МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна система моніторингу та аналізу мережевого трафіку»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Великодний Д.В.**

Студента групи ІНм-81н — **Боровик Е.О.**

СУМИ 2020

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«Інформатика»*

Затверджую:
зав. кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Боровику Едуарду Олександровичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)
Інформаційна система моніторингу та аналізу мережевого трафіку

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)

3. Вхідні данні до проекту (роботи)

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналітичний огляд відомих рішень та постановка задачі
2) Вибір методу перехоплення пакетів
3) Інформаційне та програмне забезпечення системи моніторингу та аналізу мережевого трафіку

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання_____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналітичний огляд відомих рішень та постановка задачі*		
2.	*Вибір методу перехоплення пакетів*		
3.	*Інформаційне та програмне забезпечення системи моніторингу та аналізу мережевого трафіку*		
4.	*Оформлення пояснювальної записки до дипломної роботи*		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 48 стор., 16 рис., 1 табл., 1 додаток, 10 джерел.

Мета роботи – розробка інформаційного та програмного забезпечення системи моніторингу та аналізу мережевого трафіку.

Об’єкт дослідження – трафік в локальній комп’ютерній мережі.

Предмет досліджень – характеристики навантаження, що створює трафік на локальну комп’ютерну мережу.

Методи дослідження – методи проектування. Застосування інструментарію C#, Windows Presentation Foundation та libpcap.

Результати – в ході дослідження розроблено програмне забезпечення інформаційної системи моніторингу та аналізу мережевого трафіку. Результати роботи дозволяють отримувати інформацію про кількість пакетів в мережі, їх характер, тип протоколів та навантаження на локальну комп’ютерну мережу. Графічний інтерфейс дозволяє гнучко налаштовувати фільтри та параметри відображення даних, що полегшує подальший аналіз проблемних частин мережі та прогнозування росту навантаження. Також існує можливість збереження даних в файл та побудова графіків.

ТРАФІК, ЛОКАЛЬНА КОМП’ЮТЕРНА МЕРЕЖА, WINPCAP,
NDIS, MOVING AVERAGE, PACKET CAPTURE, C#

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface

BPF – Berkeley Packet Filter

IP – Internet Protocol

ISDN – Integrated Services Digital Network

LAN – Local Area Network

MA – Moving Average

NDIS – Network Driver Interface Specification

NIC – Network Interface Controller

PCAP – Packet Capture

TAPI – Telephony Application Programming Interface

TCP – Transmission Control Protocol

TDI – Transport Driver Interface

WAN – Wide Area Network

WPF – Windows Presentation Foundation

XAML – eXtensible Application Markup Language

ВСТУП

На сьогоднішній день значно зросла роль комп’ютерних мереж у зв’язку зі швидким розвитком інформаційно-комунікаційних технологій. Досить актуальною є задача, що полягає у створенні універсальних засобів для управління та моніторингу мереж.

Необхідно усвідомлювати наскільки важливою є безперервна та правильна робота локальної комп'ютерної мережі. Проблема аналізу трафіку постає при створенні мережевих додатків, при налаштуванні нового обладнання та при вивченні принципів роботи протоколів передачі даних.

Глобальне поширення комп’ютерів та швидкісного інтернету дає можливість обмінюватися інформацією в великих обсягах. Для забезпечення безпеки даних виникає завдання створення програмного та інформаційного забезпечення контролю комп’ютерних мереж. За допомогою мережевих аналізаторів вирішуються такі завдання, як:

- аналіз стану мережі для усунення проблем;
- відновлення потоків даних між користувачами;
- дослідження статистики та прогнозування навантаження.

Таким чином, в рамках даної роботи необхідно провести дослідження існуючих методів перехоплення пакетів та аналізу отриманих даних. Метою роботи є створення програмного додатку для аналізу мережевого трафіку локальної комп’ютерної мережі.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Огляд існуючих рішень

Усвідомивши необхідність обліку і контролю мережевого трафіку, користувачі зазвичай зштовхуються з іншою проблемою: як обрати відповідний додаток. Існує досить багато програмних продуктів, що надають можливість аналізувати мережевий трафік. Але більша їх частина призначена для ОС сімейства UNIX. Багато з програм, що працюють під Windows, занадто дорогі, перевантажені функціональністю і не відрізняються гнучкістю налаштувань.

В основному, додатки для аналізу трафіку ґрунтуються на трьох технологіях: прослуховування пакетів за допомогою ввімкнення режиму прослуховування на мережевий платі, аналіз інформації, наданої іншою програмою або апаратурою та обробка потоків даних спеціальним драйвером (див. рисунок 1.1). Розглянемо переваги і недоліки цих підходів.

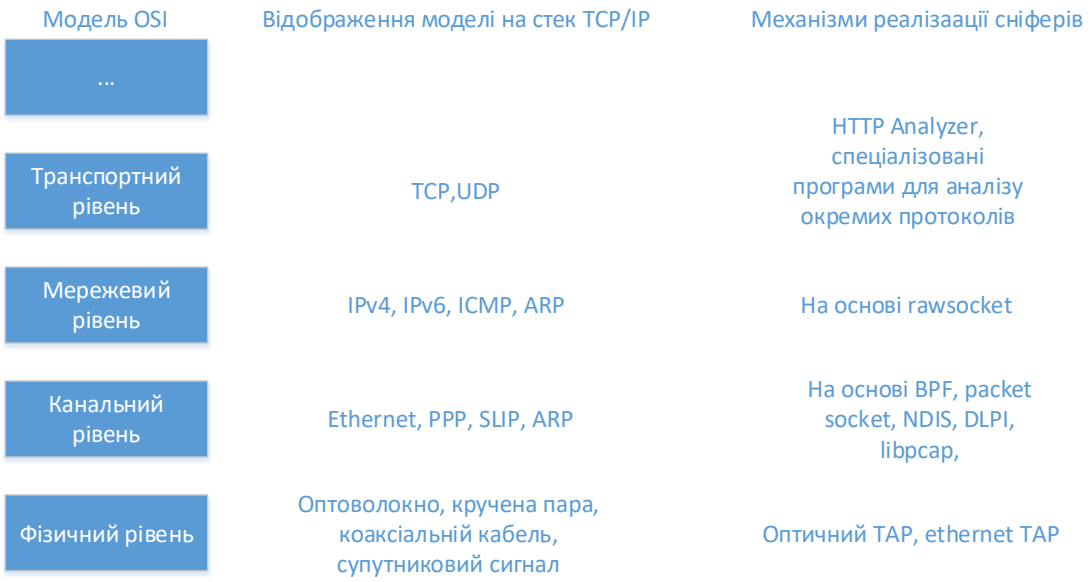

Рисунок 1.1 – Реалізація сніферів відносно моделі OSI [1]

Програми для прослуховування мережевого трафіку завдяки особливості архітектури Ethernet. Інформацію, що передається по мережі, отримує кожен пристрій. За замовчуванням мережевому інтерфейсу комп'ютера доступні тільки ті дані, що призначені саме для нього. Але прослуховуючі програми вмикають режим прийому всіх пакетів. В основі таких додатків лежать мережеві драйвери

і бібліотеки, які здійснюють більшу частину роботи. При інтенсивному завантаженні на локальну мережу комп'ютер може виявитися нездатними обробити всю інформацію, що надходить. Цим способом можна відстежувати об’єм трафіку, але не можна його обмежувати [1].

Недоліком другого методу є те, що додатки, які рахують об’єм трафіку на основі інформації отриманої з іншої програми або спеціального обладнання, не є повністю самостійними і незалежними. Це вимагає додаткових фінансових витрат і позбавляє даний підхід універсальності. До того ж при роботі через proxy-сервер підраховується не весь трафік. До переваг даного методу відноситься те, що початкова інформація залишається незайманою, тому при зміні умов можна заново налаштувати фільтри і виконати аналіз заново [1].

Додатки для обліку трафіку, засновані на драйвері, відрізняються точністю і високою продуктивністю. До можливих проблем можна віднести ситуацію: при великому навантаженні деякі пакети можуть втрачатися. Потрібно врахувати і той факт, що будь-який драйвер працює в режимі ядра, тому проблеми в додатку можуть вплинути на функціонування системи.

В даній роботі планується використання методу обліку трафіку, коли мережевий інтерфейс переходить в режим прослуховування, користувацькому додатку передається інформація про мережеві пакети через архітектуру WinPCAP. Ця програма може використовуватися незалежно від встановленого обладнання і програмного забезпечення на платформі MS Windows.

1.2 Аналоги готових продуктів

1.2.1 Microsoft Network Monitor

Мережевий аналізатор Network Monitor, створений компаніями Windows та Microsoft Systems Management Server, дозволяє здійснювати моніторинг мережевого трафіку. Існує можливість моніторингу мережевого трафіку в реальному часі, також можливо перехоплювати трафік, зберігати та аналізувати його пізніше. Збережені дані можуть використовуватися для усунення проблем

в локальних та розподілених мережах, а також з більшістю мережевих пристроїв, що здійснюють зв’язок за допомогою стеку протоколів TCP / IP [2].

До переваг Network Monitor відносяться такі особливості, як:

Загальним напрямком застосування являється пошук проблем в мережевих з'єднаннях. Network Monitor також використовується для перегляду пакетів стеку протоколів TCP / IP, це дозволяє побачити інформацію, яка передається між мережевими пристроями.

Інструментарій додатку дозволяє проводити оцінку продуктивності мережі. Інформація про навантаження, джерела трафіку та слабкі місця в мережі допомагає в створенні об’єктивної оцінки роботи ЛКМ.

Можливість пошуку обладнання, яке ініціювало видачу так званого «маяка». Це процес транслювання в мережу повідомлення про припинення передачі маркера через помилку в мережі переважно кільцевої топології. Зараз це не так актуально, адже переважна більшість мереж комутовані, але як і раніше існує можливість використовувати Network Monitor для пошуку фрагментованих пакетів [2].

Приклад графічного інтерфейсу мережевого аналізатора Network Monitor:

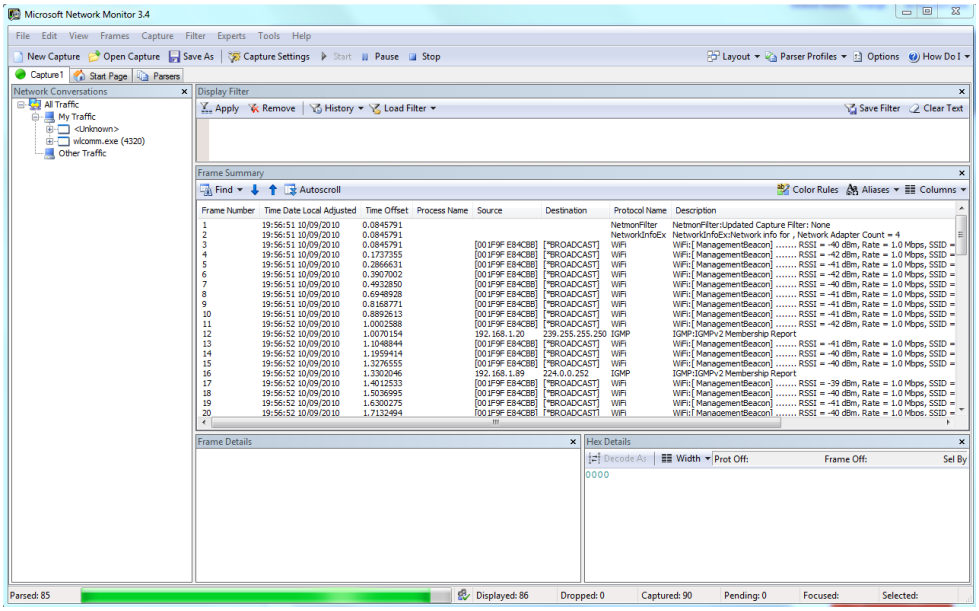

Рисунок 1.2 – Приклад програми Network Monitor

Розглянутий мережевий аналізатор має наступні недоліки:

- підтримка тільки Windows;
- складний громіздкий інтерфейс;
- високе навантаження на комп'ютер;
- менш детальне декодування протоколів в порівнянні з комерційними продуктами.

1.2.2 LanHound

LanHound складається з двох продуктів: адміністративна консоль та віддалений агент перехоплення пакетів (консоль являється агентом). Даний мережевий аналізатор працює тільки під управлінням операційних систем Windows.

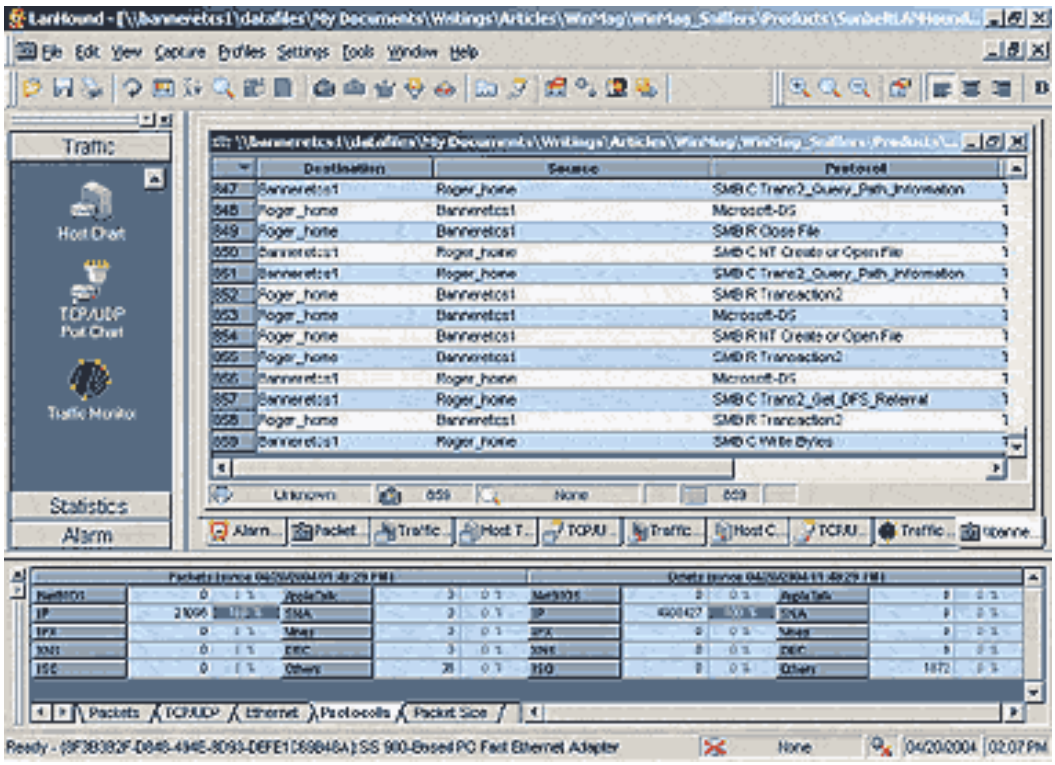

Рисунок 1.3 – Приклад програми LanHound

LanHound має простий графічний інтерфейс (див. рисунок 1.3) і набір функцій, стандартний для даного типу додатків, включаючи фільтрацію перехопленого трафіку, пошук по імені, попередження і створення звітів. До складу LanHound входить модуль експертного аналізу, лише ненабагато виходить за рамки можливостей функції попереджень. Звіти містять таблиці

активних пристроїв, гістограми, матриці трафіку, підсумкові звіти по пакетам. Також дані, зібрані LanHound, можна обробляти по-різному, однією з функцій є побудова гістограм і діаграм. Цікавою особливістю є те, що LanHound дозволяє маніпулювати збереженим трафіком і навіть повторно транслювати його по мережі – функція, не завжди доступна в недорогих продуктах такого класу.

До недоліків програми відноситься:

- висока ціна;
- слабка дешифрація багатьох протоколів;
- не працює під управлінням Windows 2003, 2008;
- відсутня модуль експертного аналізу.

1.2.3 CommView

CommView поширюється лише на комерційній основі. Для ознайомлення доступна демонстраційна версія з урізаною функціональністю. Даний пакетний аналізатор призначений для моніторингу як локальної мережі, так і підключення до глобальної. Він здатний захоплювати пакети, що проходять через мережевий інтерфейс, декодувати їх і представляти досить детальну інформацію в зручному вигляді. На відміну від більшості схожих додатків, CommView не вимагає попереднього встановлення WinPcap.

CommView підтримує операційні системи Windows. Перелік протоколів досить великий, і в цьому плані можна розраховувати на надання досить докладної інформації про перехоплених пакетах.

Пакетний аналізатор CommView підтримує можливість створення фільтрів (правил), проте недоліком тут є те, що ці фільтри не можна застосувати до вже наявних зібраним пакетам (створювані правила поширюються лише на захоплення пакетів). Графічний інтерфейс програми традиційний – три вікна, в першому з яких відображаються захоплені пакети, в другому – декодована інформація про окремому пакеті, а в третьому – вміст самого пакета (див. рисунок 1.4).

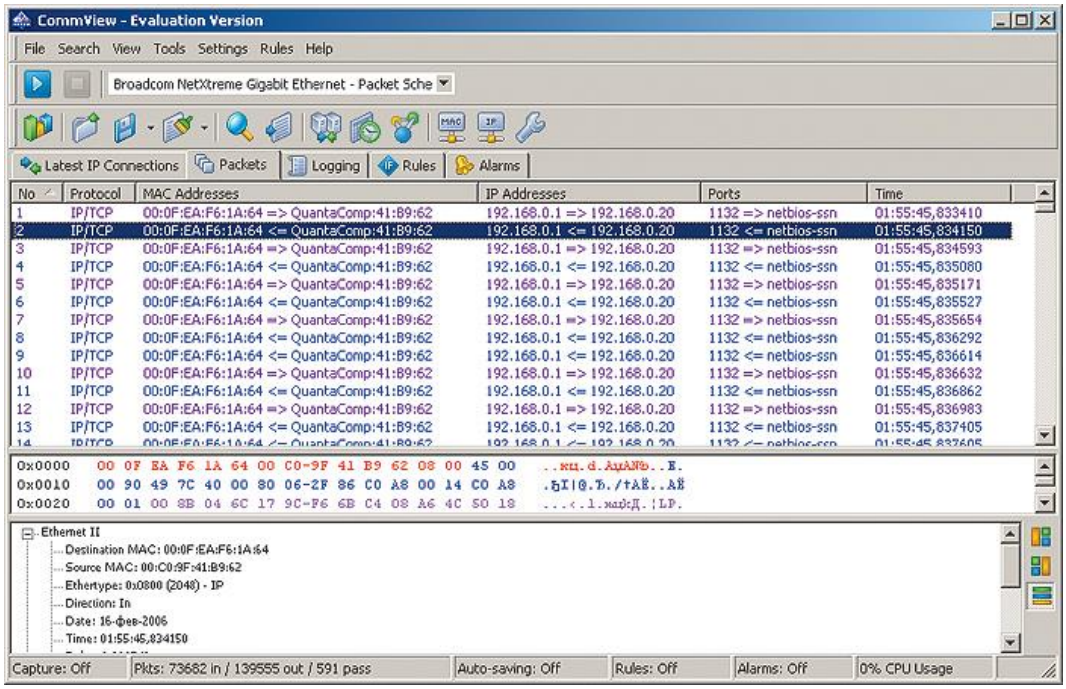

Рисунок 1.4 – Приклад програми CommView

До недоліків наведеного пакетного аналізатора належить:

- висока ціна;
- відсутність деяких Windows-декодерів, таких як Kerberos і RDP;
- роздільні версії для локальної і бездротової мережі;
- для деяких мережевих адаптерів підтримується тільки змішаний режим.

1.2.4 Wireshark

Наступним та найбільш популярним мережевим аналізатором є Wireshark.

Функціонал дозволяє не тільки перехоплення пакетів, але й деякі розширені інструменти для аналізу. Wireshark являє собою додаток з відкритим програмним кодом та існують версії майже для всіх операційних систем, зокрема серверних. Під назвою Etheral, Wireshark працює майже скрізь, зокрема в якості автономного портативного додатка [1].

Існує можливість роботи в графічному інтерфейсі (див. рисунок 1.5), що дозволяє використовувати весь інструментарій безпосередньо на робочому місці. При роботі на віддаленому сервері на через консоль можна зберігати дані в файли з розширенням .pcap, та проводити аналіз вже на локальній машині.

При запуску додатка можна завантажити файл з існуючими даними або ж почати перехоплення пакетів. Також існує можливість налаштування фільтрів.

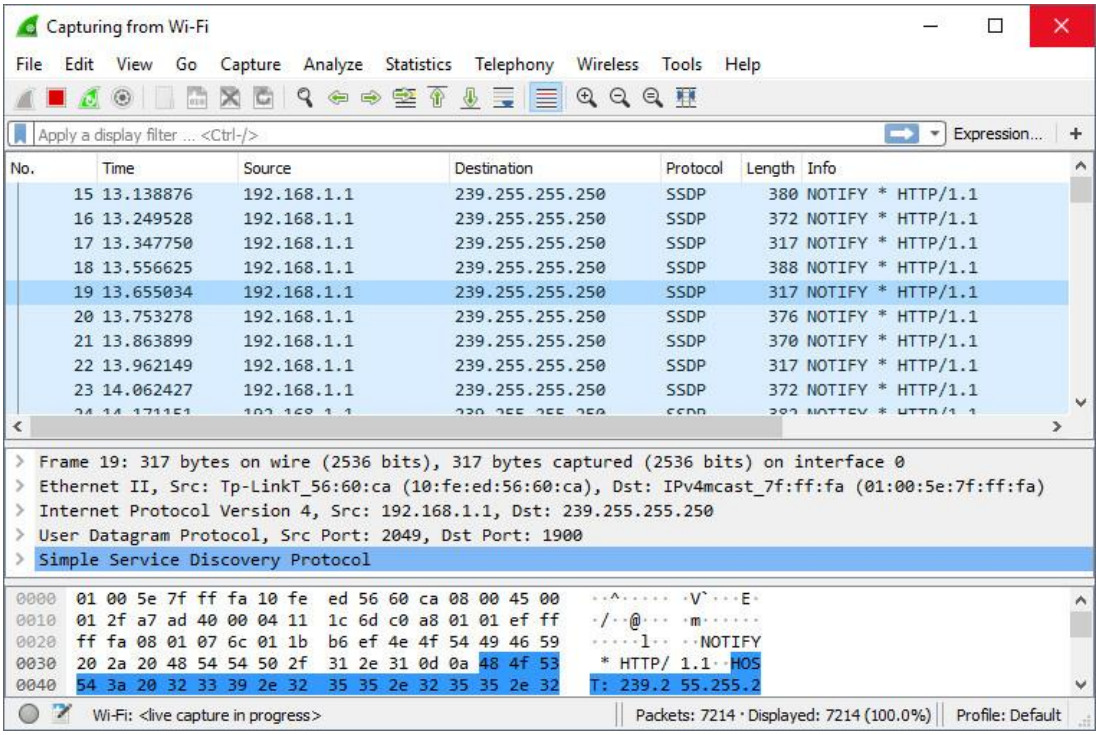

Рисунок 1.5 – Приклад програми Wireshark

До недоліків аналізатору Wireshark можна віднести:

- високий рівень складності налаштування фільтрів;
- перевантажений інтерфейс;
- менша продуктивність в порівняні з комерційними продуктами.

1.3 Постановка задачі

В даній роботі вирішується завдання розробки об'єктно-орієнтованого гнучкого програмного забезпечення для захоплення та аналізу мережевого трафіку.

Для цього необхідно вирішити наступні завдання:

1. Реалізувати функціонал захоплення трафіку.
    a. Реалізувати модуль отримання інформації про мережеві інтерфейси.
    b. Можливість самостійно обирати з якого інтерфейсу захоплювати трафік.

c. Задання довжини періоду отримання даних.

d. Можливість зберегти дані.

2. Реалізувати відображення даних.

a. Можливість застосування користувацьких фільтрів.

b. Вибір, які саме дані необхідно відображати.

3. Реалізувати модуль відображення даних.

a. Реалізувати модуль побудови графіку.

b. Можливість обрати стиль побудови графіку.

c. Реалізувати алгоритм Moving Average.

d. Можливість зберегти відображені дані.

4. Реалізувати програмне забезпечення таким чином, щоб зберігалася можливість його використання без графічного інтерфейсу (передбачити можливість інтерфейсу командної строки).

5. Всі дані про помилки, які виникли під час роботи програми, повинні записуватись в окремий файл.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Технологія захоплення трафіку WinPCAP

Архітектура WinPCAP розширює стандартні функції операційних систем сімейства Win32 можливістю приймати і передавати дані по мережі, уникаючи стек протоколів операційної системи та взаємодіючи прямо з мережевим адаптером комп'ютера. Більш того, вона надає додаткам API високого рівня для управління низькорівневими процесами [3]. WinPCAP складається з трьох компонентів:

- packet.vxd – драйвер пристрою захоплення пакетів;
- packet.dll – динамічна бібліотека низького рівня;
- libpcap – статична бібліотека високого рівня.

Дана архітектура використовується для створення додатків обробки пакетів під управлінням операційних систем Windows та UNIX.

На рисунку 2.1 приведена структура стека захоплення пакетів від мережевого адаптера до додатка верхнього рівня.

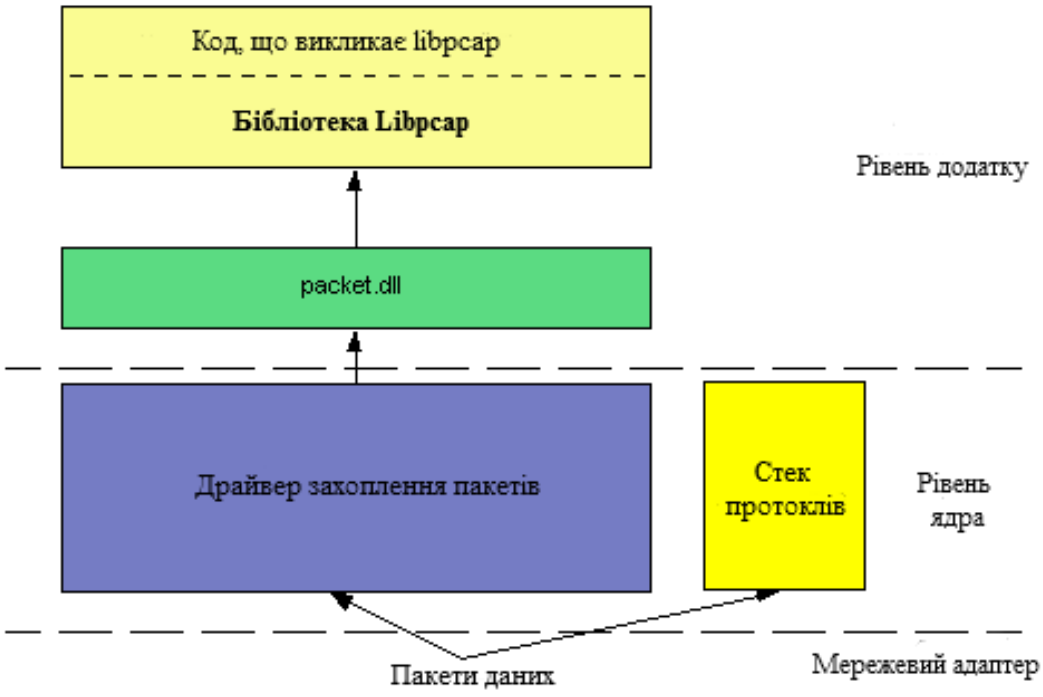

Рисунок 2.1 – Структура стека захоплення пакетів

На нижньому рівні знаходиться мережевий адаптер, який приймає всі пакети, що передаються через мережу. Драйвер захоплення пакетів packet.vxd являє собою програмний модуль низького рівня. Він працює на рівні ядра операційної системи і взаємодіє безпосередньо з драйвером мережевого адаптера. Pcap-драйвер дає набір функцій низького рівня, що забезпечують прийом і передачу даних на канальному рівні через NDIS. Даний драйвер відповідає за управління різними типами адаптерів і забезпечує зв'язок адаптера з програмним забезпеченням, що відповідає за формування пакетів різної структури [4].

Динамічна бібліотека packet.dll відокремлює додаток користувача від драйвера і дає додаткам незалежний від виду операційної системи інтерфейс. Це дозволяє додатку працювати на різних версіях Windows без перекомпіляції. Бібліотека packet.dll працює на рівні користувача, але окремо від програми.

Статична бібліотека libpcap використовується частиною програми користувача, що забезпечує перехоплення і фільтрацію пакетів. Вона задіє функції, що надаються бібліотекою packet.dll, і забезпечує програмі користувача управління процесами прийому і фільтрації даних на високому рівні. Бібліотека libpcap статично пов'язана з програмою користувача і є її частиною.

Специфікація NDIS визначає три типи мережевих драйверів:

- драйвери протоколів верхнього рівня;
- драйвери протоколів проміжного рівня ;
- драйвери мережевих карт (NIC), сюди входять драйвери мережевих карт локальних мереж (NDIS LAN Miniport NIC) і драйвери мережевих карт глобальних мереж (NDIS WAN Miniport) [4].

2.1.1 Драйвери

Драйвер протоколу верхнього рівня у своїй верхній частині дає TDI-інтерфейс або інший, що необхідний користувачам. Подібні драйвери виділяють ресурси для пакетів, копіюють дані в пакети і передають їх драйверам нижнього

рівня за допомогою використання NDIS. Нижня частина цього типу драйверів забезпечує інтерфейс для отримання даних від нижнього драйвера і передає отримані дані TDI клієнту або додаткам [5].

До драйверів протоколів верхнього рівня відноситься транспортний драйвер, що реалізує стек таких протоколів, як IPX/SPX або TCP/IP. Транспортний драйвер Windows NT реалізує TDI-інтерфейс в своїй верхній частині і використовує NDIS бібліотеку для взаємодії з драйверами мережевих карт в своїй нижній частині.

NDIS – драйвери протоколів проміжного рівня взаємодіють з драйверами протоколів верхнього рівня, а в нижній частині з NDIS драйверами мережевих адаптерів локальних і глобальних мереж. NDIS драйвери проміжного рівня можуть бути драйверами, які не підтримують інтерфейс NDIS. NDIS драйвери проміжного рівня використовуються для різних методів, включаючи фільтрацію, шифрування пакетів та подібних функцій [5].

Для драйвера протоколу верхнього рівня проміжний драйвер має вигляд драйверу віртуальної мережевої карти. Для драйвера реальної мережевої карти – як драйвер протоколу. Проміжні драйвери можуть створювати ланцюг, проте це може чинити негативний вплив на продуктивність системи.

NDIS драйвер проміжного рівня експортує «Miniport» функції на своєму верхньому рівні та «Protocol» функції на своєму нижньому рівні. Рідше проміжний драйвер може експортувати «Miniport» функції на своєму верхньому рівні, а в своїй нижній частині надавати закритий інтерфейс за допомогою використання пакетів IRP, нижньому драйверу, яка не є NDIS драйвером.

Частина інтерфейсу TAPI розташовується в режимі ядра і реалізується драйвером ndistapi.sys, який взаємодіє з драйвером ndiswan.sys. Драйвер ndiswan.sys – це NDIS драйвер проміжного рівня, що забезпечує PPP форматування, автентифікацію, стиснення і шифрування для драйверів мережевих карт. Він перетворює пакет формату NDIS_PACKET, одержаний від драйвера транспорту, в пакет формату NDIS_WAN_PACKET, і передає переформатований пакет нижньому драйверу мережевих адаптерів [5].

Ndiswan.sys надає інтерфейс стандарту 802.3 верхнім драйверам протоколів та має роль драйвера протоколу для нижніх драйверів мережевих адаптерів. Ndiswan.sys має закритий інтерфейс з драйвером ndistapi.sys для динамічного встановлення та завершення TAPI зв'язків. Ndiswan.sys перетворює формат що відправляється пакета з LAN в PPP. Велика частина розбивки фреймів, специфічною для середовища передачі, повинна виконуватися в драйвері мережевої карти глобальної мережі.

NDIS LAN Miniport NIC драйвери підтримують мережеві карти локальних мереж. NDIS WAN Miniport NIC драйвери мають додаткові вимоги і надають додаткам доступ до глобальних мереж, таким як, ISDN, Frame Relay, Switched 56.

Драйвери мережевих карт в своїй нижній частині взаємодіють з обладнанням, а у верхній частині надають інтерфейс, що дозволяє верхнім рівням посилати пакети в мережу, скидати і зупиняти мережеву карту, а також здійснювати опитування і установку параметрів драйвера мережевої карти.

Загальний вигляд структури NDIS з двома стеками захоплення пакетів, прив'язаних до одного мережевого адаптера, представлений на рисунку 2.2.

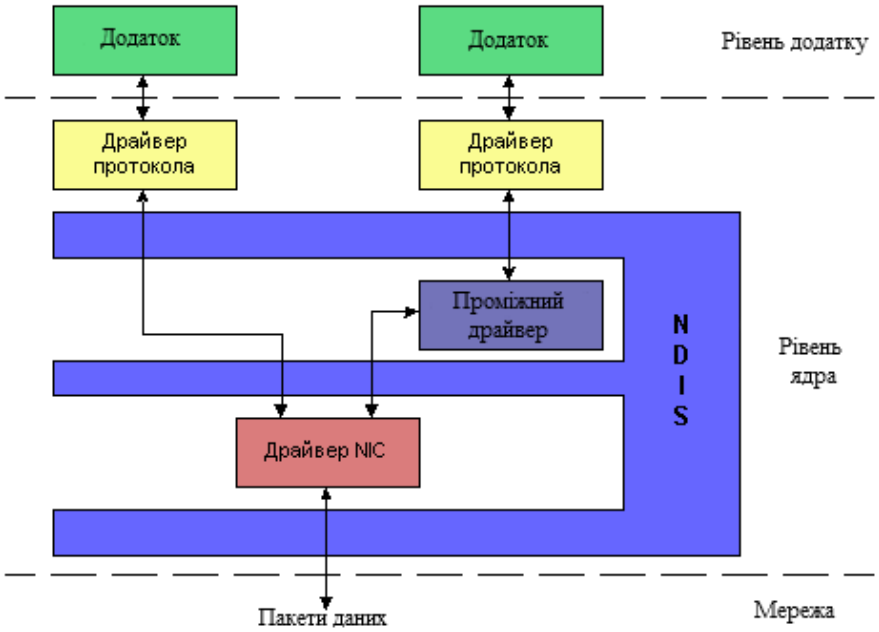

Рисунок 2.2 – Структура NDIS

Один з них складається з драйвера NIC і драйвера протоколу, інший – з драйвера NIC, проміжного драйвера і драйвера протоколу.

Існує два типи NDIS драйверів мережевих карт:

- Miniport NIC драйвер виконує специфічні для конкретної мережевої карти операції, включаючи відправлення і отримання даних. Операції загальні для всіх драйверів мережевих карт, такі як синхронізація, зазвичай забезпечуються NDIS. Miniport драйвер не викликає безпосередньо обслуговування операційної системи, він взаємодіє з операційною системою через NDIS.
- Full NIC legasy драйвери в даний час застаріли і використовуються для сумісності, вони виконують одночасно специфічні для мережевої карти операції, а також всю роботу по синхронізації, обслуговування черг, звичайно що виконуються NDIS. Full NIC драйвер, наприклад, підтримує свою власну інформацію про привязки до верхніх драйверів для індикації отриманих даних. Miniport драйвер, навпаки, не зберігає інформацію про привязки, він просто передає пакети наверх інтерфейсу NDIS, а той вже гарантує, що пакети будуть передані відповідним драйверам протоколів.

Для нормальної роботи драйверу захоплення пакетів необхідно взаємодіяти як з драйвером мережевого пристрою (для передачі і прийому даних), так і з додатком користувача (для отримання від нього даних або передачі йому прийнятих від мережевого пристрою пакетів). Тому драйвер захоплення пакетів розроблений як драйвер протоколу в структурі NDIS (див. рисунок 2.3).

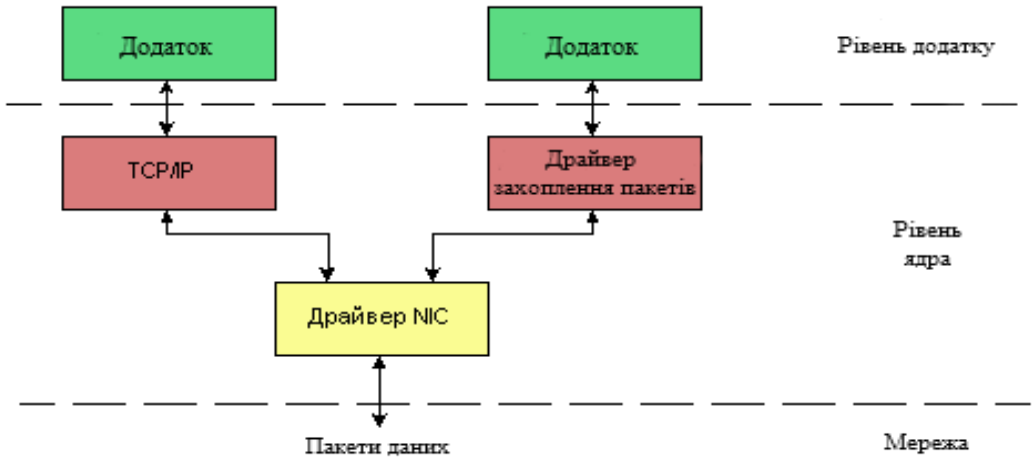

Рисунок 2.3 – Положення драйвера захоплення пакетів в структурі NDIS

Це дозволяє йому працювати з усіма мережевими пристроями, підтримуваними операційними системами Windows. Проте, на даний момент драйвер працює тільки з адаптерами Ethernet, loopback-адаптерами і забезпечує зв’язок з глобальною мережею через Ethernet-адаптер. Пакет протоколів PPP NCP-LCP «прозорий» для драйверів протоколів, оскільки PPP-з'єднання встановлюються віртуально. Тому драйвер захоплення пакетів не може працювати з такого роду сполукам [5].

2.1.2 Делегати

Для написання інформаційної системи необхідно вирішити проблему зі зверненням до компонентів форми з іншого потоку, який відрізняється від потоку, де захоплюються пакети. Для вирішення цієї проблеми необхідно створити метод для роботи з відповідними компонентами. Даний метод використовується для звернення до компонентів з іншого потоку. Він є асинхронним, що дозволяє використовувати не перериваючи роботи системи. Як параметри йому передається делегат і параметри делегата.

Делегати являють собою посиланнями на методи, вони дозволяють інкапсулювати справжні покажчики і надають зручні сервіси для роботи з ними. Посилання є об'єктами відповідного типу. Всі делегати є об'єктами та мають два типи: System.Delegate та System.MulticastDelegate.

Різниця між цими класами полягає в тому, що делегати можуть зберігати лише одне посилання на метод, а екземпляри другого можуть містити відразу кілька посилань на методи. Завдяки цьому, можна приєднувати до одного делегату кілька методів, кожен з яких при єдиному зверненні до делегату буде викликатися по ланцюжку. Таким чином, з програми буде видно лише один делегат, за яким ховається кілька методів (див. рисунок 2.4).

Така особливість делегатів дає можливість підтримки подій, оскільки дозволяє без використання додаткових механізмів приєднати до події кілька функцій обробників. Фактично, делегат представляє собою об'єкт, що приховує в собі покажчики на функції. Делегати, по суті справи, нічим не відрізняються

від звичайних об'єктів. Головна їхня особливість полягає лише в тому, що вони мають підтримку з боку середовища виконання. Про їх властивості, на відміну від звичайних об'єктів, знають навіть компілятори, що надають зручні спеціальні сервіси для роботи з ними [6].

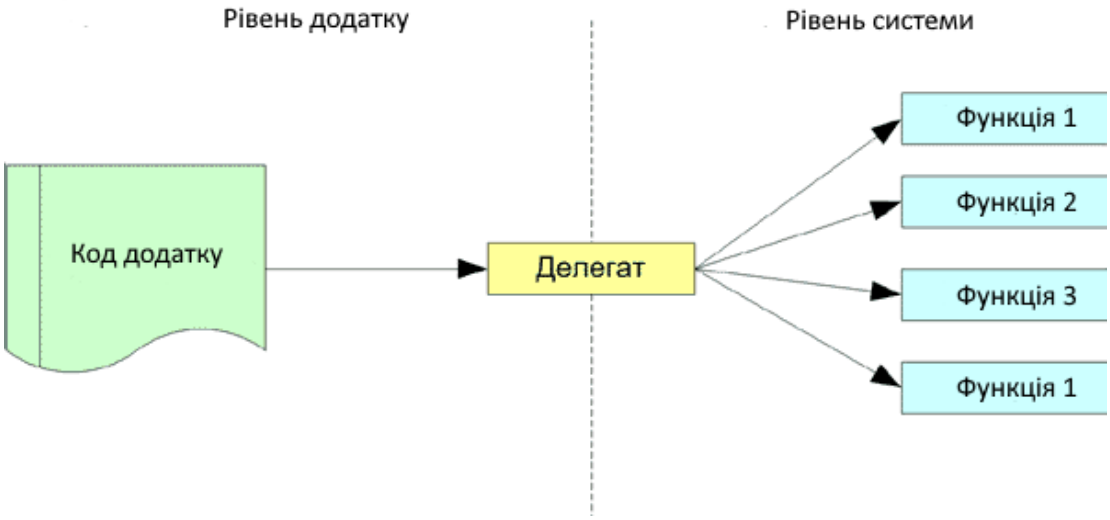

Рисунок 2.4 – Структура застосування делегатів

Методи в середовищі .NET розділяються на дві групи: static і instance. Якщо делегат посилається на статичний метод, то в цьому випадку є вся інформація для виклику методу: адреса методу і параметри. Якщо ж делегат посилається на екземплярний метод, то завдання змінюється. Для того, щоб викликати метод, делегату повинен мати інформацію про об'єкт, з яким пов’язаний певний метод. Посилання зберігається в самому об'єкті делегата і вказується при його створенні. Протягом усього існування об'єкта це посилання не змінює свого значення, вона завжди постійне і може бути задане тільки при його створенні [7].

Таким чином, незалежно від того, посилається чи делегат на статичну функцію або на екземплярний метод, звернення до нього не відрізнятиметься. Всю необхідну функціональність забезпечує сам делегат, укупі із середовищем виконання. Це значно спрощує задачу, оскільки безліч різних делегатів можна прив'язувати до однієї події.

2.2 Метод аналізу трафіку

Для побудови графіків та для більш наглядної демонстрації навантаження було вирішено використовувати середні ковзаючі лінії. Це сімейство функцій, значення яких в кожній точці свого значення рівне середньому значенню вихідної функції. Ковзаючі лінії зазвичай використовуються для більш плавного відображення зміни значення даних в невеликих часових проміжках, що дає можливість відзначити тенденції та періодичність повторення.

Середні ковзаючі лінії бувають трьох видів:

- прості;
- зважені;
- експоненціальні.

Прості ковзаючі лінії – чисельно дорівнюють середньому арифметичному значенню вихідної функції за встановлений період і розраховується згідно формули (2.1):

$$SMA_{t=}\ \frac{1}{n}\sum_{i=0}^{n-1} p_{t-i} = \frac{p_t+p_{t-1}+\cdots p_{t-i}\cdots+p_{t-n+2}+p_{t-n+1}}{n} \qquad (2.1)$$

де SMA_t – значення в точці t;

n – кількість значень вихідної функції для розрахунку змінного середнього;

p_{t-i} – значення вихідної функції в точці t-i.

Виділяють наступні недоліки простої ковзаючої середньої лінії:

- ваговий коефіцієнт має значення 1;
- подвійна реакція на кожне значення.

При побудові ковзної середньої, деякі значення вихідної функції доцільно зробити більш значущими.

Зважені ковзаючі середні лінії – при обчисленні функції вага кожного члена вихідної функції дорівнює відповідному члену арифметичній прогресії. Тобто, при обчисленні для тимчасового ряду, останні значення вихідної функції більш значущі ніж попередні, причому функція значущості лінійно спадна.

Для арифметичної прогресії з початковим значенням і та кроком 1, формула обчислення ковзної середньої має такий вигляд (2.2):

$$WMA_t = \frac{2}{n\times(n+1)}\sum_{i=0}^{n-1}(n-i)\times p_{t-i} \quad (2.2)$$

де WMA_t – значення в точці *t*;

n – кількість значень вихідної функції для розрахунку змінного середнього;

p_{t-i} – значення вихідної функції в момент часу, віддалений від поточного на *i* інтервалів.

Знаменник функції дорівнює сумі членів арифметичної прогресії з початковим членом і кроком рівними 1:

$$\frac{n\times(n+1)}{2} \quad (2.3)$$

Експоненціальні ковзаючі середні лінії – різновид зваженої ковзної, вага якої зменшуються експоненціально і ніколи не дорівнюють нулю. Формула має вигляд:

$$EMA_t = \alpha \times p_t + (1-\alpha)\times EMA_{t-1} \quad (2.4)$$

де EMA_t – значення в точці *t*;

EMA_{t-1} – експоненціальне значення в точці *t-1*, p_t- значення вихідної функції в момент часу *t*;

α – коефіцієнт, що характеризує швидкість зменшення ваги, набуває значення від 0 і до 1.

Коефіцієнт α може бути обраний довільним чином, в межах від 0 до 1, наприклад, виражений через величину різниці усереднення:

$$\alpha = \frac{2}{n+1} \quad (2.5)$$

Під час аналізу мережевого трафіка на коротких проміжках часу доцільно використовувати експоненціальні ковзаючі середні лінії, адже вони дають можливість побачити більш динамічні зміни в навантаженні на мережу. В свою чергу прості середні ковзаючі лінії більш раціонально використовувати при тривалих періодах дослідження.

2.3 Засоби розробки програмного забезпечення

Для реалізації програмного забезпечення було вирішено використати мову програмування C #. В порівнянні з іншими мовами програмування, C# має більш широкий набір інструментів для роботи з мережею, мережевими інтерфейсами та для аналізу трафіку. Велика кількість бібліотек дає можливість працювати з різноманітнім обладнанням та трафіком. Також дана мова має спеціалізований інструментарій для створення графічного інтерфейсу.

C# – одна з багатьох мов програмування платформи .NET. Вона являється об'єктно-орієнтованою і дозволяє створювати компоненти для багаторазового використання в найрізноманітніших типах програмного забезпечення. Дана мова програмування значно схожа на своїх попередників (C++, Java), але спираючись на практику їх використання вдалося позбутися моделей, які були проблематичними при розробці. Важливим моментом є те, що C # є «керованою» мовою. З цього випливає, що для роботи додатків потрібен .NET Common Language Runtime (CLR) [8].

C# забезпечує можливість реалізації графічних додатків. Технологія WPF прийшла на зміну WinForms та має більш широкий функціонал. Це підсистема для створення графічного інтерфейсу. WPF являється частиною платформи .NET. Додатки створені за допомогою WPF не залежать від роздільної здатності обладнання та мають векторну систему візуалізації, адже технологія заснована на DirectX. Це дає можливість використовувати апаратне прискорення графіки та оптимізувати роботу комп’ютера . Ще однією особливістю є можливість використання як і мови декларативної розмітки інтерфейсу XAML, так і код створений на мові C# [9].

Схематично архітектура WPF представлена на рисунку. 2.5:

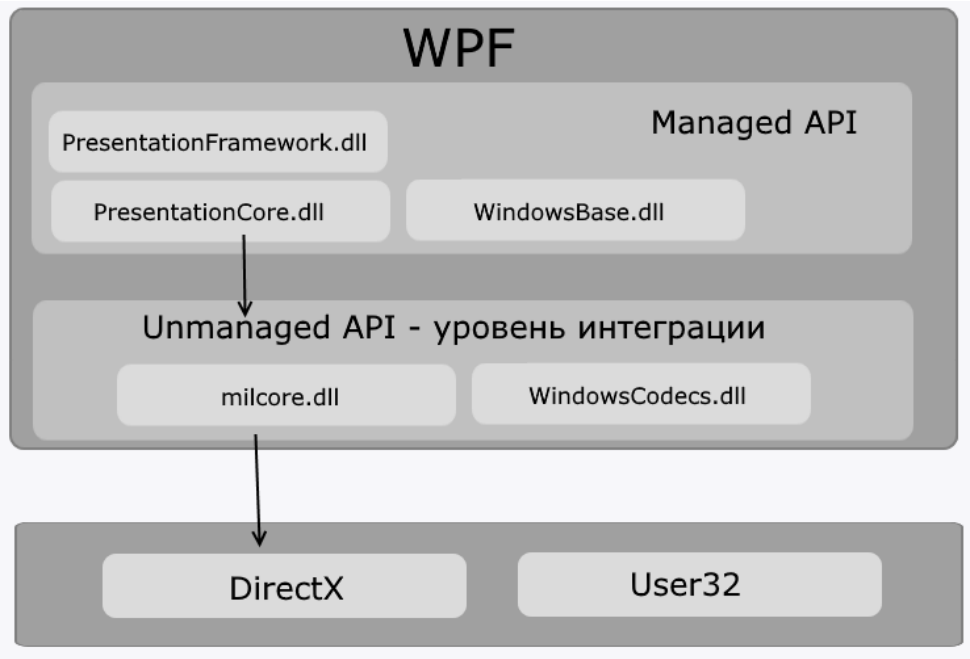

Рисунок 2.5 – Архітектура WPF

В якості середовища розробки було вирішено використовувати середовище Microsoft Visual Studio та редактор Visual Studio Code. Це спеціалізовані інструменти, які дозволяють швидко проводити розробку програмного забезпечення. Microsoft Visual Studio дозволяє створювати додатки з використанням технології WPF, а також забезпечує роботу з веб-додатками. Visual Studio Code можна використовувати, коли необхідна проста та швидка відладка коду, адже він не потребує багато ресурсів. Visual Studio також дозволяє підключати плагіни, розширювати набори інструментів це значно полегшує розробку програмного забезпечення[10].

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Опис функціоналу програми

Перщим кроком створення інформаційної системи моніторингу та аналізу мережевого трафіку є створення проекту з графічним інтерфейсом WPF. Консольний варіант додатку не відповідає поставленій задачі через відсутність інструментів для управління. В структурі системи можна виділити три головних складові:

- інтерфейсна частина;
- логічна частина;
- допоміжні бібліотеки.

Інтерфейс реалізований в вигляді графічного вікна, в якому розміщені основні робочі області в різних вкладках, елементи управління та налаштування, а також поля виведення інформації.

До логічної частини відноситься процес контролю мережевого трафіку, який можна розділити на два етапи – моніторинг та аналіз. На етапі моніторингу виконується первинний процес збору даних , що дає інформацію про кількість пакетів, об’єми даних, кількість мережевих пристроїв, їх адреси і протоколи, що використовуються. Етап аналізу даних являє собою більш складну задачу, що включає в себе процес дослідження зібраних даних, порівняння з отриманою раніше інформацією та з’ясування причин нестабільної або повільно роботи мережі.

Допоміжні бібліотеки необхідні для доступу до існуючих мережевих підключень, інтерфейсів комп’ютера та для здійснення перехоплення трафіку (пакетів). Таким чином потрібно підключити такі бібліотеки як «SharpPcap.dll» і «Pcap.NET.dll». Бібліотека «Pcap» встановлюється з пакетом «WinPcap», тому її підключення не потрібне. Також необхідно підключити бібліотеку System.Net.NetworkInformation.

Під час реалізації додатку для функціонування основних алгоритмів системи розроблено наступні класи та методи для роботи з даними:

- Pocket – клас, що реалізує алгоритм перехоплення пакетів за допомогою технології WinPcap.
- NetwDevice – клас, за допомогою якого визначаються доступні мережеві інтерфейси, що можуть здійснювати перехоплення.
- TabControlPage – клас відображення інформації в вкладках (tabs).
- Settings – клас, що відповідає за налаштування типу даних, що мають відображатися.
- PocketsPage – клас, що реалізує відображення отриманих даних в таблиці.
- Graph – клас, що забезпечує відображення отриманих даних у вигляді таблиці з використанням технології MovingAverage.
- LoadLineChartData – клас необхідний для збереження даних в файл.
- Command – клас реалізації інтерфейсу, що необхідний для обробки введеної інформації.
- HelloPage – клас відображення початкових налаштувань системи.
- Lib – статичний клас, до якого винесений допоміжний функціонал.

У таблиці 3.1 наведені основні методи, що використовуються в даній інформаційній системі:

Таблиця 3.1 – Основні методи

Назва	Опис
ExecuteAsAdmin	Перехід функціонування додатку з правами доступу адміністратора.
isWinPcapInstalled	Перевірка наявності бібліотеки WinPcap в системі.
getDevices	Отримання даних доступних інтерфейсів.
SettingsBtn_Click	Відображення системних налаштувань.
ChoosePathBtn_Click	Вибір шляху для збереження даних.

Продовження таблиці 3.1

GraphBtn_Click	Відображення графіку розміру пакетів в часі.
GetFilterPredicat	Налаштування фільтрів.
ApplyFilterBtn_Click	Застосування фільтрів.
StartCapt	Виконується початок перехоплення пакетів.
Program_OnPacketArrival	Подія, що відображає появу нового пакета.
PauseCapt	Призупиняє процес перехоплення пакетів.
StopCapt	Виконується завершення перехоплення пакетів.

В даній блок-схемі показаний алгоритм функціонування інформаційної системи:

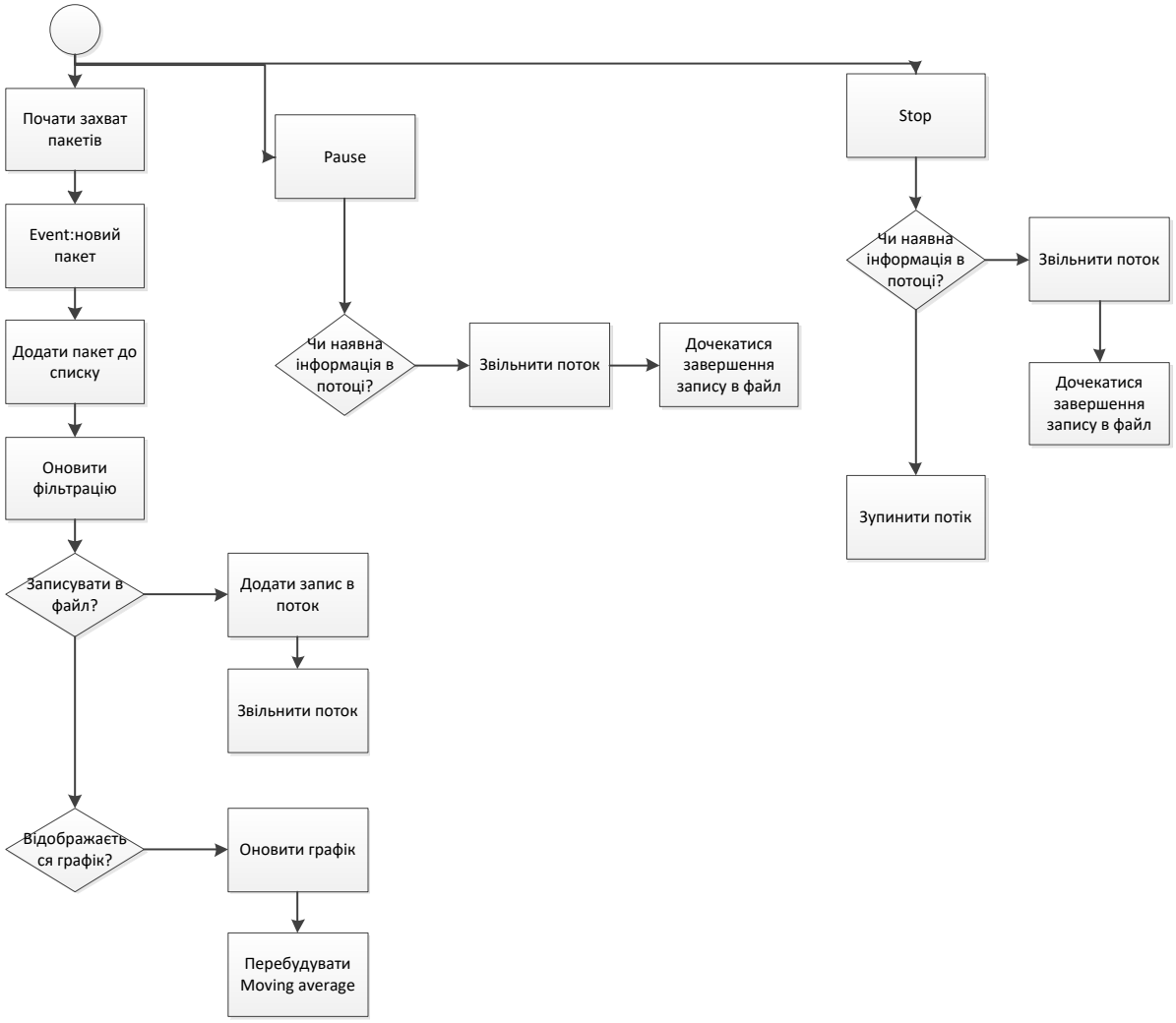

Рисунок 3.1 – Блок-схема алгоритму роботи програми

3.2 Опис програмного забезпечення

Перш за все, перед початком роботи з даною системою необхідно впевнитися, що на комп'ютері присутні мережеві інтерфейси, що вони підключені до мережі та правильно налаштовані. Також необхідно впевнитися, що існують клієнти які знаходяться в мережі.

Для запуску додатку існують рекомендації та обов'язкові умови. До рекомендацій належить мінімальна конфігурація комп'ютера на якому запускається додаток:

- Процесор intel Core 2 Duo, AMD Sempron та вище;
- Об'єм вільної оперативної пам'яті не менше 512 Мб;
- Об'єм жорсткого диска: не менше 256 Мб вільного місця;
- Операційна система не нижче Windows 7.

До обов'язкових умов належать:

- Наявність мережевого інтерфейсу, що підтримує режим прослуховування мережі;
- Версія NET.FrameWork не нижче 4.5.26;
- Версія WinPcap не ниже 4.1.3;
- Мережеві екрани мають бути вимкненими.

Перед запуском програми, необхідно впевнитися, що директорія з виконуваним файлом має вигляд, як показано на рисунку 3.2:

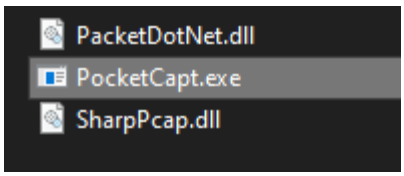

Рисунок 3.2 – Вміст папки перед запуском додатку

Упевнившись, що в системі інстальовано пакет WinPcap та присутні необхідні бібліотеки можна переходити до запуску розробленого додатку «PocketCapt».

Робота в додатку починається з вкладки «Interfaces» (див. рисунок 3.3). Спочатку користувачеві необхідно визначити наявні мережеві пристрої та інтерфейси для перехоплення пакетів. Обравши інтерфейс користувач може побачити в правій частині вікна таку інформацію, як:

- системну назву інтерфейсу;
- користувацька назва інтерфейсу;
- фізичну адресу;
- IP адресу.

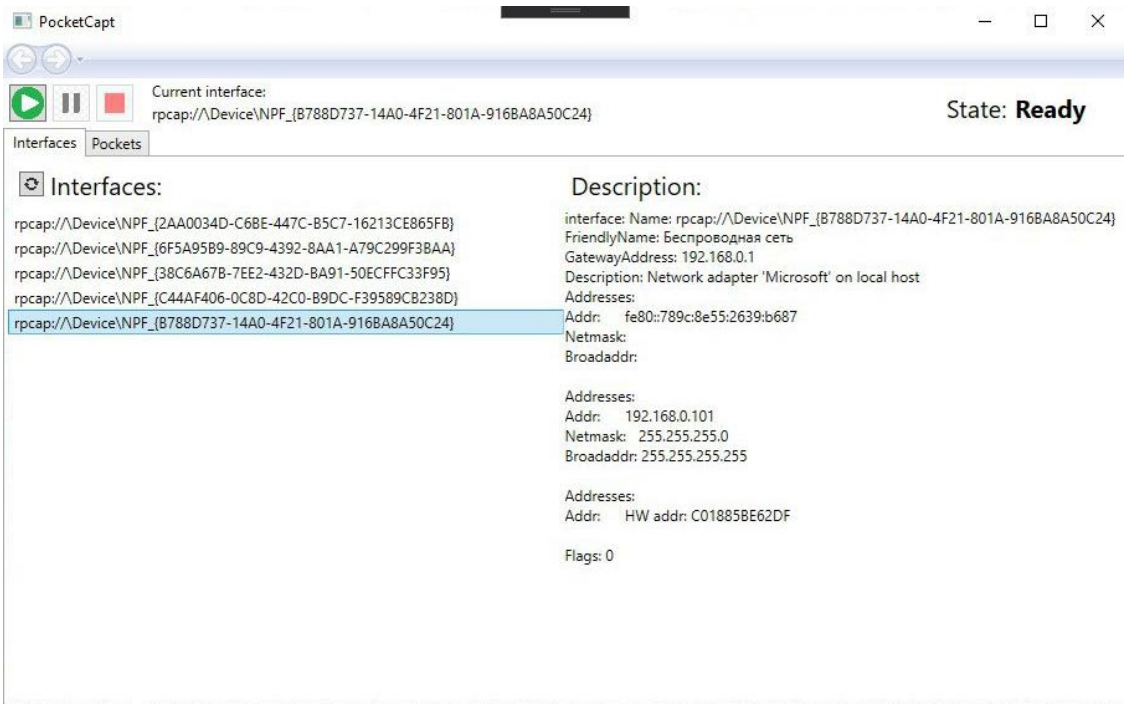

Рисунок 3.3 – Головне вікно додатку «PocketCapt»

Для початку перехоплення пакетів необхідно перейти на вкладку «Pockets» (див. рисунок 2.3), де користувачу доступні наступні елементи:

- Елементи управління перехопленням пакетів (Play, pause, stop);
- Статус (Ready, pause, running);
- Кнопка «Settings» (див. рисунок 3.5);
- Кнопка «Графік» (див. рисунок 3.6);
- Блок налаштування фільтрів та їх кольорів;
- Збереження даних в файл.

Налаштування фільтрів дає можливість обрати поля, вказати значення та колір для зручності аналізу. Також користувач може додавати нові фільтри. В разі виникнення конфлікту при застосування фільтрів перевага надається тому фільтру, що знаходиться вище. Натиснувши на назву колонки, можна відсортувати дані. Натиснувши двічі на строку можна побачити повну інформацію про пакет.

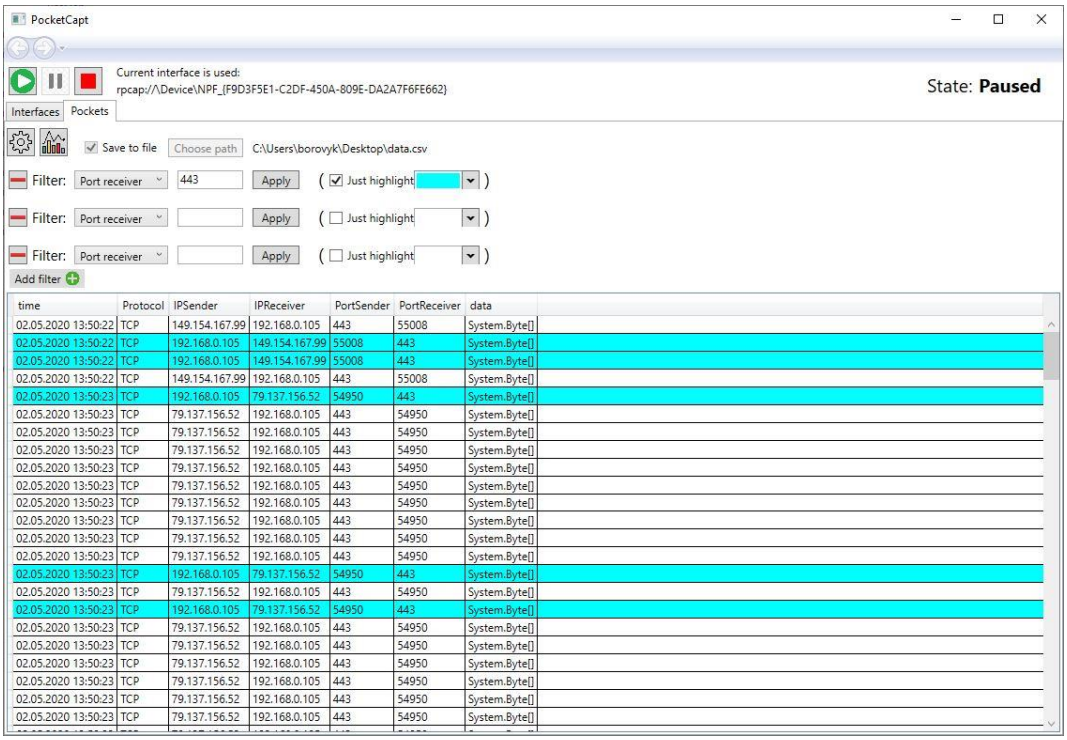

Рисунок 3.4 – Вкладка «Pockets»

Кнопка налаштувань відкриває вікно, що дозволяє обрати колонки, що будуть відображатися при виведенні даних:

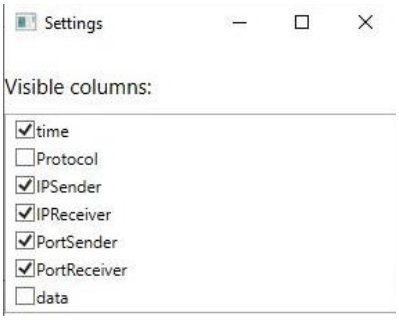

Рисунок 3.5 – Вкладка «Settings»

Для збереження даних необхідно відзначити чекбокс «Save to file» та вказати шлях до потрібної директорії перед початком перехоплення пакетів.

На основі отриманих даних можна побудувати графік. За замовчуванням на ньому відображається розмір пакетів в часі. В налаштуваннях користувач може обрати період групування пакетів.

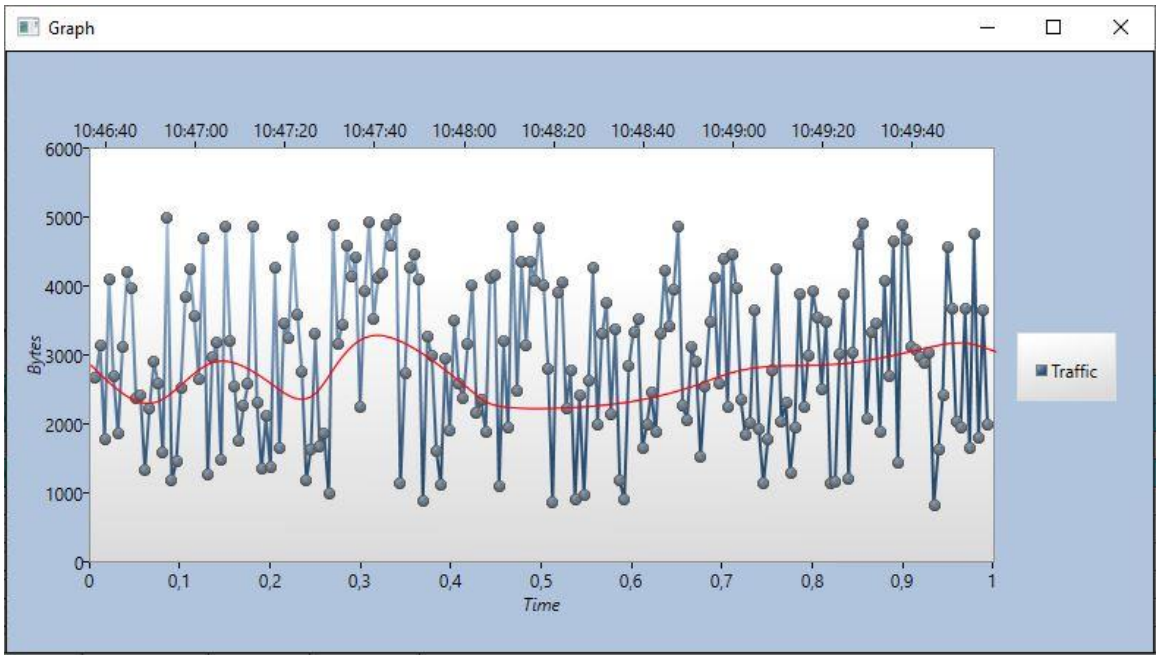

Рисунок 3.6 – Вікно відображення графіка

Для полегшення аналізу доданий функціонал відображення moving averages (лінії, які усереднюють дані з плином часу). Даний графік відображається в реальному часі, а також його можна зберегти в форматі .csv.

Даний функціонал дозволяє отримувати дані про навантаження на локальну комп’ютерну мережу та спрощує аналіз проблемних частин мережі.

ВИСНОВКИ

У ході виконання даної роботи були досліджені основні методи програмного перехоплення пакетів даних та аналізу трафіку в локальній мережі. Створений додаток являє собою об'єктно-орієнтоване гнучке програмне забезпечення реалізоване на мові програмування C#. Для створення графічного інтерфейсу використовувалася технологія Windows Presentation Foundation.

В рамках даної роботи запропонований варіант реалізації додатку для операційних систем Windows для перехоплення і аналізу мережевого трафіку та вирішені такі задачі як:

- Вибір найбільш оптимального методу;
- Використання відповідного драйвера для перехоплення мережевих даних;
- Написання модуля для виведення статистики (таблиця та графік, що оновлюються в реальному часі);
- Написання модуля для збереження та подальшого аналізу даних;
- Написання графічного інтерфейсу.

Подальші дослідження полягають у вдосконаленні та розширенні функціональних можливостей даного додатку для більш зручного та гнучкого налаштування фільтрів перехоплення пакетів та параметрів аналізу мережевого трафіку. Зокрема, реалізація статистики використання додатками мережевих з’єднань, більш детальний вивід вмісту пакетів.

СПИСОК ЛІТЕРАТУРИ

1. Goyal, Piyush, and Anurag Goyal. "Comparative study of two most popular packet sniffing tools-Tcpdump and Wireshark."2017 9th International Conference on Computational Intelligence and Communication Networks (CICN). IEEE, 2017.

2. Hamid, Isredza Rahmi A., et al. "Network monitoring system to detect unauthorized connection." Acta Electronica Malaysia 1.2, 2017.

3. Bhandari, Aishwarya, et al. "Packet Sniffing and Network Traffic Analysis Using TCP—A New Approach." Advances in Electronics, Communication and Computing. Springer, Singapore, 2018.

4. Sanders, Chris. Practical packet analysis: Using Wireshark to solve real-world network problems. No Starch Press, 2017.

5. Saxena, Praful, and Sandeep Kumar Sharma. "Analysis of Network Traffic by using Packet Sniffing Tool: Wireshark." International Journal of Advance Research, Ideas and Innovations in Technology 3.6, 2017.

6. Al-Bastami, Bashar G., and Samy S. Abu Naser. "Design and Development of an Intelligent Tutoring System for C# Language", 2017.

7. Svetlin Nakov, “Fundamentals of computer programming with C# “/ Svetlin Nakov, Veselin Kolev // Sofia: Bulgarian books, 2013.

8. Perkins, Benjamin, Jacob Vibe Hammer, and Jon D. Reid. "C# 7 Programming with Visual Studio 2017." (2018).

9. Dan Clark, “Beginning C# Object-Oriented Programming” / Dan Clark // APRESS, 2013.

10. Hans-Petter Halvorsen, “Introduction to Visual Studio and C#” / HANS- Hans-Petter Halvorsen // University College of Southeast Norway, 2016.

ДОДАТОК

Command.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
using System.Windows.Input;

namespace PocketCapt.UI {
    public class Command : ICommand {
        public Command(Action<object> execute) : this(execute,
DefaultCanExecute) {
        }
        public Command(Action<object> execute, Predicate<object>
canExecute) {
            if (execute == null) {
                throw new ArgumentNullException("execute");
            }

            if (canExecute == null) {
                throw new ArgumentNullException("canExecute");
            }

            this.execute = execute;
            this.canExecute = canExecute;
        }

        private Action<object> execute;
        private Predicate<object> canExecute;
        private event EventHandler CanExecuteChangedInternal;


        public event EventHandler CanExecuteChanged {
            add {
                CommandManager.RequerySuggested += value;
                this.CanExecuteChangedInternal += value;
            }

            remove {
                CommandManager.RequerySuggested -= value;
                this.CanExecuteChangedInternal -= value;
            }
        }

        public bool CanExecute(object parameter) {
            return this.canExecute != null && this.canExecute(parameter);
        }

        public void Execute(object parameter) {
            this.execute(parameter);
        }
```

```
        public void OnCanExecuteChanged() {
            EventHandler handler = this.CanExecuteChangedInternal;
            if (handler != null) {
                //DispatcherHelper.BeginInvokeOnUIThread(() =>
handler.Invoke(this, EventArgs.Empty));
                handler.Invoke(this, EventArgs.Empty);
            }
        }

        private static bool DefaultCanExecute(object parameter) {
            return true;
        }
    }
}
```

HelloPage.xaml.cs:

```
using PacketDotNet;
using SharpPcap;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Threading;
using System.Windows;
using System.Windows.Controls;

namespace PocketCapt.UI {
    public partial class HelloPage : Page {
        public ObservableCollection<NetwDevice> devices { get; set; }
        private NetwDevice selectedDevice;
        public NetwDevice SelectedDevice {
            get => selectedDevice;
            set {
                this.selectedDevice = value;
                ChangeSelectedDeviceEvent?.Invoke(value, new
SelectedDeviceChangedEventArgs(value));
            }
        }
        public EventHandler ChangeSelectedDeviceEvent;

        public ObservableCollection<Pocket> pockets { get; set; }
        //public Pockets pockets { get; set; }

        private PocketCaptAPI pocketCaptAPI;

        public HelloPage(){
            this.devices = new ObservableCollection<NetwDevice>();
            this.pocketCaptAPI = new PocketCaptAPI();

            this.pockets = new ObservableCollection<Pocket>();
            //Application.Current.Properties["pocketsProp"] =
this.pockets;

            InitializeComponent();

            DataContext = this;
            //test();
```

```
        }
        private void refreshInterfBtn_Click(object sender,
RoutedEventArgs e){
            // It's necessary to not remove/replace an old object.
Otherwise, data binding will be broken
            this.devices.Clear();
            foreach (NetwDevice device in this.findInterf()) {
                this.devices.Add(device);
            }
        }
        private void interfList_SelectionChanged(object sender,
SelectionChangedEventArgs e){
            Console.WriteLine("List was selected");
            foreach (var item in e.AddedItems) {
                Console.WriteLine(item);
            }
        }
        private List<NetwDevice> findInterf() {
            List<NetwDevice> devices = new List<NetwDevice>();

            try {
                devices = pocketCaptAPI.getDevices();
            }
            catch (WinPcapIsNotRunning) {
                MessageBoxResult result = MessageBox.Show("It is
necessary to run WinPcap driver\nRun it?", "",

MessageBoxButton.YesNo,

MessageBoxImage.Question);

                if (result == MessageBoxResult.Yes) {
                    lib.ExecuteAsAdmin("cmd.exe", "net start npf");
                    MessageBox.Show("WinPcap driver was run");

                    devices = findInterf();
                }
                else {
                    return new List<NetwDevice>();
                }
            }
            catch (WinPcapIsNotInstalled) {
                MessageBox.Show("Please, install WinPcap driver");
            }

            return devices;
        }
        private void test() {
            for (int i = 0; i < 10; i++) {
                App.Current.Dispatcher.Invoke((Action)delegate // <---
HERE
                {
                    //_matchObsCollection.Add(match);
                    this.pockets.Add(new Pocket("12:13", "0.0.0.0",
"1.1.1.1"));
                });
```

```
                //this.pockets.Add(new Pocket("12:13", "0.0.0.0",
"1.1.1.1"));
            }
        }
    }
}
```

PocketsPage.xaml:

```
<Page x:Class="PocketCapt.UI.PocketsPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:xctk="http://schemas.xceed.com/wpf/xaml/toolkit"
      xmlns:local="clr-namespace:PocketCapt"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="Pockets">

    <Grid>
        <Grid.Resources>
            <DataTemplate x:Key="pocketsListTemp">
                <TextBlock Text="{Binding time}"/>
            </DataTemplate>
        </Grid.Resources>

        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>

        <Grid.RowDefinitions>
            <RowDefinition Height="40"/>
            <RowDefinition Height="40"/>
            <RowDefinition Height="40"/>
            <RowDefinition Height="40"/>
            <RowDefinition Height="20"/>
            <RowDefinition/>
        </Grid.RowDefinitions>

        <StackPanel Grid.Row="0" Orientation="Horizontal">
            <Button Height="30" Margin="0, 5, 5, 5"
Click="SettingsBtn_Click">
                <Image Source="Icons/settings.png"/>
            </Button>
            <Button Height="30" Margin="0, 5, 5, 5" IsEnabled="true"
Opacity="1" Click="GraphBtn_Click">
                <Image Source="Icons/graph.png"/>
            </Button>

            <StackPanel Orientation="Horizontal" Margin="10, 0, 0, 0"
VerticalAlignment="Bottom">
                <CheckBox x:Name="checkboxSave" Content="Save to file"
Margin="2"
                          Checked="checkBox_Checked"
                          Unchecked="checkBox_Unchecked"/>
```

```
                <Button x:Name="btnChoosePath" Content="Choose path"
Margin="10, 0, 0, 4" Click="ChoosePathBtn_Click" Width="80"/>
                <TextBlock x:Name="textBlockFilePath" Text="{Binding
FilePath, Mode=TwoWay}" Margin="10, 2, 0, 0"/>
            </StackPanel>
        </StackPanel>

        <StackPanel Grid.Row="1" Orientation="Horizontal"
Margin="0,10,0,10">
            <Button Margin="2, 0, 5, 0">
                <Image Source="Icons/remove.png"/>
            </Button>

            <TextBlock Text="Filter: " VerticalAlignment="Bottom"
Margin="0,0,5,0" FontSize="15"/>
            <ComboBox Name="filterCombo" Width="100"
SelectionChanged="ComboBox_Selected" Margin="0,0,10,0">
                <ComboBoxItem>
                    <TextBlock>Port sender</TextBlock>
                </ComboBoxItem>
                <ComboBoxItem IsSelected="True">
                    <TextBlock>Port receiver</TextBlock>
                </ComboBoxItem>
                <ComboBoxItem>
                    <TextBlock>IP sender</TextBlock>
                </ComboBoxItem>
                <ComboBoxItem>
                    <TextBlock>IP receiver</TextBlock>
                </ComboBoxItem>
                <ComboBoxItem>
                    <TextBlock>Pocket size</TextBlock>
                </ComboBoxItem>
            </ComboBox>
            <TextBox Text="{Binding FilterValue}" Width="70"
Margin="0,0,10,0"/>
            <Button Content="Apply" Click="ApplyFilterBtn_Click"
Width="50"/>
            <TextBlock Text="(" FontSize="20" Margin="20,-5,5,0"/>
            <CheckBox Content="Just highlight" Margin="0,3,0,0"
Checked="justHigh_Checked" Unchecked="justHigh_Unchecked"/>
            <xctk:ColorPicker Name="ClrPcker_Background"
SelectedColorChanged="ClrPcker_Background_SelectedColorChanged"
Width="70"></xctk:ColorPicker>
            <TextBlock Text=")" FontSize="20" Margin="5,-5,0,0"/>
        </StackPanel>
        <StackPanel Grid.Row="2" Orientation="Horizontal"
Margin="0,10,0,10">
            <Button Margin="2, 0, 5, 0">
                <Image Source="Icons/remove.png"/>
            </Button>
            <TextBlock Text="Filter: " VerticalAlignment="Bottom"
Margin="0,0,5,0" FontSize="15"/>
            <ComboBox Name="filterCombo2" Width="100"
SelectionChanged="ComboBox_Selected2" Margin="0,0,10,0">
                <ComboBoxItem>
                    <TextBlock>Port sender</TextBlock>
                </ComboBoxItem>
```

```
                <ComboBoxItem IsSelected="True">
                    <TextBlock>Port receiver</TextBlock>
                </ComboBoxItem>
                <ComboBoxItem>
                    <TextBlock>IP sender</TextBlock>
                </ComboBoxItem>
                <ComboBoxItem>
                    <TextBlock>IP receiver</TextBlock>
                </ComboBoxItem>
                <ComboBoxItem>
                    <TextBlock>Pocket size</TextBlock>
                </ComboBoxItem>
            </ComboBox>
            <TextBox Text="{Binding FilterValue2}" Width="70"
Margin="0,0,10,0"/>
            <Button Content="Apply" Click="ApplyFilterBtn_Click2"
Width="50"/>
            <TextBlock Text="(" FontSize="20" Margin="20,-5,5,0"/>
            <CheckBox Content="Just highlight" Margin="0,3,0,0"
Checked="justHigh_Checked2" Unchecked="justHigh_Unchecked2"/>
            <xctk:ColorPicker Name="ClrPcker_Background2"
SelectedColorChanged="ClrPcker_Background_SelectedColorChanged2"
Width="70"></xctk:ColorPicker>
            <TextBlock Text=")" FontSize="20" Margin="5,-5,0,0"/>
        </StackPanel>
        <StackPanel Grid.Row="3" Orientation="Horizontal"
Margin="0,10,0,10">
            <Button Margin="2, 0, 5, 0">
                <Image Source="Icons/remove.png"/>
            </Button>
            <TextBlock Text="Filter: " VerticalAlignment="Bottom"
Margin="0,0,5,0" FontSize="15"/>
            <ComboBox Name="filterCombo3" Width="100"
SelectionChanged="ComboBox_Selected3" Margin="0,0,10,0">
                <ComboBoxItem IsSelected="True">
                    <TextBlock>Port sender</TextBlock>
                </ComboBoxItem>
                <ComboBoxItem IsSelected="True">
                    <TextBlock>Port receiver</TextBlock>
                </ComboBoxItem>
                <ComboBoxItem>
                    <TextBlock>IP sender</TextBlock>
                </ComboBoxItem>
                <ComboBoxItem>
                    <TextBlock>IP receiver</TextBlock>
                </ComboBoxItem>
                <ComboBoxItem>
                    <TextBlock>Pocket size</TextBlock>
                </ComboBoxItem>
            </ComboBox>
            <TextBox Text="{Binding FilterValue3}" Width="70"
Margin="0,0,10,0"/>
            <Button Content="Apply" Click="ApplyFilterBtn_Click3"
Width="50"/>
            <TextBlock Text="(" FontSize="20" Margin="20,-5,5,0"/>
            <CheckBox Content="Just highlight" Margin="0,3,0,0"
Checked="justHigh_Checked3" Unchecked="justHigh_Unchecked3"/>
```

```
            <xctk:ColorPicker Name="ClrPcker_Background3"
SelectedColorChanged="ClrPcker_Background_SelectedColorChanged3"
Width="70"></xctk:ColorPicker>
            <TextBlock Text=")" FontSize="20" Margin="5,-5,0,0"/>
        </StackPanel>

        <StackPanel Grid.Row="4" >
            <Button Width="80" Height="20" BorderThickness="0"
HorizontalAlignment="Left" Margin="3,-5,0,0">
                <StackPanel Orientation="Horizontal">
                    <TextBlock Text="Add filter"/>
                    <Image Source="Icons/add.png" Margin="5,0,0,0"
Height="15"/>
                </StackPanel>
            </Button>
        </StackPanel>

        <DataGrid x:Name="dataGrid" Grid.Row="5"
AutoGenerateColumns="False" IsReadOnly="True"
                  ItemsSource="{Binding pockets}">
        </DataGrid>
    </Grid>
</Page>
```

TabControlPage.xaml:

```
<Page x:Class="PocketCapt.UI.TabControlPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:PocketCapt.UI"
      mc:Ignorable="d"
      Loaded="Window_Loaded"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="TabControlPage">

    <Grid>
        <Grid.Resources>
            <Style x:Key="controlButtonStyle" TargetType="Button">
                <Style.Triggers>
                    <Trigger Property="IsEnabled" Value="False">
                        <Setter Property="Opacity" Value="0.5" />
                    </Trigger>
                </Style.Triggers>
            </Style>
        </Grid.Resources>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>

        <Grid.RowDefinitions>
            <RowDefinition Height="40"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
```

```
        <Grid Grid.Row="0">
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>

            <Grid.RowDefinitions>
                <RowDefinition/>
            </Grid.RowDefinitions>

            <StackPanel Orientation="Horizontal">
                <Button Margin="5,5,5,5" Command="{Binding StartCaptCmd}"
Style="{StaticResource controlButtonStyle}">
                    <Image Source="Icons/start.png"/>
                </Button>
                <Button Margin="0,5,5,5" Command="{Binding PauseCaptCmd}"
Style="{StaticResource controlButtonStyle}">
                    <Image Source="Icons/pause.png"/>
                </Button>
                <Button Margin="0, 5, 0, 5" Command="{Binding
StopCaptCmd}" Style="{StaticResource controlButtonStyle}">
                    <Image Source="Icons/stop.png"/>
                </Button>

                <StackPanel Orientation="Vertical">
                    <TextBlock Text="Current interface is used:"
Margin="15, 2, 0, 0"/>
                    <TextBlock Text="{Binding Path=SelectedDeviceName,
Mode=TwoWay}" Margin="15, 2, 0, 0"/>
                </StackPanel>
            </StackPanel>
            <TextBlock Text="State:" Width="50"
HorizontalAlignment="Right" Margin="0,10, 95, 0" FontSize="20"/>
            <TextBlock Text="{Binding CurrentStateName, Mode=TwoWay}"
Width="90" Margin="0,10, 0, 0" HorizontalAlignment="Right" FontSize="20"
FontWeight="Bold"/>
        </Grid>

        <TabControl Grid.Row="1" x:Name="tabContrl">
            <TabItem x:Name="helloPageTab" Header="Interfaces">
                <Frame Source="HelloPage.xaml"/>
            </TabItem>
            <TabItem x:Name="pocketsPageTab" Header="Pockets">
                <Frame Source="PocketsPage.xaml"/>
            </TabItem>
        </TabControl>
    </Grid>
</Page>
```

TabControlPage.xaml.cs:

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Navigation;
```

```
namespace PocketCapt.UI {
    /// <summary>
    /// Interaction logic for TabControlPage.xaml
    /// </summary>
    public partial class TabControlPage : Page {
        public TabControlPage() {
            InitializeComponent();
            DataContext = new TabControPageViewModel();

            ((TabControPageViewModel)DataContext).PropertyChanged +=
TabControlPage_PropertyChanged;
        }
        private void TabControlPage_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e) {
            if (e.PropertyName.Equals("currentState")) {
                PocketsPage dataContext =
(PocketsPage)((PocketsPage)((Frame)pocketsPageTab.Content).Content).DataC
ontext;
                switch
(((TabControPageViewModel)DataContext).currentState) {
                    case AppState.Ready:
                        if ((bool)dataContext.checkboxSave.IsChecked) {
                            dataContext.btnChoosePath.IsEnabled = true;
                            dataContext.textBlockFilePath.IsEnabled =
true;
                        }
                        dataContext.checkboxSave.IsEnabled = true;
                        break;
                    case AppState.Running:
                        dataContext.btnChoosePath.IsEnabled = false;
                        dataContext.textBlockFilePath.IsEnabled = false;
                        dataContext.checkboxSave.IsEnabled = false;
                        break;
                    case AppState.Pause:
                        dataContext.btnChoosePath.IsEnabled = false;
                        dataContext.textBlockFilePath.IsEnabled = false;
                        dataContext.checkboxSave.IsEnabled = false;
                        break;
                }
            }
        }
        private void DataContext_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e) {
            PocketsPage dataContext = ((PocketsPage)sender);

            if (e.PropertyName.Equals("FilePath"))
                ((TabControPageViewModel)DataContext).filePath =
dataContext.FilePath;

            if (e.PropertyName.Equals("IsSaveToFile"))
                ((TabControPageViewModel)DataContext).isSaveToFile =
dataContext.IsSaveToFile;
        }

        private void Window_Loaded(object sender, RoutedEventArgs e) {
            ((Frame)helloPageTab.Content).LoadCompleted += WasLoaded;
```

```
            ((Frame)pocketsPageTab.Content).LoadCompleted +=
WasLoadedPocket;
        }

        private void WasLoadedPocket(object sender, NavigationEventArgs
e) {
            PocketsPage dataContext =
(PocketsPage)((PocketsPage)((Frame)pocketsPageTab.Content).Content).DataC
ontext;
            dataContext.PropertyChanged += DataContext_PropertyChanged;
        }

        private void WasLoaded(object sender, NavigationEventArgs e) {
            HelloPage dataContext =
(HelloPage)((HelloPage)((Frame)sender).Content).DataContext;
            dataContext.ChangeSelectedDeviceEvent +=
((TabControPageViewModel)DataContext).OnSelecetedDeviceChanged;
        }
    }
}
```

TabControPageViewModel.cs:

```
using CsvHelper;
using PacketDotNet;
using SharpPcap;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Dynamic;
using System.IO;
using System.Threading;
using System.Windows;
using System.Windows.Controls;

namespace PocketCapt.UI {
    class TabControPageViewModel : INotifyPropertyChanged {
        public TabControPageViewModel() {
            startCaptCmd = new Command(StartCapt, StartCaptCanExec);
            pauseCaptCmd = new Command(PauseCapt, PauseCaptCanExec);
            stopCaptCmd = new Command(StopCapt, StopCaptCanExec);

            this.currentState = AppState.Ready;
            this.pockets = new ObservableCollection<Pocket>();
            Application.Current.Properties["pocketsProp"] = this.pockets;
        }
        public string filePath;
        public bool isSaveToFile;
        private StreamWriter writer;
        private CsvWriter CsvWriter;
        Thread t;
        bool ShouldLive = true;
        ICaptureDevice captureDevice;

        private AppState _currentState;
        public AppState currentState {
```

```
            get => _currentState;
            set {
                this._currentState = value;
                this.ChangeState(value);
                this.CurrentStateName = value.ToFriendlyString();
                OnPropertyChanged("currentState");
            }
        }

        private string currentStateName;
        public string CurrentStateName {
            get => currentStateName;
            set {
                currentStateName = value;
                OnPropertyChanged("CurrentStateName");
            }
        }

        private string selectedDeviceName;
        public string SelectedDeviceName {
            get => selectedDeviceName;
            set {
                selectedDeviceName = value;
                OnPropertyChanged("SelectedDeviceName");
            }
        }

        private NetwDevice selectedDevice;
        public NetwDevice SelectedDevice {
            get => selectedDevice;
            set {
                selectedDevice = value;
                SelectedDeviceName = value.Name;
            }
        }

        private Command startCaptCmd;
        public Command StartCaptCmd => startCaptCmd;

        private Command pauseCaptCmd;
        public Command PauseCaptCmd => pauseCaptCmd;

        private Command stopCaptCmd;
        public Command StopCaptCmd => stopCaptCmd;
        private bool isStartCaptCanExec;
        private void StartCapt(object obj) {
            if (t == null || !t.IsAlive) {
                if (isSaveToFile && !string.IsNullOrEmpty(filePath)) {
                    writer = new StreamWriter(filePath);
                    CsvWriter = new CsvWriter(writer,
System.Globalization.CultureInfo.CurrentCulture);
                    CsvWriter.WriteHeader(typeof(Pocket));
                    CsvWriter.NextRecord();
                }

                //Console.WriteLine(this.SelectedDevice.Name);
                //выбираем первое устройство в спсике(для примера)
```

```
                captureDevice = this.SelectedDevice.device;
                //регистрируем событие, которое срабатывает, когда пришел
новый пакет
                captureDevice.OnPacketArrival += new
PacketArrivalEventHandler(Program_OnPacketArrival);
                // открываем в режиме promiscuous, поддерживается также
нормальный режим
                captureDevice.Open(DeviceMode.Promiscuous, 1000);
                // начинаем захват пакетов
                t = new Thread(new ThreadStart(captureDevice.Capture));
                t.Start();
                //test();
                //Thread.Sleep(10);
                //captureDevice.Capture();
                //captureDevice.StopCapture();
            }
            else {
                t.Resume();
            }


            currentState = AppState.Running;
        }
        private bool StartCaptCanExec(object obj) {
            return isStartCaptCanExec;
        }
        public ObservableCollection<Pocket> pockets { get; set; }
        private void Program_OnPacketArrival(object sender,
CaptureEventArgs e) {
            // парсинг всего пакета
            Packet packet = Packet.ParsePacket(e.Packet.LinkLayerType,
e.Packet.Data);
            // получение только TCP пакета из всего фрейма
            var tcpPacket = TcpPacket.GetEncapsulated(packet);
            // получение только IP пакета из всего фрейма
            var ipPacket = IpPacket.GetEncapsulated(packet);
            if (tcpPacket != null && ipPacket != null) {
                DateTime time = e.Packet.Timeval.Date;
                int len = e.Packet.Data.Length;

                var protocol = ipPacket.Protocol.ToString();

                // IP адрес отправителя
                var srcIp = ipPacket.SourceAddress.ToString();
                // IP адрес получателя
                var dstIp = ipPacket.DestinationAddress.ToString();
                // порт отправителя
                var srcPort = tcpPacket.SourcePort.ToString();
                // порт получателя
                var dstPort = tcpPacket.DestinationPort.ToString();
                // данные пакета
                var data =
packet.PayloadPacket.PayloadPacket.PayloadData;

                App.Current.Dispatcher.Invoke((Action)delegate
                {
```

```
                    Pocket pocket = new Pocket(time.ToString(), protocol,
srcIp, dstIp, srcPort, dstPort, data.ToString());
                    this.pockets.Add(pocket);

                    if (isSaveToFile && !string.IsNullOrEmpty(filePath))
{
                        CsvWriter.WriteRecord(pocket);
                        CsvWriter.NextRecord();
                        CsvWriter.Flush();
                    }
                });

                Console.WriteLine("Time: {0}; Length: {1}; IP sender:
{2}; IP receiver: {3}; Port sender: {4}; Port receiver: {5}; Data: {6}",
                    time, len, srcIp, dstIp, srcPort, dstPort, data);
            }
        }

        private bool isPauseCaptCanExec;
        private void PauseCapt(object obj) {
            t.Suspend();
            currentState = AppState.Pause;
        }
        private bool PauseCaptCanExec(object obj) {
            return isPauseCaptCanExec;
        }


        private bool isStopCaptCanExec;
        private void StopCapt(object obj) {
            captureDevice.StopCapture();
            t.Abort();
            currentState = AppState.Ready;

            try {
                if (isSaveToFile && !string.IsNullOrEmpty(filePath)) {
                    writer.Close();
                }
            }
            catch(Exception ex) {

            }

            //CsvWriter.Flush();
        }
        private bool StopCaptCanExec(object obj) {
            return isStopCaptCanExec;
        }

        public void OnSelecetedDeviceChanged(object sender, EventArgs
args) {
            if(currentState == AppState.Ready)
                SelectedDevice =
((SelectedDeviceChangedEventArgs)args).Device;
        }

        private void ChangeState(AppState state) {
```

```
            switch (state) {
                case AppState.Ready:
                    isStartCaptCanExec = true;
                    startCaptCmd.OnCanExecuteChanged();

                    isPauseCaptCanExec = false;
                    pauseCaptCmd.OnCanExecuteChanged();

                    isStopCaptCanExec = false;
                    stopCaptCmd.OnCanExecuteChanged();
                    break;
                case AppState.Running:
                    isStartCaptCanExec = false;
                    startCaptCmd.OnCanExecuteChanged();

                    isPauseCaptCanExec = true;
                    pauseCaptCmd.OnCanExecuteChanged();

                    isStopCaptCanExec = true;
                    stopCaptCmd.OnCanExecuteChanged();
                    break;
                case AppState.Pause:
                    isStartCaptCanExec = true;
                    startCaptCmd.OnCanExecuteChanged();

                    isPauseCaptCanExec = false;
                    pauseCaptCmd.OnCanExecuteChanged();

                    isStopCaptCanExec = true;
                    stopCaptCmd.OnCanExecuteChanged();
                    break;
            }
        }
        public event PropertyChangedEventHandler PropertyChanged;
        public void OnPropertyChanged(string name) {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(name));
        }
    }
    public enum AppState {
        Ready = 0,
        Running,
        Pause
    }
    public static class AppStateExtensions {
        public static string ToFriendlyString(this AppState me) {
            switch (me) {
                case AppState.Ready:
                    return "Ready";
                case AppState.Running:
                    return "Running";
                case AppState.Pause:
                    return "Paused";
                default:return "ERROR"
            }
        }
    }
```