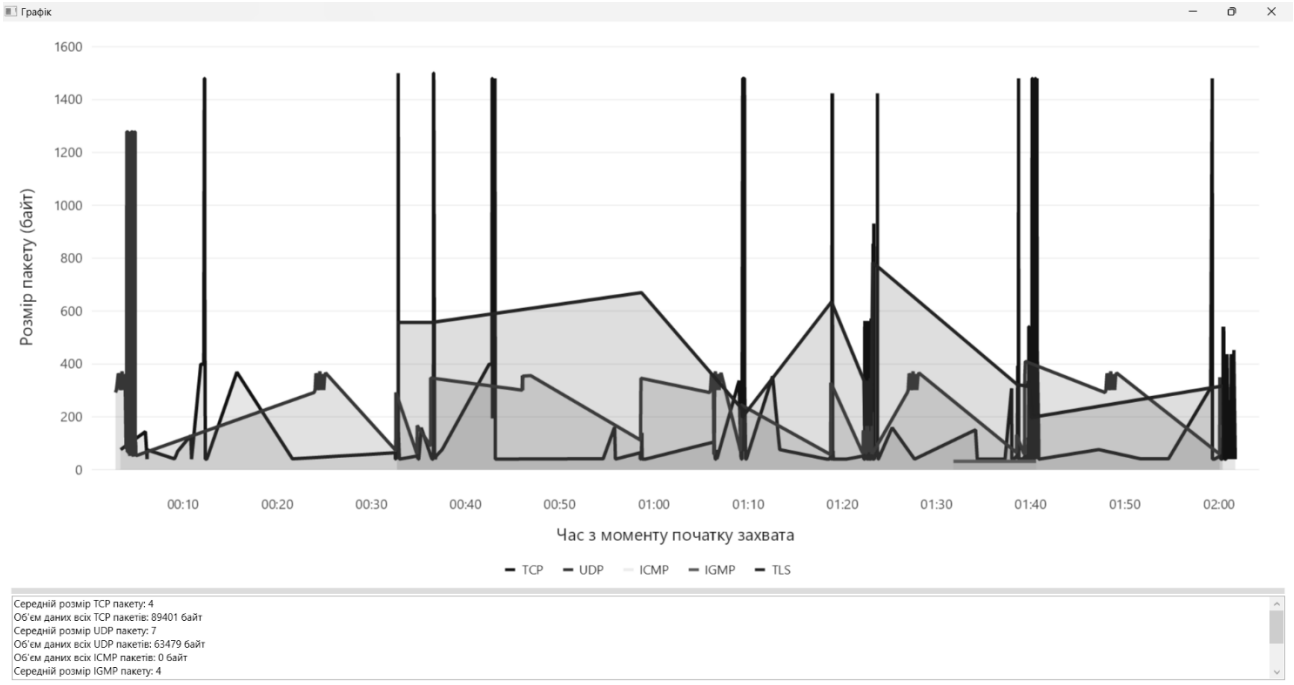

Рисунок 3.8 — Графік залежності розміру пакетів від часу початку захоплення

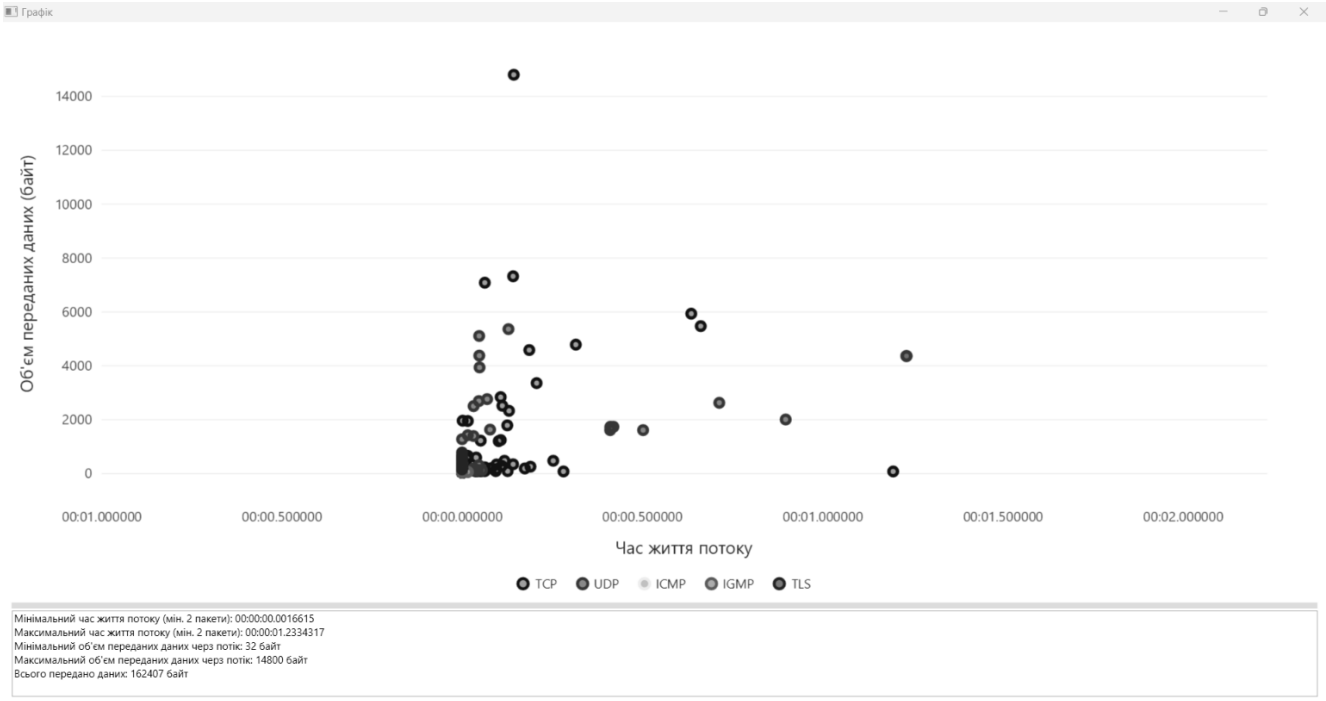

Рисунок 3.9 — Розсіювальний графік потоків

Інтерфейс головного вікна наведено на рис. Г.1.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи досліджено та опрацьовано низку теоретичних і практичних аспектів передачі пакетів у комп’ютерних мережах. По-перше, здійснено детальний аналіз моделей OSI та TCP/IP, наведено класифікацію основних мережевих протоколів (ICMP, IGMP, TCP, UDP), а також розглянуто структуру їхніх заголовків і механізми обробки помилок та керування потоком даних. По-друге, на основі отриманих теоретичних знань спроектовано й реалізовано програмний засіб для виведення та аналізу мережевого трафіку на базі C# із використанням сирих сокетів (Raw Socket) і елементів інтерфейсу WPF.

Розроблений додаток дозволяє у реальному часі захоплювати IP-пакети, аналізувати їхні заголовки та поля даних (включно з Type і Code у ICMP, Length і Checksum у UDP/TCP), а також відображати цю інформацію у зручній табличній формі з можливістю сортування та фільтрації. За допомогою графіків можна аналізувати навантаженість та стабільність мережевого трафіку. Застосування механізмів двостороннього зв’язування даних (data binding) і шаблонів оформлення забезпечило гнучкість інтерфейсу та спрощення подальших модифікацій програми. Практичні випробування показали стабільність роботи системи, здатність обробляти високі об’єми трафіку без втрат інформації і надавати користувачеві інструменти для швидкої діагностики мережевих проблем.

У перспективі доцільно розширити функціонал програми: реалізувати підтримку додаткових протоколів (SSL/TLS, QUIC), додати модулі статистичного аналізу й візуалізації залежностей трафіку, інтегрувати з базами даних для тривалого зберігання результатів моніторингу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Моніторинг комп'ютерної мережі комунікацій [Електронний ресурс] — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BD%D1%96%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3_%D0%BA%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%BE%D1%97_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D1%96.

2 Моніторинг мережі. Протоколи, найкращі практики, інструменти 2021 [Електронний ресурс] — Режим доступу до ресурсу: https://eska.global/blog/setevoj-monitoring-protokoly-luchshie-praktiki-instrumenty-2020.

3 Що таке пінгування та для чого воно використовується? [Електронний ресурс] — Режим доступу до ресурсу: https://freehost.com.ua/ukr/faq/faq/scho-take-pinguvannja-ta-dlja-chogo-vono-vikoristovujetsja/#:~:text=%D0%9F%D1%96%D0%BD%D0%B3%20(ping)%20%E2%80%94%20%D1%86%D0%B5%20%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81,%D0%B0%20%D1%82%D0%B0%D0%BA%D0%BE%D0%B6%20%D1%87%D0%B0%D1%81%D0%BE%D0%BC%20%D0%BF%D0%BE%D0%B2%D0%B5%D1%80%D0%BD%D0%B5%D0%BD%D0%BD%D1%8F%20%D0%BD%D0%B0%D0%B7%D0%B0%D0%B4..

4 Fundamentals SNMP [Електронний ресурс] — Режим доступу до ресурсу: https://habr.com/ru/articles/888692/.

5 Performance Dashboard [Електронний ресурс] — Режим доступу до ресурсу: https://learn.microsoft.com/en-us/sql/relational-databases/performance/performance-dashboard?view=sql-server-ver16.

6 ПІДСИСТЕМА ДЗЕРКАЛЮВАННЯ ТРАФІКУ У МЕРЕЖІ НА БАЗІ ОБЛАДНАННЯ CISCO [Електронний ресурс] — Режим доступу до ресурсу: https://conf.ztu.edu.ua/wp-content/uploads/2017/11/3.pdf.

7 Network tap CISCO [Електронний ресурс] — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Network_tap?utm_source=chatgpt.com.

8 Захоплення мережевих пакетів [Електронний ресурс] — Режим доступу до ресурсу: https://help.keenetic.com/hc/uk/articles/360000401420-%D0%97%D0%B0%D1%85%D0%BE%D0%BF%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B5%D0%B2%D0%B8%D1%85-%D0%BF%D0%B0%D0%BA%D0%B5%D1%82%D1%96%D0%B2.

9 Голь В.Д., Ірха М.С. Системи передачі даних: конспект лекцій. Київ : ІСЗЗІ КПІ ім. Ігоря Сікорського, 2021. 126 с.

10 Віртуальне з'єднання [Електронний ресурс] — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%92%D1%96%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D0%B7%27%D1%94%D0%B4%D0%BD%D0%B0%D0%BD%D0%BD%D1%8F.

11 Лекція 11. Комутація пакетів OSI [Електронний ресурс] — Режим доступу до ресурсу: https://e-tk.lntu.edu.ua/mod/page/view.php?id=3561&forceview=1.

12 Лекція Еталонна модель OSI [Електронний ресурс] — Режим доступу до ресурсу: https://learn.ztu.edu.ua/pluginfile.php/320992/mod_resource/content/1/%D0%95%D1%82%D0%B0%D0%BB%D0%BE%D0%BD%D0%BD%D0%B0%20%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C%20OSI.pdf.

13 Основи та застосування протоколів TCP/IP, повний путівник OSI [Електронний ресурс] — Режим доступу до ресурсу: https://hackyourmom.com/kibervijna/osnovy-ta-zastosuvannya-protokoliv-tcp-ip-povnyj-putivnyk/.

14 Розробка та реалізація мережних протоколів: Навчальний посібник [Електронний ресурс]: навч. посіб. для студ. спеціальності 121 «Інженерія програмного забезпечення» та 126 «Інформаційні системи та технології»,

спеціалізації «Інженерія програмного забезпечення інформаційно управляючих систем» та «Інформаційне забезпечення робототехнічних систем»/ Б. Ю. Жураківський, І. О. Зенів; КПІ ім. Ігоря Сікорського. — Електронні текстові дані (1 файл: 11,5 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2020. – 462 с.

15 .NET [Електронний ресурс] — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/.NET.

16 Windows Presentation Foundation Overview [Електронний ресурс] — Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/.

17 Patterns - WPF Apps With The Model-View-ViewModel Design Pattern [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern.

18 LiveCharts2 [Електронний ресурс] — Режим доступу до ресурсу: https://livecharts.dev/#frameworks.

19 Socket Class [Електронний ресурс] — Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=net-8.0.

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н.. Азаров О.Д.

21.03.2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

“Система моніторингу та аналізу мережевого трафіку”

08-54.БКР.026.00.000 ТЗ

Керівник роботи к.т.н. доц. каф. ОТ

Тарновський М. Г

Студент групи 2СП–21б

Данчук Д. О.

ВНТУ 2025

1 Підстава для виконання комплексного бакалаврської кваліфікаційної роботи (БКР)

1.1 Актуальність роботи обумовлена тим, що системи моніторингу та аналізу мережевого трафіку відіграють ключеву роль у забезпечені безпеки та стабільності сучасних мереж. Постійне ускладнення протоколів передачі даних і поява нових сценаріїв обміну інформацією вимагають гнучких і масштабованих інструментів для збору, обробки та візуалізації статистики. Зростання кількості кібератак та витоків даних, своєчасне виявлення аномалій у мережевому трафіку набуває критично важливого значення для попередження інцидентів безпеки та оперативного реагування на них.

1.2 Наказ про затвердження теми БКР.

2 Мета БКР і призначення розробки

2.1 Мета роботи — вдосконалення та розширення функціональних можливостей засобів моніторингу та аналізу мережевого трафіку.

2.2 Призначення розробки — збір та аналізу вхідних й вихідних пакетів мережі для забезпечення фільтрації та класифікації трафіку за різними критеріями, що дає змогу виявити аномалії, загрози безпеці чи критичні збої в роботі мережі.

3 Вихідні дані для виконання БКР

3.1 Інформаційні джерела — спіцифікація з протоколів сімества сімества TCP/IP.

3.2 Платформа — операційна система Windows.

3.3 Технології розробки — мова програмування C#, платформа .NET Framework, графічна підсистема WPF, шаблон MVVM.

3.4 Бібліотека LiveCharts2 для побудови графіків та діаграм.

3.5 Середовище розробки — Visual Studio Community.

4 Вимоги до виконання БКР

4.1 Провести аналіз предметної області.

4.2 Вибрати та обгрунтувати аналогів.

4.3 Вибрати та обґрунтувати інструменти для розробки системи.

4.4 Розробити функціональні модулі системи.

4.5 Розробити інтерфейс програми.

5 Етапи БКП та очікувані результати

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

Таблиця А.1 — Етапи БКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз предметної області.	22.03.25	26.03.25	Вступ, Розділ 1
2	Вибір та обгрунтування аналогів	27.03.25	03.04.25	Розділ 1
3	Підбір та аргументація засобів розробки системи	04.04.25	11.04.25	Розділ 2
4	Створення системи моніторингу та аналізу мережевого трафіку	12.04.25	27.04.25	Розділ 3
5	Оформлення пояснювальної записки, графічного матеріалу і презентації	28.04.25	8.05.25	ПЗ, графічний матеріал і презентація
6	Підготовка та підпис супроводжуючих документів, нормоконтроль та тест на плагіат	09.05.25	13.05.25	Оформленні документи

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні і ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відгук наукового керівника, відгук рецензента, анотації до БКР українською та іноземною мовами.

7 Порядок контролю виконання та захисту БКР

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

8 Вимоги до оформлювання та порядок виконання БКР

8.1 При оформлюванні БКР використовуються:

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

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

— ГОСТ 2.104–2006 «Єдина система конструкторської документації. Основні написи»;

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

— документи на які посилаються у вище вказаних.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21.

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: ____система моніторигу та аналізу мережевого трафіку_____

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

Підрозділ _____кафедра обчислювальної техніки_____
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism _1,5_ %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____Азаров О.Д., зав. каф. ОТ _____ _____
(прізвище, ініціали, посада) (підпис)

_____ Тарновський М.Г., доц. каф. _____ _____
(прізвище, ініціали, посада) (підпис)

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

З висновком експертної комісії ознайомлений(-на)

Керівник _____ _____Тарновський М.Г., доц. каф._____
(підпис) (прізвище, ініціали, посада)

Здобувач _____ _____Данчук Д.О._____
(підпис) (прізвище, ініціали)

ДОДАТОК В

ГРАФІЧНА ЧАСТИНА

СИСТЕМА МОНІТОРИНГУ ТА АНАЛІЗУ МЕРЕЖЕВОГО ТРАФІКУ

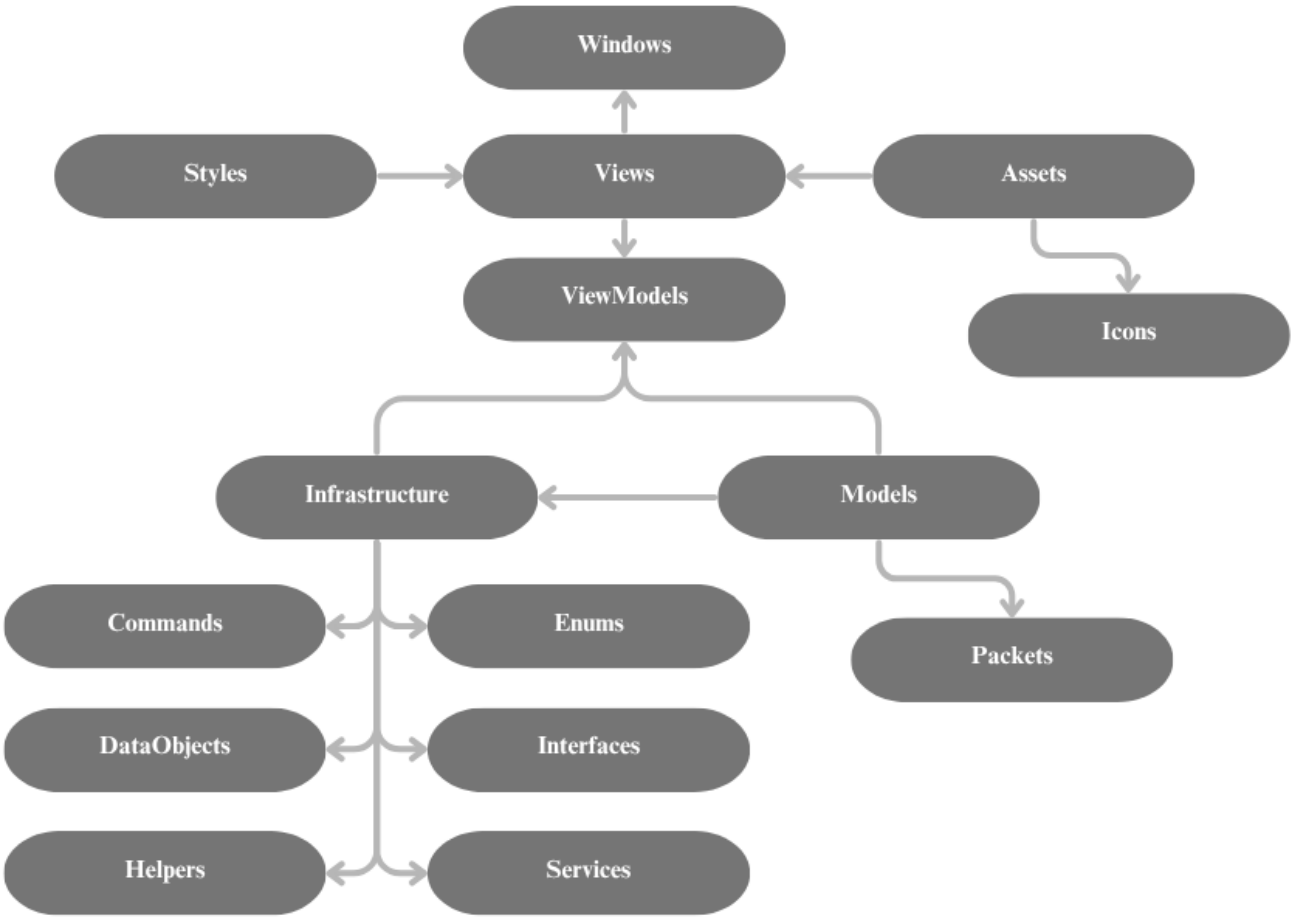

Рисунок В.1 — Блок-схема архітектури програми

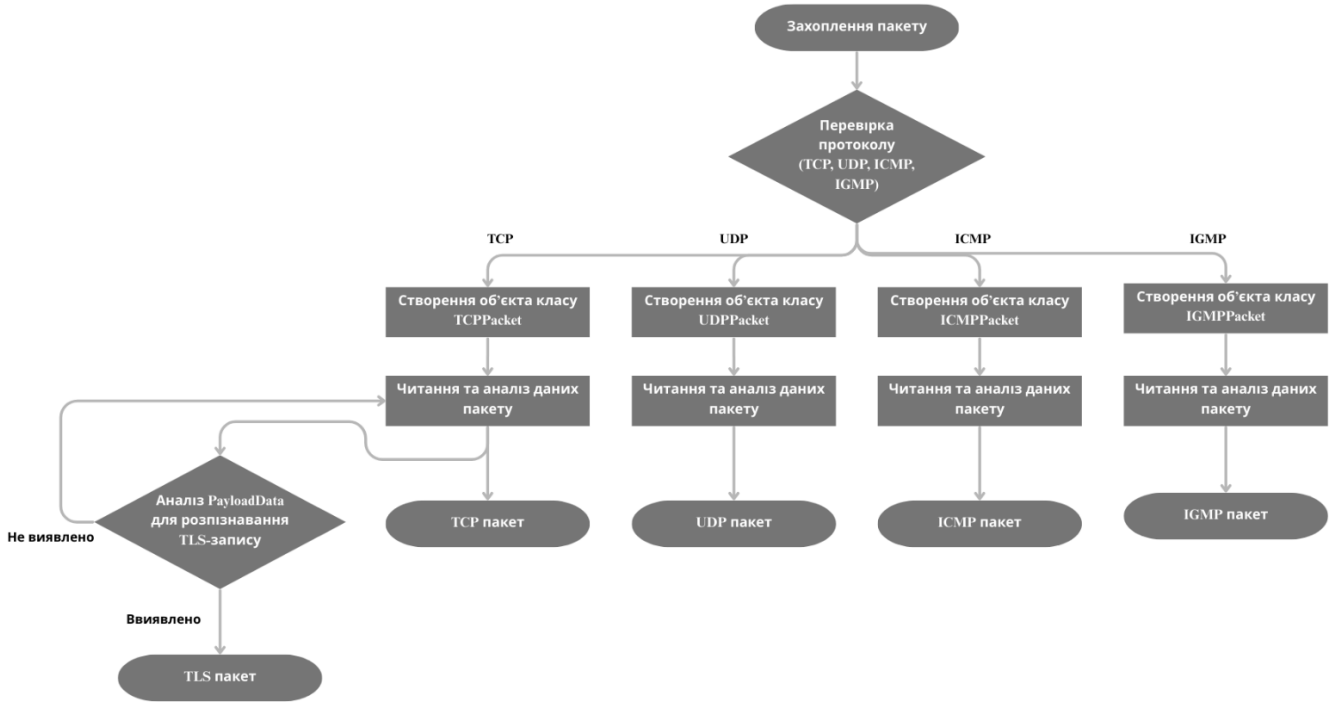

Рисунок В.2 — Блок-схема алгоритму захоплення пакетів

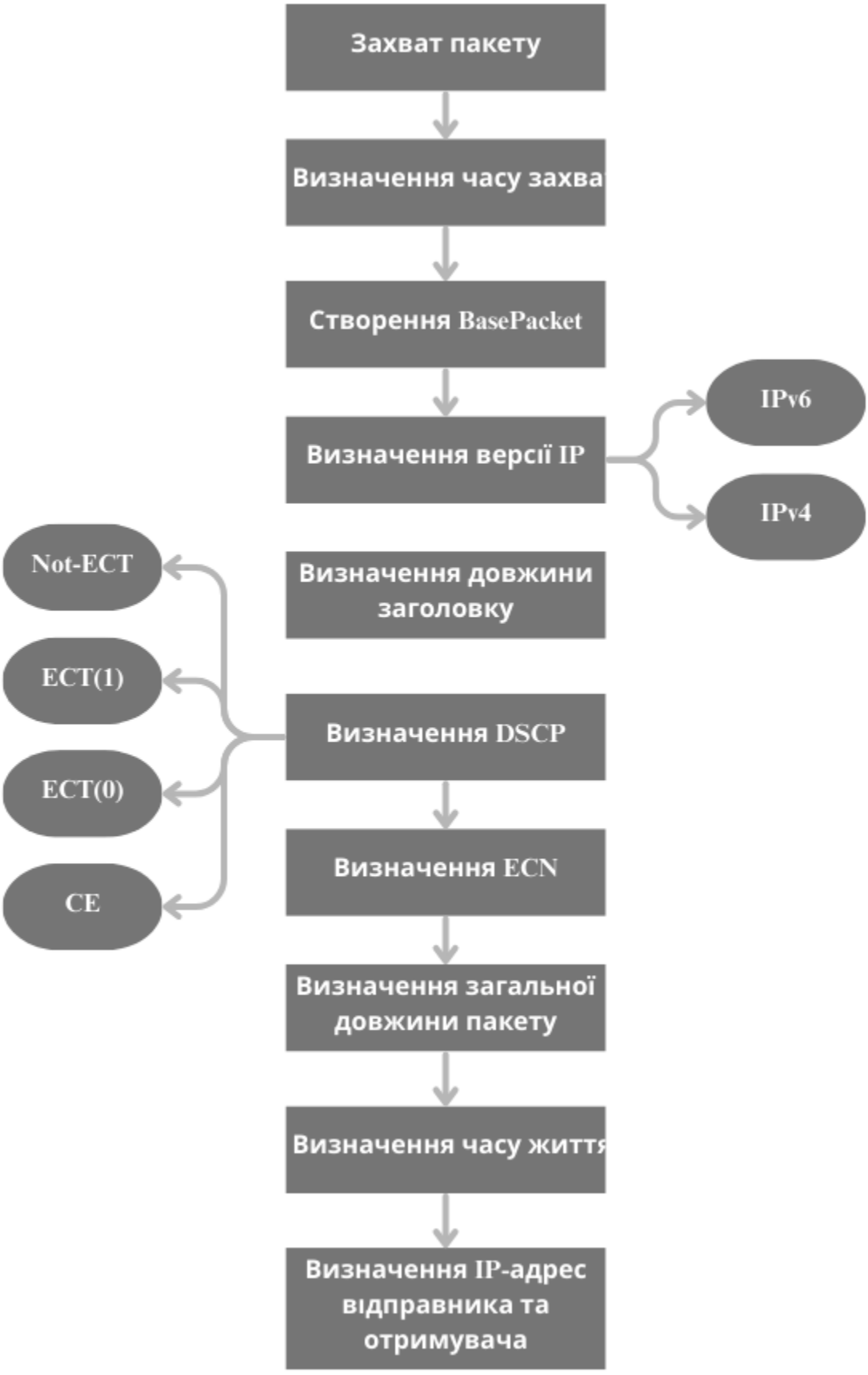

Рисунок В.3 — Блок-схема аналізу пакетів класу BasePacket

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

СИСТЕМА МОНІТОРИНГУ ТА АНАЛІЗУ МЕРЕЖЕВОГО ТРАФІКУ

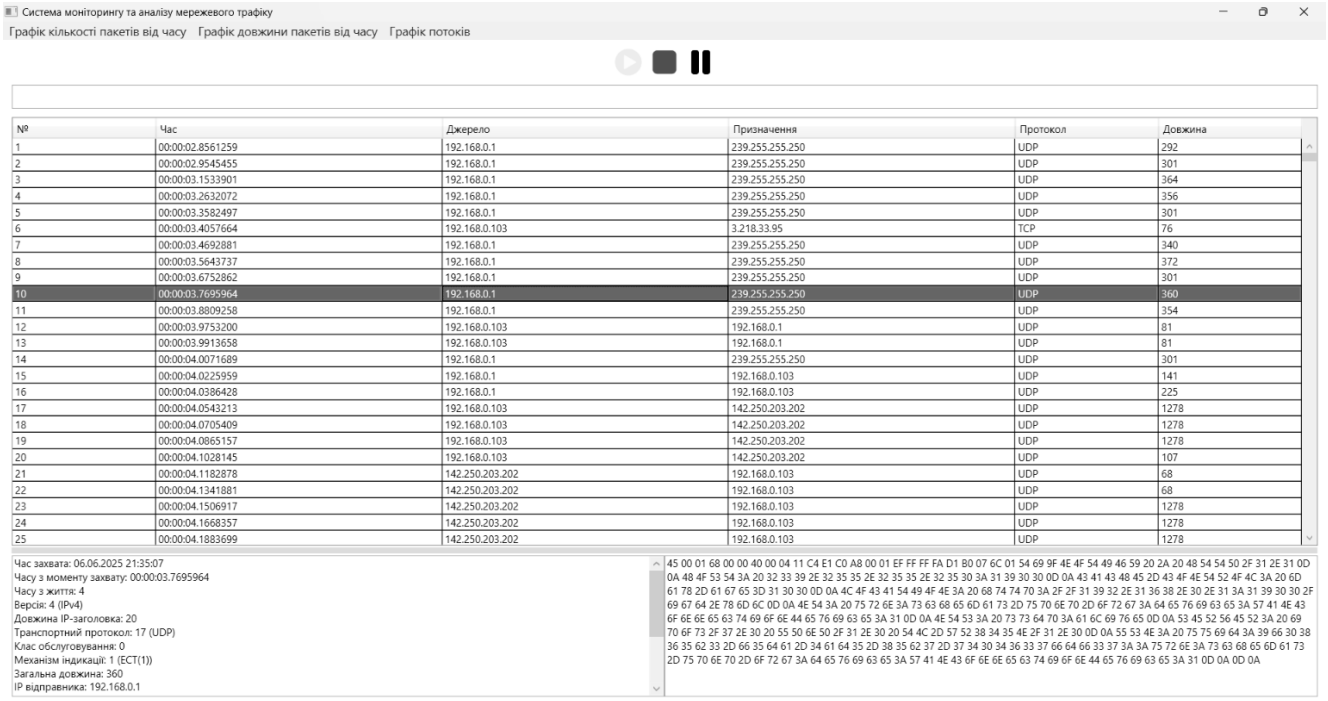

Рисунок Г.1 — Інтерфейс головного вікна

ДОДАТОК Д

Лістинг програмного забезпечення

Лістинг Д.1 – Повний код класу PacketsWatcher

```
using System.Net.Sockets;
using System.Net;
using System.Windows;
using System.Collections.ObjectModel;
using BachelorsThesis.Models.Packets;

namespace BachelorsThesis.Infrastructure.Services
{
    internal class PacketsWatcher : IDisposable
    {
        private CancellationTokenSource? _cts;

        private readonly ManualResetEventSlim _pauseEvent = new(true);

        public DateTime CaptureStartTime { get; private set; }

        public ObservableCollection<BasePacket> Packets { get; } = [];

        public void Start()
        {
            _cts?.Cancel();
            _cts = new CancellationTokenSource();
            _pauseEvent.Set();
            _ = Task.Run(() => StartMonitoringAsync(_cts.Token), _cts.Token);
            Packets.Clear();
        }

        public void Resume()
        {
            _pauseEvent.Set();
        }

        public void Pause()
        {
            _pauseEvent.Reset();
        }

        public void Stop()
        {
```

```
        _pauseEvent.Set();
        _cts?.Cancel();
    }

    private static Socket CreateSoket()
    {
        Socket socket = new(AddressFamily.InterNetwork, SocketType.Raw,
ProtocolType.IP);
        IPHostEntry host = Dns.GetHostEntry(Dns.GetHostName());
        IPAddress localIP = host.AddressList.FirstOrDefault(ip => ip.AddressFamily
== AddressFamily.InterNetwork)
            ?? throw new InvalidOperationException("Не знайдено локальну IPv4-
адресу.");
        socket.Bind(new IPEndPoint(localIP, 0));
        socket.SetSocketOption(SocketOptionLevel.IP,
SocketOptionName.HeaderIncluded, true);
        byte[] optionInValue = [1, 0, 0, 0];
        byte[] optionOutValue = new byte[4];
        socket.IOControl(IOControlCode.ReceiveAll, optionInValue,
optionOutValue);
        return socket;
    }

    private async Task StartMonitoringAsync(CancellationToken token)
    {
        try
        {
            using Socket socket = CreateSoket();
            byte[] buffer = new byte[65535];
            CaptureStartTime = DateTime.Now;

            while (!token.IsCancellationRequested)
            {
                _pauseEvent.Wait(token);
                if (socket.Available > 0)
                {
                    int bytesRead = await socket.ReceiveAsync(buffer,
SocketFlags.None);
                    if (bytesRead > 0)
                    {
                        byte[] realData = new byte[bytesRead];
                        Array.Copy(buffer, realData, bytesRead);
                        DateTime captureTimePacket = DateTime.Now;
                        ProtocolType protocol = (ProtocolType)realData[9];
```

```
                    switch (protocol)
                    {
                        case ProtocolType.Icmp:
                            AddPackeToPacketsList(new ICMPPacket(realData,
captureTimePacket));
                            break;
                        case ProtocolType.Igmp:
                            AddPackeToPacketsList(new IGMPPacket(realData,
captureTimePacket));
                            break;
                        case ProtocolType.Tcp:
                            AddPackeToPacketsList(new TCPPacket(realData,
captureTimePacket));
                            break;
                        case ProtocolType.Udp:
                            AddPackeToPacketsList(new UDPPacket(realData,
captureTimePacket));
                            break;
                        default:
                            break;
                    }
                }
            }
            await Task.Delay(1, token);
        }
    }
    catch (OperationCanceledException)
    {
        // моніторинг зупинено
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка: {ex.Message}");
    }
}

private void AddPackeToPacketsList(BasePacket packet)
{
    Application.Current.Dispatcher.Invoke(() =>
    {
        Packets.Add(packet);
    });
}
```

```
        public void Dispose()
        {
            Stop();
            _cts?.Dispose();
            _pauseEvent.Dispose();
        }
    }
}
```