Хмельницький національний університет
Факультет програмування та
комп'ютерних і телекомунікаційних систем
Кафедра комп'ютерної інженерії та системного програмування

ДИПЛОМНА РОБОТА МАГІСТРА

Галузь знань 12 – Інформаційні технології

Спеціальність 123 – Комп'ютерна інженерія

на тему «Централізована розподілена система виявлення атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу»

ДРКІСПр. 015094.19.01.21 ПЗ

Виконав: студент 2 курсу, група КІ2м-19-1 _____ Шагін В.Ю.
Підпис Ініціали, прізвище

Керівник д.т.н. проф. _____ Березький О.М.
Науковий ступінь, вчене звання Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КІСП, д.т.н., с.н.с., проф. _____ Т.О. Говорущенко
__ _____ 2021 р. Підпис Ініціали, прізвище

Хмельницький, 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ПРОГРАМУВАННЯ ТА КОМП'ЮТЕРНИХ І ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА СИСТЕМНОГО ПРОГРАМУВАННЯ

Освітній рівень МАГІСТР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма ОСВІТНЬО-НАУКОВА ПРОГРАМА «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

"07" 09 2021 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Шагін В'ячеслав Юрійович

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Централізована розподілена система виявлення атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу

Керівник проекту (роботи) Березький О.М. д.т.н. проф.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.01.2021 р. № 7

2. Строк подання студентом проекту (роботи) на кафедру 11.05.2020 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

аналіз відомих методів виявлення аномалій в комп'ютерних мережах;

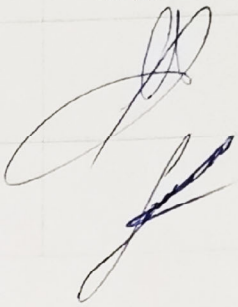
проєктування архітектури розподіленої системи та методу підтримки її цілісності;

метод виявлення аномалій в комп'ютерних мережах

дослідження ефективності запропонованих рішень, реалізація розподіленої системи.

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

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., професор кафедри КІСП		
Антиплагіат	Нічепорук А.О., доцент кафедри КІСП		

7. Дата видачі завдання «_01»_____09_____2020р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики ДРМ з керівником	01.09.2020	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	05.10.2020	виконано
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	05.11.2020	виконано
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	05.12.2020	виконано
5	Робота над науковою статтею	05.01.2021	виконано
6	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	15.02.2021	виконано
7	Робота над розділом 4 – проектування та розробка розподіленої системи для вирішення поставленої задачі, експериментальна частина	05.04.2021	виконано
8	Оформлення пояснювальної записки згідно вимог	15.04.2021	виконано
9	Попередній захист ДРМ	25.04.2021	виконано
10	Захист ДРМ на засіданні ЕК	До 31.05.2021	

Студент _____ Шагін В.Ю.
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Березький О.М.
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема дипломної роботи: «Централізована розподілена система виявлення атак в корпоративних комп’ютерних мережах на основі мультифрактального аналізу»

Автор роботи: Шагін В’ячеслав Юрійович

Керівник роботи: Березький Олег Миколайович

Пояснювальна записка: 80 ст., 41 рис., 2 табл., 3 дод., 61 джерел.

ПЕРЕЛІК КЛЮЧОВИХ СЛІВ: аномалії, мультифрактальний аналіз, вейвлет-аналіз, розподілена система, комп’ютерна система, комп’ютерна мережа.

Об’єкт дослідження є процес функціонування централізованих розподілених систем виявлення мережевих атак в комп’ютерних мережах.

Предмет дослідження є методи і засоби створення централізованих розподілених систем виявлення мережевих атак в комп’ютерних мережах.

Метою роботи є покращення ефективності виявлення аномалій в комп’ютерних системах шляхом розробки розподіленої системи виявлення мережевих атак в корпоративних комп’ютерних мережах на основі мультифрактального аналізу.

Наукова новизна отриманих результатів полягає в наступному:

1) удосконалено архітектуру розподіленої системи виявлення мережевих атак в комп’ютерних мережах, в якій скомбіновано вимоги розподіленості, централізованості, самоорганізованості та багаторівневості, що дало змогу вдосконалити її внутрішню структуру та принцип взаємодії центру прийняття рішень з компонентами системи з чітким розмежуванням верхнього та нижнього рівнів ієрархії із впровадженням методу збереження цілісності;

2) удосконалено метод виявлення мережевих атак на основі мультифрактального аналізу в комп’ютерних мережах, що надає можливість його використання в комп’ютерних мережах з багатьма вузлами, на яких встановлено розподілену систему виявлення мережевих атак.

Практичне значення отриманих результатів. В результаті виконаних в роботі досліджень розроблено архітектуру та компоненти розподіленої системи виявлення мережевих атак, в якій поєднано вимоги централізованості, розподіленості, самоорганізованості та на її основі здійснено розробку централізованої розподіленої системи визначення мережевих атак в корпоративних комп'ютерних мережах. Проведені експериментальні дослідження з реалізованою централізованою розподіленою системою визначення мережевих атак в комп'ютерних мережах підтвердили ефективність функціонування в комп'ютерній мережі.

Теоретичних та практичних результатів роботи було досягнуто при виконанні науково-дослідних робіт в Хмельницькому національному університеті.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД - база даних

ЦПР - центр прийняття рішень

РС – розподілена система

ОС - операційна система

ПЗ - програмне забезпечення

DDoS - Distributed Denial of Service (розподілена відмова в обслуговуванні) IDS - система виявлення вторгнень

КС - комп’ютерні системи

КМ – комп’ютерні мережі

ВСТУП

Актуальність роботи. Зі зростанням об’ємів інформації, що передаються через мережу зростає і число апаратних та програмних засобів для втручання в процес передачі даних. Щодня все більше користувачів в інтернеті стають жертвами дій зловмисного програмного забезпечення. Особливу небезпеку становлять атаки та корпоративні системи, що можуть спричинити фінансові збитки організаціям чи безпосередньо їх працівникам.

Для мінімізації кількості успішних мережевих атак в корпоративних комп’ютерних мережах може бути використана математична статистика. Оскільки в корпоративних комп’ютерних мережах, як правило, присутня велика кількість комп’ютерів необхідні ефективні методи виявлення та реагування на активність в мережі. Існує твердження, що немає абсолютно захищеної системи, проте можливе суттєве зниження шансу на несанкціонований доступ до даних чи комп’ютерної системи загалом. Для протидії активності зловмисників необхідний комплексний підхід до розробки методів виявлення вторгнень та систем, в які інтегруються ці методи.

Поява нового типу чи методу мережевих атак завжди привертає до себе увагу наслідками своїх дій. Тому необхідна система, що дозволить визначати загрози ще до нанесення шкоди. Для оперативності опрацювання даних та мінімізації впливу людини на процес загалом розподілена система повинна бути автономною та самоорганізованою.

В роботі пропонується використання розподілених систем визначення аномалій в комп’ютерних мережах, розроблених за принципами централізації та самоорганізації.

Метою роботи є розробка розподіленої системи виявлення мережевих атак в корпоративних комп’ютерних мережах на основі мультифрактального аналізу.

В роботі сформульовано наступні задачі дослідження:

1) дослідити та визначити особливості прояву аномалій в комп’ютерних мережах під час активності зловмисного програмного забезпечення і здійснення

мережевих атак в локальній мережі, проаналізувати архітектури розподілених систем, доступні засоби виявлення мережевих атак та їх особливості;

2) удосконалити архітектуру розподіленої системи виявлення вторгнень в комп'ютерних мережах методом синтезу вимог розподіленості та централізованості з можливістю самообслуговування, для створення на її основі системи, що працюватиме під керуванням одного центру прийняття рішень про рівень безпеки мережевого трафіку;

3) розробити метод підтримки цілісності розподіленої системи виявлення мережевих атак, на основі якого система змогла б за допомогою центру прийняття рішень самостійно змінювати свою архітектуру та подальшу стратегію роботи без втручання користувача;

4) удосконалити метод централізованого виявлення мережевих атак за алгоритмом мультифрактального аналізу для аналізу мережевого трафіку на різних часових проміжках та з різним масштабом;

5) розробити програмне забезпечення централізованої розподіленої системи виявлення мережевих атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу для підтвердження можливості реалізації таких систем згідно запропонованої архітектури та використання в експериментальних дослідженнях для порівняння з відомими аналогами.

Предмет дослідження – методи і засоби створення централізованих розподілених систем виявлення мережевих атак в комп'ютерних мережах.

Об'єкт дослідження – процес функціонування централізованих розподілених систем виявлення мережевих атак в комп'ютерних мережах.

Методи дослідження. Для досягнення результатів поставлених задач використано основні положення:

1) теорії розподілених систем, що можуть бути основою для розробки програмних чи апаратно-програмних засобів;

2) методи мультифрактального аналізу для виявлення мережевих атак;

3) теорії комп'ютерних мереж для розгортання та функціонування розподіленої системи.

Наукова новизна отриманих результатів полягає в наступному:

3) удосконалено архітектуру розподіленої системи виявлення мережевих атак в комп'ютерних мережах, в якій скомбіновано вимоги розподіленості, централізованості, самоорганізованості та багаторівневості, що дало змогу вдосконалити її внутрішню структуру та принцип взаємодії центру прийняття рішень з компонентами системи з чітким розмежуванням верхнього та нижнього рівнів ієрархії із впровадженням методу збереження цілісності;

4) удосконалено метод виявлення мережевих атак на основі мультифрактального аналізу в комп'ютерних мережах, що надає можливість його використання в комп'ютерних мережах з багатьма вузлами, на яких встановлено розподілену систему виявлення мережевих атак.

Обґрунтованість і достовірність наукових положень, висновків і рекомендацій. Наукові положення, рекомендації та висновки, представлені в дипломній роботі магістра обґрунтовані використанням математичного апарату, алгоритмами визначення мережевих атак в комп'ютерних мережах, практичним впровадженням результатів, що демонструє відповідність теоретичних розробок та практичних результатів.

Практичне значення отриманих результатів. В результаті виконаних в роботі досліджень розроблено архітектуру та компоненти розподіленої системи виявлення мережевих атак, в якій поєднано вимоги централізованості, розподіленості, самоорганізованості та на її основі здійснено розробку централізованої розподіленої системи визначення мережевих атак в корпоративних комп'ютерних мережах. Проведені експериментальні дослідження з реалізованою централізованою розподіленою системою визначення мережевих атак в комп'ютерних мережах підтвердили ефективність функціонування в комп'ютерній мережі.

Теоретичних та практичних результатів роботи було досягнуто при виконанні науково-дослідних робіт в Хмельницькому національьному університеті.

Особистий внесок здобувача. Всі результати досліджень, що представлено до захисту роботи було одержано автором особисто. Опублікована в співавторстві робота

Структура кваліфікаційної роботи. Дипломна робота складається з анотації, вступу, чотирьох розділів, висновків, списку використаних джерел з 61 найменування на 6 сторінках та трьох додатків на 70 сторінках. Загальний обсяг роботи становить 80 сторінок, з них 70 сторінок основного тексту, 41 рисунок, 2 таблиці.

1 РОЗПОДІЛЕНІ СИСТЕМИ

1.1 Поняття розподіленої системи

В науковій літературі зустрічаються різноманітні визначення розподіленої системи, та кожне з них відрізняється. В загальному поняття розподіленої системи полягає в об’єднанні автономних обчислювальних одиниць в єдину когерентну систему[1, 46, 51]. Кожна обчислювальна одиниця слугує вузлом, що може бути як програмним процесом так і окремою машиною. Оскільки для користувача система є суцільною – всі вузли повинні співпрацювати.

В загальному розподілені системи можна розділити на такі типи:

1) Кластери – група пов’язаних обчислювальних одиниць, що співпрацюють для спільної мети. Як правило вони пов’язані між собою за допомогою швидких локальних мереж. На рисунку 1.1 один із варіантів побудови кластера.

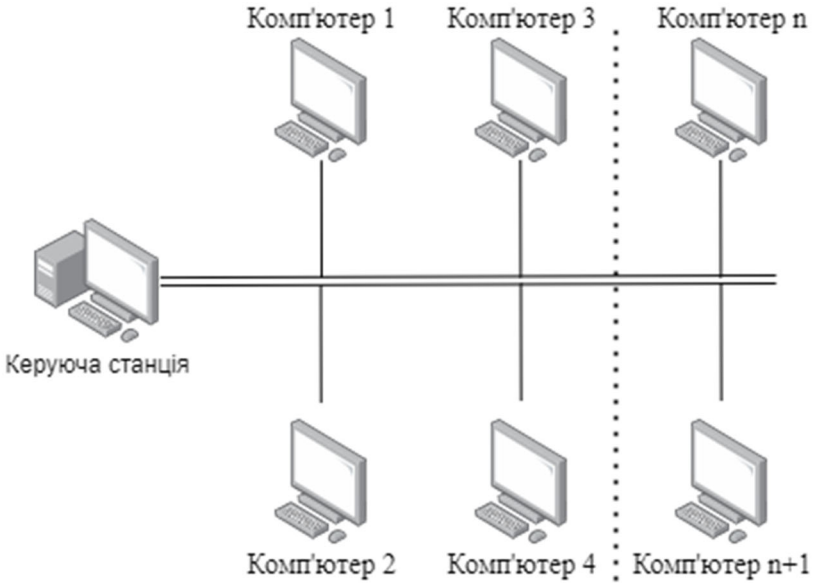


Рисунок 1.1 - Кластер

Для користувача системи робота з кластером виглядає як взаємодія з єдиною цілісною системою;

2) Сітки – об’єднані кластери, що утворюють систему масивних обчислювальних потужностей. Загальну схему об’єднаних кластерів зображено на рисунку 1.2;

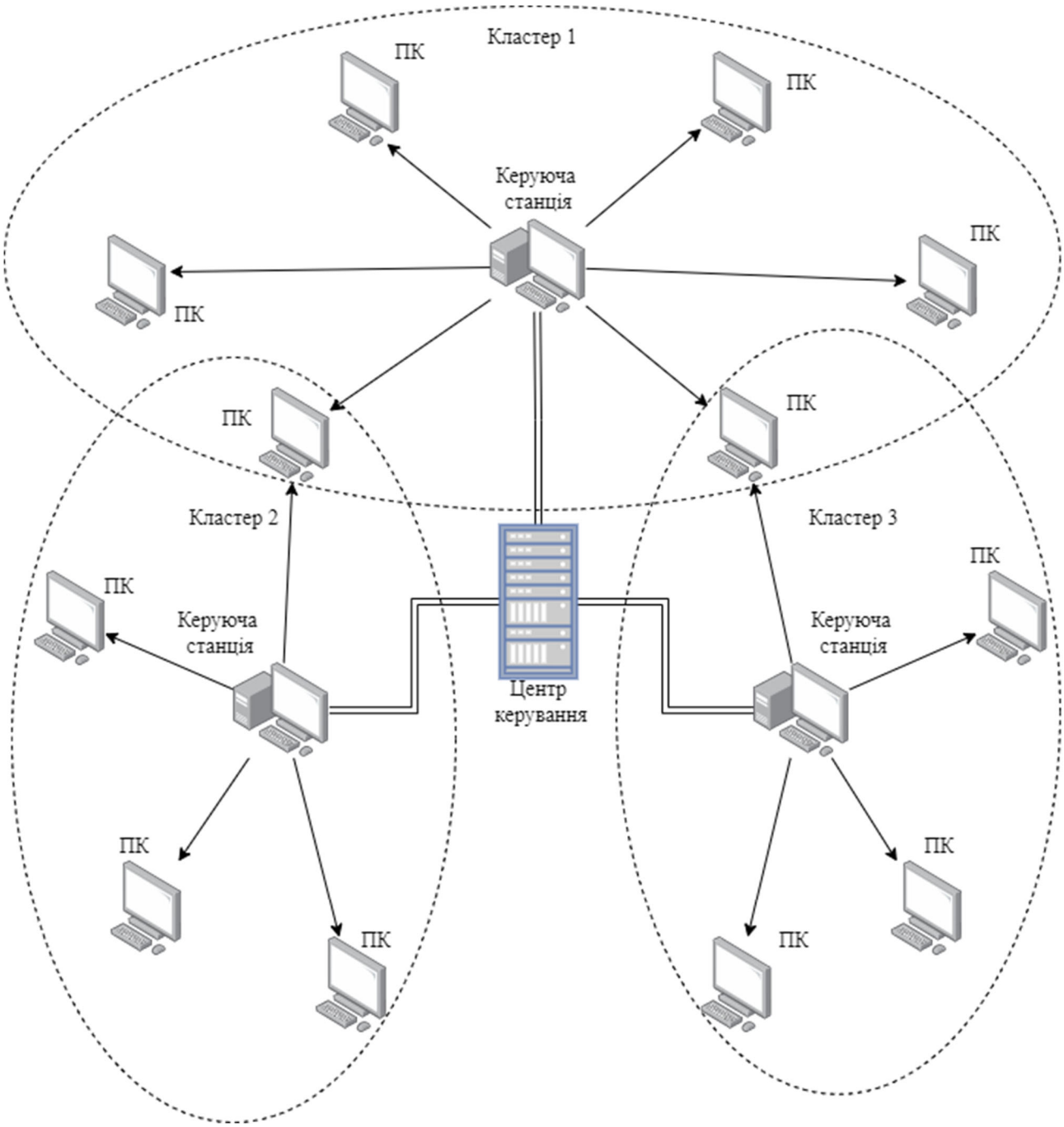


Рисунок 1.2 – Сітка

3) Peer-to-Peer – система, в якій користувачі або вузли комунікують між собою самостійно [45, 52]. Як правило схема підключення таких систем – зірка. На рисунку 1.3 зображено загальну схему Peer-to-Peer системи.

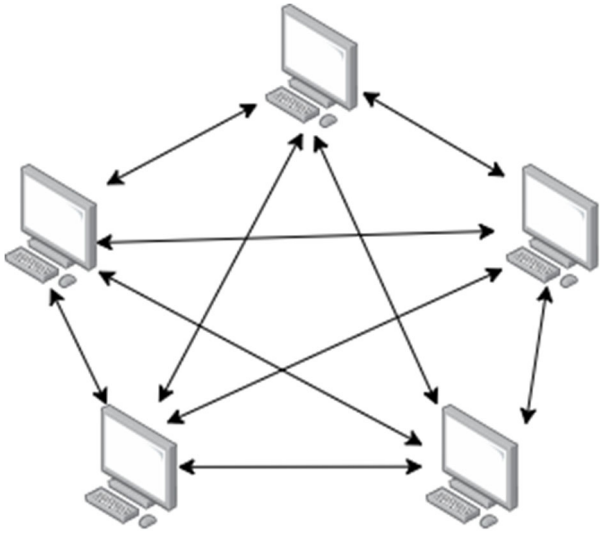


Рисунок 1.3 – Peer-to-Peer

1.1 Концепт розподіленої системи

Питання надійності комп’ютерної мережі зараз актуальне як ніколи. Тому для бізнесу широко використовуються розподілені мережі, які можуть бути реалізовані за допомогою програмного продукту. Концепт розподіленої мережі можна використати по різному. В широкому розумінні це робочі станції, що обмінюються інформацією та спільно її обробляють, при цьому кожна робоча станція може мати різну обчислювальну потужність та ємність сховища даних.

Модель розподіленої мережі передбачає поділ завдань між клієнтам чи серверами, що залежить від того, чи підходить конкретна машина для обробки даних. За цим типом архітектури частина додатку виконується клієнтом, а інша частина сервером. З клієнт-серверним типом побудови програмного забезпечення користувач працює лише з обмеженою кількістю даних, включаючи інтерфейс користувача, користувацький ввід та запити до бази даних. Контроль доступу до бази даних, отримання чи обробку даних користувачем здійснює безпосередньо сервер. При цьому важливо перевіряти чи справді клієнт має дозвіл на оперування інформацією. Для цього проводять процедуру двох голосувань. Під час першого голосування кожен вузол отримує довіру, розподіляючи інформацію про відкликання сусідів розподіленої мережі. Сусідні вузли створюють довірене

оточення. Якщо вузол обмінюється інформацією з іншими в достатній мірі – він може і далі спілкуватись з рештою в мережі, в іншому випадку він відхиляється [5]. Якщо вузол загрожує мережі та проявляє підозрілу поведінку – відбувається друга процедура відкликання голосування. В цьому випадку, якщо оточуючі вузли вказують на підозрілу активність свого сусіда – блокуються весь обмін даними з цим вузлом по всій мережі.

Основними властивостями розподіленої системи за теоремою CAP [29, 50] є цілісність, доступність та стійкість до відмов (Consistency, Availability, Partition tolerance). Проте будь-яка розподілена система може мати не більше двох властивостей.

Цілісність або узгодженість системи за теоремою CAP означає її лінійність. Кожна наступна операція повинна мати інформацію про результат попередньої. Доступність системи означає безперебійну успішну обробку запитів на всіх активних вузлах. Якщо частина вузлів не відповідає чи система відповідає не на всі запити – за теоремою CAP це непостійна доступність. Стійкість до відмов в першу чергу забезпечує стабільну роботу всієї системи. Наприклад, якщо кластер із декількох серверів втратив з'єднання з половиною своїх вузлів – робота системи продовжується, хоч і було втрачено доступність. В іншому випадку частини кластера працюють незалежно один від одного та відповідають на запити користувачів. В такому випадку тільки після відновлення зв'язку між всіма ланками розподіленої системи буде зрозуміло, що це була єдина система.

Згідно теореми CAP маємо три варіанти проєктування розподіленої системи:

1) AC – доступність та цілісність;

2) CP – цілісність та стійкість;

3) AP – доступність та стійкість.

AC системи мають суттєвий недолік – відсутність стійкості до відмов мережі. Використання цього типу систем вимагає чіткого усвідомлення всіх ризиків. В іншому випадку необхідно обирати між цілісністю та доступністю системи.

1.2 Аналіз відомих методів та засобів проєктування розподілених мереж та систем виявлення мережевих атак

1.2.1 Аналіз моделей розподіленої системи

Для розподілених обчислень можуть бути використані різноманітні апаратні та програмні моделі архітектури [41, 47-49]. Найнижчий рівень потребує з'єднання декількох процесорів з будь-якою мережею, що може бути утворена на друкованій платі чи складена зі з'єднаних випадковим чином пристроїв та кабелів.

На високому рівні необхідно зв'язати процеси, що виконуються на різних процесорах в єдине ціле за допомогою єдиної системи зв'язку [7, 61]. Для цього існує розподілене програмування, що можна розділити на наступні категорії:

1. В клієнт-серверній моделі підключення клієнтів здійснюється безпосередньо до сервера , що може знаходитись в одній мережі з користувачем чи з'єднуватись через мережу інтернет, та розподілення робочих навантажень між постачальником послуги та отримувачем, тобто клієнтом.

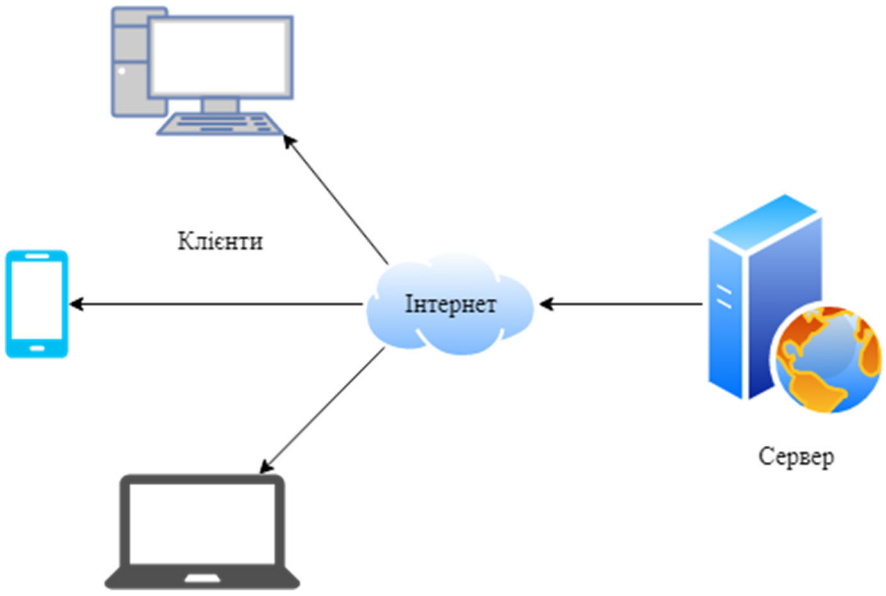


Рисунок 1.4 – Клієнт-серверна архітектура

2. 3-рівнева архітектура являє собою вдосконалену клієнт-серверну модель. Вона має 3 рівні компонентів:

1) На рівні представлення відбувається виведення інформації користувачам за допомогою графічного інтерфейсу, тобто безпосередньо веб-інтерфейсу чи візуальної складової додатку;

2) На прикладному рівні здійснюється контроль доступу до даних та інші можливості програми, що керуються сервером;

3) На рівні даних міститься інформація, яку використовує прикладний рівень.

Схема 3-рівневої архітектури представлена на рисунку 1.5.

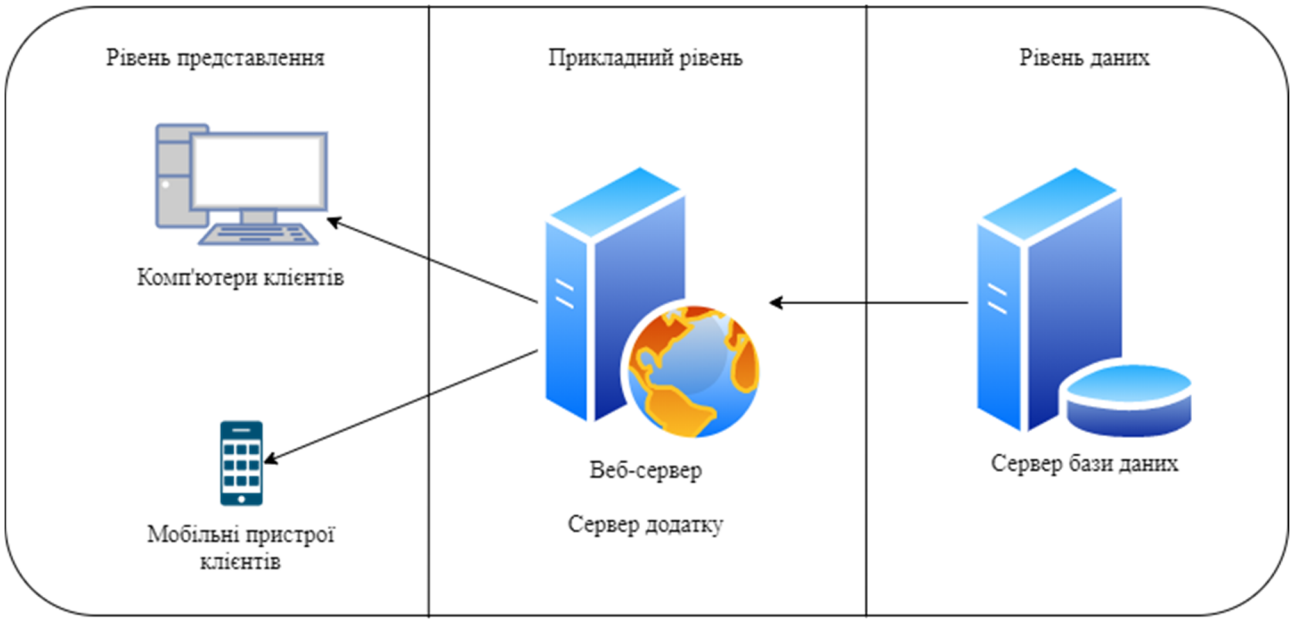


Рисунок 1.5 – 3-рівнева архітектура

3. Багаторівнева архітектура за своєю сутністю це розширена трирівнева архітектура, основна перевага якої в тому, що рівні можуть працювати на різних процесорах та операційних системах та оновлюватись незалежно один від одного.

4. Тісно зв’язана модель являє собою кластер, що виконує єдину задачу паралельно на всіх обчислювальних одиницях системи.

5. Модель Peer-to-Peer – однорангова мережа, в якій кожен комп’ютер може виконувати роль як сервера так і клієнта. В такій системі кожен вузол має спільний доступ до обчислювальних потужностей та сховища даних різних машин.

На початку 1990-х років об’єднано-орієнтовне програмування [10] принципово змінили підхід до розробки програмного забезпечення на користь модульних компонентів. Сьогодні подібні зміни відбуваються із розробкою розподілених систем. Мікросервісна архітектура, що будується з контейнерних програмних компонентів, набирає дедалі більшої популярності та добре підходить в якості об’єкту розподіленої системи.

Для розробки розподілених систем за допомогою контейнерів існує декілька шаблонів проєктування, які можна розділити на 2 основні групи: з єдиним вузлом та багатоконтейнерні. Шаблонами з єдиним вузлом є:

1. Шаблон «розширення» [8] – складається з двох контейнерів (рисунок 1.6). Першим є контейнер додатку, що містить логіку програми. Без нього додаток не буде існувати. Другим є контейнер розширення, що доповнює та вдосконалює контейнер додатку, часто незалежно від вмісту першого. В простому розумінні контейнер розширення розширює можливості контейнера додатку, який складно вдосконалити в інший спосіб. Для прикладу перший контейнер може бути веб сервером, що об’єднаний з контейнером розширення, що виконує функцію логування. Також способом застосування цього патерну може бути періодична синхронізація даних з іншим сервером чи репозиторієм git [9].

Рисунок 1.6 - Шаблон «розширення»

2. Шаблон «посередник» – контейнери з’єднано між собою за допомогою проксі [11], тому додаток розцінює підключення з іншими вузлами розподіленої системи як з’єднання з локальним сервером (рисунок 1.7). Це спрощує розробку таких програм тому, що налаштовувати потрібно лише підключення до локального сервера. Це можливо тому, що контейнери, що розміщені за одній машині підключаються до одного і того ж локального мережевого інтерфейсу.

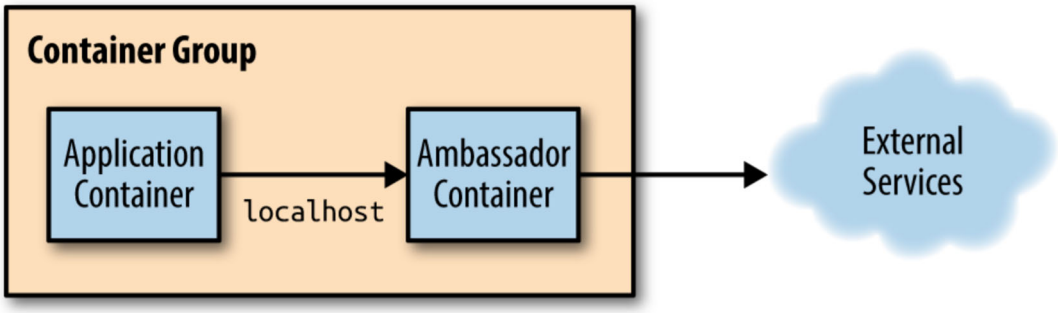


Рисунок 1.7 - Шаблон «посередник»

3. Шаблон «адаптер» [12] – контейнер-адаптер перетворює вихідні дані контейнера додатку так, щоб їх можна було використати будь-де, тобто стандартизує вивід програми для використання в іншому середовищі (рисунок 1.8). Також шаблон адаптера забезпечує взаємодію різного роду додатків, що були створені раніше, без модифікації їх вихідного коду.

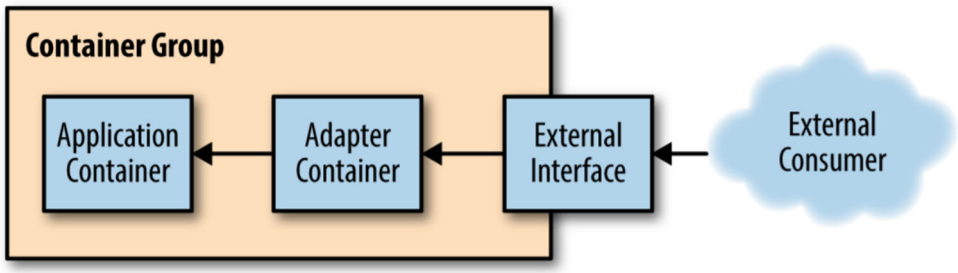


Рисунок 1.8 - Шаблон «адаптер»

До багатоконтейнерних шаблонів проєктування програмного забезпечення можна віднести наступні:

4. Відтворювані сервіси з керованим навантаженням [13, 38]. Це найпростіший шаблон, при якому кожен сервер має ідентичні характеристики та пропускну здатність мережі (рисунок 1.9). Патерн складається з легко масштабованої кількості серверів та контейнера, що керує навантаженням кожного реплікованого вузла. Відтворювані сервіси не потребують наявності стану для коректної роботи. В найпростішому додатку без стану індивідуальні запити можуть бути розділені між екземплярами системи. Сервіси без стану можуть включати сервери зі статичним контентом та складні системи проміжного програмного

забезпечення, що отримує та об’єднує запити з безлічі різних систем. Системи без стану відтворюють для забезпечення надмірності та масштабування. Незалежно від розміру сервісу необхідно мати щонайменше два його екземпляри для забезпечення високого рівня доступності ресурсу.

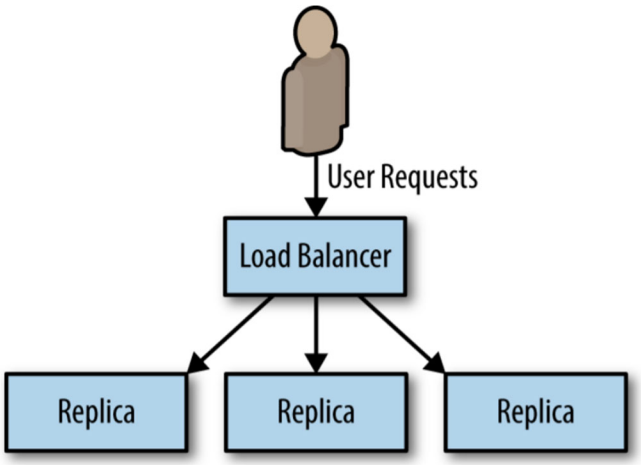


Рисунок 1.9 - Базовий відтворюваний сервіс без стану

5. Шаблон «сегментування» [14] – в такій системі клієнт надсилає початковий запит до кореневого вузла (рисунок 1.10). Цей вузол передає запит великій кількості серверів для паралельного обчислення. Кожен вузол повертає часткові дані в корінь, що збирає всю інформацію в єдину відповідь на користувацький запит. Розробка такої системи передбачає використання великої кількості типових кодів, таких як: поширення запитів, збір відповідей, взаємодія з клієнтом тощо. Досить велика частина цього коду є загальною та може бути реконструйована в єдиний шаблон, шо буде використовуватись довільними контейнерами. Для реалізації системи сегментування необхідно задати два контейнери. Спочатку контейнер, що обчислює свою частину даних та повертає відповідний результат. Другий – контейнер злиття, що збирає всі часткові результати в єдину відповідь.

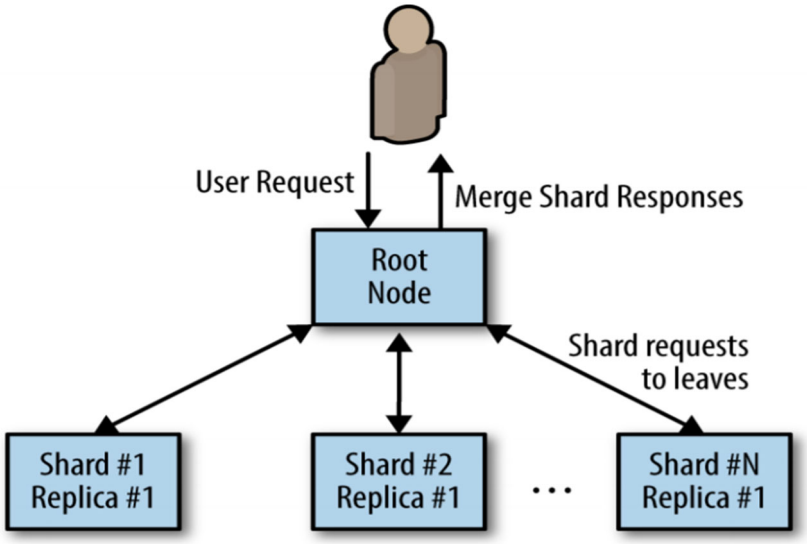


Рисунок 1.10 – Шаблон «сегментування»

6. Вибір лідера [15] – в мережі з єдиним центром питання вибору «лідера» мережі не існує, оскільки існує тільки одна програма, що встановлює права власності та гарантує, що певний суб'єкт володіє тільки конкретними даними. Однак подібні обмеження знижують надійність системи. Для того, щоб підвищити доступність системи необхідна система встановлення власності. Розглянемо рисунок 1.11. Нехай існує три копії контейнера, що можуть бути власниками. Початково власником даних є перший контейнер. Потім ця репліка перестає функціонувати і їй на зміну власником стає третя репліка. Після відновлення роботи першої репліки третя все ще залишається лідером / власником. Часто вибір лідера є найважливішим та найскладнішим аспектом при розробці розподілених систем.

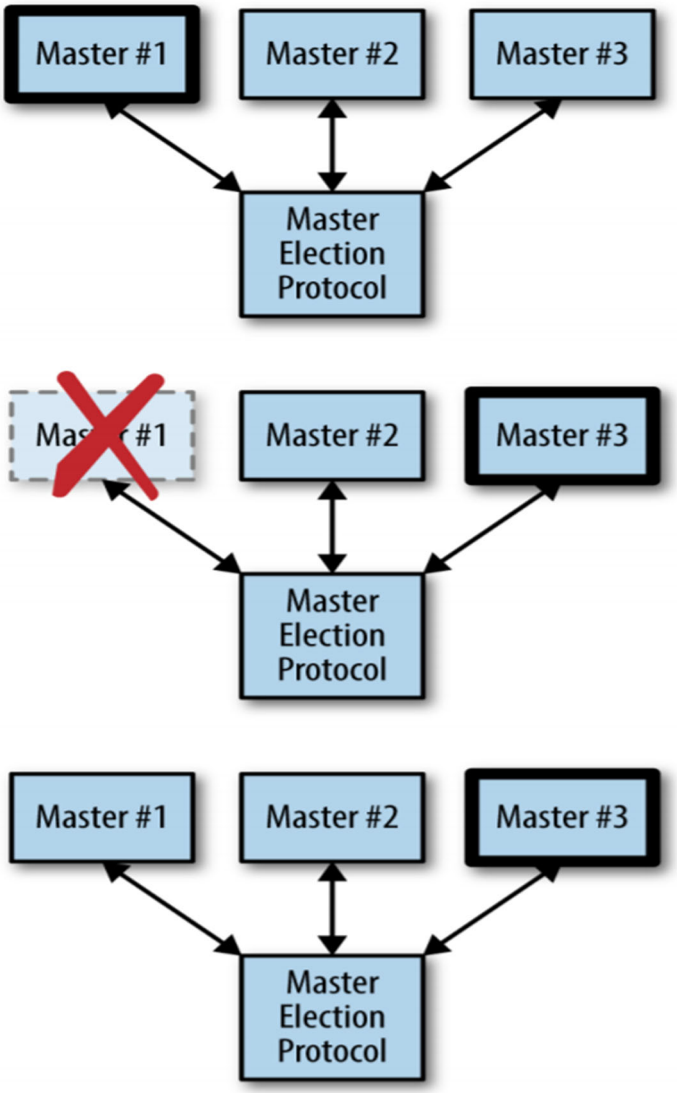


Рисунок 1.11 – Вибір лідера

В багатьох випадках розподілені системи чи додатки розробляються з використанням визначеного архітектурного стилю. Проте обраний стиль може бути незастосовним для всіх випадків та потребує адаптивної зміни поведінки проміжного програмного забезпечення [31]. Проміжне ПЗ використовують для забезпечення взаємодії клієнтів, що мають різні операційні системи чи інші користувацькі вимоги. Для визначення цих умов використовуються так звані перехоплювачі, що визначають алгоритм взаємодії з сервером на основі отриманих даних про клієнта. Перехоплювачі існують двох типів: зовнішні та внутрішні. Зовнішні перехоплювачі можуть перехоплювати як системні виклики, так і бібліотеки. В першому випадку відбувається підключення до основного процесу за

допомогою його ідентифікатора в системі, що дає можливість витягувати та модифікувати значення, що повернули системні виклики чи значення аргументів. Перехоплення бібліотеки дозволяє змінювати визначені за замовчуванням алгоритми в бібліотеці перехопленого процесу. В даному випадку можна вибірково модифікувати виведені підпрограмами дані, їх аргументи та семантику. Вбудовані перехоплювачі дозволяють змінювати користувацькі запити та відповіді сервера ще до їх відправки.

Отже ідея розподіленої архітектури полягає в представленні компонентів у різний спосіб та їх взаємодії через уніфікований протокол передачі даних. Для використання різних конфігурацій компонентів та інтерфейсів можна застосувати такі архітектурні стилі: багатошарова, об’єктно-орієнтована, ресурсна та орієнтована на події архітектура [37].

Багатошарова архітектура полягає в тому, що компоненти організовано в рівні. Компонент рівня звертається до компонента на рівень нижче та очікує на його відповідь. Лише у виключних випадках компоненти можуть звертатись до компонентів вищого рівня [32].

Об’єктно-орієнтована та сервісна архітектури складаються з менш структурованого набору мало зв’язаних об’єктів. Кожен об’єкт можна назвати компонентом. Об’єднання компонентів відбувається за допомогою віддаленого виклику процедур. Прикладами є Java RMI та виклики REST API. У випадку розподіленої системи викликані об’єкти не обов’язково мають виконуватись на тій же машині, де було виконано виклик. Об’єктно-орієнтована архітектура приваблива тим, що забезпечує інкапсуляцію даних (станів об’єкта) та операцій (методів) в єдине ціле. Наданий об’єктом інтерфейс приховує деталі реалізації, що робить його незалежним від середовища. Саме тому об’єкт можна розглядати як окрему сутність, що також може містити в собі інші сервіси. Розділення сервісів на незалежно функціонуючі частини і полягає в сервісно-орієнтованій архітектурі. Тобто додаток будується як набір різних служб [33, 34].

Архітектура на основі ресурсів може розглядатись як великий набір функцій, якими компоненти можуть оперувати індивідуально. Ресурсами можна оперувати

програмно: додавати, вилучати, отримувати та змінювати. Цей підхід більш відомий як REST. Архітектура RESTful набула популярності завдяки своїй простоті [35].

Архітектура, орієнтована на події має жорстке розділення між обробкою та координуванням. Система розглядається як сукупність незалежних процесів. В цій моделі координація стосується взаємодії між процесами. Особливістю є те, що процеси не мають явного посилання один на одного, а обмін даними відбувається з описом подій, що цікавлять клієнта [36].

1.2.2 Аналіз засобів виявлення мережевих атак з відкритим вихідним кодом

Для своєчасного та ефективного реагування на мережеві атаки [16] необхідно мати широкі знання про існуючі інструменти та системи, що доступні у вільному доступі. Особливо це важливо для малих та середніх підприємств, що не можуть дозволити собі комерційні засоби виявлення кібератак. Нижче представлено декілька систем виявлення вторгнень:

1. Snort [17] – найпопулярніший інструмент з відкритим вихідним кодом. Здатний працювати під керуванням операційних систем Windows, Unix та Linux. Аналізує трафік в режимі реального часу та має три режими роботи: сніфер пакетів, реєстратор пакетів та виявлення вторгнень. В основі останнього режиму є набір правил, що можна створити вручну або завантажити з офіційного сайту Snort. Цей програмний продукт може виявляти сканування портів, зондування SMB [18], спробу отримання відбитку ОС та багато інших атак за допомогою методів, що базуються на мережевих аномаліях та цифрових підписах.

2. Zeek [20] – як і Snort використовує методи виявлення вторгнень на основі аномалій в мережі та цифрових підписів. Може працювати під керуванням ОС Linux, Unix і Mac OS. Zeek реєструє та аналізує мережевий трафік на рівні програми, тому може відстежувати служби з різних рівнів моделі OSI [19], наприклад FTP, SNMP, HTTP тощо.

3. Suricata – один із головних конкурентів Snort. Має можливість багатопотокової обробки, прискорення за допомогою графічного процесора та виявляє статистичні відхилення за кількома моделями. Також ця система сумісна зі структурою даних Snort, що дозволяє імпортувати політики безпеки. Suricata перевіряє сертифікати TLS/SSL, транзакції DNS та запити HTTP.

4. OpenWIGS-ng – безкоштовна система виявлення вторгнень з відкритим вихідним кодом від розробників відомого інструменту Aircrack-ng. Може використовуватись як сніфер пакетів, або для виявлення мережевих атак. Працює ця система лише під керуванням ОС Linux, що є недоліком.

5. Sguil [22] – надає інформацію про події в мережі в режимі реального часу. Містить компоненти для моніторингу мережевої безпеки. Може працювати в будь-якій операційній системі, що використовує tcl/tk [23].

6. Security Onion – операційна система на базі Linux, призначена для виявлення вторгнень та моніторингу безпеки мережі. Платформа надає комплексне виявлення кібератак, моніторинг мережевої безпеки та керування журналами. Поєднує можливості вище описаних програмних продуктів та інших систем виявлення вторгнень.

Описані вище застосунки об’єднує спосіб перехоплення мережевих пакетів. Розглянемо як приклад інструмент для мережевого аналізу для операційної системи Windows Npcap. Він включає низькорівневу та високорівневу бібліотеки для фільтрування пакетів. Для отримання необроблених даних, що передаються в мережі система захоплення повинна обійти деякі протоколи операційних систем. Для цього потрібна частина, що працює в середині ядра ОС для взаємодії з драйверами мережевого інтерфейсу. В пакеті Npcap вона реалізована як драйвер пристрою, що дозволяє перехоплювати та надсилати пакети, а також має систему фільтрації та механізм моніторингу. Далі система захоплення має експортувати інтерфейс, який буде використовуватись додатками високого рівня. Для цього Npcap надає дві різні бібліотеки: packet.dll та wpcap.dll. Перша бібліотека надає API низького рівня для взаємодії безпосередньо з драйвером пристрою. Wpcap.dll надає

високорівневий набір методів захоплення пакетів,що сумісні з libpcap, відомою бібліотекою захоплення для Unix-систем.

Драйвер Npcap (NPF) виконує найважливішу роль: обробку пакетів, що проходять в мережі та експортує можливості перехоплення, надсилання та аналізу на рівень користувача. NPF реалізований як драйвер фільтру. Для того щоб надати доступ до необробленого трафіку він реєструється як модифікуючий драйвер фільтрації FilterClass.

Для моніторингу мережі Npcap містить модуль моніторингу на рівні ядра, що здатний обрахувати просту статистику мережевого трафіку. Для відображення результатів моніторингу мережі не обов'язково копіювати статистичні дані до програми. Це дозволяє уникнути значного навантаження на обчислювальну одиницю. Механізм моніторингу складається з класифікатора та лічильника. Пакети класифікуються за допомогою NPF, що дозволяє обрати частину трафіку для подальшої обробки. Відфільтровані дані надходять до лічильника, який зберігає деякі змінні, такі як кількість пакетів та кількість байтів, отриманих фільтром та оновлює їх даними вхідних пакетів. Ці змінні регулярно передаються додатку на рівні користувача з визначеною вручну періодичністю.

1.2.3 Аналіз методів виявленні мережевих атак

Методи виявлення мережевих атак можна розділити на наступні категорії:

1. Сигнатурний аналіз [25] – методи цієї групи сканують характеристики одного чи декількох мережевих пакетів для визначення підозрілих дій в мережі. Наприклад правило Snort може бути написане для визначення керуючого трафіку між зараженим пристроєм та зловмисником. Якщо при цьому використовується шифрування – такі пакети складніше ідентифікувати як загрозу. Найбільшим обмеженням цих методів є те, що необхідно мати екземпляр вірусу чи розуміння мережевої атаки, щоб скласти сигнатуру для їх виявлення. Це означає, що програма, яка використовує лише бібліотеку сигнатур не зможе ідентифікувати невідомі та нові мережеві загрози [27].

2. Поведінковий аналіз [26, 39-40] – на відміну від сигнатурного виявлення методи поведінкового аналізу не шукають унікальні характеристики загрози, а аналізують результат її дій. Іншими словами ці алгоритми спостерігають за системою і визначають симптоми – результат дії зловмисника в мережі. Перевага цього типу аналізу полягає в можливості визначення невідомих атак. Однак можливе помилкове спрацьовування. Наприклад в пікові години навантаження мережі система може розпізнати активність як DDOS-атаку з бот-мережі [28, 53]. Додаткові відомості допомагають впорядкувати результати сканування мережі, але коли хибних спрацювань системи більше ніж справжніх – рішення може спричинити більше проблем ніж користі. Крім того аналіз поведінки потребує більших затрат ресурсів, тому фінансово не вигідно використовувати його для визначення відомих загроз. Також є ризик пропустити загрозу, що можна визначити за допомогою сигнатурного аналізу.

3. Методи інтелектуального визначення мережевих атак [42-43, 54-55]. До цієї категорії можна віднести машинне навчання та обчислювальній інтелект. В роботі [44, 57] проаналізовано методи штучного інтелекту, для розробки адаптивних систем виявлення мережевих атак. До методів машинного навчання можна віднести Байєсівські мережі, дерева рішень, алгоритми регресії та інші. Генетичні алгоритми, нейронні мережі та системи з нечіткою логікою відносяться до методів обчислювального інтелекту. Ці методи застосовуються для визначення аномалій та зловживань, оскільки вихідні дані для системи складаються як і з нормального трафіку так і аномальної активності в мережі [56-60].

Для забезпечення збалансованого та багаторівневого захисту від кібератак слід комбінувати методи поведінкового та сигнатурного аналізу. Близько 80% загроз можна легко ідентифікувати за допомогою сигнатурного аналізу. Насправді цей підхід є найефективнішим для виявлення відомих загроз, тому залишається досить важливим методом. Однак інші 20% неможливо ідентифікувати за допомогою підписів. Як правило цілеспрямовані атаки на організації не є відомими, тому оцінка поведінки є безсумнівно важливою.

1.3 Висновки до першого розділу

В сучасному інформаційному просторі надзвичайно актуальним є питання безпеки. Шляхів досягнення цієї мети є безліч. Кожен метод має як переваги так і недоліки. Аналіз відомих методів захисту інформації показав, що на ефективність методу впливає структура мережі, протоколи підключення, тип підключення та безпосередньо алгоритм роботи системи. Серед безкоштовних систем захисту від вторгнень можна виділити два типи аналізаторів: сигнатурний та поведінковий.

Спосіб реалізації розподіленої системи відіграє не менш важливу роль в безпеці системи та швидкості її роботи. Досліджені шаблони проєктування дозволяють розробляти самодостатні системи для широкого спектру завдань.

Отже, з метою підвищення рівня безпеки розподілених систем доцільним є вдосконалення відомих та розробка нових методів визначення мережевих атаки в розподілених системах.

2 АРХІТЕКТУРА РОЗПОДІЛЕНОЇ СИСТЕМИ

В попередньому розділі було описано низку методів реалізації розподіленої системи. Кожна представлена архітектура має свої особливості реалізації, логіку роботи та обмеження. Тому неможливо використати одну архітектуру для реалізації будь-якої системи. Для прикладу існують архітектурні стилі, що підходять для розробки настільних додатків, але незастосовні для реалізації веб-застосунку.

Для розробки розподіленої системи було обрано технологію .NET Core та сформовано наступні компоненти рішення:

1) веб-клієнт додатку для відображення статистичної інформації;

2) налаштування сутностей бази даних;

3) налаштування таблиць бази даних та відношень від ними;

4) репозиторій – сховище методів для віддаленого виклику процедур;

5) спільні налаштування всіх компонентів;

6) клієнтська частина – програмний додаток для перехоплення та аналізу трафіку «на льоту».

Компоненти пов’язані структурно та залежать один від одного. Таким чином було сформовано багаторівневу структуру. Для вирішення завдання розподіленої системи компоненти програмного забезпечення повинні взаємодіяти і в той же час бути автономними. Тому було обрано сервісно-орієнтований підхід. Методи віддалених компонентів можна виконати за допомогою віддаленого виклику процедур, що реалізовано в репозиторії. Обмін даними відбувається через реляційну базу даних. Клієнтський додаток містить наступні складові: головний клас Program, модуль для фонового виконання Worker, обробник події прийому пакетів Capture Handler, алгоритм служби CaptureTask, клас визначення та підключення до мережевого адаптера PcapDevice, клас для збору інформації про клієнта та метод для підключення до бази даних. Програмне забезпечення повністю сумісне з операційними системами Windows та Linux, оскільки технологія .NET крос-платформна. З її допомогою розробляють веб-додатки, настільне та мобільне

програмне забезпечення, включаючи 2D та 3D ігри, мікросервіси, хмарні сервіси, штучний інтелект. Клієнтський додаток потребує встановлення додаткових бібліотек в операційну систему. Для Windows це бібліотеки Npcap, а для Linux – libpcap. Веб-клієнт також використовує технологію .NET та отримує дані з СУБД використовуючи методи зі спільного репозиторію. Це значно зменшує розмір додатку через відсутність необхідності повторювати реалізацію методів. Репозиторій виконує роль API та дозволяє працювати з базою даних. Класи репозиторію реалізують відповідні інтерфейси. Його структуру зображено на рисунку 2.1.

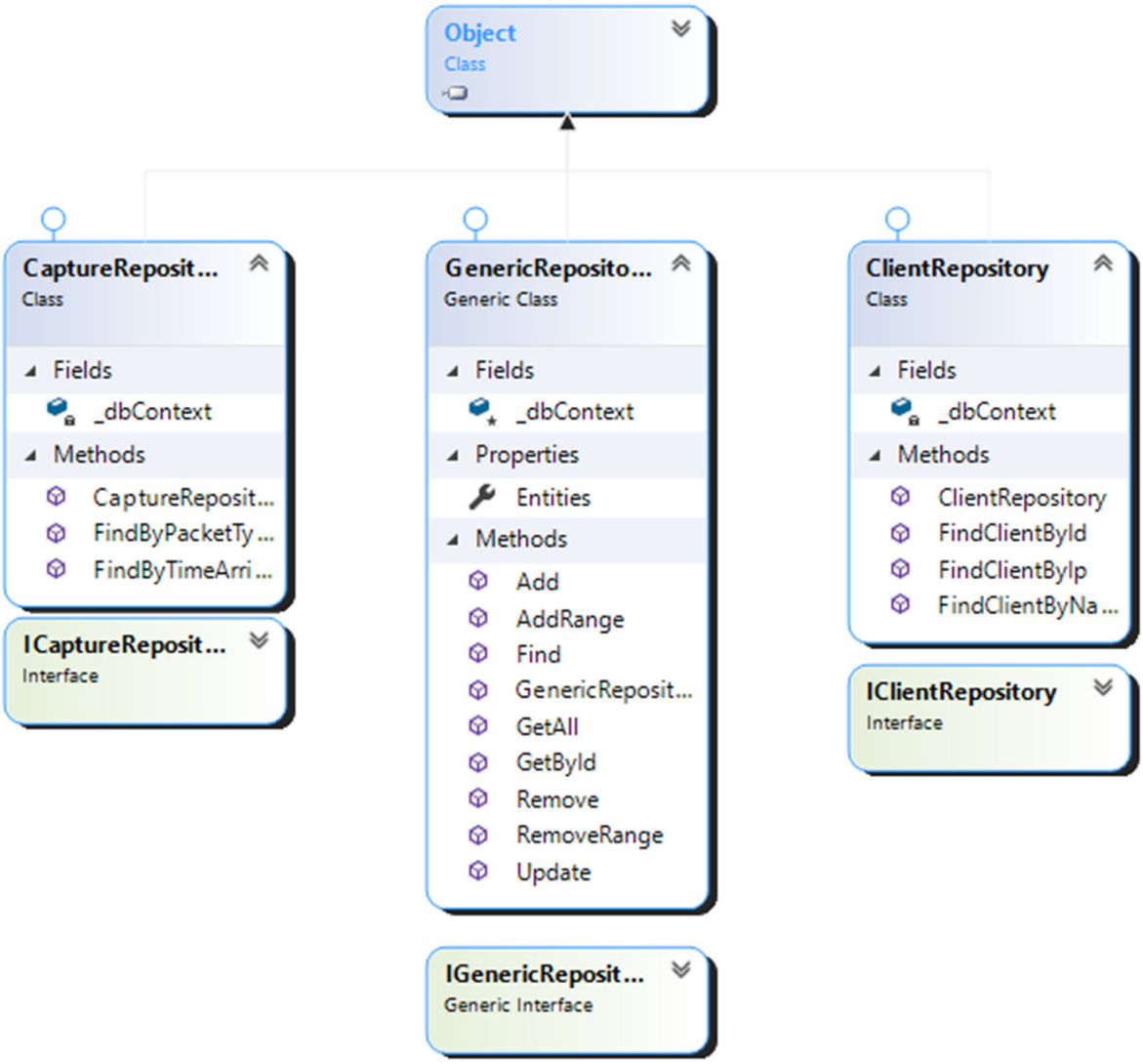


Рисунок 2.1 – репозиторій

Конфігурацію бази даних здійснюється з використанням Entity Framework Core. Як і бібліотека фреймворк містить в собі код для багаторазового виконання. Але на відміну від бібліотеки потребує виконання вимог щодо структури коду. Entity Framework Core – об’єктно-орієнтована технологія, розроблена Microsoft, що дозволяє працювати з базами даних. Вона дозволяє абстрагуватись від таблиць бази даних та працювати з ними незалежно від типу сховища. Тому зникає необхідність працювати з таблицями, індексами, первинними та зовнішніми ключами. На концептуальному рівні, що надає EF Core, робота відбувається з об’єктами. Слід зазначити, що дана технологія не має прив’язки до єдиної операційної системи та типу додатка. Це робить Entity Framework універсальною технологією доступу до даних, що також являє собою API для роботи з даними. Таблиці бази даних представлено у вигляді сутностей, що містять дані про назви полів даних (стовпців таблиці) та їх типи даних. Для побудови реляційної бази даних за відповідною схемою необхідно створити класи, що визначають її структуру. Для поставленого завдання база даних повинна зберігати інформацію про клієнта: ідентифікатор, IP-адреса мережеве ім’я, часова мітка останнього підключення, зовнішній ключ, що буде пов’язаний з іншою таблицею, що містить необхідну для визначення інтенсивності трафіку інформацію: ідентифікатор, час та дата отримання пакету, джерело та пункт призначення пакету, тип пакету та ключ таблиці користувачів. На рисунку 2.2 зображено діаграму класів для роботи з Entity Framework Core.

Рисунок 2.2 – діаграма класів Entity Framework Core

Сутності, які використовуються фреймворком містять інформацію про поля даних, що в подальшому будуть використані як стовпці таблиць бази даних. Сутності бази даних представлено на рисунку 2.3.

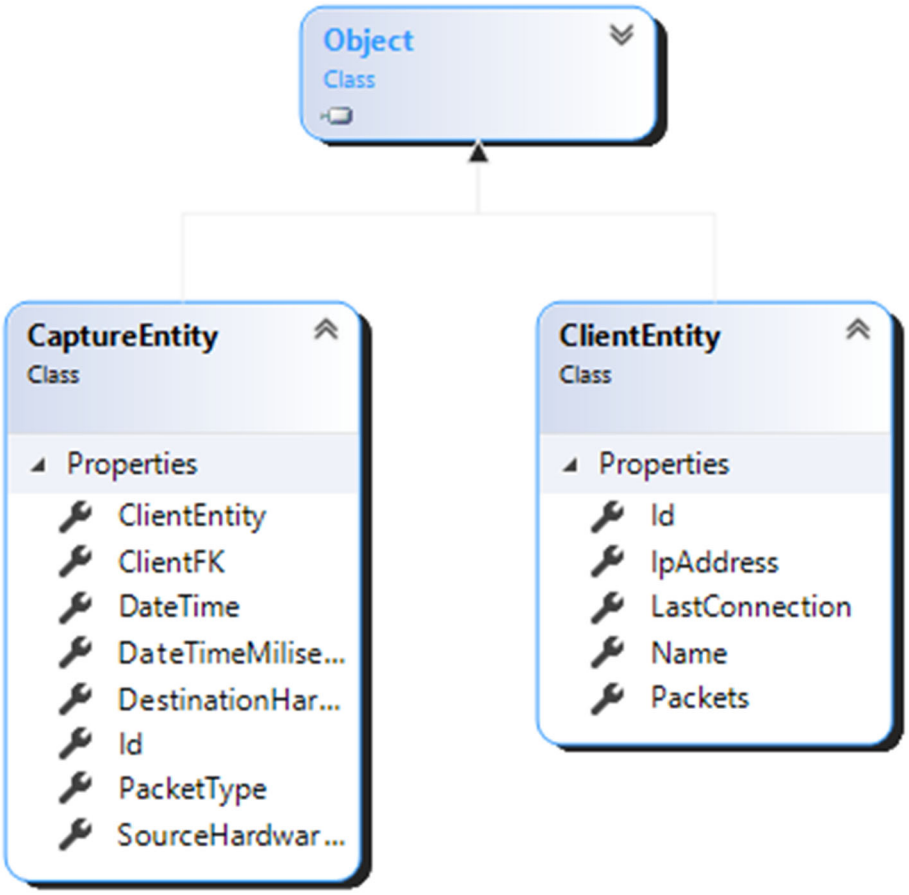


Рисунок 2.3 – діаграма класів сутностей

Відношення в таблицях сформовані так, щоб для кожного унікального клієнта формувалась вибірка мережевих пакетів

Спосіб підключення до даних є спільним для всіх компонентів рішення, тому було визначено загальну конфігурацію підключення, що зображена на рисунку 2.4. Цей компонент слугує для швидкої модифікації рядка підключення у випадку зміни адреси сервера бази даних, оскільки ця інформація за замовчуванням міститься в файлі appsetings.json. Тому за необхідності редагування рядка підключення не потрібно змінювати його в усіх компонентах.

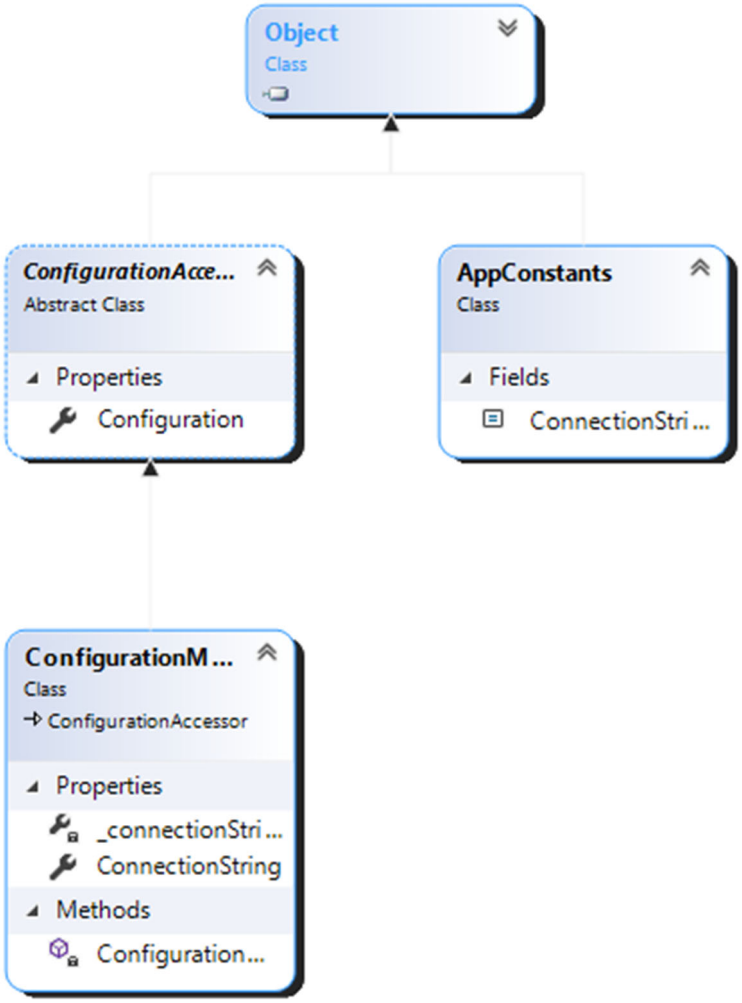


Рисунок 2.4 – конфігурація підключення до бази даних

2.1 Корпоративна мережа

Мережа, що контролюється певною компанією для внутрішньої комунікації та надає доступ до спільних ресурсів і пристроїв виключно для працівників організації може вважатись корпоративною. Як і будь-яка мережа вона використовує комунікаційні стандарти Інтернету та працює на базі протоколу TCP/IP. Як правило корпоративні мережі є географічно розподіленими та об’єднують підрозділи компанії, що знаходяться далеко один від одного.

Структурно корпоративна мережа складається з деякої кількості локальних мереж різних розмірів та способів підключення: провідного та безпровідного.

Для організації мережі необхідні наступні компоненти:

1) щонайменше один сервер;

2) комутатори та патч панелі для з’єднання пристроїв у мережі;

3) маршрутизатори, для з’єднання різних мереж, наприклад з’єднання локальної мережі з інтернетом або створення власного інтранету.

4) кабелі з’єднання, навіть якщо локальна мережа повністю безпровідна вони потрібні для з’єднання з зовнішніми мережами. Найчастіше використовують оптоволоконні кабелі через їх надійність та швидкість передачі даних.

На рисунку 2.5 зображено загальну структуру корпоративної мережі.

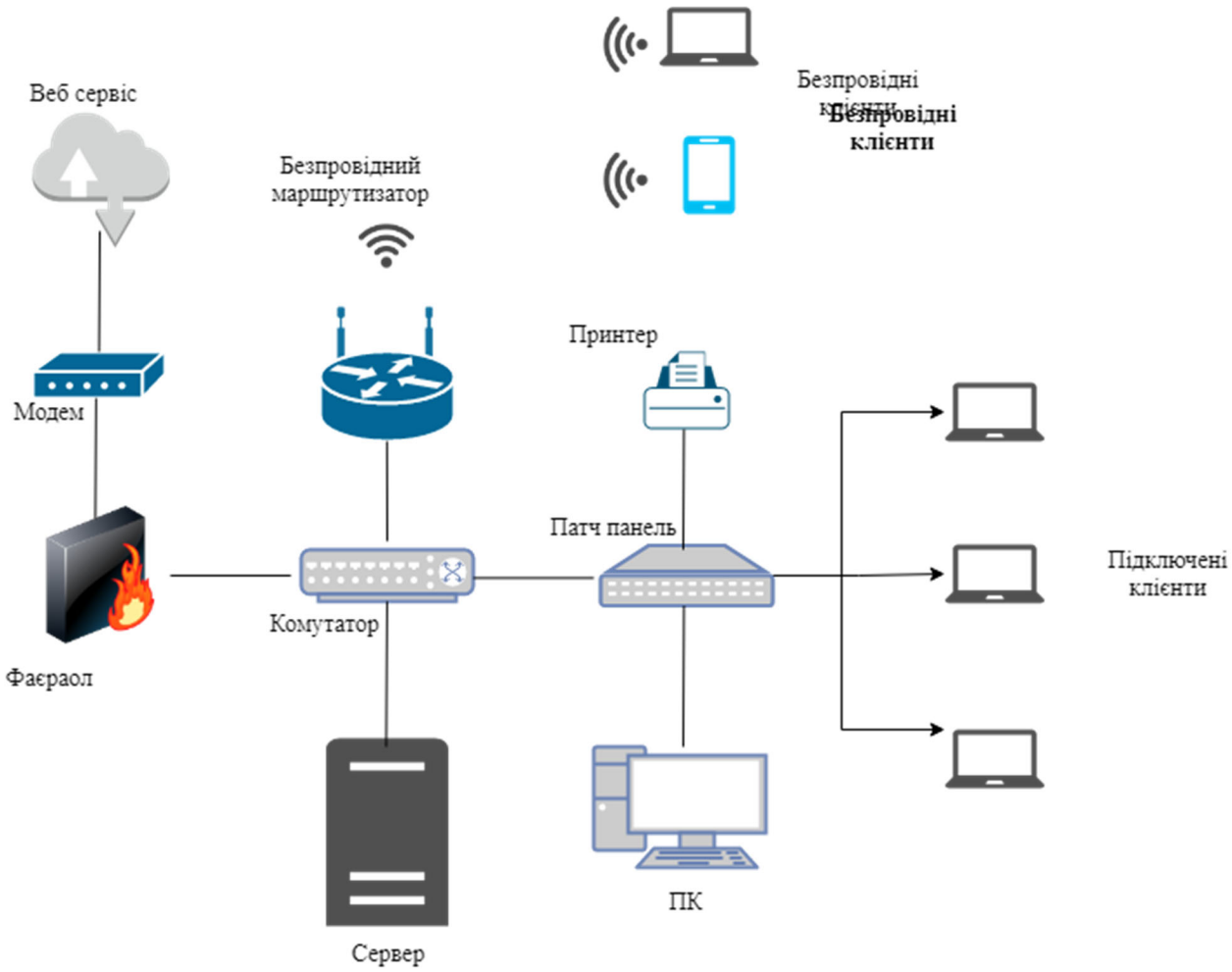


Рисунок 2.5 – Загальна структура корпоративної мережі

Альтернативним способом розгортання корпоративної мережі є створення віртуальної приватної мережі, що дозволяє об’єднувати структурні підрозділи, що розподілені географічно. За своєю сутністю віртуальна приватна мережа є деяким рядом віртуальних з’єднань, що шифрують трафік між клієнтом та ресурсом в мережі інтернет. Враховуючи, що звичайне з’єднання з використанням протоколів

мережі інтернет зашифровано, то з використанням віртуальної приватної мережі дані шифруються щонайменше двічі. Користувачами VPN можуть стати будь-які пристрої, що мають доступ до інтернету: ПК, смартфони, виділені сервери та деякі пристрої інтернету речей.

В результаті підключені за допомогою єдиної віртуальної мережі пристрої можуть співпрацювати так, ніби знаходяться в межах однієї локальної мережі. VPN забезпечує роботу протоколів інтернету та додатково шифрує інформацію, що підвищує рівень безпеки даних та надає додатковий рівень захисту всій системі.

Для ефективної роботи в межах одного підрозділу компанії застосовується програмне забезпечення для адміністрування, планування ресурсів підприємства (ERP-система) та користувацькі додатки, що під'єднані до СУБД – системи управління базами даних. З'єднання між користувачами здійснюється з використанням стандартних протоколів моделі OSI.

Модель OSI – концептуальна модель, створена міжнародною організацією стандартизації (ISO [30]), що дозволяє різноманітним системам зв'язку обмінюватись даними, використовуючи стандартні протоколи. Ця модель має 7 рівнів, що виконують унікальні завдання для мережевого зв'язку.

Прикладний рівень – єдиний рівень, що працює напряму із користувацькими даними. Програмне забезпечення, таке як веб-браузери та поштові клієнти лише ініціалізують підключення, використовуючи прикладний рівень та виводять дані в зрозумілому для користувача вигляді.

Рівень представлення відповідає за підготовку даних для прикладного рівня. Також слугує для перекладу, шифрування та стиснення даних.

Різні пристрої можуть передавати дані з різним шифруванням, тому 6 рівень відповідає за переклад вхідних даних на зрозумілий для програмного забезпечення отримувача синтаксис. Якщо пристрої з'єднані через зашифрований канал – рівень представлення відповідає за шифрування та дешифрування даних для прикладного рівня. Для того, щоб мінімізувати об'єм даних та прискорити їх передачу в мережі – 6 рівень стискає дані перед тим як передати їх на сеансовий рівень.

На п’ятому рівні моделі OSI відбувається керування сеансом. Механізм цього рівня складається із запитів та відповідей між компонентами програмного забезпечення. Найчастіше використовується для віддаленого виклику процедур. У випадку втрати з’єднання протокол сеансового рівня може відновити підключення.

Протоколи транспортного рівня забезпечують підключення між додатками хостів. Серед можливостей є орієнтована на підключення підтримка потоку даних, контроль потоку, мультиплексування та забезпечення надійності.

Мережевий рівень часто називають рівнем пакетів. Цей рівень відповідає за переадресацію пакетів в мережі, включаючи проміжні точки маршрутизації. Також мережевий рівень керує якістю обслуговування, розпізнає та пересилає повідомлення з локального хосту на транспортний рівень.

Рівень каналу передачі даних відповідає за обмін даними між сусідніми вузлами в глобальній чи локальній мережі. Забезпечує функціональні та процедурні засоби для передачі даних між вузлами мережі та може надавати засоби виявлення та виправлення помилок, що можуть виникнути на фізичному рівні. Рівень каналу даних передає інформацію локально, а міжмережева маршрутизація та глобальна маршрутизація – є функціями вищого рівня.

Найнижчим рівнем моделі OSI є фізичний. Цей рівень обробляє та передає дані через фізичне з’єднання, використовуючи механічні, електричні та інші функціональні засоби. Надає спільне використання передавального середовища, шляхом статичного та динамічного мультиплексування.

На рисунку 2.6 зображено загальну схему моделі OSI та об’єкт взаємодії кожного рівня.

Дані	Рівень
Дані	Прикладний
Дані	Представлення
Дані	Сеансовий
Блоки	Транспортний
Пакети	Мережевий
Кадри	Канальний
Біти	Фізичний

Рисунок 2.6 – Загальна схема моделі OSI

2.2 Центр прийняття рішень

Досить велика кількість мережевих атак пов’язана з людським фактором, що часто стає причиною витоку даних чи перехоплення контролю над системою зловмисниками. Тому необхідно, щоб система самостійно реагувала на мережеві атаки. Для цього повинен бути центр, що керуватиме підключенням клієнтів та їх взаємодію між собою. В такому випадку користувач не зможе втручатись в роботу системи. Самодіагностика та автоматичне реагування суттєво зменшить кількість раптових збоїв чи відмов інформаційної системи. На рисунку 2.7 зображено архітектуру системи з самообслуговуванням.

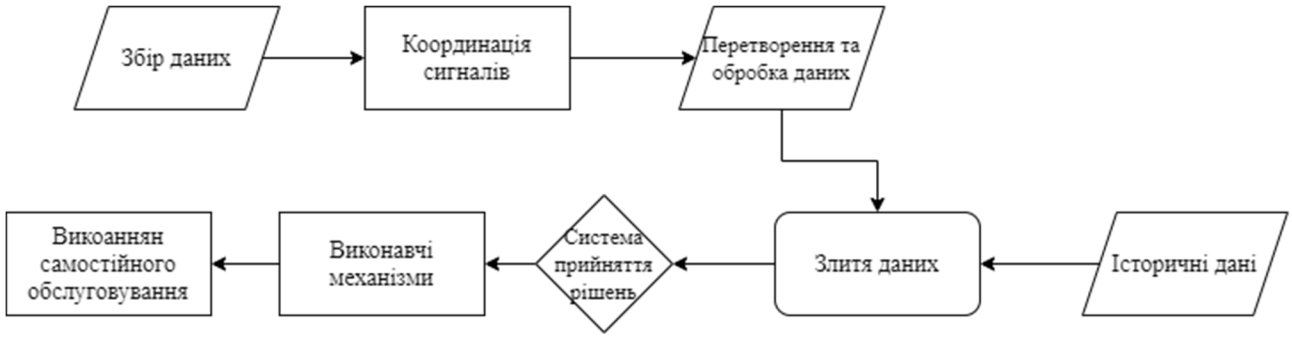


Рисунок 2.7 – Архітектура системи з самообслуговуванням

Збір даних також має бути автоматичним, тому завдання вибору мережевого пристрою для моніторингу, початкова обробка та отримання даних про систему також повинні виконуватись без участі користувача.

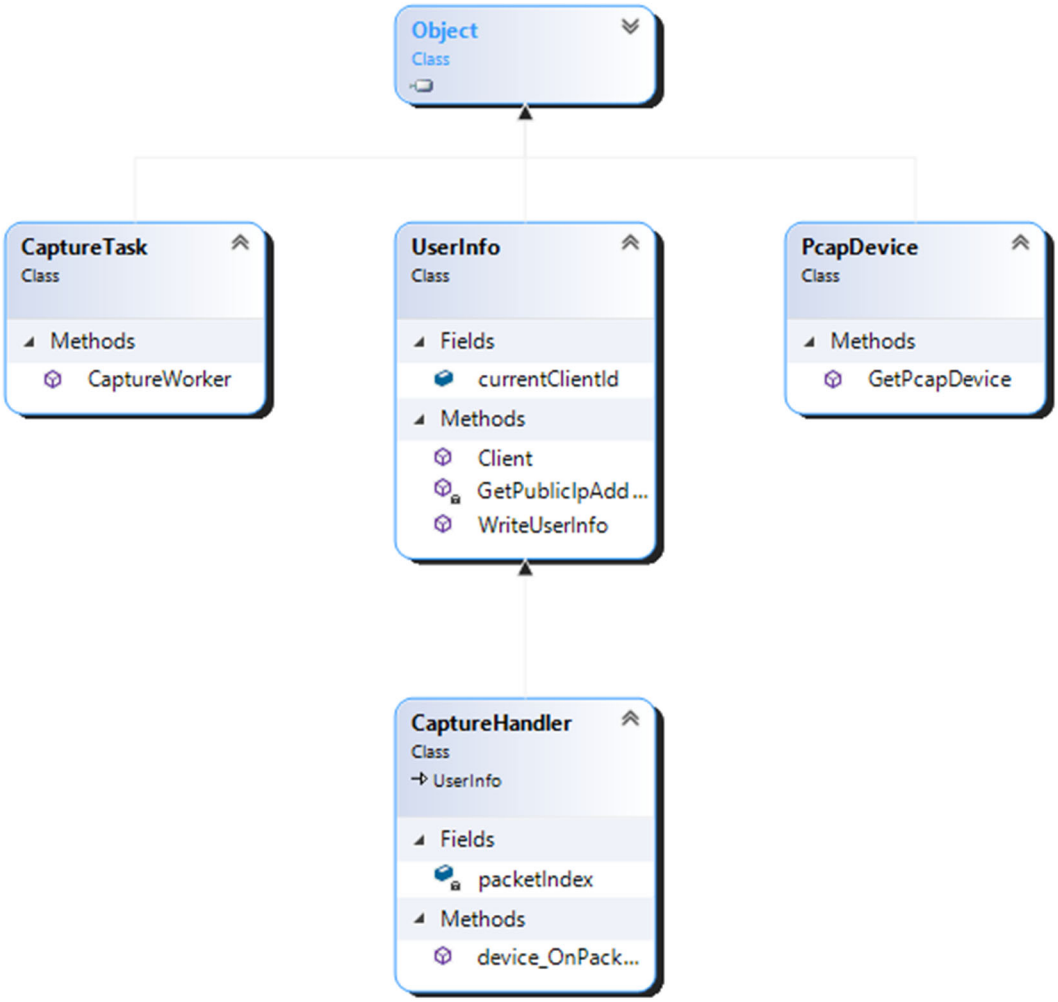


Рисунок 2.8 – Збір інформації

Для прийняття системою правильного рішення та уникнення помилкових спрацювань потрібно не тільки задати критерії підозрілого трафіку, необхідно зберігати історію мережевого трафіку всієї системи. Це дозволить не тільки безпомилково визначати загрози, а й адаптуватись до змін у використанні мережі, оскільки висока мережева активність не завжди означає наявність мережевої атаки.

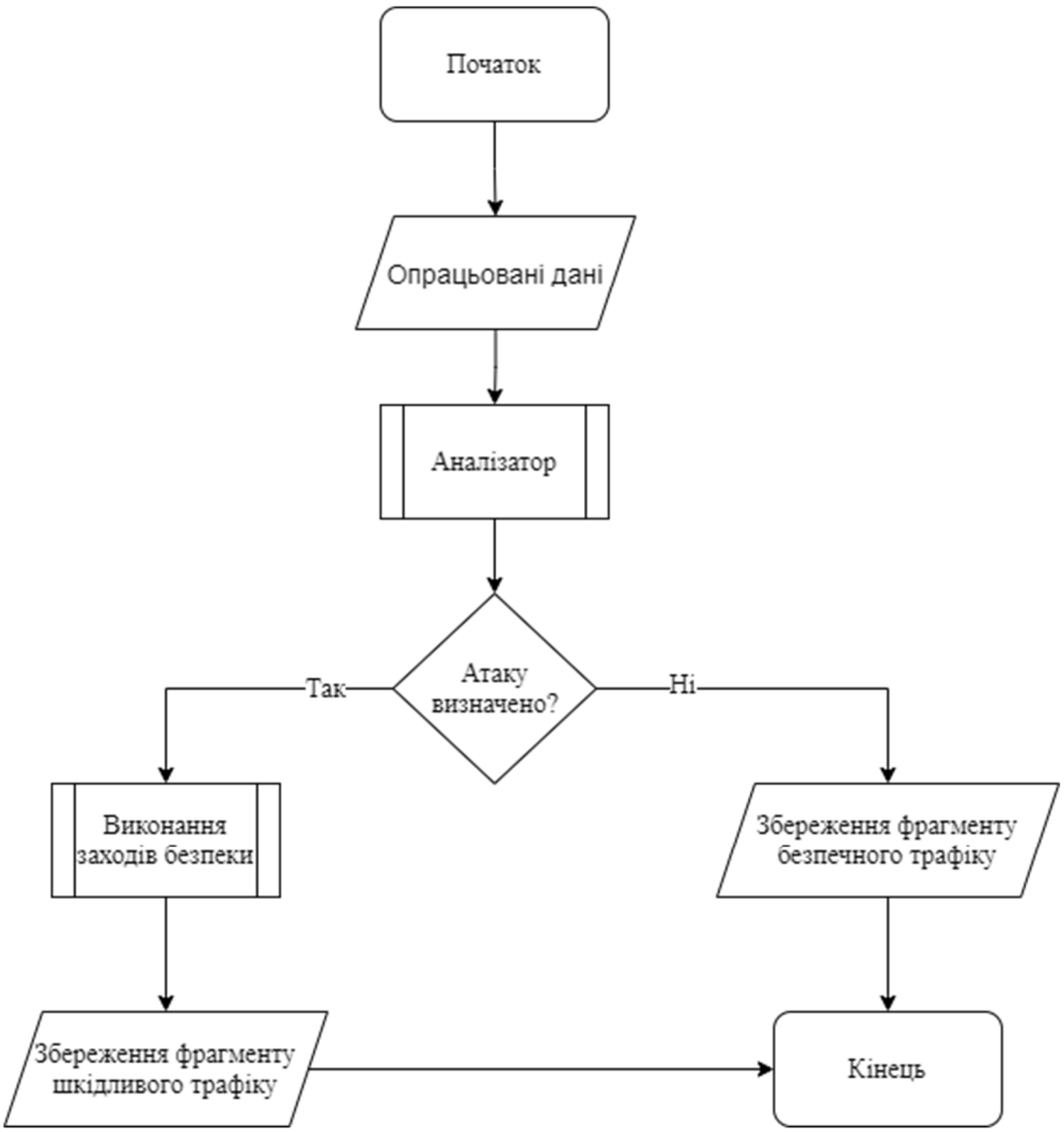


Рисунок 2.9 – Центр прийняття рішень

На рисунку 2.9 зображено загальну логіку роботи центру прийняття рішень. Перевірка трафіку в цьому випадку повинна виконуватись з певною періодичністю та з різним масштабом, оскільки різні типи атак мають власні характеристики.

Заходи безпеки мережі повинні передбачати вилучення враженого вузла з мережі на деякий час та пошук альтернативних шляхів для передачі інформації. Тому для ізоляції загрози буде доцільним використання системного брандмауера та керування його правилами.

2.3 Стійкість до відмов та цілісність розподіленої системи

Згідно представленої в першому розділі архітектури «вибір лідера» та загальної концепції розподілених систем, при якій проводиться опитування вузлів та встановлення загальної оцінки довіри до вузла. При виявленні підозрілої активності вузла проводиться повторне опитування, що відкликає результати першого, в результаті чого обмін даними з цим вузлом блокується по всій мережі. Головною умовою підтримка цілісності є наявність вузлів, що можуть взяти на себе керуючу роль, що забезпечить відмовостійкість системи у випадку успішної мережевої атаки. Кількість таких вузлів має бути більше половини, для забезпечення кворуму – мінімальної кількості керуючих одиниць та безперебійного функціонування кластеру. Нехай маємо 9 вузлів в мережі, включаючи керуючий – рисунок 2.10.

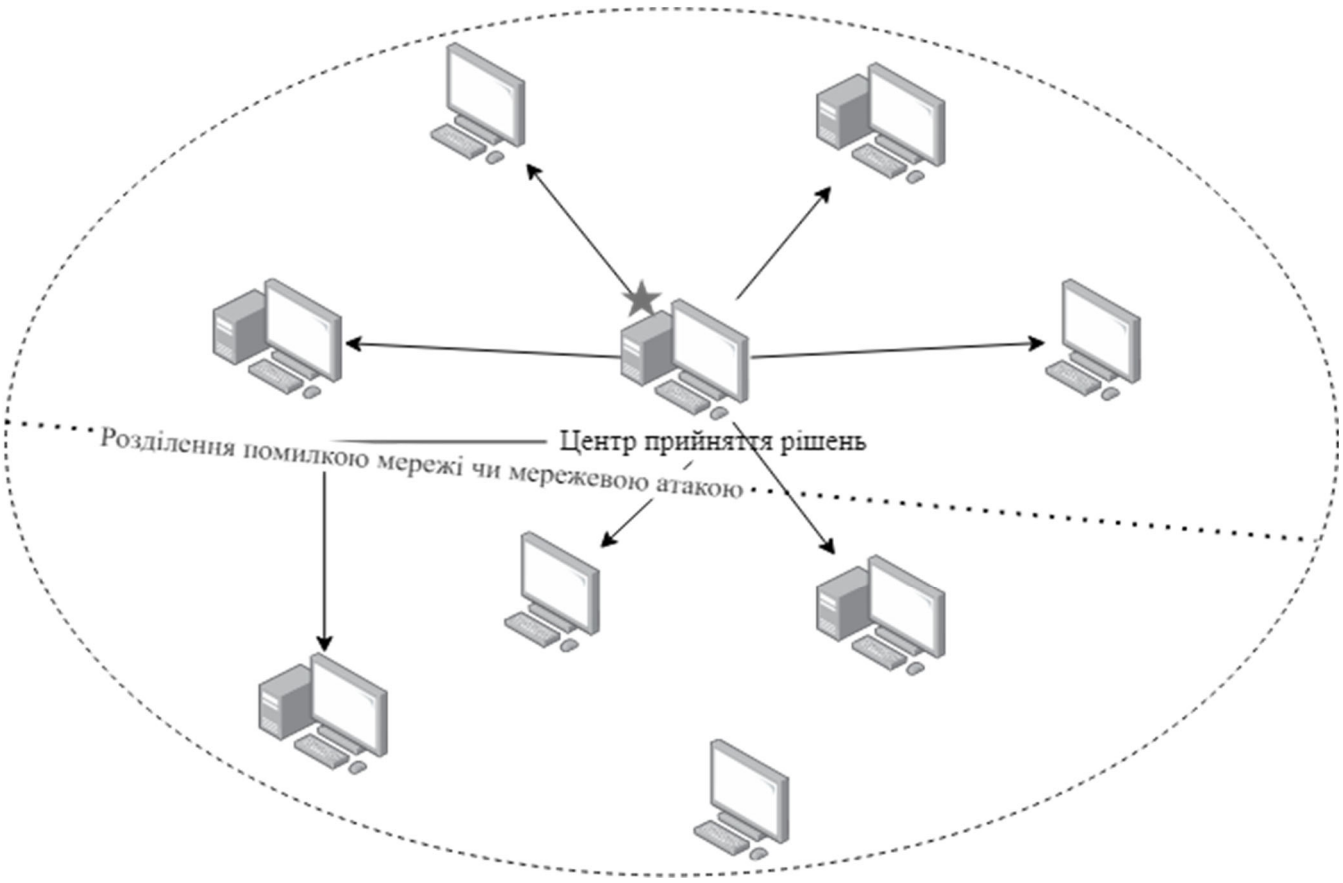


Рисунок 2.10 – Розподілена мережа з центром керування

В результаті мережевої атаки відбулось розділення вузлів, після якого центр керування керує лише 4 вузлами з 8, в той час як інші не мають лідера. Процес вибору лідера починається за умови, коли звичайний вузол довго не отримує вказівки від керуючого. Далі цей вузол отримує статус кандидата. Інші вузли проводять голосування за того кандидата, від якого вони отримали перший запит. В результаті маємо наступну схему підключення, що зображено на рисунку 2.11.

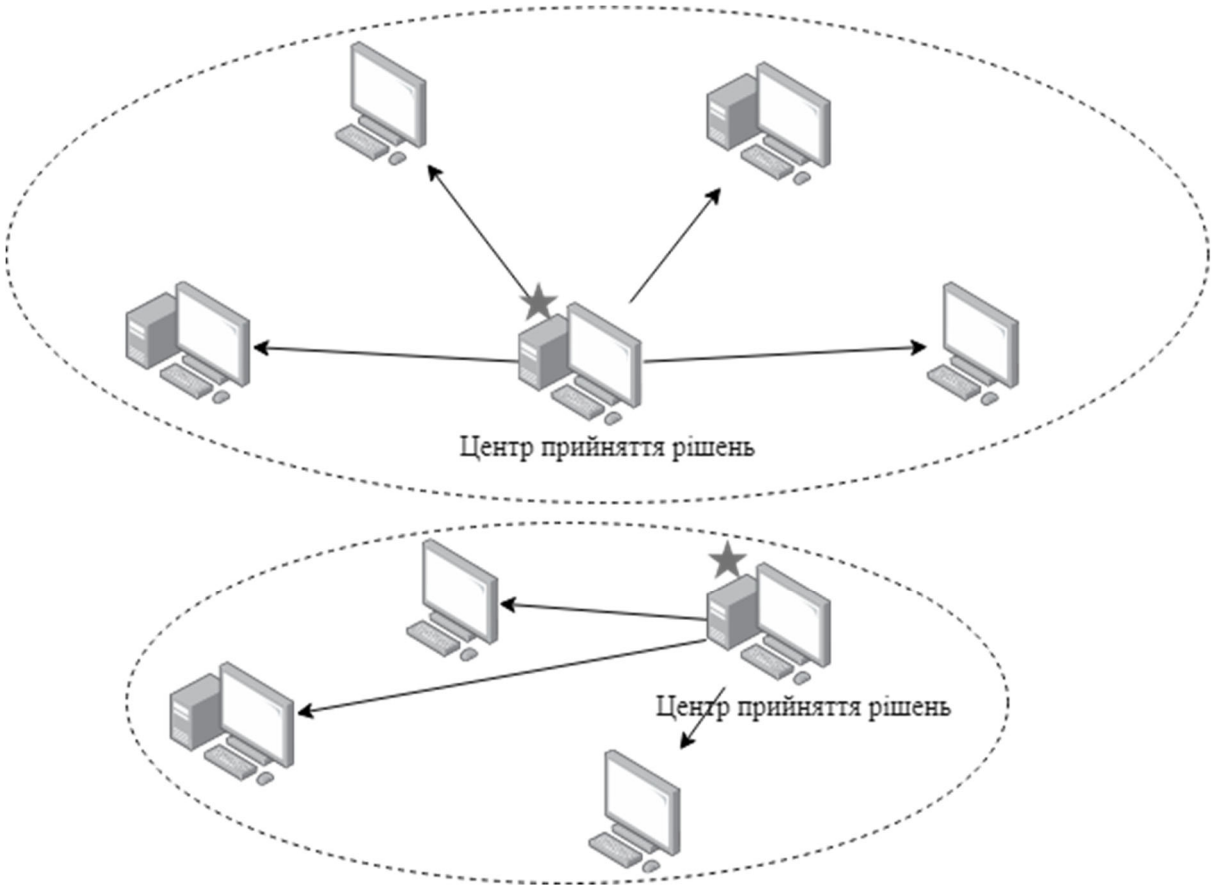


Рисунок 2.11 – Нова схема розподіленої мережі

Після зміни конфігурації маємо дві незалежні мережі. Функціонування системи продовжується, оскільки центр прийняття рішень було обрано новий для другої групи вузлів. При відновленні підключення кластери об’єднуються із поверненням компонентів системи до попереднього стану та відновлення центру прийняття рішень за замовчуванням, якщо він доступний. Обмеженням цього методу є потреба в непарній кількості вузлів для забезпечення роботи системи, оскільки для 4 вузлів необхідно мати 3 центри, що насправді не підвищує стійкість

до відмов та не виправдовує економічні затрати на підтримку додаткового керуючого вузла.

Розподілена з багатьма менеджерами може бути розділена на зони доступності – додаткове місце, де може розміщуватись сервер. У випадку втрати однієї зони доступності – залишаються інші, що підтримують роботу системи. Для забезпечення кворуму рекомендовано розподілити систему на 3 зони доступності.

Таблиця 2.1 – розподіл керуючих вузлів на зони доступності

Кількість керуючих вузлів	Розподіл на зони доступності
3	1-1-1
5	2-2-1
7	3-2-2
9	3-3-3

Для розподілення на зони доступності необхідно слідувати кворуму для забезпечення стійкості до відмов у кожній з цих груп. У випадку, якщо система втрачає зв’язок з одним із трьох вузлів – все ще є підключення до інших двох. Для п’яти менеджерів неможливо провести рівний розподіл на зони. Тому маємо по два вузла в двох зонах та один в третій. У випадку втрати однієї зони ми все ще маємо щонайменше три менеджера, що задовольняють кворум. В такий же спосіб працює розподіл для семи та дев’яти вузлів.

2.4 Висновки до другого розділу

В другому розділі було описано архітектуру розполіденої системи та вимоги до мережі, що дозволять вважати її корпоративною. За основу було взято сервісно-орієнтований підхід, що дозволяє створювати гнучкі системи з можливістю подальшого вдосконалення та динамічного розширення. В цій системі сервер виконує роль сховища даних та центру прийняття рішень. Клієнтська частина виконує збір мережевої інформації та передачу даних на сервер. З’єднання

компонентів виконується за допомогою двостороннього зв’язку клієнта та сервера з метою своєчасного прийняття рішення в разі визначення мережевої атаки. Система передбачає встановлення на операційні системи Windows та Linux, що робить її універсальним рішенням для корпоративних комп’ютерних мереж. Підтримка цілісності та стійкість системи до відмов базуються на динамічному виборі лідера, у випадку розділення кластера на частини чи відмови поточного керуючого вузла.

3 МУЛЬТИФРАКТАЛЬНИЙ АНАЛІЗ

В минулому розділі описано архітектуру розподіленої системи та компоненти, необхідні для організації корпоративної розподіленої мережі. В цьому розділі буде описано метод визначення мережевих атак в розподіленій мережі та детектори системи.

Мультифракталами називають складні фрактали, що зустрічаються, як правило, в природі. Фактично мультифрактальний підхід означає, що деякий досліджуваний об’єкт можна розділити на частини, що мають власні характеристики подібності, відмінні від інших. Мережевий трафік є самоподібним на деяких часових проміжках. Тому для його аналізу буде використано метод максимумів модулів вейвлет-перетворення, що дозволяє визначити особливості сигналу. Вейвлет аналіз полягає в побудові коефіцієнтів, що використовуються при розподіленні вихідного сигналу на базисні функції. В якості сигналу може виступати інтенсивність мережевого трафіку чи дані кореляції кінцевих IP-адрес. Вейвлет-перетворення дозволяє перетворити найбільш вагомі дані в сигнал, що відповідає вказаній амплітуді коливання та відкинути менш корисну інформацію з малою амплітудою, класифікуючи її як шум.

Для аналізу параметрів мультифрактального спектру існує наступний алгоритм:

1) декомпозиція вихідного сигналу $f(t)$ на коефіцієнти батьківським вейвлетом $\psi(t)$:

$$W_f(u,j) = \left(f(t), \psi_{u,s}(t)\right) = 2^{-j/2} \int_{-\infty}^{\infty} \frac{t-u}{2^j} dt; \quad (3.1)$$

де u – параметр масштабу, j – просторова координата чи момент часу;

2) в масиві коефіцієнтів знаходимо позиції локальних максимумів $\{u_p(j)\}_{p\in Z}$ та знаходимо їх абсолютне значення та формуємо масив максимумів

$$\left|W_f(u_p,j)\right|; \quad (3.2)$$

3) визначаємо функцію розбиття:

$$S(q,j) = \sum_p \left|W_f(u_p,j)\right|^q; \tag{3.3}$$

4) для кожного q ∈ R обчислюємо показник масштабу:

$$\tau(q,j) = lim \inf_{j\to 0} \frac{\ln S(q,j)}{\ln 2^j}; \tag{3.4}$$

5) обчислюємо мультифрактальний спектр за допомогою перетворення Лежандра:

$$f_L(\alpha) = \min_{q\in R}[q(\alpha + 1/2) - \tau(q)]; \tag{3.5}$$

6) для кожного проміжку j обчислюємо мультифрактальні розмірності порядку q:

$$D_{q,j} = \frac{1}{q-1}[q(\alpha(q,j) - f(\alpha(q),j)]\,. \tag{3.6}$$

Для ілюстрації та аналізу мережевого трафіку обрано його інтенсивність, тобто кількість відправлених та прийнятих пакетів за одиницю часу.

Принцип виявлення вторгнень наступний. Нехай X – часовий ряд звичайного трафіку, Y – часовий ряд шкідливого трафіку, Z – часовий ряд аномалій. Звідси Y=X+Z. Незалежно від наявності властивостей самоподібності в часовому ряді аномалій – Y все ще буде самоподібним процесом, якщо X стаціонарний самоподібний процес. Проте ступінь самоподібності може змінюватись. Нехай $s_X s_Y s_Z$ функції автокореляції для X, Y та Z відповідно. Тоді під час атаки акцентуємо увагу на $\|s_Y - s_X\|$, при цьому $s_Y = s_X + s_Z$. Для кожного $H \in (0.5, 1)$ існує лише одна функція автокореляції з самоподібністю. Тому розглядається $\|H_Y - H_X\|$, де H_Y та H_X – середнє значення показників Херста X та Y відповідно.

Коефіцієнт Херста вводиться для підвищення точності оцінки самоподібності системи. Недоліком підходу є необхідність перезапуску визначення порогу самоподібності трафіку для кожного масштабу. Тому сигнал про зміну самоподібності буде подано незалежно від того, чи існує він для іншого масштабу. Після визначення мережевої атаки трафік розбивається на декілька частин. Інтенсивність атаки можна визначити за допомогою аналізу показника Херста та швидкості його зміни, тобто різницю між показниками Херста до факту атаки та після.

Визначення точки зміни самоподібності трафіку базується на тому, що ентропія послідовності зі змінною граничною точкою самоподібності більша ніж ентропія послідовності з фіксованою точкою. На рисунку 3.1 зображено загальну схему визначення мережевої атаки.

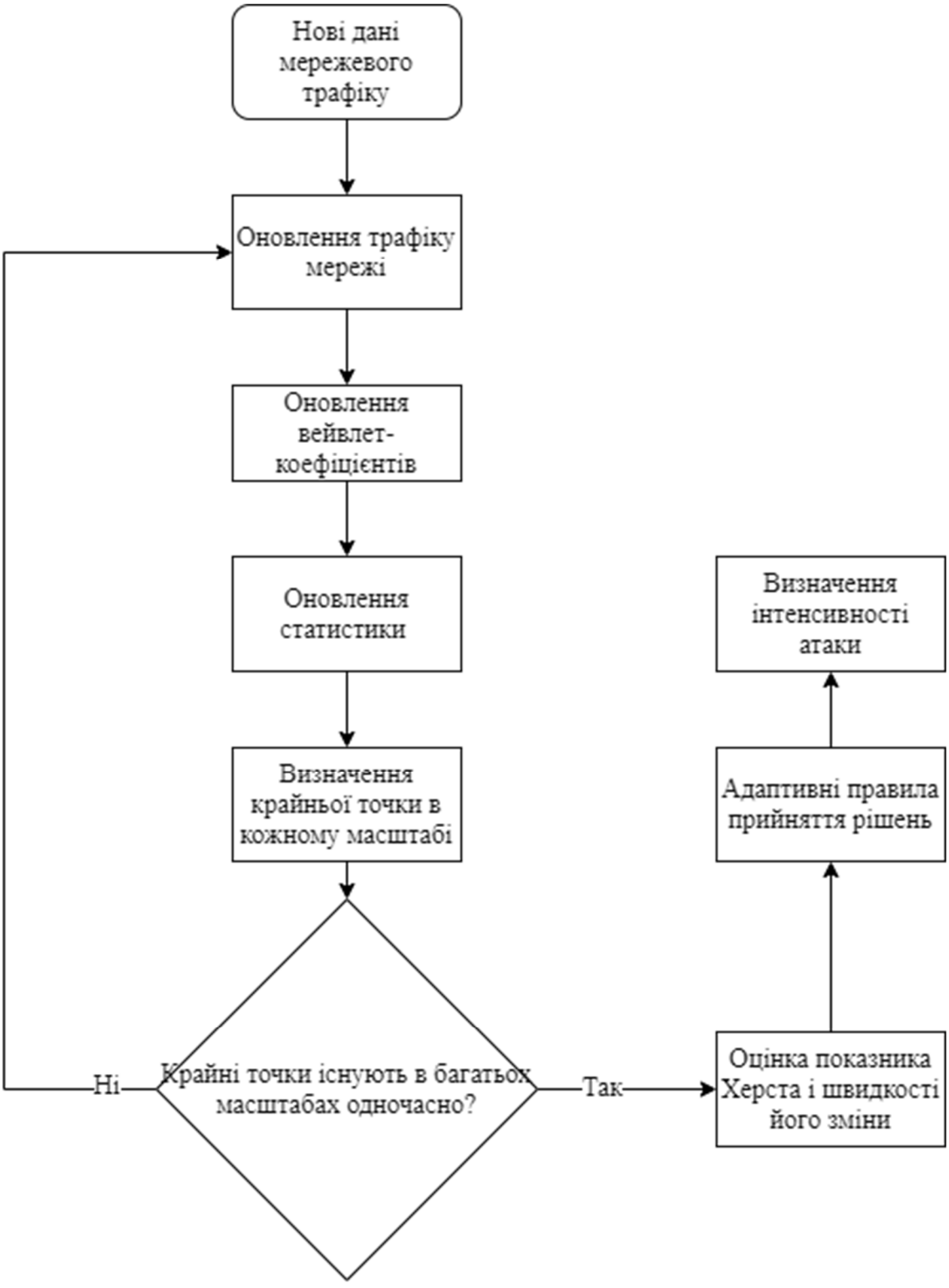


Рисунок 3.1 – Загальна схема визначення мережевої атаки

На рисунку 3.2 представлено схему оцінки безпеки мережевого трафіку. Алгоритм складається з п’яти етапів:

1) збір трафіку;

2) статистичний аналіз;

3) оцінка показника Херста;

4) визначення аномалій;

5) оцінка безпеки.

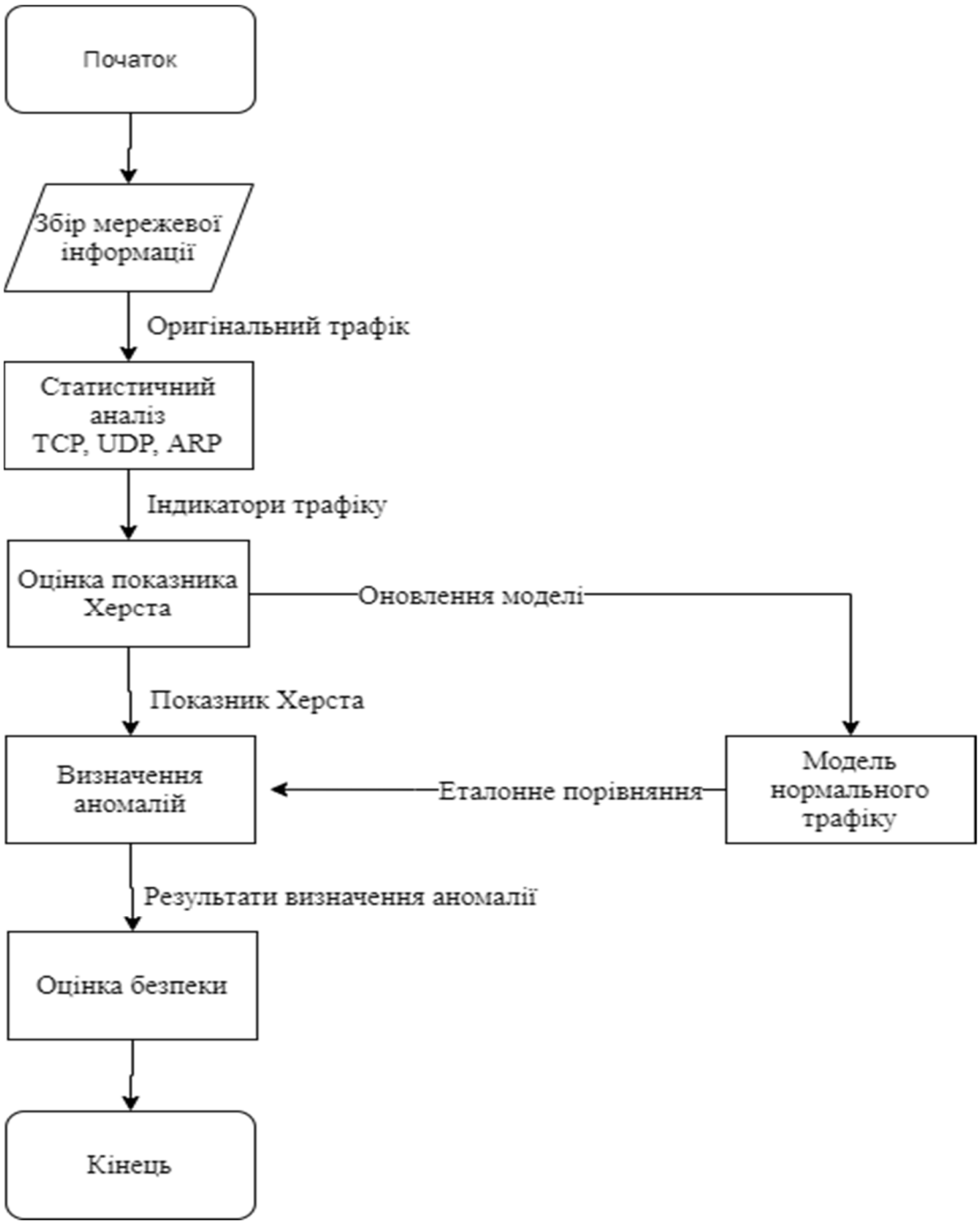


Рисунок 3.2 – Визначення оцінки безпеки мережевого трафіку

Для зменшення впливу на функціонування мережі трафік дублюється на сервер, що збирає мережеву інформацію кожного з підключених клієнтів.

Мультифрактальний аналіз є поведінковим методом, що характеризується наявністю помилкових спрацювань. Причиною цих помилок слугує складність повного і точного опису дій користувача. Крім того для більшості подібних систем в процесі роботи характерним є необхідність проведення попереднього налаштування, при якому система набуває досвіду для створення моделі нормальної поведінки користувача в мережі. Як правило цей процес займає досить багато часу. Вказані недоліки часто стають причиною того, що розробники відмовляються від поведінкових методів на користь систем, що мають чітке уявлення про небезпечну активність в мережі. Статистичний аналіз також можна назвати поведінковим, оскільки використовуються історичні дані про активність в мережі, порівнюються з новими даними про поведінку.

3.1 Детектори системи

Для реєстрації мережевих пакетів було обрано бібліотеку з відкритим вихідним кодом SharpPcap, на основі якої розроблено таке ПЗ, як Wireshark для Windows та Linux, оскільки ця бібліотека не потребує вдосконалення.

SharpPcap підтримує наступні мережеві протоколи: Ethernet, LinuxSLL, Ip (IPv4 and IPv6), Tcp, Udp, ARP, ICMPv4 и ICMPv6, IGMPv2, PPPoE, PTP, Link Layer Discovery Protocol (LLDP), Wake-On-LAN (WOL).

Основними можливостями цього пакету є визначення активних мережевих пристроїв, формування статистики, зчитування мережевої інформації з активних мережевих пристроїв та читання з файлів, фільтрація пакетів, збереження мережевої інформації в файл. Форматом даних є Pcap та pcap-ng, за умови використання npcap чи libpcap версії 1.1.0 та вище.

SharpPcap має багатошарову архітектуру. На верхньому рівні доступні наступні класи: CaptureDeviceList та ICaptureDevice. Перший дозволяє отримати всі наявні в системі мережеві пристрої. Другий надає інтерфейси для кожного сумісного пристрою. В архітектурі також доступні наступні простори імен: LibCap, WinCap та AirCap. Кожен простір імен має власні методи доступу до пристроїв.

Проте кожен пристрій, що повернуто за допомогою CaptureDeviceList мають тип або LibPcapLiveDevice, або WinPcapDevice, або AirPcapDevice. Це дозволяє отримати весь список мережевих інтерфейсів системи і відфільтрувати його залежно від потрібного типу.

Для початку перехоплення мережевих пакетів необхідно «відкрити» адаптер. Метод Open() виконує цю операцію та має два перевантаження аргументів: Open(DeviceMode mode) та Open(DeviceMode mode, int readTimeout). Аргумент DeviceMode дозволяє обрати режим роботи мережевого пристрою. Режимів є два: нормальний та змішаний. При роботі пристрою в нормальному режимі перехоплюються тільки ті пакети, що безпосередньо були призначені саме для цього адаптера, пакети, що призначені для інших вузлів мережі ігноруються. В змішаному режимі відбувається перехоплення всіх пакетів, незалежно від адресата. Змішаний режим як правило використовується за замовчуванням в більшості додатків для перехоплення мережевих пакетів. Слід зазначити, що сканування мережі в змішаному режимі можуть виявити інші користувачі мережі. Аргумент методу Open() read_timeout встановлює час затримки. Для прикладу метод GetNextPacket() буде завжди виконано після затримки, навіть якщо немає жодного пакету в мережі для перехоплення. Також затримка може бути використана для формування звітів за вказаний проміжок часу. Якщо значення цього аргументу рівне нулю – зчитування інформації не відбуватиметься до появи мережевого пакету. Значення -1 змушує адаптер завжди повертати поточні дані.

Як тільки мережевий адаптер «відкрито» - можна розпочинати перехоплення пакетів методами StartCapture() або Capture(int packetCount). Ці два методи дуже схожі. Їх різниця полягає в тому, що перший є неблокуючим методом запускає процес перехоплення в новому потоці, в той час як метод Capture(int packetCount) блокує пристрій до перехоплення вказаної кількості мережевих пакетів. Для зупинки процесу перехоплення викликаного методом StartCapture() слфд використовувати метод StopCapture(). Обидва методи вимагають обробника подій для роботи з мережевими пакетами. Виклик обробника подій відбувається з

надходженням з мережі нового пакета. В результаті повертається мережевий пакет з усіма заголовками протоколу.

SharpPcap також надає статистику адаптера, що доступна за дпопмогою виклику властивості ICaptureDevice.Statistics. Ця властивість підтримується всіма типами ICaptureDevice. WinPcap дозволяє зворотні виклики статистики, що може бути більш ефективним ніж повторне отримання статистики пристрою.

Зі зростанням кількості користувачів системи буде зростати кількість даних з отриманих мережевих пакетів. Тому виділяється інформація про тип пакету, а також наступні метрики: загальна кількість пакетів, число TCP, UDP та ARP пакетів за одиницю часу для конкретного вузла мережі – клієнта. Це знижує навантаження на канал зв’язку та безпосередньо на сервер. Оскільки система повинна порівнювати поведінку мережі з відомими даними було вирішено використовувати набір даних KDD Cup, що містить інформацію про нормальний трафік та відомі мережеві атаки, повний список яких наведено в таблиці 3.1.

Таблиця 3.1 – Мережеві атаки

Тип атаки	Атаки з набору даних
DOS	Back, Land, Neptune, Pod, Smurf, Teardrop
Probe	Ipsweep, Nmap, Portsweep, Satan
U2R	Buffer_overflow, Perl, Rootkit, Loadmodule
R2L	Ftp_write, Guess_password, Imap, Multihop, Phf, Spy, Warezclient, Warezmaster

DOS атаки спрямовані на відмову системи. Probe атаки дозволяють збір інформації про систему та її користувачів. U2R атаки спрямовані на незаконне отримання кореневого доступу до системи за наявності користувацького доступу. R2L атаки слугують для отримання неавторизованого віддаленого доступу до комп’ютера чи мережі в цілому.

Аналіз цього набору даних показав суттєві відмінності графіків нормального та аномального трафіку. Графік нормального трафіку зображено на рисунку 3.3. Для порівняння на рисунку 3.4 зображено один з різновидів DOS атаки.

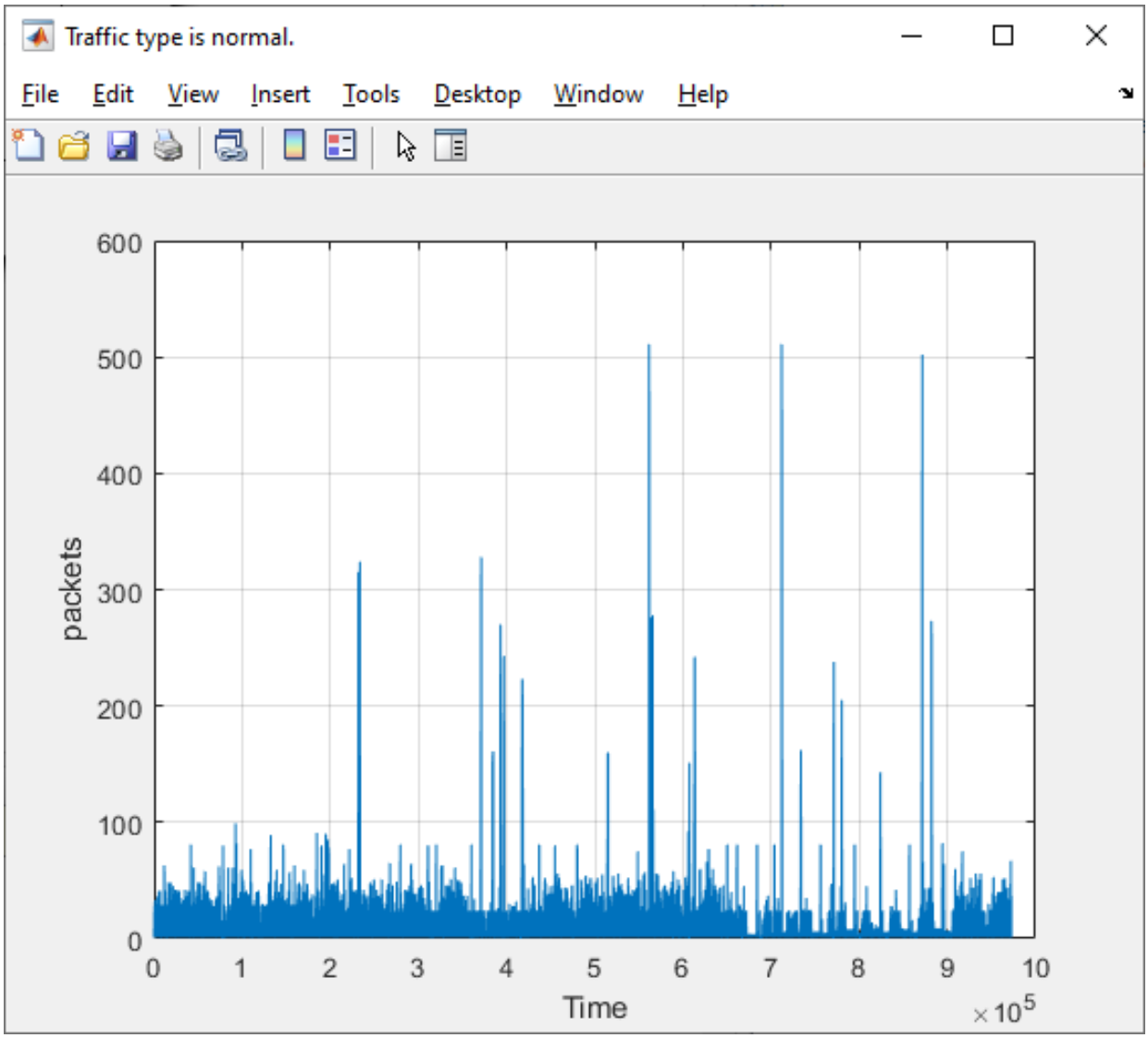


Рисунок 3.3 – Графік нормального трафіку

За нормальної мережевої активності можна передбачити подальшу мережеву активність. На графіку можна спостерігати підвищення та зниження активності з плином часу, що в більшості випадків мають приблизно ту ж саму амплітуду та періодичність. Використовуючи ці дані не важко змоделювати стохастичні варіації активності додатків, що використовують підключення до мережі. Передбачуваність мережевого трафіку дає можливість адаптувати мережу для запобігання перевантажень. Тому розробники оптимізують мережеві додатки для

використання каналів мережі в повній мірі, використовуючи спеціалізовані протоколи передачі даних.

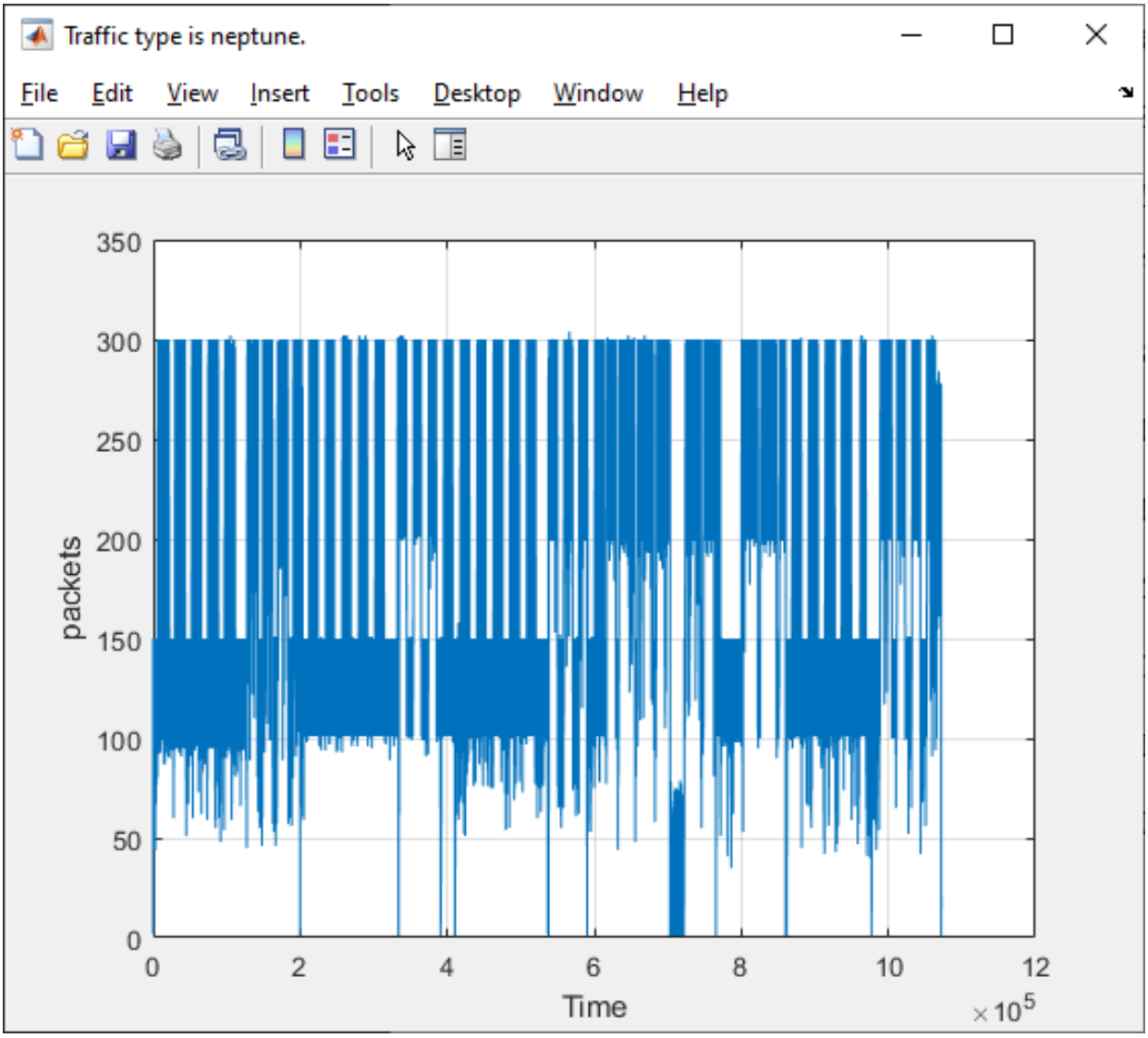


Рисунок 3.4 – Графік різновиду DOS атаки

3.2 Аналіз даних

Як показують дослідження мережевий трафік має ознаки самоподібності, тому немає необхідності в обробці всіх отриманих даних. Натомість потрібно аналізувати лише частину, щоб прискорити роботу програми та вчасно прийняти міри безпеки в разі виявлення вторгнення. Для перетворення даних, зібраних користувацькою компонентою, в зручний для взаємодії формат необхідно обрати масштаб та обрахувати кількість мережевих пакетів на кожному часовому проміжку. Далі з використанням розробленого методу побудувати мультифрактальний спектр та порівняти спочатку з нормальним трафіком. У

випадку виявлення суттєвих розбіжностей проаналізувати схожість з відомими атаками для визначення типу мережевої атаки. Додатковим критерієм для підвищення точності роботи методу взято адресу, з якої надходять мережеві пакети. Для прикладу якщо з однієї адреси надходить велика кількість запитів, довжина яких не перевищує 50 байтів – найімовірніше, що це сканування портів. І навпаки, якщо з адреси надходять великі об’єми даних приміром протягом останніх п’яти хвилин – це є ознакою DOS атаки. Надходження подібної кількості даних з різних джерел класифікується як DDOS атака.

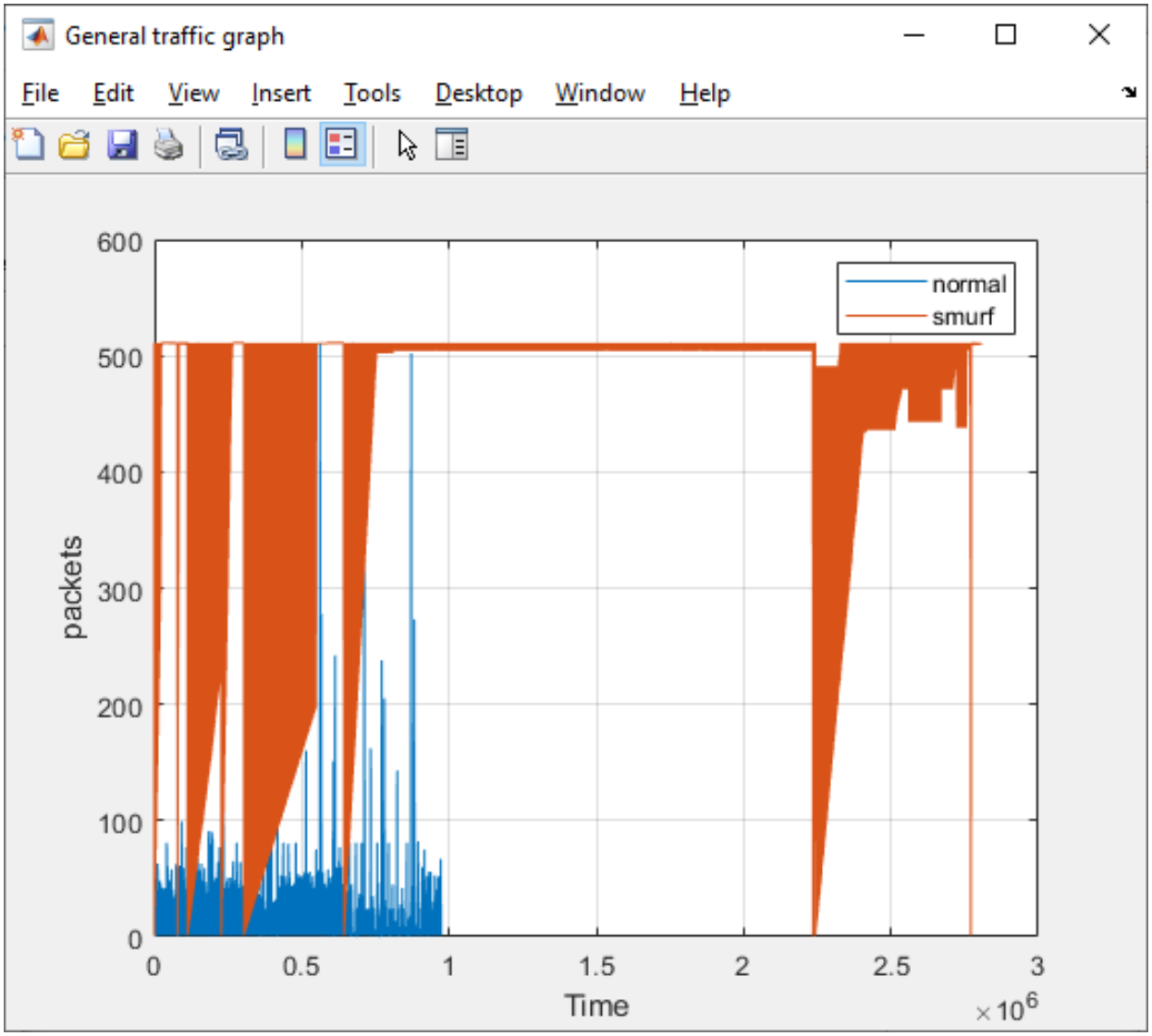


Рисунок 3.5 – Порівняння двох фрагментів мережевої активності

Оскільки об’єм історичного трафіку може відрізнятись від об’єму отриманих даних – порівняння значень може значно знизити продуктивність системи в цілому.

На рисунку 3.5 зображено графік нормального трафіку та DOS атаки. Кількість пакетів кардинально відрізняється. Візуально ми можемо бачити різницю, але машина аналізує значення координат, що може бути довгим та ресурсоємним процесом. Саме тому необхідно мінімізувати витрати системних ресурсів та зменшити час обробки. Для цього в системі і застосовується мультифрактальний аналіз.

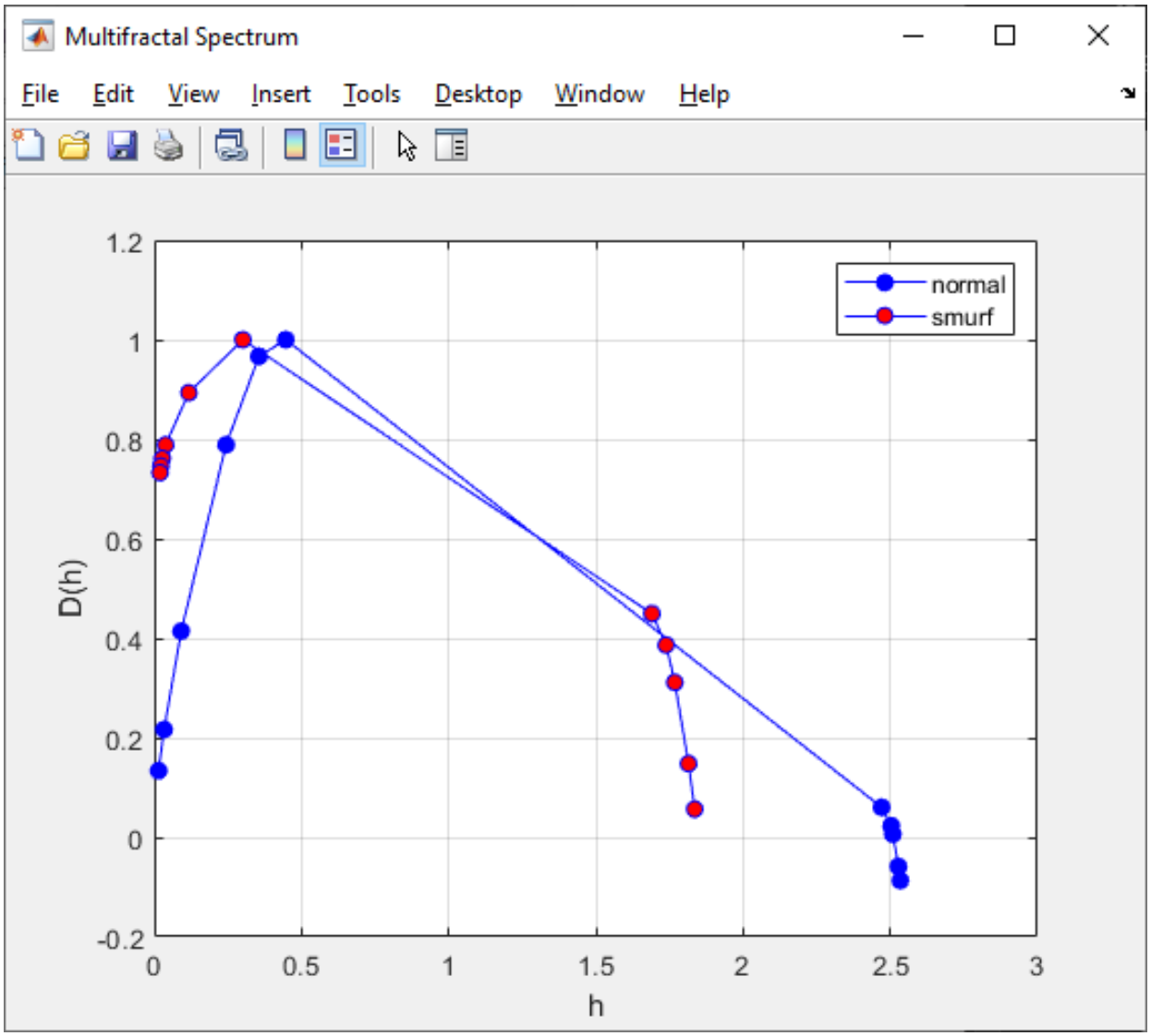


Рисунок 3.6 – Порівняння мультифрактального спектру

На рисунку 3.6 зображено порівняння двох мультифрактальних спектрів: нормального трафіку та активності під час мережевої атаки. Всього графік має 11 точок, які і враховуються при визначенні небезпечної мережевої активності. Якщо всі точки мають відхилення від значень нормального трафіку менше 15%, то

вважаємо значення координати нормальним У випадку, якщо суттєві відхилення присутні не більше ніж в трьох точках – вважаємо трафік безпечним. Якщо більше трьох точок мають відхилення вище 15% – має місце мережева атака. Відсоток відхилення враховує зміну в активності користувача для мінімізації випадкових спрацювань алгоритму, оскільки це і є головним недоліком поведінкового аналізу.

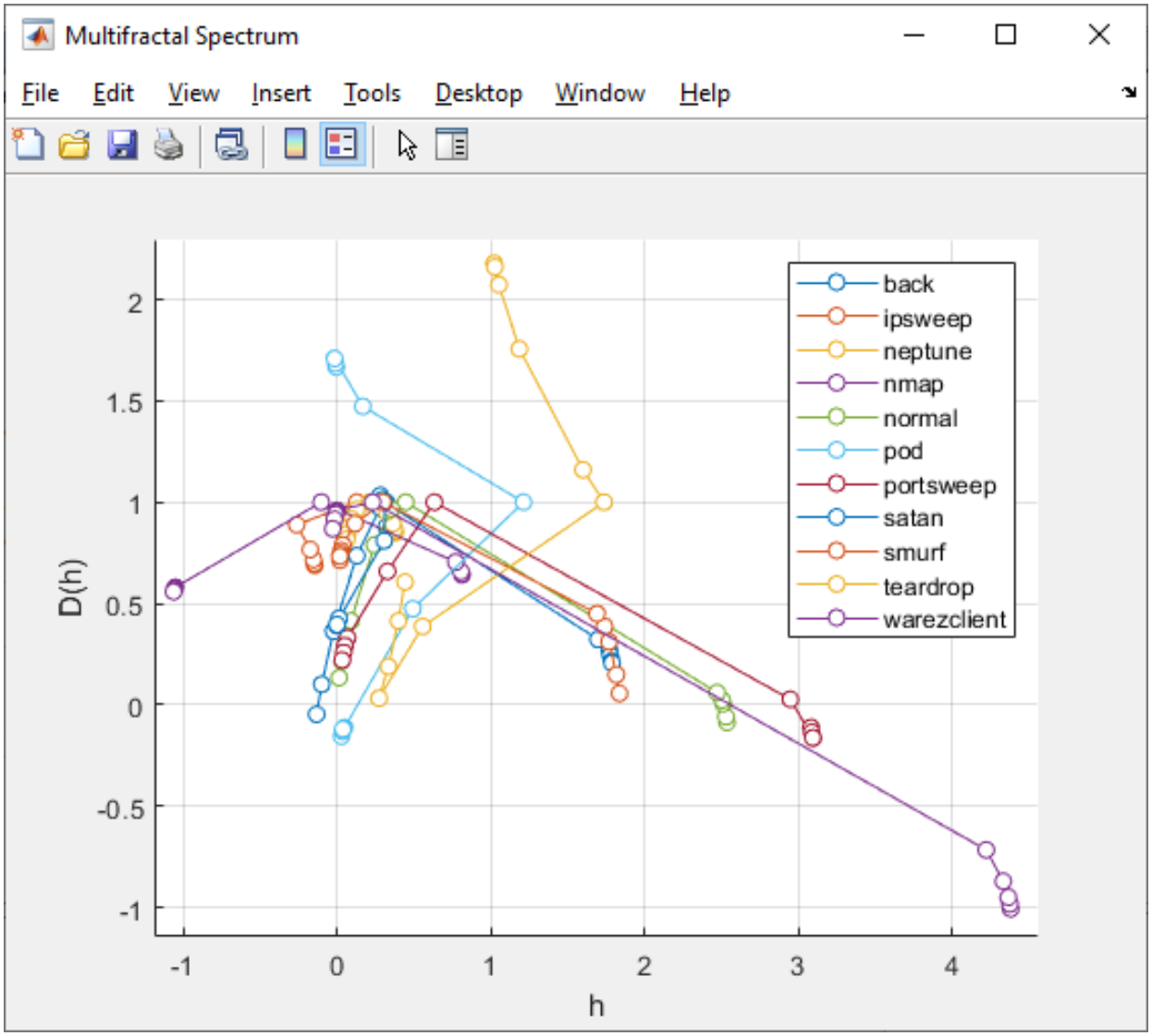


Рисунок 3.6 – Зведений графік мультифрактального спектру

На рисунку 3.6 представлено мультифрактальні спектри всіх підтримуваних типів трафіку, що представлені в наборі даних KDD Cup. Графік наочно демонструє різницю між різними мережевими атаками та нормальним трафіком. Для формування зведеного графіку було використано Matlab R2019a та наступні фрагменти коду:

```
labels=unique(kddcup.label);
figure('Name', 'Multifractal Spectrum ','NumberTitle','off');
for row=1:rows
 packets=kddcup(kddcup.label== labels(row,:) , :);
 try
  [dh1,h1,cp1,tauq1] = dwtleader(packets.p_count);
  hp = plot(h1,dh1,'-o');
 catch
  exception=(sprintf('Too short signal of %s', labels(row,:))); warning(exception);
 end end
```

Повний код програми доступний в додатку А

Використана функція dwtleader(x) повертає спектр сингулярності $D(h)$ та показники Холдера h для $x \in R$, показники масштабування «ср» та «tauq» - показники масштабування для лінійно розташованих моментів від -5 до 5. Для найтонших масштабів лідери вейвлетів не визначено.

3.3 Висновки до третього розділу

В третьому розділі проаналізовано властивості мережевих пакетів та визначено набір даних, необхідних для аналізу мережевих пакетів. Визначено допустимі відхилення значень мультифрактального спектру, при якому мережевий трафік вважається безпечним. Проведено дослідження різних типів мережевих атак та їх основні особливості поведінки, що вирізняють їх серед інших мережевих атак чи безпечного трафіку. Визначено алгоритм роботи мультифрактального аналізу для подальшого застосування в розподілених системах.

4 РЕАЛІЗАЦІЯ ТА ВИПРОБУВАННЯ РОЗПОДІЛЕНОЇ СИСТЕМИ ВИЗНАЧЕННЯ МЕРЕЖЕВИХ АТАК НА ОСНОВІ МУЛЬТИФРАКТАЛЬНОГО АНАЛІЗУ

Попередній розділ описує суть використаного методу та його застосування в розподіленій системі. Алгоритм показує високу швидкість та точність обчислень. Архітектуру системи було описано в другому розділі, звідси маємо систему, компоненти якої пов’язані між собою та залежать лише від доступності сервера.

4.1 Реалізація розподіленої системи

Для розробки було використано Microsoft Visual Studio Community 2019. В середовищі розробки створено рішення під назвою «Distributed.System.ServerAPI». Для зручності всі компоненти, що стосуються серверної частини використовують назву рішення як спільний простір імен. Відповідно маємо наступні модулі: «WebAPI», «Common», «Database», «EntityFramework», «Repository» та «Services». Клієнтський додаток має власний простір імен «WorkerCaptureService», оскільки він не має такого тісного зв’язку із серверними компонентами. Клієнтський додаток виступає в ролі фонової служби та не має графічного інтерфейсу чи ключів запуску для повної автономності. Тому використовуючи методи бібліотеки SharpPcap додаток самостійно знаходить активний мережевий інтерфейс та відкриває підключення до нього. Нижче представлено фрагмент методу для доступу до мережевого інтерфейсу. Повний код можна переглянути в додатку А.

```
var nics = NetworkInterface.GetAllNetworkInterfaces();
foreach (var device in LibPcapLiveDeviceList.Instance)
{
  var friendlyName = device.Interface.FriendlyName ?? string.Empty;
  if (friendlyName.ToLower().Contains("loopback") || friendlyName == "any")
  {
    continue;
  }
  var nic = nics.FirstOrDefault(ni => ni.Name == friendlyName);
```

```
if (nic?.OperationalStatus != OperationalStatus.Up)
{
  continue;
```

В цьому фрагменті коду також використано оператор try-catch, оскільки допоміжні драйверні бібліотеки libpcap для Linux та Npcap для Windows можуть не підтримувати деякі мережеві інтерфейси. Служба також повинна запускатись від імені адміністратора для повного доступу до мережевої інформації. Реалізація роботи додатку в якості служби виконана з використанням шаблону Worker Service. Тому маємо наступний метод для асинхронного виклику методу та безперервної роботи програми:

```
protected override async Task ExecuteAsync(CancellationToken stoppingToken)
{
  CaptureTask captureTask = new CaptureTask();
  captureTask.CaptureWorker();
  while (!stoppingToken.IsCancellationRequested)
  {
    await Task.Delay(1000, stoppingToken);
  }
}
```

Клієнтський додаток також збирає дані про користувача: його ім’я, ip адресу та час останнього підключення до системи. Для цього використано стандартну бібліотеку System.Net, що надає інформацію про мережеве ім’я користувача та дозволяє отримати його адресу в глобальній мережі. Нижче представлено метод отримання даних та їх відправку до бази даних:

```
public void WriteUserInfo()
  {
   String strHostName = new String("");
   strHostName = Dns.GetHostName();
   IPHostEntry ipEntry = Dns.GetHostEntry(strHostName);
   var db = new LocalAppDbContext();
   using (db)
   {
    var newClient = new ClientEntity()
```

```
        {
          IpAddress = GetPublicIpAddress(),
          Name = strHostName,
          LastConnection = DateTimeOffset.Now
        };
        db.Clients.Add(newClient);
        db.SaveChanges();
        currentClientId += newClient.Id;
      }
    }
```

Метод GetPublicIpAddress() реалізовано окремо та представлено в додатку А. Отримання IP адреси відбувається за допомогою HTTP запиту. Результатом виконання є рядок із адресою в глобальній мережі.

Коли система реєструє новий мережевий пакет спрацьовує метод device_OnPacketArrival(), що використовує об’єкт currentClientId класу UserInfo для закріплення за цим клієнтом мережевих пакетів в базі даних. Для збору мережевої інформації використовується метод Packet.ParsePacket з бібліотеки PacketDotNet, що є складовою SharpPcap. Вихідний код методу представлено в додатку А

Методи реєстрації пакетів та отримання інформації про клієнта викликаються методом CaptureWorker(). Принцип його роботи наступний:

1) для поточного сеансу створюється новий екземпляр класу UseInfo;

2) викликається метод запису інформації в базу даних з екземпляру класу;

3) викликається метод визначення активних мережевих інтерфейсів;

4) виводиться в налагоджувальну консоль назва підключеного мережевого адаптера;

5) викликається обробник події отримання мережевим інтерфейсом пакету;

6) встановлюються змінні для «відкриття» мережевого адаптеру в режимі «Promiscuous» та вказаними затримками читання даних;

7) викликається метод, що стартує захоплення мережевих пакетів.

Як було описано вище - для роботи з пристроєм використовується режим «Promiscuous», оскільки він повідомляє операційну систему про потребу

перехоплення всіх мережевих пакетів, включаючи ті, що не призначені для мережевого адаптера.

Інформація, отримана клієнтом відправляється на сервер, що зберігає та обробляє дані кожного клієнта. Методи WriteUserInfo() та device_OnPacketArrival() записують дані в таблиці «Client» та «CapturedPackets» відповідно. Для створення цих таблиць було використано Entity Framework:

```
public class ClientEntity
  {
    public long Id { get; set; }
    public string IpAddress { get; set; }
    public string Name { get; set; }
    public DateTimeOffset LastConnection { get; set; }
    public List<CaptureEntity> Packets { get; set; }
  }
public class CaptureEntity
  {
    public long Id { get; set; }
    public DateTimeOffset DateTime { get; set; }
    public int DateTimeMiliseconds { get; set; }
    public string DestinationHardwareAddress { get; set; }
    public string SourceHardwareAddress { get; set; }
    public string PacketType { get; set; }
    public long ClientFK { get; set; }
    public ClientEntity ClientEntity { get; set; }
  }
```

Головні та внутрішні ключі, реалізовані в коді, додатково налаштовано за допомогою FluentAPI та повідомляють програму про спосіб роботи з таблицями та відношення між ними:

```
public class FKConfig : IEntityTypeConfiguration<CaptureEntity>
  {
    public void Configure(EntityTypeBuilder<CaptureEntity> modelBuilder)
    {
      modelBuilder.HasOne(y => y.ClientEntity)
          .WithMany(x => x.Packets);
```

```
    }
  }
protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
      base.OnModelCreating(modelBuilder);
      modelBuilder.ApplyConfiguration(new FKConfig());
      modelBuilder.Entity<ClientEntity>()
          .ToTable("Client")
          .HasKey(e => e.Id);
      modelBuilder.Entity<CaptureEntity>()
          .ToTable("CapturedPackets")
          .HasKey(e => e.Id);
    }
```

Звідси маємо унікальний ідентифікатор для кожного записаного мережевого пакету та посилання на унікального користувача, що надіслав цей трафік на сервер. Далі трафік кожного клієнта аналізується на сервері. Для доступу до даних використовується репозиторій. В ньому реалізовано наступні методи: Add(), AddRange(), Update(), Remove(), RemoveRange() та GetById(). Перераховані вище методи використовуються в загальному випадку та використовують головний ключ. Для здійснення більш гнучкої вибірки даних реалізовано окремі репозиторії для клієнтів та отриманих пакетів. Пошук серед клієнтів здійснюють асинхронні методи FindClientById(), FindClientByIp() та FindClientByName(). Вибірку мережевих пакетів надають методи FindByPacketType() та FindByTimeArrived(). Всі вище вказані методи для пошуку в базі даних використовують метод FirstOrDefaultAsync(), що повертає перший елемент, який задовольняє умову, або стандартне значення, якщо нічого не було знайдено.

4.2 Дослідження розробленого методу

Для дослідження методу було використано пакет прикладних програм аналізу та програмування Matlab. Візуалізація мультифрактального спектру відбувається вбудованими засобами програмного забезпечення. Для цього було реалізовано програмний код, що міститься в додатку А. Алгоритм роботи наступний:

1) завантаження даних;

2) вибірка унікальних найменувань мережевих атак з блоку даних;

3) формування загального графіку за кожним типом мережевого трафіку;

4) обчислення автокореляційної послідовносі для кожного типу мережевої активності;

5) формування графіку автокореляційної послідовності для кожного типу мережевої активності;

6) обчислення мультифрактального спектру за кожним типом активності;

7) формування графіку на основі даних, отриманих на попередньому кроці

8) вивід попередження, якщо неможливо сформувати графік мультифрактального спектру через недостатню кількість даних.

Програма використовує набір даних KDD Cup, який конвертовано в таблицю засобами Matlab. Для виведення графіків трафіку використано значення кількості пакетів. В даному наборі даних кожен рядок можна розглядати як статистику активності за одиницю часу. Перед створенням графіків програма фільтрує статистичні дані за кожним наявним типом мережевих атак. Далі формуються графіки мережевого трафіку для кожного виду мережевих атак та окремо графік нормального трафіку.

Результатом виконання представленого вище коду є графік мережевого трафіку, графік автокореляційної послідовності та мультифрактальний спектр На рисунках 4.1, 4.2 та 4.3 зображено графіки DOS атаки відповідно.

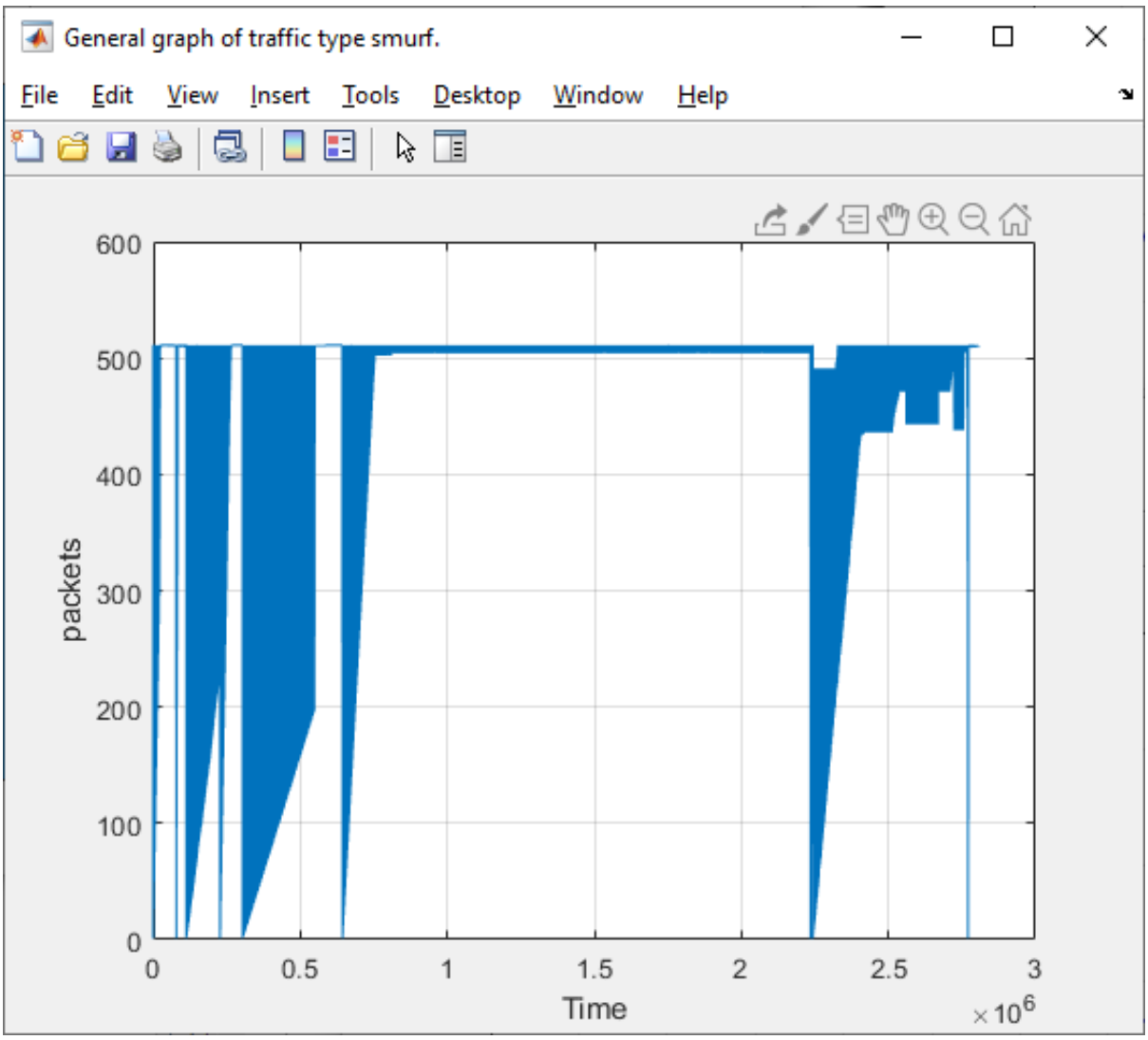


Рисунок 4.1 – Графік активності мережевої атаки smurf

Мережеву активність з такою природою поведінки нескладно розпізнати, оскільки відбувається перевантаження великою кількістю даних за короткий проміжок часу. Така активність також передбачувана, оскільки має чітку амплітуду та частоту, залежно від типу мережевої атаки.

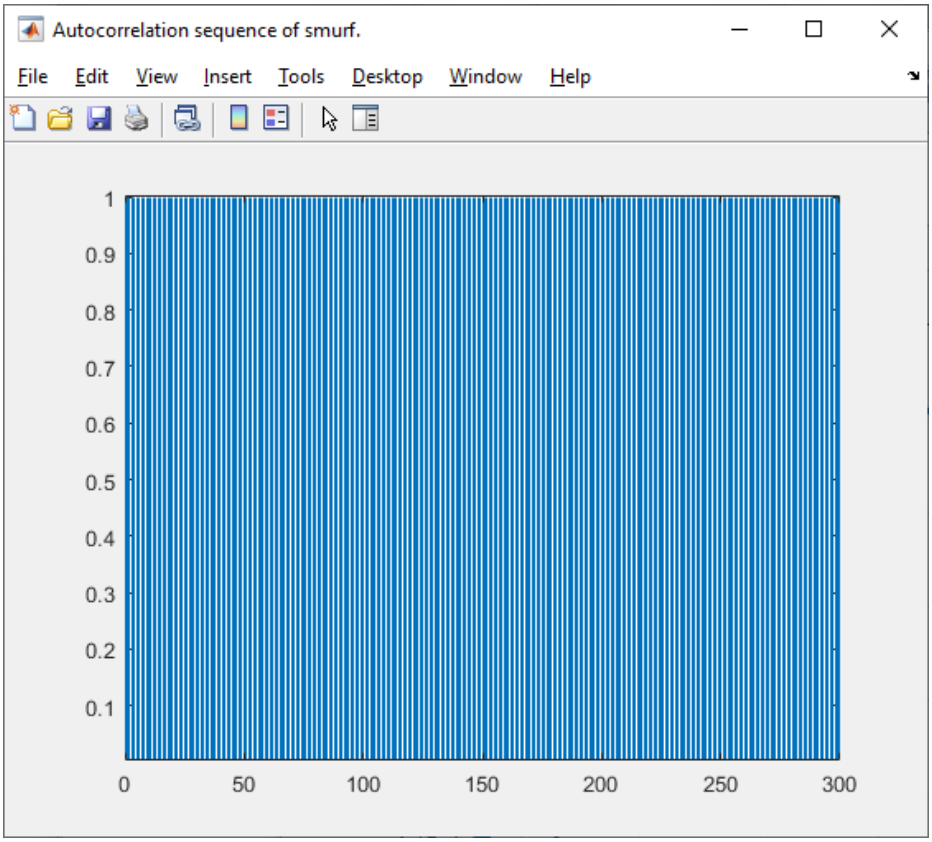


Рисунок 4.2 – Автокореляційна послідовність мережевої атаки smurf

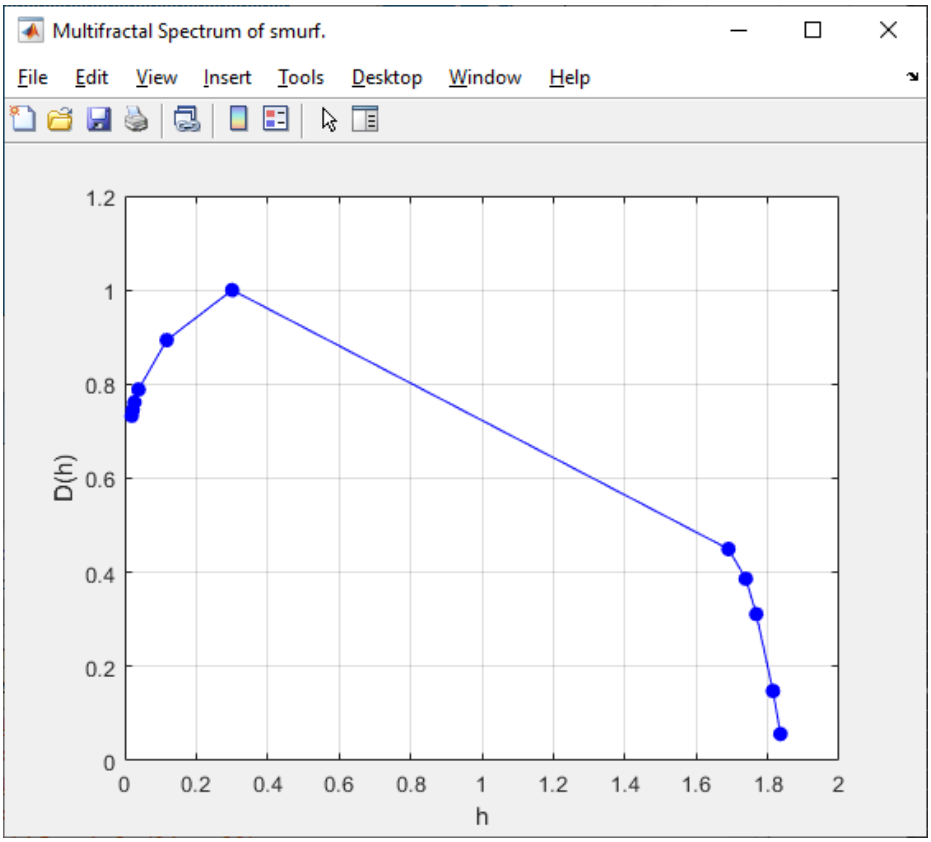


Рисунок 4.3 – Мультифрактальний спектр мережевої атаки smurf

Наступним кроком є порівняння історичних даних з новим блоком мережевих пакетів. Однак метод має обмеження: при недостатній кількості значень в автокореляційній послідовності неможливо сформувати мультифрактальний спектр. На це впливає кількість випадків визначеного типу атаки на часовій шкалі. Яскравим прикладом з набору даних є активність мережевої атаки rootkit, що зображено на рисунку 4.3 та її автокореляційна послідовність, – рисунок 4.4.

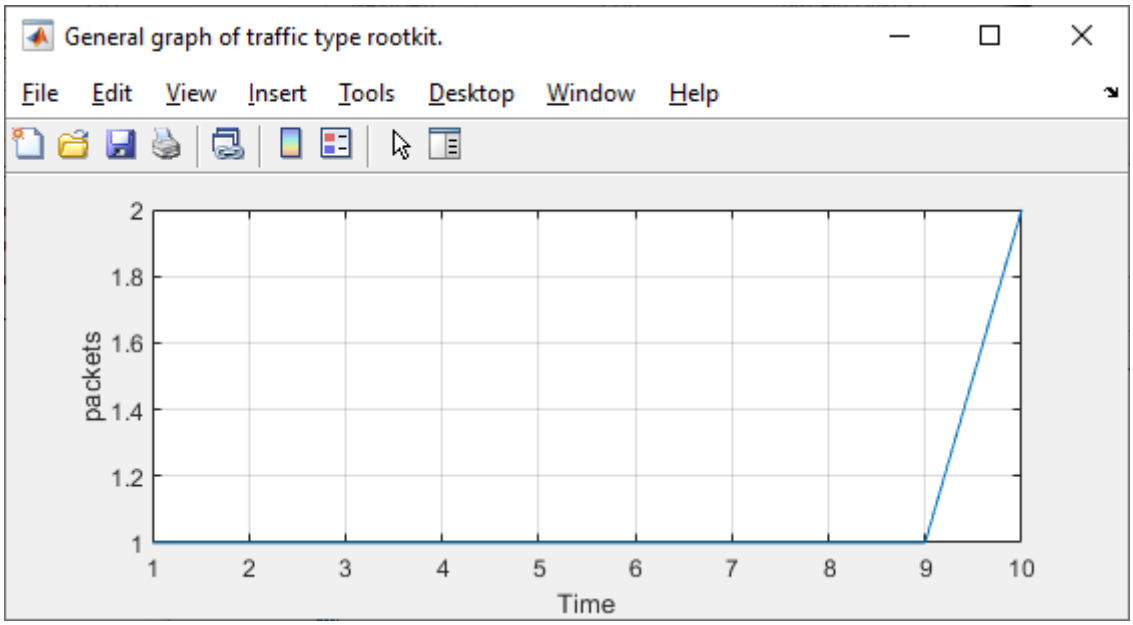


Рисунок 4.3 – Графік активності мережевої атаки rootkit

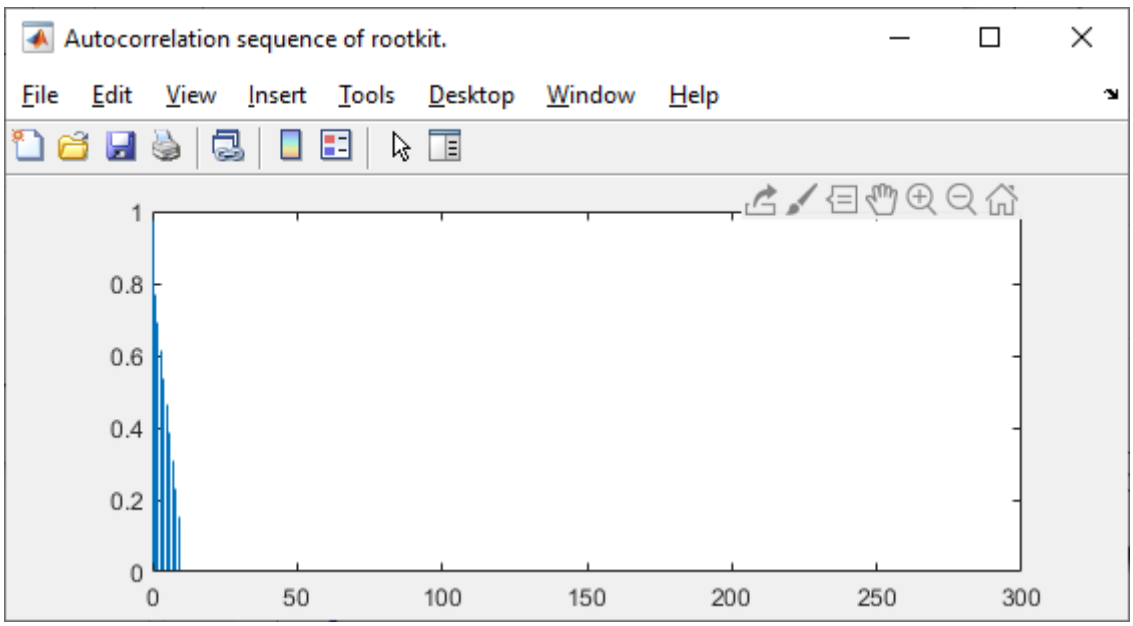


Рисунок 4.4 – Автокореляційна послідовність мережевої атаки rootkit

Тому для формування мультифрактального спектру необхідно більше даних, що може спричинити ігнорування системою загрози взагалі.

Метод дозволяє не просто класифікувати атаку, а й розрізняти атаки в межах типу. На рисунку 4.5 зображено графіки двох видів DOS атак.

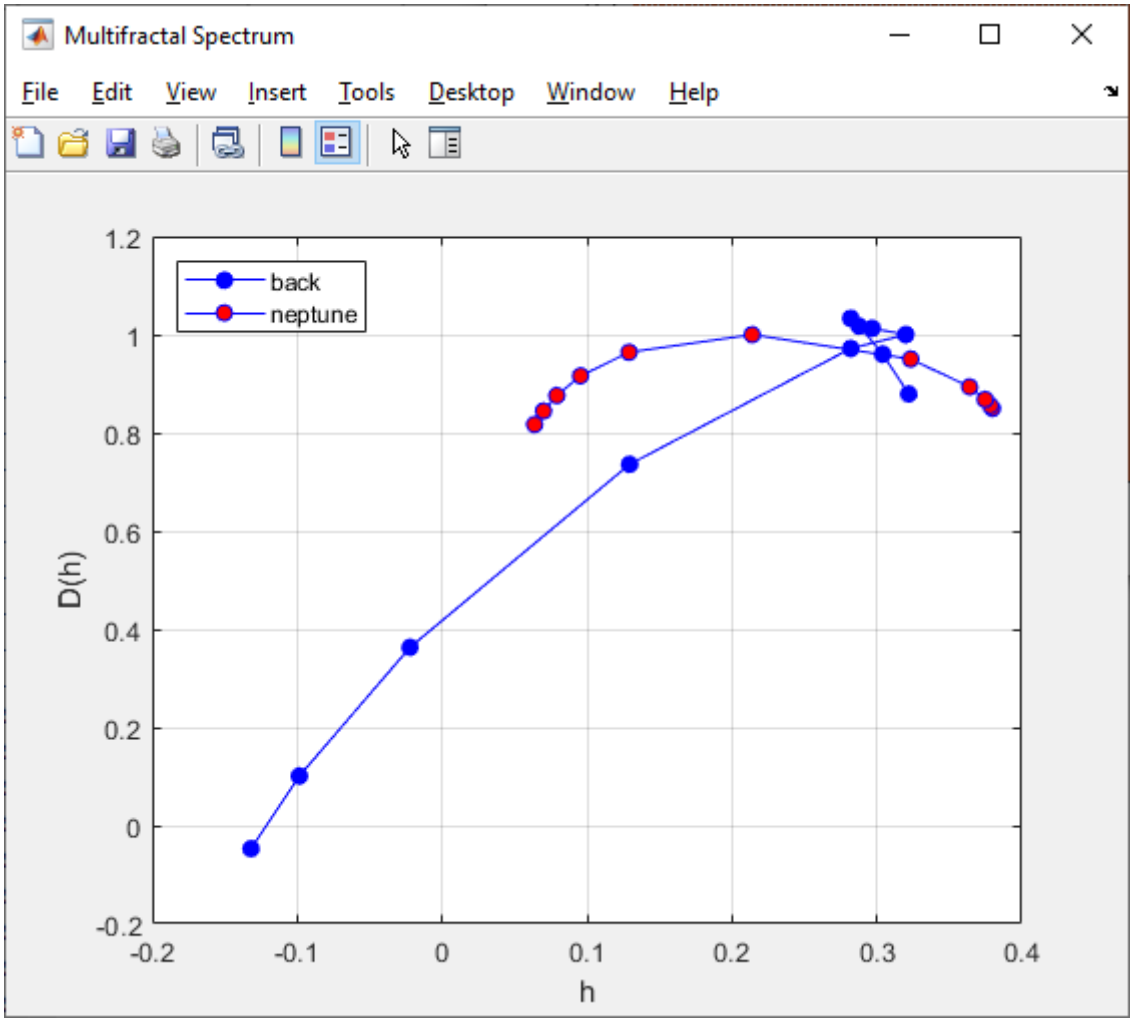


Рисунок 4.5 – Мультифрактальний спектр двох різновидів DOS атак

В коді програми присутній таймер, що показує швидкість роботи алгоритму. Загальний час обробки даних складає близько 8 секунд, тоді як об’єм даних складає майже 4.9 мільйони рядків, в форматі текстового документу ці дані займають більше 700 мегабайт. Це свідчить про високу швидкість роботи алгоритму та ефективність використання системних ресурсів.

4.3 Дослідження розподіленої системи

З метою перевірки швидкості роботи розподіленої системи та ефективності використання системних ресурсів проводяться експериментальні дослідження. Програмний продукт вимагає додаткового встановлення найновішої версії .NET Framework. Рекомендована швидкість мережі становить 1 Мбіт/с та вище. Для серверної компоненти необхідне велике сховище даних понад 1ТБ для збереження історії мережевої активності підключених вузлів. Для підключення компонентів рекомендується використання окремого мережевого інтерфейсу для запобігання перевантаження основного каналу зв'язку. Підтримувані операційні системи: Windows та Linux. Додатковим програмним забезпеченням для функціонування клієнтських компонентів є NPcap для ОС Windows та libpcap для Linux. Також є можливість розгортання системи в Docker контейнерах незалежно від їх типу. Для створення контейнера використовується автоматично згенерований DockerFile. Поширення контейнера відбувається безпосередньо інструментами Docker, а саме Docker Swarm, що надає централізоване керування всіма підключеними контейнерами незалежно від типу. Ініціалізація Swarm менеджера на прикладі Linux контейнерів зображена на рисунку 4.7.

```
cloud_user        :~$ docker swarm init --advertise-addr 1  .3 .9 .1
Swarm initialized: current node (bu56lkahvsar8mwna5cwu3gu1) is now a mana
ger.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0yb4qnxvyjszs2ke7tukm6nx8vj9uvuez9
2gtlp113twv5aulk-bvf4h5h54359t4hyw7vxlwshb 1  .3 .9 .1  :2377

To add a manager to this swarm, run 'docker swarm join-token manager' and
 follow the instructions.
```

Рисунок 4.7 – Ініціалізація Swarm менеджера

Після ініціалізації менеджера контейнер, з якого було виконано команду стає менеджером і водночас залишається вузлом кластеру. Фрагмент виводу команди docker info зображено на рисунку 4.8.

Рисунок 4.8 – Інформація про кластер

Додавання вузлів до кластеру відбувається за допомогою токену, що надає менеджер кластеру. Можливе додавання як звичайних вузів так і додаткових керуючих. Для генерування токену використовується команда docker swarm join-token worker (або manager). Результатом виконання команди є автоматично сформована команда підключення додаткових вузлів із вказаним токеном, IP-адресою та портом для підключення. Після підключення звичайний вузол також має доступ до інформації про кластер.

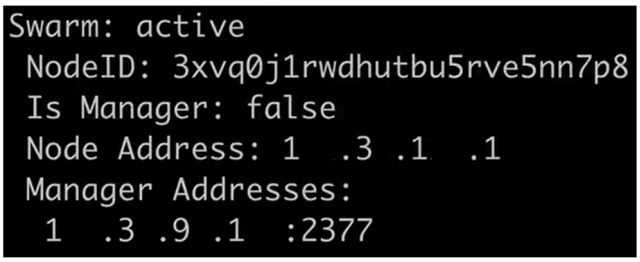


Рисунок 4.9 – Інформація про кластер, надана звичайним вузлам

Як можна побачити на рисунку 4.9 – інформація про кластер обмежена та містить лише необхідні для співпраці з керуючим вузлом дані. В свою чергу менеджеру доступні всі дані про кластер та функції керування вузлами. Однією із функцій менеджера є створення сервісів в межах кластеру та їх поширення між вузлами. За умови наявності готового контейнеру, що готовий до роботи, поширення відбувається за командою docker service create -env

NODE_HOSTNAME=”{{.Node.Hostname}}” --replicas 3 target_container. Ця команда автоматично визначає змінні середовища всіх доступних контейнерів та проводить встановлення сервісу на вказану кількість вузлів. Керування сервісами також відбувається з керуючого вузла. Для прикладу за потреби оновлення конфігурації чи версії сервісу на контейнерах використовується команда docker service update, аргументами якої може бути нова кількість задіяних вузлів та безпосередньо назва контейнера для копіювання. Альтернативним способом поширення є глобальний спосіб створення сервісу, що встановлюється аргументом команди замість вказання кількості контейнерів та автоматично запускає завдання на всіх доступних вузлах. Слід зазначити, що незалежно від типу контейнерів необхідно в ручному режимі встановити бібліотеки відповідної платформи для запуску додатку в кластері.

Виконання клієнтського програмного забезпечення не потребує багато ресурсів та невидиме для користувача, оскільки відбувається в якості фонової служби. Для налагодження програми було використано консоль, де відбувається виведення активності вузла – рисунок 4.10.

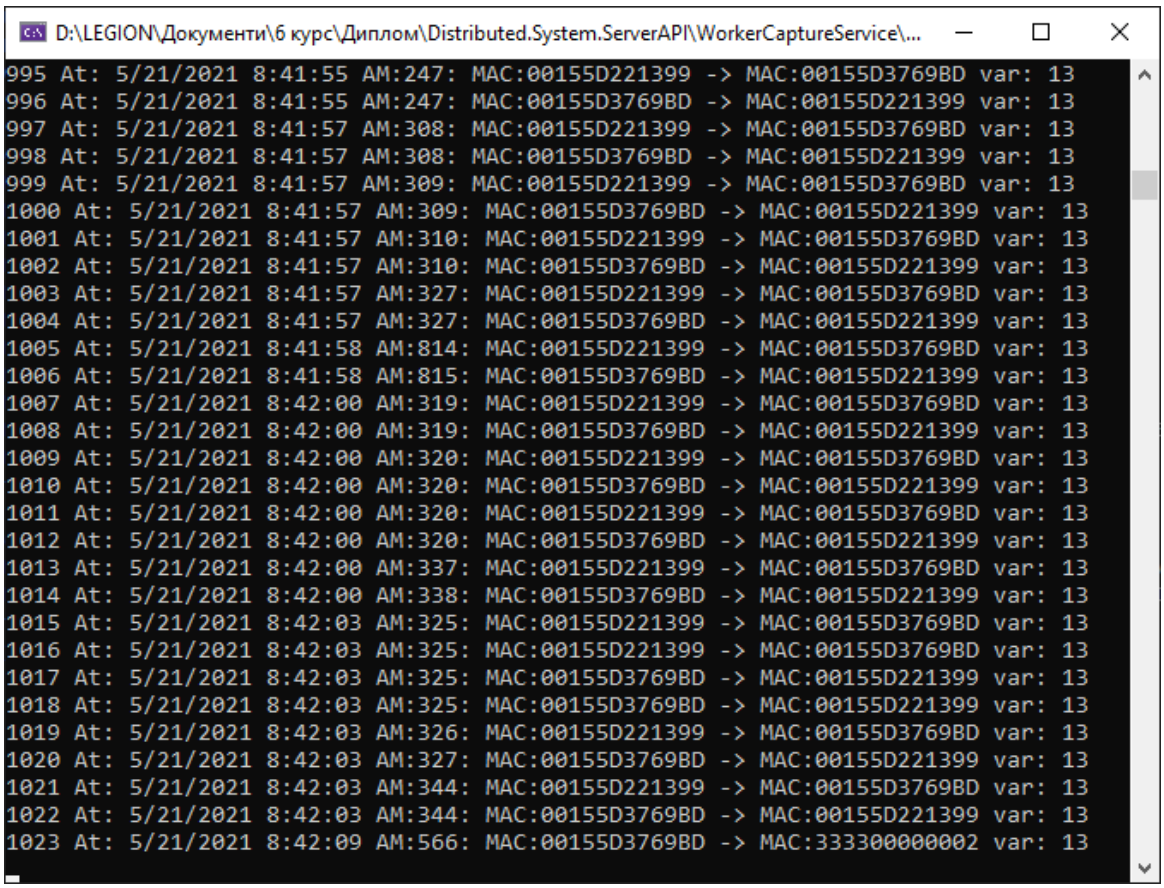


Рисунок 4.10 – Консоль клієнтського додатку

Програма надсилає дані мережевих пакетів на сервер бази даних для подальшої обробки та зберігання історії трафіку.

Id	DateTime	DateTimeMilliseconds	DestinationHardwareAddress	SourceHardwareAddress	PacketType	ClientFK	ClientEntityId
2084	2021-05-21 08:41:57.3089430 +00:00	308	00155D221399	00155D3769BD	IPv4	0	13
2085	2021-05-21 08:41:57.3094060 +00:00	309	00155D3769BD	00155D221399	IPv4	0	13
2086	2021-05-21 08:41:57.3094830 +00:00	309	00155D221399	00155D3769BD	IPv4	0	13
2087	2021-05-21 08:41:57.3102680 +00:00	310	00155D3769BD	00155D221399	IPv4	0	13
2088	2021-05-21 08:41:57.3103830 +00:00	310	00155D221399	00155D3769BD	IPv4	0	13
2089	2021-05-21 08:41:57.3274730 +00:00	327	00155D3769BD	00155D221399	IPv4	0	13
2090	2021-05-21 08:41:57.3276130 +00:00	327	00155D221399	00155D3769BD	IPv4	0	13
2091	2021-05-21 08:41:58.8146470 +00:00	814	00155D3769BD	00155D221399	Arp	0	13
2092	2021-05-21 08:41:58.8151470 +00:00	815	00155D221399	00155D3769BD	Arp	0	13
2093	2021-05-21 08:42:00.3190200 +00:00	319	00155D3769BD	00155D221399	IPv4	0	13
2094	2021-05-21 08:42:00.3191700 +00:00	319	00155D221399	00155D3769BD	IPv4	0	13
2095	2021-05-21 08:42:00.3200490 +00:00	320	00155D3769BD	00155D221399	IPv4	0	13
2096	2021-05-21 08:42:00.3201390 +00:00	320	00155D221399	00155D3769BD	IPv4	0	13
2097	2021-05-21 08:42:00.3208970 +00:00	320	00155D3769BD	00155D221399	IPv4	0	13
2098	2021-05-21 08:42:00.3209840 +00:00	320	00155D221399	00155D3769BD	IPv4	0	13
2099	2021-05-21 08:42:00.3379560 +00:00	337	00155D3769BD	00155D221399	IPv4	0	13

Рисунок 4.11 – Фрагмент таблиці отриманих мережевих пакетів

На рисунку 4.11 зображено фрагмент історії мережевої активності користувача з ідентифікатором 13, що зв’язано з даними таблиці підключених користувачів

.

4.4 Ефективність роботи розробленої розподіленої системи

Швидкі темпи розвитку апаратного забезпечення як правило компенсують неефективне використання ресурсів. Серед усіх видів програмного забезпечення лише ігри можуть повною мірою використати апаратні ресурси та є чутливими до ефективності. Для більшості програм апаратних ресурсів насправді надто багато, тому розробники поступово знижують вимоги до оптимізації ефективності використання системних ресурсів. Проте це не погіршує конкурентоспроможність системи, а навпаки може навіть покращити, оскільки для розробки більш ефективного ПЗ необхідні більші економічні затрати. Не зважаючи на те, що з розвитком технологій знижується їх вартість – неякісне програмне забезпечення має потенційну можливість нанести шкоди системі. Тенденції розвитку все ж сприяють появі ПЗ з низьким рівнем ефективності. Накопичення таких програмних продуктів в межах одного комп’ютера з часом призведе до сповільнення його роботи, що може бути критичним для користувача чи навіть всієї компанії, що володіє цим ПК. Тому оптимізація програм є невід’ємною частиною розробки.

Прискорення роботи програмного забезпечення повністю залежить від оптимізації програмного коду. Це стосується не тільки використаних алгоритмів, а й самого принципу роботи мови програмування, що може як сприяти швидкості виконання команд так і сповільнити процес.

Початковим етапом оптимізації програмного забезпечення є визначення вимог до продуктивності чи ефективності. Оптимізація продуктивності та ефективності потрібна лише тоді, коли існують вимоги щодо обмежень в часі чи ресурсах. Тому якщо задано ліміти: обчислювальних ресурсів p, оперативної пам’яті m, ресурсів даних d – визначаємо як реагуватиме система R та можливі

варіанти оптимізації для виконання обробки події e за час T, що знаходиться в заданому обмеженні в часі T_c. Звідси маємо:

$$T = R\ (p, m, d; e\ ) \leq T_c; \tag{4.1}$$

Вважаємо, що система ефективніша тоді, коли час відгуку найкоротший, тоді:

$$T = min[R\ (p, m, d; e\ )]. \tag{4.2}$$

На практиці зацікавлені сторони мають власні методи оцінки ефективності системи, отже метрики вимог та кінцеві результати оцінювання можуть суттєво відрізнятись.

В процесі оптимізації важливу роль відіграє оптимізація архітектури, що стосується розподілення системних ресурсів. Оптимізувати систему можна за наступним алгоритмом, що складається з п’яти основних етапів:

1) збір інформації про програмну та апаратну архітектуру системи: продуктивність процесора, об’єм пам’яті, швидкість пам’яті. Також необхідно визначити як програмне забезпечення відповідає на запити та загальні витрати ресурсів;

2) визначення цілі оптимізації: досягнення ефективності чи продуктивності системи. Встановлення лімітів на час виконання та використання ресурсів;

3) побудова моделі відповіді R та обчислення поточної продуктивності. Формування звіту про споживання ресурсів заданою моделлю;

4) застосування описаних вище правил для оптимізації системи;

5) повернення моделі до архітектурного вигляду. Результатом оптимізації є вдосконалена архітектура програмного забезпечення, що може бути застосована на практиці.

Експериментальні дослідження мережевого трафіку з використанням мережевого аналізатора Wireshark показали, що кожні десять хвилин сумарна кількість мережевих пакетів становить чверть мільйона. Цього трафіку достатньо

для проведення загального аналізу активності мережі та сформувати мультифрактальний спектр для нього. Оскільки описаний в третьому розділі алгоритм обробляє майже 4.9 мільйони наявних записів про мережеву активність за 8 секунд, тобто близько 500000 унікальних записів у базі даних за одну секунду, або 73 мегабайти даних. Для порівняння було проведено аналіз продуктивності Snort та Suricata. Розмір кожного пакету становить 1024 байти. Порівняльний графік зображено на рисунку 4.12.

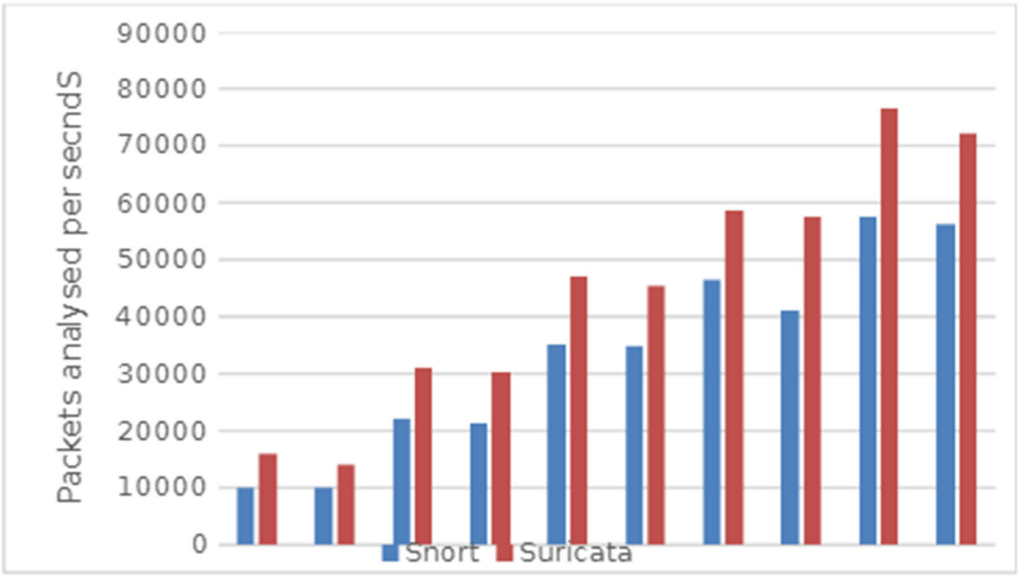


Рисунок 4.12 – Порівняльний графік ефективності Snort та Suricata

Найвищі результати експерименту показує система Suricata, обробляючи близько 73 мегабайтів даних за секунду, як і розроблена система. Проте для оптимальної роботи системи Suricata необхідним є багатоядерний процесор для вищої продуктивності, мінімальний об’єм оперативної пам’яті, необхідний для роботи становить 4Гб. В той же час клієнтська частина розподіленої системи не вимагає наявності продуктивного процесора та може працювати з мінімальним об’ємом оперативної пам’яті, оскільки для роботи служби достатньо менше 100 мегабайт вільної оперативної пам’яті. Для роботи серверної частини системні вимоги ідентичні Suricata. Проте слід зазначити, що сервер розподіленої системи обробляє дані багатьох клієнтів. Тому розроблена централізована розподілена

система визначення мережевих атак на основі мультифрактального аналізу працює ефективніше за найпопулярніші безкоштовні аналоги.

4.5 Висновки до четвертого розділу

Розроблена централізована розподілена система підтверджує можливість реалізації програмного забезпечення, згідно архітектури, описаної в 2 розділі.

Проведено випробування методу визначення мережевих атак на основі мультифрактального аналізу, результи яких показують високу швидкість роботи алгоритму та ефективність використання системних ресурсів при аналізі мережевого трафіку. Випробування також показали обмеження даного алгоритму, що полягає у відсутності можливості аналізу надто малих об’ємів даних, тому за низької активності в мережі необхідно проводити аналіз на більшому часовому проміжку ніж початковий.

В результаті випробувань розподіленої системи в різних середовищах, а саме Windows та Linux, було підтверджено високу швидкість роботи фонової служби, що досягнуто мінімізацією об’єму клієнтського додатку та ефективне використання ресурсів комп’ютера та мережі при передачі даних на сервер та віддаленому виклику процедур. Також було проведено випробування в кластері Docker. Поширення додатку в такому випадку спрощено використанням Docker Swarm.

Порівняння швидкості обробки мережевого трафіку показує, що розроблена система працює з такою ж швидкістю, як відома система виявлення вторгнень з відкритим вихідним кодом Suricata. Проте за відсутності графічного інтерфейсу реалізована система потребує значно менше системних ресурсів, що підвищує загальну швидкість роботи системи та ефективність використання системних ресурсів в цілому.

ВИСНОВКИ

В результаті виконання теоретичних та практичних досліджень, описаних в роботі було розроблено централізовану розподілену систему визначення мережевих атак на основі мультифрактального аналізу для підвищення ефективності визначення мережевих атак та безпеки системи в цілому.

Основними результатами досліджень є:

1) проаналізовано відомі архітектурні стилі та обрано найбільш гнучкий стиль для розробки розподіленої системи, що не тільки утворює багаторівневу структуру, а й за необхідності забезпечує автономність як окремих вузлів так і груп комп'ютерів. Такий підхід також усуває потребу повторення коду та надає можливість віддаленого виклику процедур;

2) визначено особливості прояву мережевих атак та розроблено на їх основі алгоритм вибірки лише необхідних для аналізу даних з мережевого пакету для зменшення кількості оброблюваної інформації та прискорення роботи мультифрактального аналізу.

3) удосконалено архітектуру централізованої розподіленої системи, в якій удосконалено спосіб підключення та взаємодії вузлів. На основі даних про поведінку в мережі центр прийняття рішень формує нову конфігурацію, що надає можливість динамічної зміни кількості компонентів системи та нарощування функцій в системі без модифікації базового коду;

4) розроблено метод підтримки цілісності та стійкості до відмов, що враховує поточний стан компонентів системи та автоматично обирає новий центр прийняття рішень і схему підключення доступних вузлів у випадку помилки мережі чи безпосередньо мережевої атаки на окремі комп'ютери в системі чи на систему в цілому. Це дозволяє системі автоматично змінювати конфігурацію підключення компонентів розподіленої системи та змінювати стратегію роботи без втручання користувача;

5) удосконалено метод виявлення мережевих атак методом мультифрактального аналізу, що надає можливість застосування методу в кластері

чи іншій багатокомп’ютерній системі, де встановлено централізовану розподілену систему виявлення мережевих атак на основі мультифрактального аналізу. Застосування методу прискорило обмін даними між компонентами системи та підвищило ефективність використання ресурсів при виконанні аналізу мережевого трафіку.

За темою дипломної роботи магістра опубліковано тези у збірнику Всеукраїнської науково-практичної конференції АПКН 2020, метою якої є висвітлення актуальних проблем інформаційних технологій, інформатики та комп’ютерних наук.

СПИСОК ДЖЕРЕЛ ПОСИЛАНЬ

1. Praveen Balda et al, *International Journal of Computer Science and Mobile Computing*, Vol.4 Issue.4, April 2015, pg. 761-767

2. Verissimo, Paulo; Rodrigues, Luis. *Distributed systems for system architects*. Springer Science & Business Media, 2012.

3. Fleischmann, Albert. *Distributed systems: software design and implementation*. Springer Science & Business Media, 2012.

4. Jia, Weijia; Zhou, Wanlei. *Distributed network systems: from concepts to implementations*. Springer Science & Business Media, 2004.

5. Y. Tsujimoto, Y. Miki, S. Shimatani, and K. Kiyono, Fast algorithm for scaling analysis with higher-order detrending moving average method, Phys. Rev. E 93(5), 053304 (2016)

6. Garcia-Morchon, Oscar, et al. Cooperative security in distributed networks. *Computer Communications*, 2013, 36.12: 1284-1297.

7. *Iosr Journal of Computer Engineering (IOSR-JCE)* ISSN : 2278-0661, ISBN : 2278-8727, PP : 53-56

8. Sidecar pattern URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar (дата звернення 15.02.2021).

9. GitHub URL: https://github.com/ (дата звернення 15.02.2021).

10. B. Stroustrup. What is object-oriented programming? *IEEE Software*, 1988. vol. 5, no. 3, pp. 10-20.

11. Proxy URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy (дата звернення 15.02.2021).

12. Karlgren, Klas; Ramberg, Robert. The use of design patterns in overcoming misunderstandings in collaborative interaction design. CoDesign, 2012, 8.4: 231-246.

13. Vardhan, Manu, et al. A demand based load balanced service replication model. *GSTF Journal on Computing (JoC)*, 2014, 2.4.

14. Sharding pattern URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/sharding (дата звернення 15.02.2021).

15. Leader election URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/leader-election (дата звернення 15.02.2021).

16. Network Attacks and Network Security Threats URL: https://www.cynet.com/network-attacks/network-attacks-and-network-security-threats/ (дата звернення 15.02.2021).

17. Snort URL: https://www.snort.org/ (дата звернення 15.02.2021).

18. Overview of file sharing using the SMB 3 protocol in Windows Server URL: https://docs.microsoft.com/en-us/windows-server/storage/file-server/file-server-smb-overview (дата звернення 15.02.2021).

19. What is the OSI Model? URL: https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/ (дата звернення 15.02.2021).

20. Zeek URL: https://zeek.org/ (дата звернення 15.02.2021).

21. Suricata URL: https://suricata-ids.org/ (дата звернення 19.02.2021).

22. Sguil: The Analyst Console for Network Security Monitoring URL: https://bammv.github.io/sguil/index.html (дата звернення 15.02.2021).

23. Tcl Developer Xchange URL: https://www.tcl.tk/ (дата звернення 19.02.2021).

24. Security Onion URL: https://securityonionsolutions.com/ (дата звернення 19.02.2021).

25. Kruegel, Christopher; Toth, Thomas. Using decision trees to improve signature-based intrusion detection. In: International Workshop on Recent Advances in Intrusion Detection. Springer, Berlin, Heidelberg, 2003. p. 173-191.

26. Cao, Jin; Drabeck, Lawrence; He, Ran. Statistical network behavior based threat detection. In: 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, 2017. p. 420-425.

27. Li, Jing, et al. A review on signature-based detection for network threats. In: 2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN). IEEE, 2017. p. 1117-1121.

28. Lysenko, Sergii & Bobrovnikova, Kira & Matiukh, Serhii & Hurman, Ivan & Savenko, Oleg. Detection of the botnets' low-rate DDoS attacks based on self-similarity. *International Journal of Electrical and Computer Engineering,* 2020. Vol.10. 3651. 10.11591/ijece.v10i4.pp3651-3659.

29. Shim, Simon SY. The CAP theorem's growing impact. Computer, 2012, 45.2: 21-22.

30. International Organization for Standardization URL: https://www.iso.org/about-us.html (дата звернення 19.02.2021).

31. Puder, Arno; Römer, Kay; Pilhofer, Frank. Distributed systems architecture: a middleware approach. Elsevier, 2011.

32. C. Sarkar, A. U. Nambi S. N., R. V. Prasad, A. Rahim, R. Neisse and G. Baldini, Diat: A Scalable Distributed Architecture for IoT, *IEEE Internet of Things Journal*, vol. 2, no. 3, pp. 230-239, June 2015, doi: 10.1109/JIOT.2014.2387155.

33. F. Jammes et al., Technologies for SOA-based distributed large scale process monitoring and control systems, IECON 2012 - 38th Annual Conference IEEE Industrial Electronics Society, 2012, pp. 5799-5804, doi: 10.1109/IECON.2012.6389589.

34. Rotem-Gal-Oz, Arnon; Bruno, Eric; Dahan, Udi. Soa patterns. Shelter Island: Manning, 2012.

35. Sarkar, Chayan, et al. Diat: A scalable distributed architecture for IoT. *IEEE Internet of Things journal*, 2014, 2.3: 230-239.

36. Lin, Cong. LePlaza: an event-centered social networking platform with location based personalized recommendation service. *PhD Thesis. University of British Columbia*. 2011

37. Kshemkalyani, Ajay D.; Singhal, Mukesh. Distributed computing: principles, algorithms, and systems. Cambridge University Press, 2011.

38. S. Helen, IRM: Integrated file replication and consistency maintainence in P2P Systems, *IEEE Trans. on Parallel and Distributed Systems*, Vol. 21, No. 1, January 2010, pp. 100-113.

39. A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras and B. Stiller, An overview of ip flow-based intrusion detection, *IEEE communications surveys & tutorials*, vol. 12, no. 3, pp. 343-356, 2010.

40. R. Perdisci, W. Lee and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces", NSDI, pp. 391-404, 2010.

41. Шагін В. Ю., Ковальчук Д. В., Каштальян А. С. Централізована розподілена система виявлення атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу. *Збірник наукових праць за матеріалами XII всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2020»*. Хмельницький – 2020. С. 345-347.

42. R. Lopez Perez, F. Adamsky, R. Soua and T. Engel, "Machine Learning for Reliable Network Attack Detection in SCADA Systems, 2018 *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2018, pp. 633-638, doi: 10.1109/TrustCom/BigDataSE.2018.00094.

43. C. Lee, Y. Su, Y. Lin and S. Lee, "Machine learning based network intrusion detection," 2017 2nd *IEEE International Conference on Computational Intelligence and Applications (ICCIA)*, 2017, pp. 79-83, doi: 10.1109/CIAPP.2017.8167184.

44. A. Gupta, O. J. Pandey, M. Shukla, A. Dadhich, S. Mathur and A. Ingle, Computational intelligence based intrusion detection systems for wireless communication and pervasive computing networks, 2013 *IEEE International Conference on Computational Intelligence and Computing Research*, 2013, pp. 1-7, doi: 10.1109/ICCIC.2013.6724156.

45. Kreitz G, Niemelä F (2010) Spotify-large scale, low latency, P2P music-on-demand streaming. In:Tenth international conference IEEE*, IEEE Computer Society Press, Los Alamitos, CA, Peer-to-PeerComputing*, pp 266–275.

46. Newman M (2010) Networks: an introduction. Oxford University Press, Oxford.

47. Voulgaris S, Dobson M, van Steen M (2016) Decentralized network-level synchronization in mobileAd Hoc networks. *ACM Trans Sensor Netw* 12(1). doi:10.1145/2880223

48. Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: state of the art and research challenges. JInternet Serv Appl 1(1):7–18.

49. Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson DA, Rabkin A,Stoica I, Zaharia M A view of cloud computing. Commun ACM 2010, 53(4):50–58.

50. Brewer E (2012) CAP twelve years later: how the “Rules” have changed. *IEEE Comput* 45(2):23–29.

51. Mottola L, Picco GP (2011) Programming wireless sensor networks: fundamental concepts and stateof the art. ACM Comput Surv 43(3):19:1–19:51.

52. Tarkoma S Overlay networks: toward information networking. CRC Press, Boca Raton. 2010

53. Todd Booth, Karl Andersson, Critical infrastructure network DDoS defense via cognitive learning" *Consumer Communications & Networking Conference (CCNC) 2017 14th IEEE Annual*, pp. 1-6, 2017.

54. Huan Yang, Liang Cheng, Mooi Choo Chuah, Deep-Learning-Based Network Intrusion Detection for SCADA Systems, Communications and Network Security (CNS) 2019 IEEE Conference on, pp. 1-7, 2019.

55. Bruno B. Bulle, Altair O. Santin, Eduardo K. Viegas, Roger R. dos Santos, A Host-based Intrusion Detection Model Based on OS Diversity for SCADA, *Industrial Electronics Society (IECON) 2020 The 46th Annual Conference of the IEEE*, pp. 691-696, 2020.

56. E. K. Viegas, A. O. Santin, V. V. Cogo and V. Abreu, Facing the unknown: A stream learning intrusion detection system for reliable model updates in Advanced Information Networking and Applications, Springer International Publishing, pp. 898-909, 2020.

57. E. Kakihata, H. Sapia, R. Oikawa, D. Pereira, J. Papa, V. Alburquerque, et al., Intrusion detection system based on flows using machine learning algorithms, *IEEE Latin America Transactions*, vol. 15, no. 10, pp. 1988-1993, Oct. 2017.

58. S. Yusuf, W. Luk, M. Sloman, N. Dulay, E. C. Lupu and G. Brown, Reconfigurable architecture for network flow analysis, *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 1, pp. 57-65, Jan. 2008.

59. A. L. França, R. Jasinski, P. Cemin, V. A. Pedroni and A. O. Santin, The energy cost of network security: A hardware vs. software comparison, Proc. *IEEE Int. Symp. Circuits Syst*., pp. 81-84, 2015.

60. S. Pontarelli, G. Bianchi and S. Teofili, Traffic-aware design of a high-speed FPGA network intrusion detection system, *IEEE Trans. Comput*., vol. 62, no. 11, pp. 2322-2334, Nov. 2013.

61. Vilmar Abreu, Altair O. Santin, Eduardo K. Viegas, Vinicius V. Cogo, Advanced Information Networking and Applications, vol. 1151, pp. 1215, 2020.

ДОДАТОК А
ЛІСТИНГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЦЕНТРАЛІЗОВАНОЇ РОЗПОДІЛЕНОЇ СИСТЕМИ

Startup.cs

```
using Distributed.System.ServerAPI.Common;
using Distributed.System.ServerAPI.EntityFramework;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.OpenApi.Models;

namespace Distributed.System.ServerAPI.Web.Api
{
   public class Startup
   {
      public Startup(IConfiguration configuration)
      {
         Configuration = configuration;
         ConfigurationAccessor.Configuration = configuration;
      }

      public IConfiguration Configuration { get; }

      // This method gets called by the runtime. Use this method to add services to the container.
      public void ConfigureServices(IServiceCollection services)
      {
         services.AddDbContext<AppDbContext>();
```

```
            services.AddControllers();
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title =
"Distributed.System.ServerAPI.Web.Api", Version = "v1" });
            });
        }

        // This method gets called by the runtime. Use this method to configure the HTTP
request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                app.UseSwagger();
                app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json",
"Distributed.System.ServerAPI.Web.Api v1"));
            }

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
```

```
}
```

Program.cs

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Distributed.System.ServerAPI.Web.Api
{
   public class Program
   {
      public static void Main(string[] args)
      {
         CreateHostBuilder(args).Build().Run();
      }

      public static IHostBuilder CreateHostBuilder(string[] args) =>
         Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
               webBuilder.UseStartup<Startup>();
            });
   }
}
```

Dockerfile

```
FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /src
COPY
["Distributed.System.ServerAPI.Web.Api/Distributed.System.ServerAPI.Web.Api.csproj", "Distributed.System.ServerAPI.Web.Api/"]
COPY
["Distributed.System.ServerAPI.EntityFramework/Distributed.System.ServerAPI.EntityFramework.csproj", "Distributed.System.ServerAPI.EntityFramework/"]
COPY
["Distributed.System.ServerAPI.Common/Distributed.System.ServerAPI.Common.csproj", "Distributed.System.ServerAPI.Common/"]
COPY
["Distributed.System.ServerAPI.Database/Distributed.System.ServerAPI.Database.csproj", "Distributed.System.ServerAPI.Database/"]
COPY
["Distributed.System.ServerAPI.Repository/Distributed.System.ServerAPI.Repository.csproj", "Distributed.System.ServerAPI.Repository/"]
COPY
["Distributed.System.ServerAPI.Services/Distributed.System.ServerAPI.Services.csproj", "Distributed.System.ServerAPI.Services/"]
RUN dotnet restore
"Distributed.System.ServerAPI.Web.Api/Distributed.System.ServerAPI.Web.Api.csproj"
COPY . .
WORKDIR "/src/Distributed.System.ServerAPI.Web.Api"
```

```
RUN dotnet build "Distributed.System.ServerAPI.Web.Api.csproj" -c Release -o
/app/build

FROM build AS publish
RUN dotnet publish "Distributed.System.ServerAPI.Web.Api.csproj" -c Release -o
/app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "Distributed.System.ServerAPI.Web
```

appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=LEGION;Initial
Catalog=CaptureStorage;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;
MultiSubnetFailover=False"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

ClientController.cs

```
using Distributed.System.ServerAPI.Database.Entities;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Threading.Tasks;

namespace Distributed.System.ServerAPI.Web.Api.Controllers
{
   [Route("[controller]")]
   public class ClientController : GenericController
   {
      [HttpGet]
      public async Task<ClientEntity> Get(int id, string IpAddress, string Name,
DateTimeOffset LastConnection)
      {
         return new ClientEntity();

         //return NoContent();
      }
   }
}
```

GenericController

```
using Microsoft.AspNetCore.Mvc;

namespace Distributed.System.ServerAPI.Web.Api.Controllers
{
   [ApiController]
   public class GenericController: ControllerBase
   {
```

```
    }
}
```

launchsettings.json

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:7441",
      "sslPort": 0
    }
  },
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "swagger",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "Distributed.System.ServerAPI.Web.Api": {
      "commandName": "Project",
      "launchBrowser": true,
      "launchUrl": "swagger",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
```

```
    },
    "dotnetRunMessages": "true",
    "applicationUrl": "http://localhost:5000"
   },
   "Docker": {
    "commandName": "Docker",
    "launchBrowser": true,
    "launchUrl": "{Scheme}://{ServiceHost}:{ServicePort}/swagger",
    "publishAllPorts": true
   }
  }
}
```

Distributed.System.ServerAPI.Web.Api.csproj

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
   <TargetFramework>net5.0</TargetFramework>
   <DockerDefaultTargetOS>Linux</DockerDefaultTargetOS>
   <OutputType>Exe</OutputType>
  </PropertyGroup>

  <ItemGroup>
   <PackageReference Include="Microsoft.EntityFrameworkCore.Tools"
Version="5.0.5">
    <PrivateAssets>all</PrivateAssets>
    <IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>
   </PackageReference>
```

```
    <PackageReference
Include="Microsoft.VisualStudio.Azure.Containers.Tools.Targets" Version="1.10.13"
/>
    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.1.2" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference
Include="..\Distributed.System.ServerAPI.Common\Distributed.System.ServerAPI.Co
mmon.csproj" />
    <ProjectReference
Include="..\Distributed.System.ServerAPI.EntityFramework\Distributed.System.Server
API.EntityFramework.csproj" />
    <ProjectReference
Include="..\Distributed.System.ServerAPI.Repository\Distributed.System.ServerAPI.Re
pository.csproj" />
  </ItemGroup>

</Project>
```

Worker.cs

```
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace WorkerCaptureService
```

```
{
    public class Worker : BackgroundService
    {
        private readonly ILogger<Worker> _logger;

        public Worker(ILogger<Worker> logger)
        {
            _logger = logger;
        }

        protected override async Task ExecuteAsync(CancellationToken stoppingToken)
        {
            CaptureTask captureTask = new CaptureTask();
            captureTask.CaptureWorker();
            while (!stoppingToken.IsCancellationRequested)
            {

                await Task.Delay(1000, stoppingToken);
            }
        }
    }
}
```

UserInfo.cs

```
using Distributed.System.ServerAPI.Database.Entities;
using Distributed.System.ServerAPI.EntityFramework;
using System;
using System.IO;
using System.Linq;
using System.Net;
```

```
namespace WorkerCaptureService
{
    public class UserInfo
    {
        public static long currentClientId = 0;
        public void WriteUserInfo()
        {
            String strHostName = new String("");
            strHostName = Dns.GetHostName();
            IPHostEntry ipEntry = Dns.GetHostEntry(strHostName);
            //IPAddress[] addr = ipEntry.AddressList;

            var db = new LocalAppDbContext();
            using (db)
            {
                var newClient = new ClientEntity()
                {
                    IpAddress = GetPublicIpAddress(),
                    Name = strHostName,
                    LastConnection = DateTimeOffset.Now
                };
                db.Clients.Add(newClient);
                db.SaveChanges();

                currentClientId += newClient.Id;

            }
        }
```

```
        private string GetPublicIpAddress()
        {
            var request = (HttpWebRequest)WebRequest.Create("http://ifconfig.me");

            request.UserAgent = "curl"; // this will tell the server to return the information as if the request was made by the linux "curl" command

            string publicIPAddress;

            request.Method = "GET";
            using (WebResponse response = request.GetResponse())
            {
                using (var reader = new StreamReader(response.GetResponseStream()))
                {
                    publicIPAddress = reader.ReadToEnd();
                }
            }

            return publicIPAddress.Replace("\n", "");
        }

        public ClientEntity Client()
        {
            var db = new LocalAppDbContext();

            return db.Clients.FirstOrDefault(c => c.Id == currentClientId);
        }
    }
}
```

Program.cs

```
using Distributed.System.ServerAPI.Common;
using Distributed.System.ServerAPI.EntityFramework;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using System;

namespace WorkerCaptureService
{
   public class Program
   {
      public Program(IConfiguration configuration)
      {
         Configuration = configuration;
         ConfigurationAccessor.Configuration = configuration;
      }

      public IConfiguration Configuration { get; }

      public static void Main(string[] args)
      {
         CreateHostBuilder(args).Build().Run();
      }

      public static IHostBuilder CreateHostBuilder(string[] args) =>
         Host.CreateDefaultBuilder(args)
            .UseWindowsService()
            .ConfigureServices((hostContext, services) =>
            {
               services.AddDbContext<LocalAppDbContext>();
```

```
            services.AddHostedService<Worker>();

        });
    }
}
```

PcapDevice.cs

```
using PacketDotNet;
using SharpPcap;
using SharpPcap.LibPcap;
using System;
using System.Linq;
using System.Net.NetworkInformation;

namespace WorkerCaptureService
{
    public class PcapDevice
    {
        /// <summary>
        /// Find the first Ethernet adapter that is actually connected to something
        /// </summary>
        /// <returns>Device</returns>
        public static LibPcapLiveDevice GetPcapDevice()
        {

            var nics = NetworkInterface.GetAllNetworkInterfaces();
            foreach (var device in LibPcapLiveDeviceList.Instance)
            {
                var friendlyName = device.Interface.FriendlyName ?? string.Empty;
                if (friendlyName.ToLower().Contains("loopback") || friendlyName == "any")
```

```
        {
            continue;
        }
        var nic = nics.FirstOrDefault(ni => ni.Name == friendlyName);
        if (nic?.OperationalStatus != OperationalStatus.Up)
        {
            continue;
        }

        LinkLayers link;
        try
        {
            device.Open();
            link = device.LinkType;
        }
        catch (PcapException ex)
        {
            Console.WriteLine(ex);
            continue;
        }
        finally
        {
            if (device.Opened) device.Close();
        }

        if (link == LinkLayers.Ethernet)
        {
            return device;
        }
    }
```

```
            throw new InvalidOperationException("No ethernet pcap supported devices found, are you running" +
                                        " as a user with access to adapters (root on Linux)?");
        }
    }
}
```

LocalAppDbContext.cs

```
using Distributed.System.ServerAPI.Common.Configuration;
using Distributed.System.ServerAPI.Database.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace WorkerCaptureService
{

    public class FKConfig : IEntityTypeConfiguration<CaptureEntity>
    {
        public void Configure(EntityTypeBuilder<CaptureEntity> modelBuilder)
        {
            modelBuilder.HasOne(y => y.ClientEntity)
                        .WithMany(x => x.Packets);
            //.HasForeignKey(x => x.ClientFK);
        }
    }

    public class LocalAppDbContext : DbContext
    {
        public DbSet<ClientEntity> Clients { get; set; }
        public DbSet<CaptureEntity> Packets { get; set; }
```

```
    public LocalAppDbContext(DbContextOptions<LocalAppDbContext> options)
        : base(options)
    {
        Database.Migrate();
    }

    public LocalAppDbContext()
    {
    }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer("Data Source=LEGION;Initial
Catalog=CaptureStorage;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;
MultiSubnetFailover=False");
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.ApplyConfiguration(new FKConfig());


        modelBuilder.Entity<ClientEntity>()
                .ToTable("Client")
                .HasKey(e => e.Id);

        //modelBuilder.Entity<ClientEntity>()
```

```
        //          .Property(e => e.Id).IsRequired();

        modelBuilder.Entity<CaptureEntity>()
                .ToTable("CapturedPackets")
                .HasKey(e => e.Id);

        //modelBuilder.Entity<CaptureEntity>()
        //          .Property(e => e.Id).IsRequired();
    }
  }
}
```

Dockerfile

```
FROM mcr.microsoft.com/dotnet/runtime:5.0 AS base
WORKDIR /app

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /src
COPY ["WorkerCaptureService/WorkerCaptureService.csproj",
"WorkerCaptureService/"]
RUN dotnet restore "WorkerCaptureService/WorkerCaptureService.csproj"
COPY . .
WORKDIR "/src/WorkerCaptureService"
RUN dotnet build "WorkerCaptureService.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "WorkerCaptureService.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
```

```
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "WorkerCaptureService.dll"]

CaptureTask.cs
using SharpPcap;
using SharpPcap.LibPcap;
using System;
using Distributed.System.ServerAPI.EntityFramework;

namespace WorkerCaptureService
{
   public class CaptureTask
   {
      public void CaptureWorker()
      {
         UserInfo nfo = new UserInfo();
         nfo.WriteUserInfo();

         LibPcapLiveDevice device = PcapDevice.GetPcapDevice();

         Console.WriteLine("-- Listening on {0} {1}", device.Interface.FriendlyName,
device.Description);
         device.OnPacketArrival +=
            new PacketArrivalEventHandler(CaptureHandler.device_OnPacketArrival);
         int readTimeoutMilliseconds = 1000;
         device.Open(mode: DeviceMode.Promiscuous, read_timeout:
readTimeoutMilliseconds);
         device.StartCapture();
      }
   }
```

```
}
```

CaptureHandler.cs

```
using Distributed.System.ServerAPI.Database.Entities;
using Distributed.System.ServerAPI.EntityFramework;
using Distributed.System.ServerAPI.Repository.ClientRepository;
using Microsoft.EntityFrameworkCore;
using PacketDotNet;
using SharpPcap;
using System;
using System.Linq;

namespace WorkerCaptureService
{
   public class CaptureHandler : UserInfo
   {
      private static int packetIndex = 0;

      public static void device_OnPacketArrival(object sender, CaptureEventArgs e)
      {
         LocalAppDbContext context = new LocalAppDbContext();

         ClientRepository clientRepository = new ClientRepository(new
AppDbContext());

         if (e.Packet.LinkLayerType == LinkLayers.Ethernet ||
            e.Packet.LinkLayerType == LinkLayers.Ieee80211 ||
            e.Packet.LinkLayerType == LinkLayers.ArcNet ||
            e.Packet.LinkLayerType == LinkLayers.Raw)
         {
```

```
            var packet = Packet.ParsePacket(e.Packet.LinkLayerType, e.Packet.Data);
            var ethernetPacket = (EthernetPacket)packet;

            using (context)
            {
               var captured = new CaptureEntity
               {
                  DateTime = e.Packet.Timeval.Date,
                  DateTimeMiliseconds = e.Packet.Timeval.Date.Millisecond,
                  SourceHardwareAddress =
ethernetPacket.SourceHardwareAddress.ToString(),
                  DestinationHardwareAddress =
ethernetPacket.DestinationHardwareAddress.ToString(),
                  PacketType = ethernetPacket.Type.ToString(),
                  ClientEntity = context.Clients.FirstOrDefault(c => c.Id ==
currentClientId)

               };
               context.Packets.Add(captured);
               context.Database.ExecuteSqlRaw("SET IDENTITY_INSERT dbo.Client
ON;");
               context.SaveChanges();
               context.Database.ExecuteSqlRaw("SET IDENTITY_INSERT dbo.Client
OFF;");
            };

            Console.WriteLine("{0} At: {1}:{2}: MAC:{3} -> MAC:{4} var: {5}",
                        packetIndex,
                        e.Packet.Timeval.Date.ToString(),
                        e.Packet.Timeval.Date.Millisecond,
```

```
                    ethernetPacket.SourceHardwareAddress.ToString(),
                    ethernetPacket.DestinationHardwareAddress.ToString(),
                    currentClientId);
            packetIndex++;
        }
    }
  }
}
```

AbbDbContext.cs

```
using Distributed.System.ServerAPI.Common.Configuration;
using Distributed.System.ServerAPI.Database.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Distributed.System.ServerAPI.EntityFramework
{
  public class FKConfig : IEntityTypeConfiguration<CaptureEntity>
  {
    public void Configure(EntityTypeBuilder<CaptureEntity> modelBuilder)
    {
      modelBuilder.HasOne(y => y.ClientEntity)
            .WithMany(x => x.Packets);
            //.HasForeignKey(x => x.ClientFK);
    }
  }

  public class AppDbContext : DbContext
  {
```

```
public DbSet<ClientEntity> Clients { get; set; }
public DbSet<CaptureEntity> Packets { get; set; }

public AppDbContext(DbContextOptions<AppDbContext> options)
   : base(options)
{
   Database.Migrate();
}

public AppDbContext()
{
}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
   optionsBuilder.UseSqlServer(ConfigurationManager.ConnectionString);
}

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
   base.OnModelCreating(modelBuilder);
   modelBuilder.ApplyConfiguration(new FKConfig());


   modelBuilder.Entity<ClientEntity>()
            .ToTable("Client")
            .HasKey(e => e.Id);

   //modelBuilder.Entity<ClientEntity>()
```

```
            //          .Property(e => e.Id).IsRequired();

            modelBuilder.Entity<CaptureEntity>()
                    .ToTable("CapturedPackets")
                    .HasKey(e => e.Id);

            //modelBuilder.Entity<CaptureEntity>()
            //          .Property(e => e.Id).IsRequired();
        }
    }
}
```

AppDbContextModelSnapshot.cs

```
// <auto-generated />
using System;
using Distributed.System.ServerAPI.EntityFramework;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

namespace Distributed.System.ServerAPI.EntityFramework.Migrations
{
    [DbContext(typeof(AppDbContext))]
    partial class AppDbContextModelSnapshot : ModelSnapshot
    {
        protected override void BuildModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
```

```
        modelBuilder
            .HasAnnotation("Relational:MaxIdentifierLength", 128)
            .HasAnnotation("ProductVersion", "5.0.5")
            .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);


modelBuilder.Entity("Distributed.System.ServerAPI.Database.Entities.CaptureEntity",
b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint")
                    .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

                b.Property<long?>("ClientEntityId")
                    .HasColumnType("bigint");

                b.Property<long>("ClientFK")
                    .HasColumnType("bigint");

                b.Property<DateTimeOffset>("DateTime")
                    .HasColumnType("datetimeoffset");

                b.Property<int>("DateTimeMiliseconds")
                    .HasColumnType("int");

                b.Property<string>("DestinationHardwareAddress")
                    .HasColumnType("nvarchar(max)");
```

```
            b.Property<string>("PacketType")
                .HasColumnType("nvarchar(max)");

            b.Property<string>("SourceHardwareAddress")
                .HasColumnType("nvarchar(max)");

            b.HasKey("Id");

            b.HasIndex("ClientEntityId");

            b.ToTable("CapturedPackets");
        });

modelBuilder.Entity("Distributed.System.ServerAPI.Database.Entities.ClientEntity", b
=>
        {
            b.Property<long>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("bigint")
                .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

            b.Property<string>("IpAddress")
                .HasColumnType("nvarchar(max)");

            b.Property<DateTimeOffset>("LastConnection")
                .HasColumnType("datetimeoffset");
```

```
                b.Property<string>("Name")
                    .HasColumnType("nvarchar(max)");

                b.HasKey("Id");

                b.ToTable("Client");
            });

modelBuilder.Entity("Distributed.System.ServerAPI.Database.Entities.CaptureEntity",
b =>
            {
                b.HasOne("Distributed.System.ServerAPI.Database.Entities.ClientEntity",
"ClientEntity")
                    .WithMany("Packets")
                    .HasForeignKey("ClientEntityId");

                b.Navigation("ClientEntity");
            });

modelBuilder.Entity("Distributed.System.ServerAPI.Database.Entities.ClientEntity", b
=>
            {
                b.Navigation("Packets");
            });
#pragma warning restore 612, 618
        }
    }
}
```

CaptureEntity.cs

```
using System;

namespace Distributed.System.ServerAPI.Database.Entities
{
   public class CaptureEntity
   {
      public long Id { get; set; }
      public DateTimeOffset DateTime { get; set; }
      public int DateTimeMiliseconds { get; set; }

      public string DestinationHardwareAddress { get; set; }
      public string SourceHardwareAddress { get; set; }

      public string PacketType { get; set; }

      public long ClientFK { get; set; }
      public ClientEntity ClientEntity { get; set; }
   }
}
```

ClientEntity.cs

```
using System;
using System.Collections.Generic;

namespace Distributed.System.ServerAPI.Database.Entities
{
   public class ClientEntity
```

```
    {
        public long Id { get; set; }
        public string IpAddress { get; set; }
        public string Name { get; set; }
        public DateTimeOffset LastConnection { get; set; }

        public List<CaptureEntity> Packets { get; set; }
    }
}

namespace Distributed.System.ServerAPI.Common
{
    public class AppConstants
    {
        public const string ConnectionStringName = "DefaultConnection";
    }
}
using Microsoft.Extensions.Configuration;

namespace Distributed.System.ServerAPI.Common
{
    public abstract class ConfigurationAccessor
    {
        public static IConfiguration Configuration { get; set; }
    }
}
using Microsoft.Extensions.Configuration;

namespace Distributed.System.ServerAPI.Common.Configuration
```

```
{
    public class ConfigurationManager : ConfigurationAccessor
    {
        private static string _connectionString { get; }

        static ConfigurationManager()
        {
            _connectionString =
Configuration.GetConnectionString(AppConstants.ConnectionStringName);
        }

        public static string ConnectionString => _connectionString;
    }
}

using Distributed.System.ServerAPI.Database.Entities;
using Distributed.System.ServerAPI.EntityFramework;
using Microsoft.EntityFrameworkCore;
using System;
using System.Threading.Tasks;

namespace Distributed.System.ServerAPI.Repository.CaptureRepository
{
    public class CaptureRepository : ICaptureRepository
    {
        private readonly AppDbContext _dbContext;

        public CaptureRepository(AppDbContext dbContext)
        {
```

```
            _dbContext = dbContext;
        }

        public async ValueTask<CaptureEntity> FindByPacketType(string packetType)
        {
            return await _dbContext.Packets.FirstOrDefaultAsync(e => e.PacketType ==
packetType);
        }

        public async ValueTask<CaptureEntity> FindByTimeArrived(DateTimeOffset
timeStamp)
        {
            return await _dbContext.Packets.FirstOrDefaultAsync(e => e.DateTime ==
timeStamp);
        }
    }
}

using Distributed.System.ServerAPI.Database.Entities;
using System;
using System.Threading.Tasks;

namespace Distributed.System.ServerAPI.Repository.CaptureRepository
{
    public interface ICaptureRepository
    {
        ValueTask<CaptureEntity> FindByPacketType(string packetType);

        ValueTask<CaptureEntity> FindByTimeArrived(DateTimeOffset timeStamp);
    }
```

```
}

using Distributed.System.ServerAPI.Database.Entities;
using Distributed.System.ServerAPI.EntityFramework;
using Microsoft.EntityFrameworkCore;
using System.Threading.Tasks;

namespace Distributed.System.ServerAPI.Repository.ClientRepository
{
   public class ClientRepository : IClientRepository
   {
      private readonly AppDbContext _dbContext;

      public ClientRepository(AppDbContext dbContext)
      {
         _dbContext = dbContext;
      }

      public async ValueTask<ClientEntity> FindClientById(long id)
      {
         return await _dbContext.Clients.FirstOrDefaultAsync(e => e.Id == id);
      }

      public async ValueTask<ClientEntity> FindClientByIp(string ipAddress)
      {
         return await _dbContext.Clients.FirstOrDefaultAsync(e => e.IpAddress ==
ipAddress);
      }
```

```
        public async ValueTask<ClientEntity> FindClientByName(string name)
        {
            return await _dbContext.Clients.FirstOrDefaultAsync(e => e.Name == name);
        }
    }
}

using Distributed.System.ServerAPI.Database.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Distributed.System.ServerAPI.Repository.ClientRepository
{
    interface IClientRepository
    {
        ValueTask<ClientEntity> FindClientById(long id);

        ValueTask<ClientEntity> FindClientByIp(string ipAddress);

        ValueTask<ClientEntity> FindClientByName(string name);
    }
}

using Distributed.System.Api.Repository;
using Distributed.System.ServerAPI.EntityFramework;
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

namespace Distributed.System.ServerAPI.Repository
{
   public class GenericRepository<T> : IGenericRepository<T> where T : class
   {
      protected readonly AppDbContext _dbContext;

      public GenericRepository(AppDbContext dbContext)
      {
         _dbContext = dbContext;
      }

      public IQueryable<T> Entities => _dbContext.Set<T>();

      public virtual void Add(T entity)
      {
         _dbContext.Add(entity);
      }

      public virtual void AddRange(IEnumerable<T> entities)
      {
         _dbContext.AddRange(entities);
      }

      public virtual void Update(T entity)
      {
```

```
            _dbContext.Update(entity);
        }

        public virtual IEnumerable<T> Find(Expression<Func<T, bool>> expression)
        {
            return _dbContext.Set<T>().Where(expression);
        }

        public virtual IEnumerable<T> GetAll()
        {
            return _dbContext.Set<T>().ToList();
        }

        public virtual T GetById(int id)
        {
            return _dbContext.Set<T>().Find(id);
        }

        public virtual void Remove(T entity)
        {
            _dbContext.Remove(entity);
        }

        public virtual void RemoveRange(IEnumerable<T> entities)
        {
            _dbContext.RemoveRange(entities);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

namespace Distributed.System.Api.Repository
{
    public interface IGenericRepository<T> where T : class
    {
        IQueryable<T> Entities { get; }
        T GetById(int id);
        IEnumerable<T> GetAll();
        IEnumerable<T> Find(Expression<Func<T, bool>> expression);
        void Add(T entity);
        void AddRange(IEnumerable<T> entities);
        void Update(T entity);
        void Remove(T entity);
        void RemoveRange(IEnumerable<T> entities);
    }
}
```

ДОДАТОК Б
СТАТТІ ЗА РЕЗУЛЬТАТАМИ ДОСЛІДЖЕННЯ

Актуальні проблеми комп'ютерних наук

УДК 004.75:004.9

Шагін В. Ю., Ковальчук Д. В., Каштальян А. С.

Хмельницький національний університет

ЦЕНТРАЛІЗОВАНА РОЗПОДІЛЕНА СИСТЕМА ВИЯВЛЕННЯ АТАК В КОРПОРАТИВНИХ КОМП'ЮТЕРНИХ МЕРЕЖАХ НА ОСНОВІ МУЛЬТИФРАКТАЛЬНОГО АНАЛІЗУ

Розглянуто різні моделі для розробки розподілених мережевих систем, парадигми зв'язку, що використовуються в розподіленій мережевій системі, та принципи надійності та безпеки при проєктуванні розподілених мережевих систем. Представлено підходи до створення централізованої розподіленої системи виявлення атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу.

Considered different models for developing distributed network systems, the communication paradigms used in a distributed network system, and the principles of reliability and security in the design of distributed network systems. Approaches to the creation of a centralized distributed attack detection system in corporate computer networks based on multifractal analysis are presented.

В сучасному інформаційному просторі дедалі важливішим стає питання розподілених [1] мережевих систем через постійне збільшення об'ємів оброблюваної інформації.

Розподілена система складається із компонентів, які розміщено на різних машинах, що обмінюються даними і координують дії як єдина цілісна система. Компонентам розподіленої системи можуть бути комп'ютери, сервери, віртуальні машини, контейнери та інші пристрої, що можуть підключитись до мережі, мають власну локальну пам'ять та можуть обмінюватись повідомленнями. Існує два загальних способи функціонування розподілених систем: кожна машина працює задля спільної мети і для кінцевого користувача і результат формується як єдине ціле; кожна машина має власного кінцевого користувача, а розподілена система полегшує обмін обчислювальними ресурсами або послугами зв'язку.

Розподілені системи переважно мають такі основні характеристики: всі компоненти працюють одночасно, немає глобального годинника, і всі компоненти виходять з ладу не залежно один від одного.

Головними перевагами розподілених систем є масштабованість, надійність, продуктивність.

Проблематика при використанні розподілених систем:

1) планування – розподілена система повинна вирішувати коли, як і де виконувати завдання;

2) латентність – чим ширше розповсюджується мережа, тим більше затримок може виникнути під час обміну даними;
3) можливість спостереження – збір, обробка та моніторинг метрик використання апаратного забезпечення для великих кластерів сьогодні є досить складним завданням.

Типи (моделі) розподілених систем. Клієнт-сервер – клієнт отримує дані від серверу, форматує та відображає кінцевому користувачу.

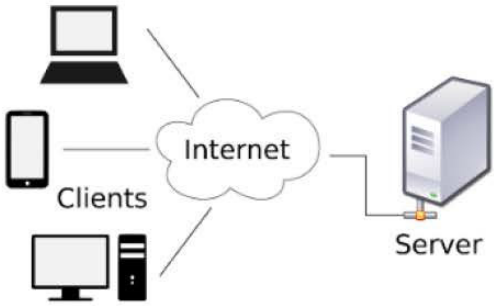


Рисунок 1 – Модель клієнт-сервер [2]

Трирівнева – інформація про клієнта зберігається на середньому рівні, щоб спростити розгортання застосунків. Ця модель найбільш поширена для веб-додатків.

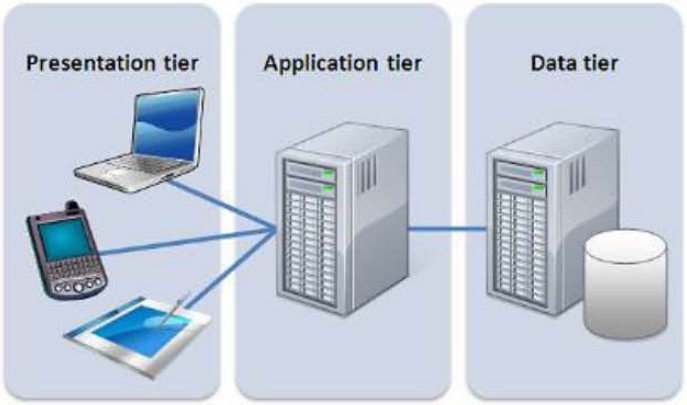


Рисунок 2 – Трирівнева модель [3]

Багаторівнева – зазвичай використовуються, коли застосунку або серверу потрібно переслати запити на додаткові корпоративні служби в мережі.

Peer-to-peer – для надання послуг або управління ресурсами сервер не використовується. Обов'язки рівномірно розподілені між вузлами, що можуть працювати як в режимі клієнта так і в режимі сервера.

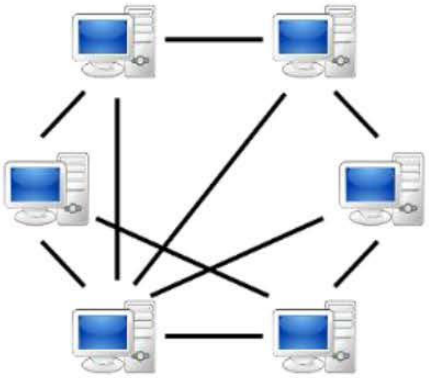


Рисунок 3 – Модель Peer-to-peer [4]

Розглянуті моделі розподілених систем є основою для створення централізованої розподіленої системи. Ця система буде призначена для виявлення атак в корпоративних комп'ютерних мережах і методи, які вона використовуватиме базуватимуться на основі мультифрактального аналізу. Що досягти цього результату важливим є інтеграція розподіленої системи та методів фрактального аналізу. Ефективність такої системи суттєво залежатиме від закладеної в неї моделі та методів виявлення [5].

Висновки

В результаті дослідження проаналізовано різні моделі для розробки розподілених мережевих систем, парадигми зв'язку, що використовуються в розподіленій мережевій системі, та принципи надійності та безпеки при проєктуванні розподілених мережевих систем. Представлено підходи до створення централізованої розподіленої системи виявлення атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу.

Перелік посилань

1. Distributed networking [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Distributed_networking
2. Three-tier architecture [Електронний ресурс]. – Режим доступу: https://managementmania.com/en/three-tier-architecture
3. Peer-to-peer [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Peer-to-peer
4. Multitier architecture [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Multitier_architecture
5. Савенко О. С. Дослідження методів антивірусного діагностування комп'ютерних мереж / О. С. Савенко, С. М. Лисенко // Вісник Хмельницького національного університету. Технічні науки. – 2007. – № 2, т. 2. – С. 120–126.

УДК 004.092

ШАГІН В. Ю., НІЧЕПОРУК А.А., КАШТАЛЬЯН А. С.
Хмельницький національний університет

ЦЕНТРАЛІЗОВАНА РОЗПОДІЛЕНА СИСТЕМА ВИЯВЛЕННЯ АТАК В КОРПОРАТИВНИХ КОМП'ЮТЕРНИХ МЕРЕЖАХ НА ОСНОВІ МУЛЬТИФРАКТАЛЬНОГО АНАЛІЗУ

В роботі запропоновано архітектуру та компоненти розподіленої системи виявлення мережевих атак, в якій поєднано вимоги централізованості, розподіленості, самоорганізованості та на її основі здійснено розробку централізованої розподіленої системи визначення мережевих атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу. Проведені експериментальні дослідження з реалізованою централізованою розподіленою системою визначення мережевих атак в комп'ютерних мережах підтвердили ефективність функціонування в комп'ютерній мережі.

Ключові слова:Розподілена система, Виявлення атаки, Мультифрактальний аналіз, Комп'ютерна мережа

SHAGIN V. Y., NICHEPORUK A. A., KASHTALIAN A.S.
Khmelnytskyi National University

CENTRALIZED DISTRIBUTED ATTACK DETECTION SYSTEM IN CORPORATE COMPUTER NETWORKS BASED ON MULTIFRACTAL ANALYSIS

The architecture and components of a distributed network attack detection system are proposed in the paper, which combines the requirements of centralization, distribution, self-organization and on its basis develops a centralized distributed network attack detection system in corporate computer networks based on multifractal analysis. Experimental studies with a centralized distributed system for detecting network attacks in computer networks have confirmed the effectiveness of functioning in a computer network.

The proposed centralized system is based on the use of multifractals. Multifractals are complex fractals that occur, as a rule, in nature. In fact, the multifractal approach means that some object under study can be divided into parts that have their own similarity characteristics that are different from others. Network traffic is self-similar at some intervals. Therefore, the method of maxima of wavelet transform modules will be used for its analysis, which allows to determine the features of the signal.

To conduct experimental research, a distributed system was implemented and software was deployed to detect attacks on local computer networks. The study found that the total data processing time is about 8 seconds, while the amount of data is almost 4.9 million lines, in the format of a text document, this data is more than 700 megabytes. Thus, this indicates the high speed of the algorithm and the efficiency of system resources.

Keywords: Distributed System, Attack Detection, Multifractal Analysis, Computer Network

Вступ. Зі зростанням об'ємів інформації, що передаються через мережу зростає і число апаратних та програмних засобів для втручання в процес передачі даних. Щодня все більше користувачів в інтернеті стають жертвами дій зловмисного програмного забезпечення. Особливу небезпеку становлять атаки та корпоративні системи, що можуть спричинити фінансові збитки організаціям чи безпосередньо їх працівникам.

Для мінімізації кількості успішних мережевих атак в корпоративних комп'ютерних мережах може бути використана математична статистика. Оскільки в корпоративних комп'ютерних мережах, як правило, присутня велика кількість комп'ютерів необхідні ефективні методи виявлення та реагування на активність в мережі. Існує твердження, що немає абсолютно захищеної системи, проте можливе суттєве зниження шансу на несанкціонований доступ до даних чи комп'ютерної системи загалом. Для протидії активності зловмисників необхідний комплексний підхід до розробки методів виявлення вторгнень та систем, в які інтегруються ці методи.

Концепція розподілених систем. На сьогоднішній день питання надійності комп'ютерної мережі є надзвичайно важливим. Тому для бізнесу широко використовуються розподілені мережі, які можуть бути реалізовані за допомогою програмного продукту. Концепт розподіленої мережі можна трактувати по різному. В широкому розумінні це робочі станції, що обмінюються інформацією та спільно її обробляють, при цьому кожна робоча станція може мати різну обчислювальну потужність та ємність сховища даних.

Модель розподіленої мережі передбачає поділ завдань між клієнтам чи серверами, що залежить від того, чи підходить конкретна машина для обробки даних [1-5]. За цим типом архітектури частина додатку виконується клієнтом, а інша частина сервером. З клієнт-серверним типом побудови програмного забезпечення користувач працює лише з обмеженою кількістю даних, включаючи інтерфейс користувача, користувацький ввід та запити до бази даних. Контроль доступу до бази даних, отримання чи обробку даних користувачем здійснює безпосередньо сервер. При цьому важливо перевіряти чи справді клієнт має дозвіл на оперування інформацією. Для цього проводять процедуру двох голосувань. Під час першого голосування кожен вузол

отримує довіру, розподіляючи інформацію про відкликання сусідів розподіленої мережі. Сусідні вузли створюють довірене оточення. Якщо вузол обмінюється інформацією з іншими в достатній мірі – він може і далі спілкуватись з рештою в мережі, в іншому випадку він відхиляється. Якщо вузол загрожує мережі та проявляє підозрілу поведінку – відбувається друга процедура відкликання голосування. В цьому випадку, якщо оточуючі вузли вказують на підозрілу активність свого сусіда – блокуються весь обмін даними з цим вузлом по всій мережі.

Основними властивостями розподіленої системи за теоремою CAP [6] є цілісність, доступність та стійкість до відмов (Consistency, Availability, Partition tolerance). Проте будь-яка розподілена система може мати не більше двох властивостей.

Цілісність або узгодженість системи за теоремою CAP означає її лінійність. Кожна наступна операція повинна мати інформацію про результат попередньої. Доступність системи означає безперебійну успішну обробку запитів на всіх активних вузлах. Якщо частина вузлів не відповідає чи система відповідає не на всі запити – за теоремою CAP це непостійна доступність. Стійкість до відмов в першу чергу забезпечує стабільну роботу всієї системи. Наприклад, якщо кластер із декількох серверів втратив з'єднання з половиною своїх вузлів – робота системи продовжується, хоч і було втрачено доступність. В іншому випадку частини кластера працюють незалежно один від одного та відповідають на запити користувачів. В такому випадку тільки після відновлення зв'язку між всіма ланками розподіленої системи буде зрозуміло, що це була єдина система.

Згідно теореми CAP можна виділити три варіанти проєктування розподіленої системи: AC – доступність та цілісність; CP – цілісність та стійкість; AP – доступність та стійкість.

AC системи мають суттєвий недолік – відсутність стійкості до відмов мережі. Використання цього типу систем вимагає чіткого усвідомлення всіх ризиків. В іншому випадку необхідно обирати між цілісністю та доступністю системи.

Таким чином, з огляду на широку розповсюдженість централізованої архітектури, важливим завданням є проектування системи, яка дозволить здійснити виявлення атак та підвищити загальний рівень безпеки всієї мережі.

Централізована система виявлення атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу. В основі запропонованої централізовано системи закладено використання мультифракталів. Мультифракталами називають складні фрактали, що зустрічаються, як правило, в природі. Фактично мультифрактальний підхід означає, що деякий досліджуваний об'єкт можна розділити на частини, що мають власні характеристики подібності, відмінні від інших. Мережевий трафік є самоподібним на деяких часових проміжках. Тому для його аналізу буде використано метод максимумів модулів вейвлет-перетворення, що дозволяє визначити особливості сигналу.

Для аналізу параметрів мультифрактального спектру використаємо наступний алгоритм:

1. Декомпозиція вихідного сигналу $f(t)$ на коефіцієнти батьківським вейвлетом $\psi(t)$:

$$W_f(u,j) = \left(f(t), \psi_{u,s}(t)\right) = 2^{-j/2} \int_{-\infty}^{\infty} \frac{t-u}{2^j} dt$$

де u – параметр масштабу, j – просторова координата чи момент часу;

2. В масиві коефіцієнтів знаходимо позиції локальних максимумів $\{u_p(j)\}_{p\in Z}$ та знаходимо їх абсолютне значення та формуємо масив максимумів

$$\left|W_f(u_p,j)\right|;$$

3. Визначення функції розбиття:

$$S(q,j) = \sum_p \left|W_f(u_p,j)\right|^q;$$

4. Для кожного q ∈ R обчислюємо показник масштабу:

$$\tau(q,j) = lim \inf_{j\to 0} \frac{\ln S(q,j)}{\ln 2^j};$$

5. Обчислюємо мультифрактальний спектр за допомогою перетворення Лежандра:

$$f_L(\alpha) = \min_{q\in R}[q(\alpha + 1/2) - \tau(q)];$$

6. Для кожного проміжку j обчислюємо мультифрактальні розмірності порядку q:

$$D_{q,j} = \frac{1}{q-1}[q(\alpha(q,j) - f(\alpha(q),j)].$$

Для ілюстрації та аналізу мережевого трафіку обрано його інтенсивність, тобто кількість відправлених та прийнятих пакетів за одиницю часу.

Принцип виявлення вторгнень запропонованої системи наступний. Нехай X – часовий ряд звичайного трафіку, Y – часовий ряд шкідливого трафіку, Z – часовий ряд аномалій. Звідси Y=X+Z. Незалежно від наявності властивостей самоподібності в часовому ряді аномалій – Y все ще буде самоподібним процесом, якщо X стаціонарний самоподібний процес. Проте ступінь самоподібності може змінюватись. Нехай $s_X s_Y s_Z$ функції автокореляції для X, Y та Z відповідно. Тоді під час атаки акцентуємо увагу на $\|s_Y - s_X\|$, при цьому $s_Y = s_X + s_Z$. Для кожного $H \in (0.5, 1)$ існує лише одна функція автокореляції з самоподібністю. Тому розглядається $\|H_Y - H_X\|$, де H_Y та H_X – середнє значення показників Херста X та Y відповідно. Коефіцієнт Херста вводиться для підвищення точності оцінки самоподібності системи. Недоліком підходу є необхідність

перезапуску визначення порогу самоподібності трафіку для кожного масштабу. Тому сигнал про зміну самоподібності буде подано незалежно від того, чи існує він для іншого масштабу. Після визначення мережевої атаки трафік розбивається на декілька частин. Інтенсивність атаки можна визначити за допомогою аналізу показника Херста та швидкості його зміни, тобто різницю між показниками Херста до факту атаки та після [7].

Визначення точки зміни самоподібності трафіку базується на тому, що ентропія послідовності зі змінною граничною точкою самоподібності більша ніж ентропія послідовності з фіксованою точкою.

На рис. 1 представлено схему оцінки безпеки мережевого трафіку. Алгоритм складається з п'яти етапів:

1. Збір трафіку;
2. Статистичний аналіз;
3. Оцінка показника Херста;
4. Визначення аномалій;
5. Оцінка безпеки.

Для зменшення впливу на функціонування мережі трафік дублюється на сервер, що збирає мережеву інформацію кожного з підключених клієнтів.

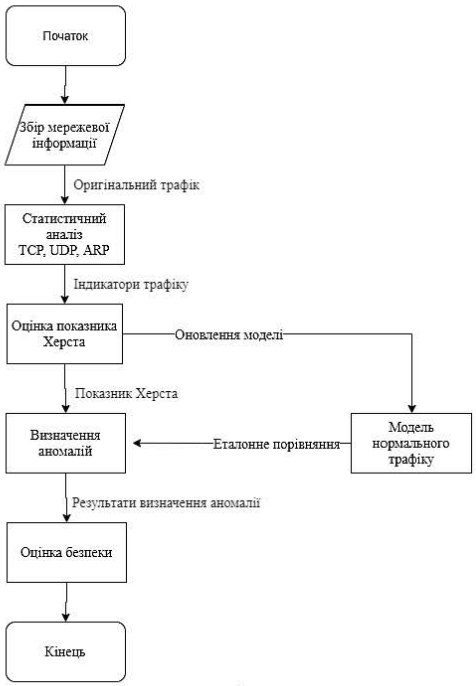


Рис. 1. Визначення оцінки безпеки мережевого трафіку

Експериментальні дослідження. Для розробки централізованої системи виявлення атак було використано середовище Microsoft Visual Studio Community 2019. Досліджувана централізована система складалась з одного сервера та двох клієнтських станцій.

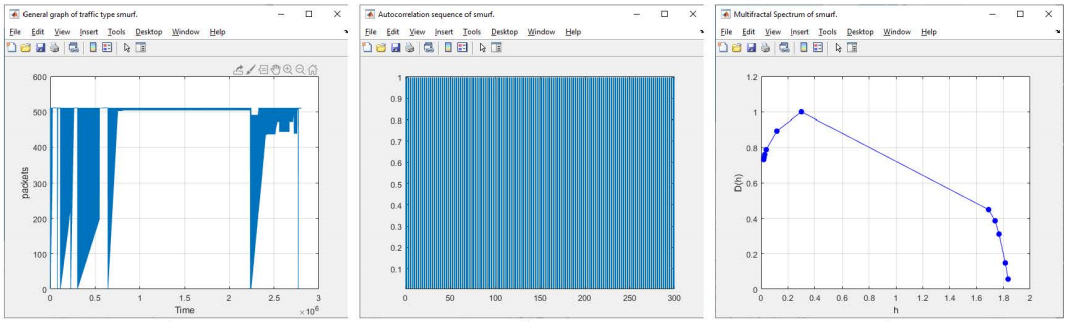
Для реєстрації мережевих пакетів було обрано бібліотеку з відкритим вихідним кодом SharpPcap. Бібліотека SharpPcap підтримує наступні мережеві протоколи: Ethernet, LinuxSLL, Ip (IPv4 and IPv6), Tcp, Udp, ARP, ICMPv4 и ICMPv6, IGMPv2, PPPoE, PTP, Link Layer Discovery Protocol (LLDP), Wake-On-LAN (WOL). Основними можливостями цього пакету є визначення активних мережевих пристроїв, формування статистики, зчитування мережевої інформації з активних мережевих пристроїв та читання з файлів, фільтрація пакетів, збереження мережевої інформації в файл. Форматом даних є Pcap та pcap-ng, за умови використання npcap чи libpcap версії 1.1.0 та вище.

В результаті дослідження було реалізовано наступні модулі: «WebAPI», «Common», «Database», «EntityFramework», «Repository» та «Services». Клієнтський додаток виступає в ролі фонової служби та не має графічного інтерфейсу чи ключів запуску для повної автономності. Тому використовуючи методи бібліотеки SharpPcap додаток самостійно знаходить активний мережевий інтерфейс та відкриває підключення до нього.

Для дослідження методу було використано пакет прикладних програм аналізу та програмування Matlab. Візуалізація мультифрактального спектру відбувається вбудованими засобами програмного забезпечення.

Розроблене програмне забезпечення використовує набір даних KDD Cup [8], який конвертовано в таблицю засобами Matlab. Для виведення графіків трафіку використано значення кількості пакетів. В даному наборі даних кожен рядок можна розглядати як статистику активності за одиницю часу. Перед створенням графіків програма фільтрує статистичні дані за кожним наявним типом мережевих атак. Далі формуються графіки мережевого трафіку для кожного виду мережевих атак та окремо графік нормального трафіку.

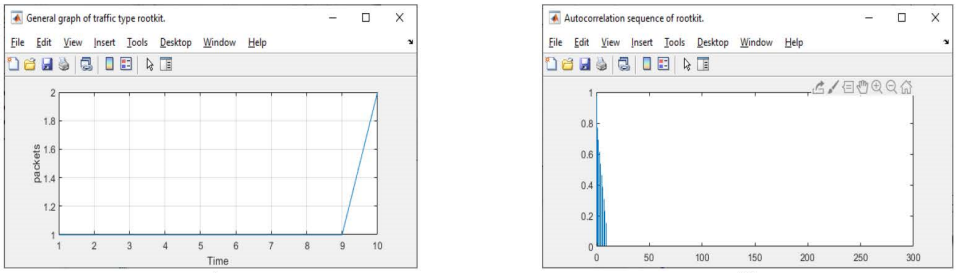
Результатом виконання представленого вище коду є графік мережевого трафіку, графік автокореляційної послідовності та мультифрактальний спектр (рис. 2).



а) б) в)

Рис. 2. Результати проведеного експерименту: а) графік активності мережевої атаки smurf; б) автокореляційна послідовність мережевої атаки smurf; в) мультифрактальний спектр мережевої атаки smurf

Слід відзначити, що запропонований метод має обмеження: при недостатній кількості значень в автокореляційній послідовності неможливо сформувати мультифрактальний спектр. На це впливає кількість випадків визначеного типу атаки на часовій шкалі. Яскравим прикладом з набору даних є активність мережевої атаки rootkit, що зображено на рисунку та її автокореляційна послідовність (рис. 3).



а) б)

Рис. 3. Результати системи при недостатній кількості значень в автокореляційній послідовності на прикладі мережевої атаки RootKit: а) графік активності мережевої атаки; б) автокореляційна послідовність мережевої атаки

Проте, разом з тим запропонована система, при достатній кількості даних, дозволяє здійснити не тільки факт виявлення атаки, а й розрізнити її тип (рис. 4).

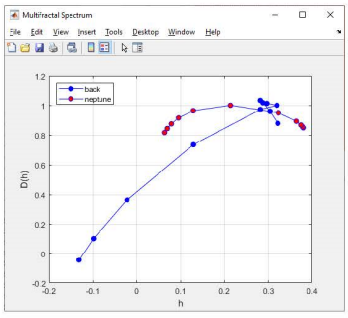
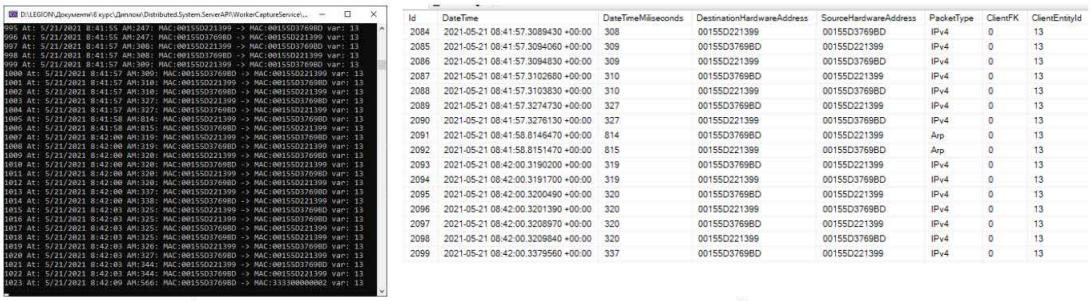


Рис. 4. Мультифрактальний спектр двох різновидів DOS атак

Виконання клієнтського програмного забезпечення не потребує багато ресурсів та невидиме для користувача, оскільки відбувається в якості фонової служби. Для налагодження програми було використано консоль, де відбувається виведення активності вузла (рис. 5).



Id	DateTime	DateTimeMiliseconds	DestinationHardwareAddress	SourceHardwareAddress	PacketType	ClientFK	ClientEntityId
2084	2021-05-21 08:41:57.3089430 +00:00	308	00155D221399	00155D3769BD	IPv4	0	13
2085	2021-05-21 08:41:57.3094060 +00:00	309	00155D3769BD	00155D221399	IPv4	0	13
2086	2021-05-21 08:41:57.3094830 +00:00	309	00155D221399	00155D3769BD	IPv4	0	13
2087	2021-05-21 08:41:57.3102680 +00:00	310	00155D3769BD	00155D221399	IPv4	0	13
2088	2021-05-21 08:41:57.3103830 +00:00	310	00155D221399	00155D3769BD	IPv4	0	13
2089	2021-05-21 08:41:57.3274730 +00:00	327	00155D3769BD	00155D221399	IPv4	0	13
2090	2021-05-21 08:41:57.3276130 +00:00	327	00155D221399	00155D3769BD	IPv4	0	13
2091	2021-05-21 08:41:58.8146470 +00:00	814	00155D3769BD	00155D221399	Arp	0	13
2092	2021-05-21 08:41:58.8151470 +00:00	815	00155D221399	00155D3769BD	Arp	0	13
2093	2021-05-21 08:42:00.3190200 +00:00	319	00155D3769BD	00155D221399	IPv4	0	13
2094	2021-05-21 08:42:00.3191700 +00:00	319	00155D221399	00155D3769BD	IPv4	0	13
2095	2021-05-21 08:42:00.3200490 +00:00	320	00155D3769BD	00155D221399	IPv4	0	13
2096	2021-05-21 08:42:00.3201390 +00:00	320	00155D221399	00155D3769BD	IPv4	0	13
2097	2021-05-21 08:42:00.3208970 +00:00	320	00155D3769BD	00155D221399	IPv4	0	13
2098	2021-05-21 08:42:00.3209840 +00:00	320	00155D221399	00155D3769BD	IPv4	0	13
2099	2021-05-21 08:42:00.3379560 +00:00	337	00155D3769BD	00155D221399	IPv4	0	13

а) б)

Рис. 5. Результати роботи розробленого програмного забезпечення: а) консоль клієнтського додатку, б) фрагмент таблиці отриманих мережевих пакетів

В процесі реалізації запропонованої системи було визначено також швидкість її роботи. Так загальний час обробки даних складає близько 8 секунд, тоді як об'єм даних складає майже 4.9 мільйони рядків, в форматі текстового документу ці дані займають більше 700 мегабайт. Таким чином, це свідчить про високу швидкість роботи алгоритму та ефективність використання системних ресурсів.

Висновки. В результаті проведеного дослідження розроблено архітектуру та компоненти розподіленої системи виявлення мережевих атак, в якій поєднано вимоги централізованості, розподіленості, самоорганізованості та на її основі здійснено розробку централізованої розподіленої системи визначення мережевих атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу. Проведені експериментальні дослідження з реалізованою централізованою розподіленою системою визначення мережевих атак в комп'ютерних мережах підтвердили ефективність функціонування в комп'ютерній мережі.

Література

1. C. Sarkar, A. U. Nambi S. N., R. V. Prasad, A. Rahim, R. Neisse and G. Baldini, "DIAT: A Scalable Distributed Architecture for IoT," in IEEE Internet of Things Journal, vol. 2, no. 3, pp. 230-239, June 2015, doi: 10.1109/JIOT.2014.2387155.
2. F. Jammes et al., "Technologies for SOA-based distributed large scale process monitoring and control systems," IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society, 2012, pp. 5799-5804, doi: 10.1109/IECON.2012.6389589.
3. S. Helen, IRM: Integrated file replication and consistency maintainence in P2P Systems, IEEE Trans. on Parallel and Distributed Systems, Vol. 21, No. 1, January 2010, pp. 100-113.
4. A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras and B. Stiller, "An overview of ip flow-based intrusion detection", IEEE communications surveys & tutorials, vol. 12, no. 3, pp. 343-356, 2010.
5. A. Gupta, O. J. Pandey, M. Shukla, A. Dadhich, S. Mathur and A. Ingle, "Computational intelligence based intrusion detection systems for wireless communication and pervasive computing networks," 2013 IEEE International Conference on Computational Intelligence and Computing Research, 2013, pp. 1-7, doi: 10.1109/ICCIC.2013.6724156.
6. S. S. Y. Shim, "Guest Editor's Introduction: The CAP Theorem's Growing Impact," in Computer, vol. 45, no. 2, pp. 21-22, Feb. 2012, doi: 10.1109/MC.2012.54.
7. P. Dymora, M. Mazurek, Anomaly detection in iot communication network based on spectral analysis and hurst exponent, Applied Sciences 9, 5319 (2019). doi: 10.3390/app9245319
8. KDD Cup 1999 Dataset, URL: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

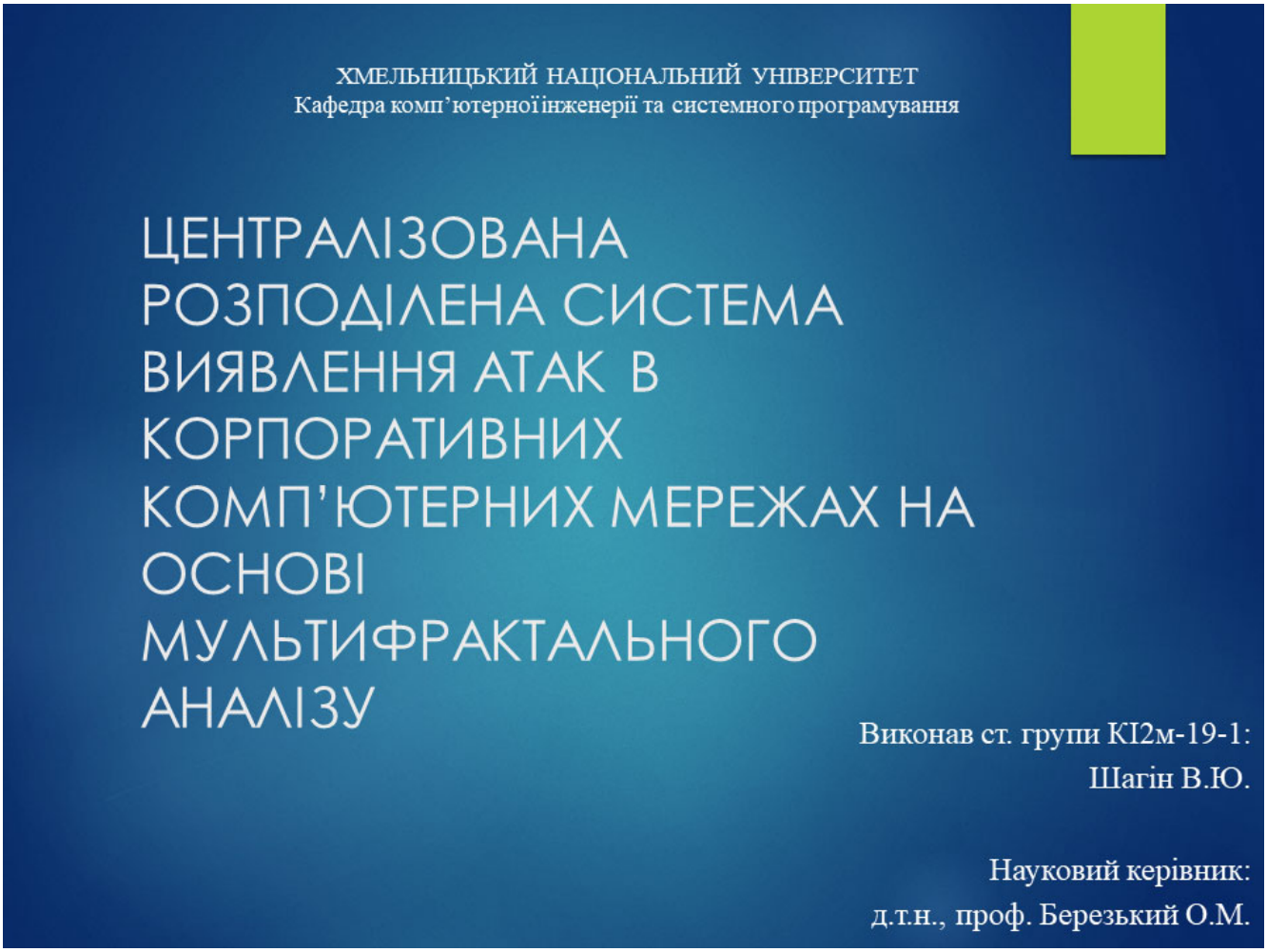
ДОВІДКА

Видана Шагіну В.Ю., що стаття «Централізована розподілена система виявлення атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу», співавтором якої він є, здана у №1 за 2021 рік журналу «Вимірювальна та обчислювальна техніка в технологічних процесах».

Головний редактор журналу  Мартинюк В.В.

ДОДАТОК В

ПРЕЗЕНТАЦІЯ ДОПОВІДІ



ОБ'ЄКТ, МЕТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- Об'єктом дослідження є процес функціонування централізованих розподілених систем виявлення мережевих атак в комп'ютерних мережах.
- Предмет дослідження є методи і засоби створення централізованих розподілених систем виявлення мережевих атак в комп'ютерних мережах
- Метою роботи є покращення ефективності виявлення аномалій в комп'ютерних системах шляхом розробки розподіленої системи виявлення мережевих атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу.

НАУКОВА НОВИЗНА

- удосконалено архітектуру розподіленої системи виявлення мережевих атак в комп'ютерних мережах, в якій скомбіновано вимоги розподіленості, централізованості, самоорганізованості та багаторівневості, що дало змогу вдосконалити її внутрішню структуру та принцип взаємодії центру прийняття рішень з компонентами системи з чітким розмежуванням верхнього та нижнього рівнів ієрархії із впровадженням методу збереження цілісності;
- удосконалено метод виявлення мережевих атак на основі мультифрактального аналізу в комп'ютерних мережах, що надає можливість його використання в комп'ютерних мережах з багатьма вузлами, на яких встановлено розподілену систему виявлення мережевих атак.

ПРАКТИЧНА ЦІННІСТЬ

В результаті виконаних в роботі досліджень розроблено архітектуру та компоненти розподіленої системи виявлення мережевих атак, в якій поєднано вимоги централізованості, розподіленості, самоорганізованості та на її основі здійснено розробку централізованої розподіленої системи визначення мережевих атак в корпоративних комп'ютерних мережах. Проведені експериментальні дослідження з реалізованою централізованою розподіленою системою визначення мережевих атак в комп'ютерних мережах підтвердили ефективність функціонування в комп'ютерній мережі.

ЗАДАЧІ ДОСЛІДЖЕННЯ

- дослідити та визначити особливості прояву аномалій в комп'ютерних мережах під час активності зловмисного програмного забезпечення і здійснення мережевих атак в локальній мережі, проаналізувати архітектури розподілених систем, доступні засоби виявлення мережевих атак та їх особливості;
- удосконалити архітектуру розподіленої системи виявлення вторгнень в комп'ютерних мережах методом синтезу вимог розподіленості та централізованості з можливістю самообслуговування, для створення на її основі системи, що працюватиме під керуванням одного центру прийняття рішень про рівень безпеки мережевого трафіку;

ЗАДАЧІ ДОСЛІДЖЕННЯ

- розробити метод підтримки цілісності розподіленої системи виявлення мережевих атак, на основі якого система змогла б за допомогою центру прийняття рішень самостійно змінювати свою архітектуру та подальшу стратегію роботи без втручання користувача;
- удосконалити метод централізованого виявлення мережевих атак за алгоритмом мультифрактального аналізу для аналізу мережевого трафіку на різних часових проміжках та з різним масштабом;
- розробити програмне забезпечення централізованої розподіленої системи виявлення мережевих атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу для підтвердження можливості реалізації таких систем згідно запропонованої архітектури та використання в експериментальних дослідженнях для порівняння з відомими аналогами.

МЕТОДИ ДОСЛІДЖЕННЯ

Для досягнення результатів поставлених задач використано основні положення:

- теорії розподілених систем, що можуть бути основою для розробки програмних чи апаратно-програмних засобів;
- методи мультифрактального аналізу для виявлення мережевих атак;
- теорії комп'ютерних мереж для розгортання та функціонування розподіленої системи.

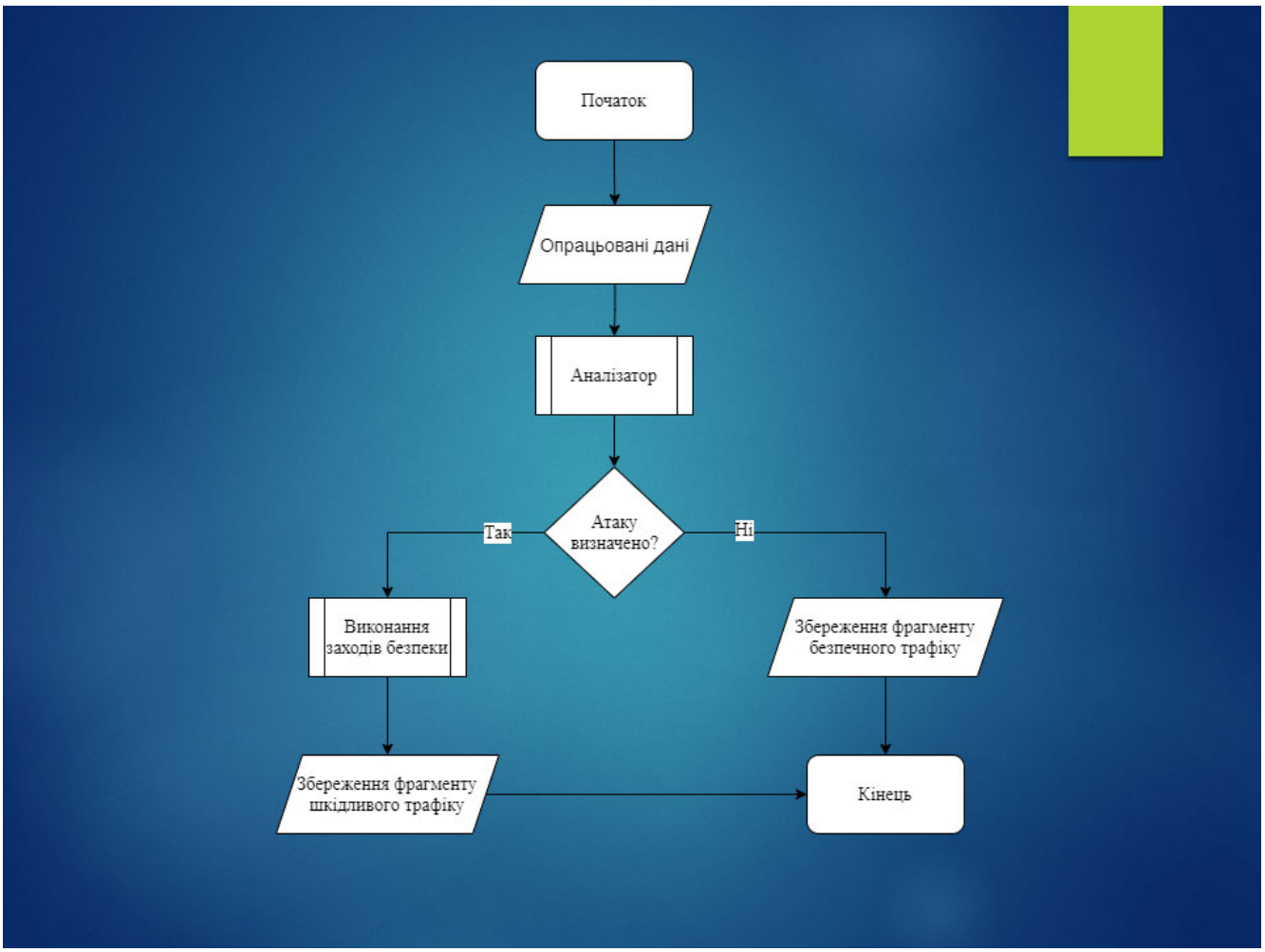
МЕТОДИ ДОСЛІДЖЕННЯ

Для досягнення результатів поставлених задач використано основні положення:

- теорії розподілених систем, що можуть бути основою для розробки програмних чи апаратно-програмних засобів;
- методи мультифрактального аналізу для виявлення мережевих атак;
- теорії комп'ютерних мереж для розгортання та функціонування розподіленої системи.

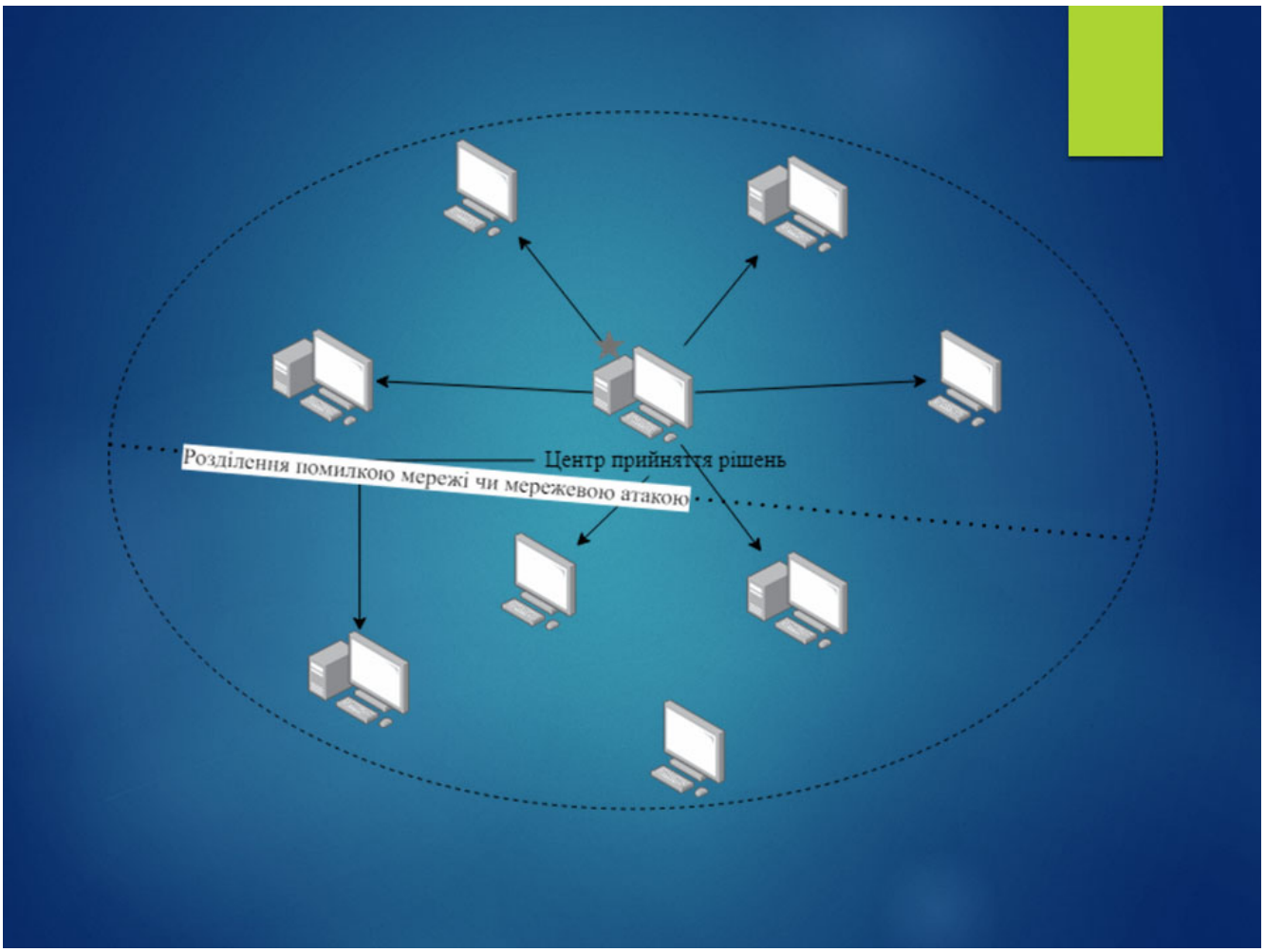
Центр прийняття рішень

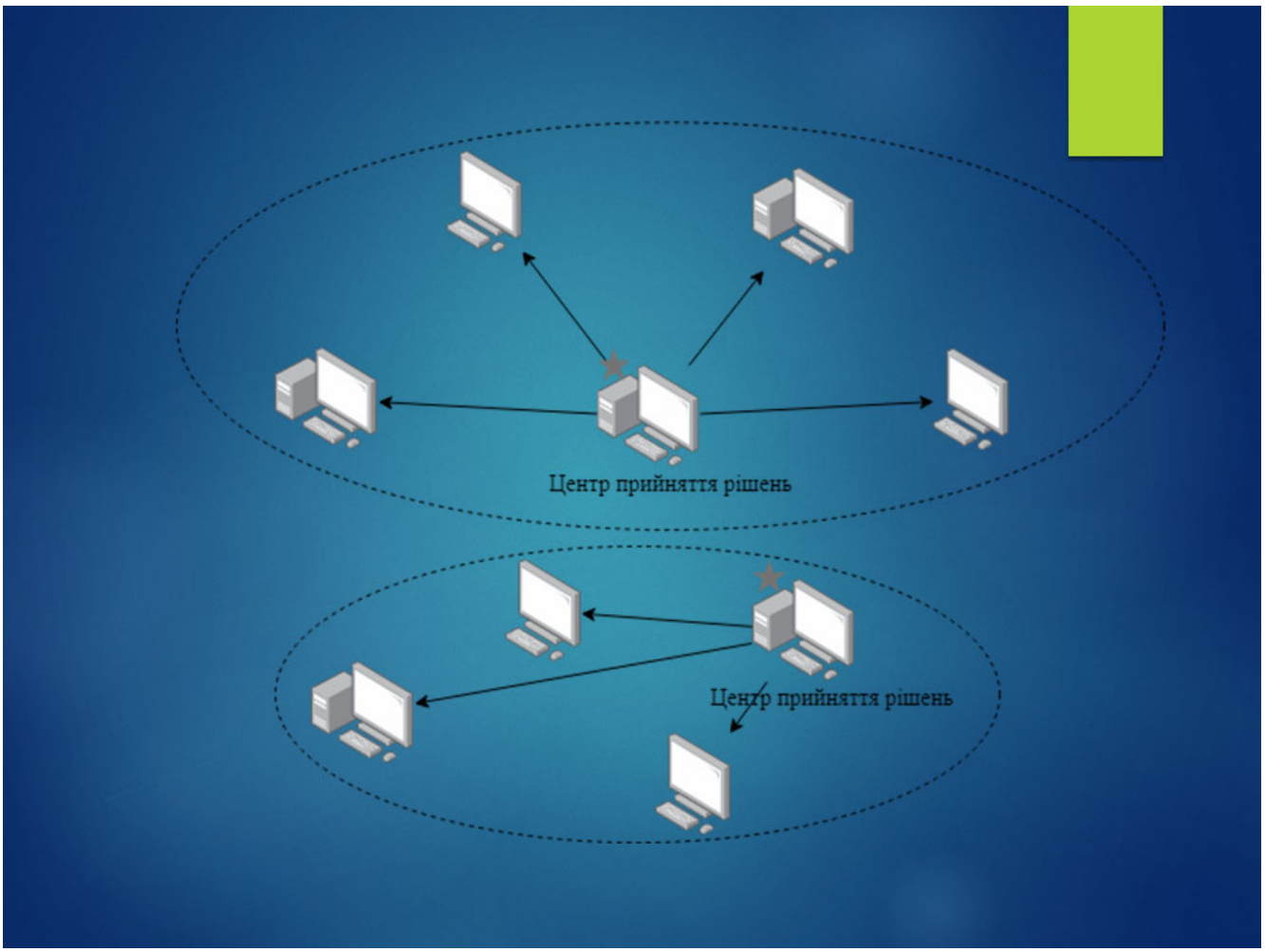
- Досить велика кількість мережевих атак пов'язана з людським фактором, що часто стає причиною витоку даних чи перехоплення контролю над системою зловмисниками. Тому необхідно, щоб система самостійно реагувала на мережеві атаки. Для цього повинен бути центр, що керуватиме підключенням клієнтів та їх взаємодію між собою. В такому випадку користувач не зможе втручатись в роботу системи. Самодіагностика та автоматичне реагування суттєво зменшить кількість раптових збоїв чи відмов інформаційної системи.


Початок
Опрацьовані дані
Аналізатор
Атаку визначено?
Так
Ні
Виконання заходів безпеки
Збереження фрагменту безпечного трафіку
Збереження фрагменту шкідливого трафіку
Кінець

Центр прийняття рішень

- Згідно представленої в першому розділі архітектури «вибір лідера» та загальної концепції розподілених систем, при якій проводиться опитування вузлів та встановлення загальної оцінки довіри до вузла. При виявленні підозрілої активності вузла проводиться повторне опитування, що відкликає результати першого, в результаті чого обмін даними з цим вузлом блокується по всій мережі. Головною умовою підтримка цілісності є наявність вузлів, що можуть взяти на себе керуючу роль, що забезпечить відмовостійкість системи у випадку успішної мережевої атаки. Кількість таких вузлів має бути більше половини, для забезпечення кворуму – мінімальної кількості керуючих одиниць та безперебійного функціонування кластеру


Центр прийняття рішень
Розділення помилкою мережі чи мережевою атакою

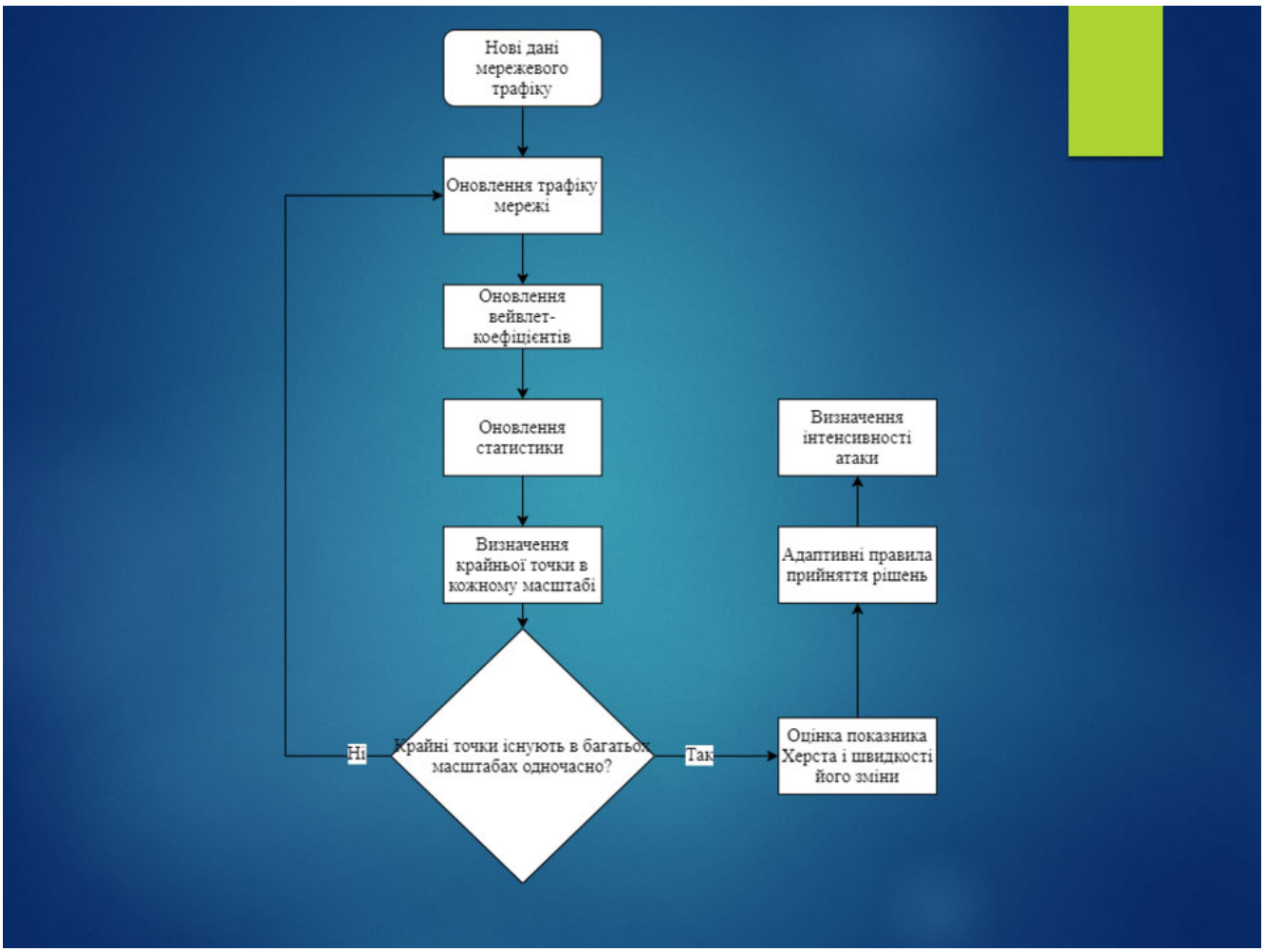

Центр прийняття рішень
Центр прийняття рішень

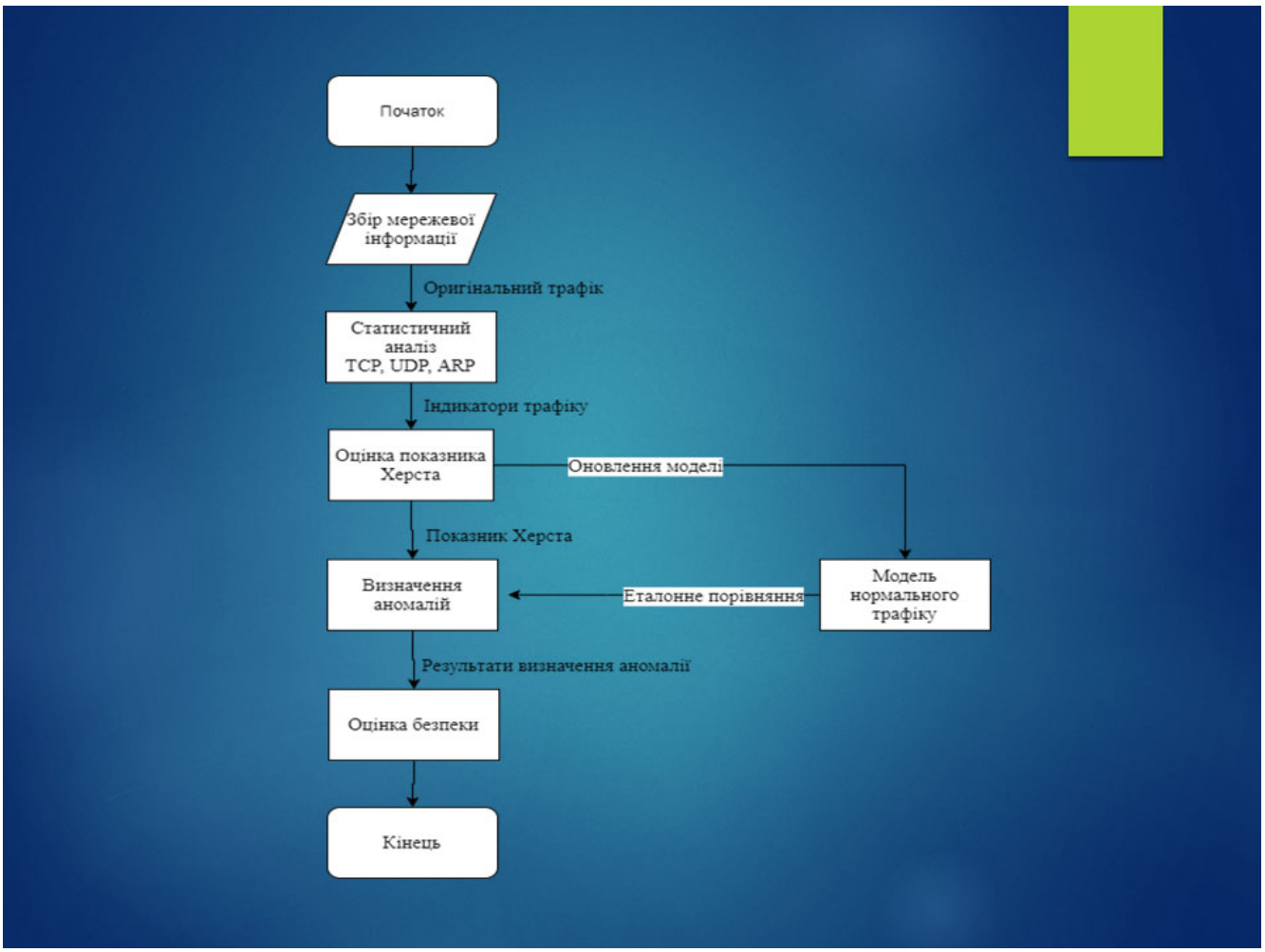
Центр прийняття рішень

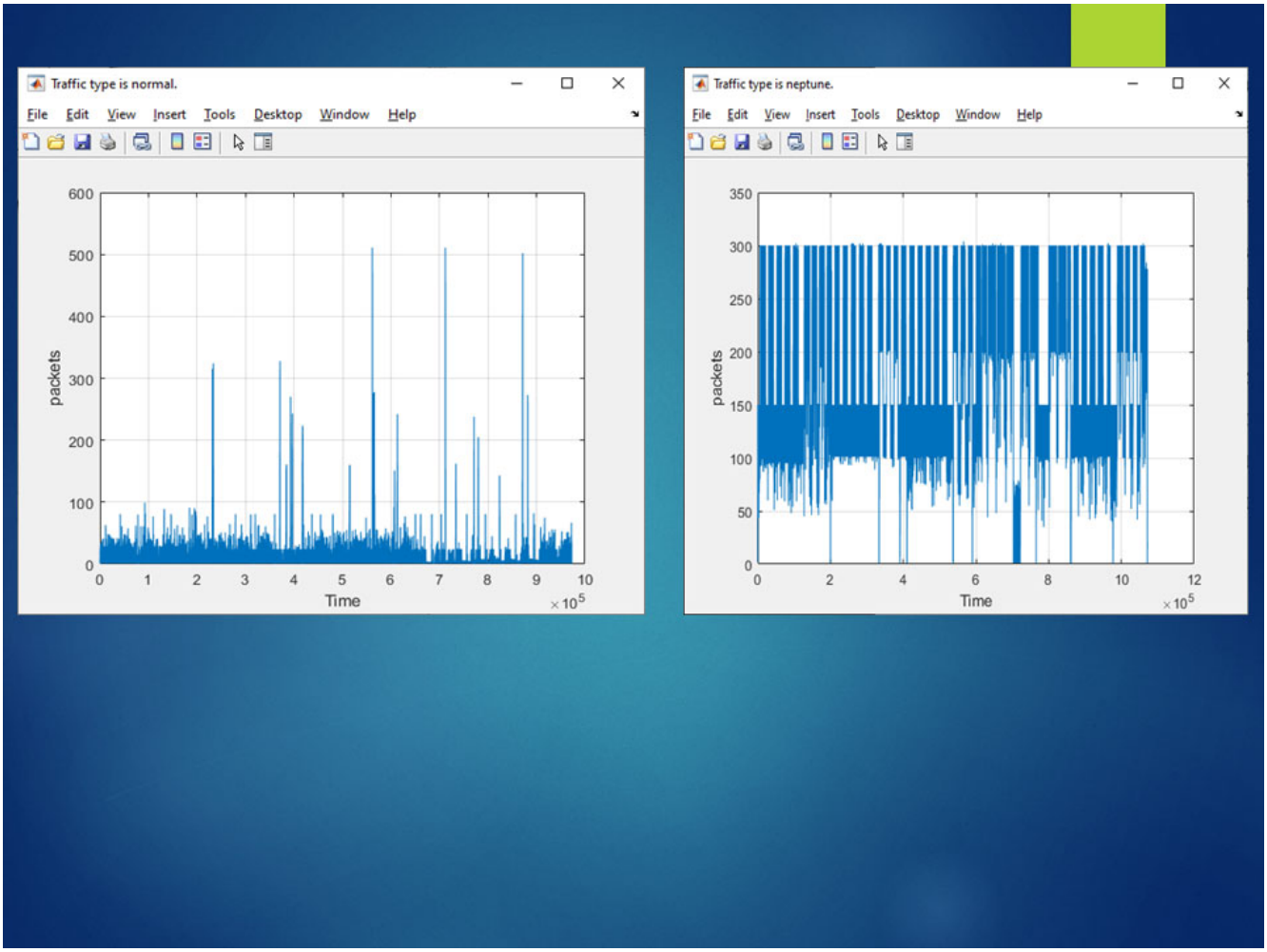
Кількість керуючих вузлів	Розподіл на зони доступності
3	1-1-1
5	2-2-1
7	3-2-2
9	3-3-3

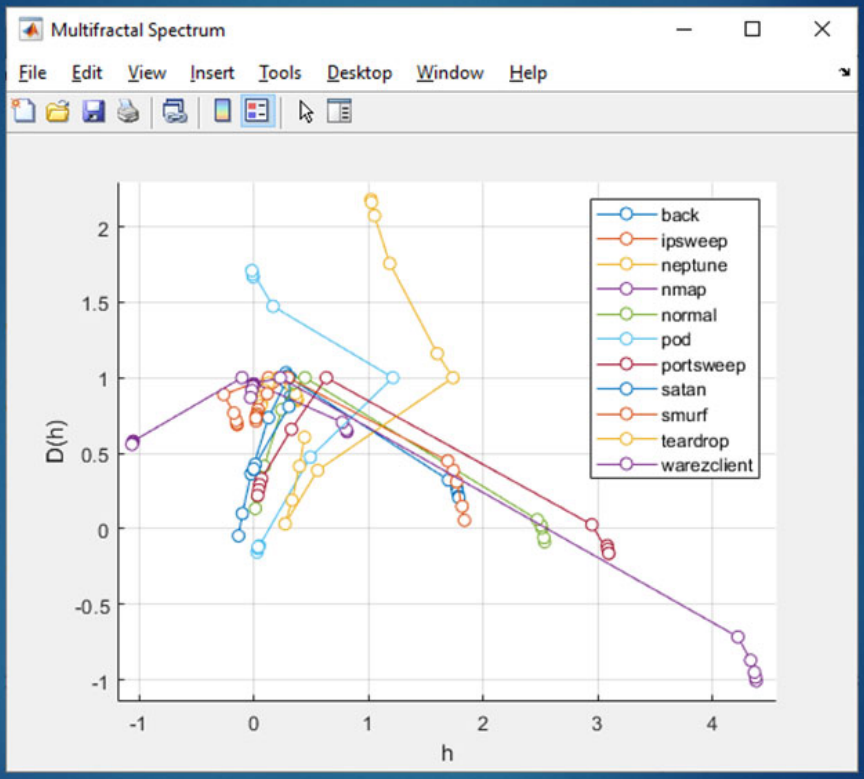
Мультифрактальний аналіз

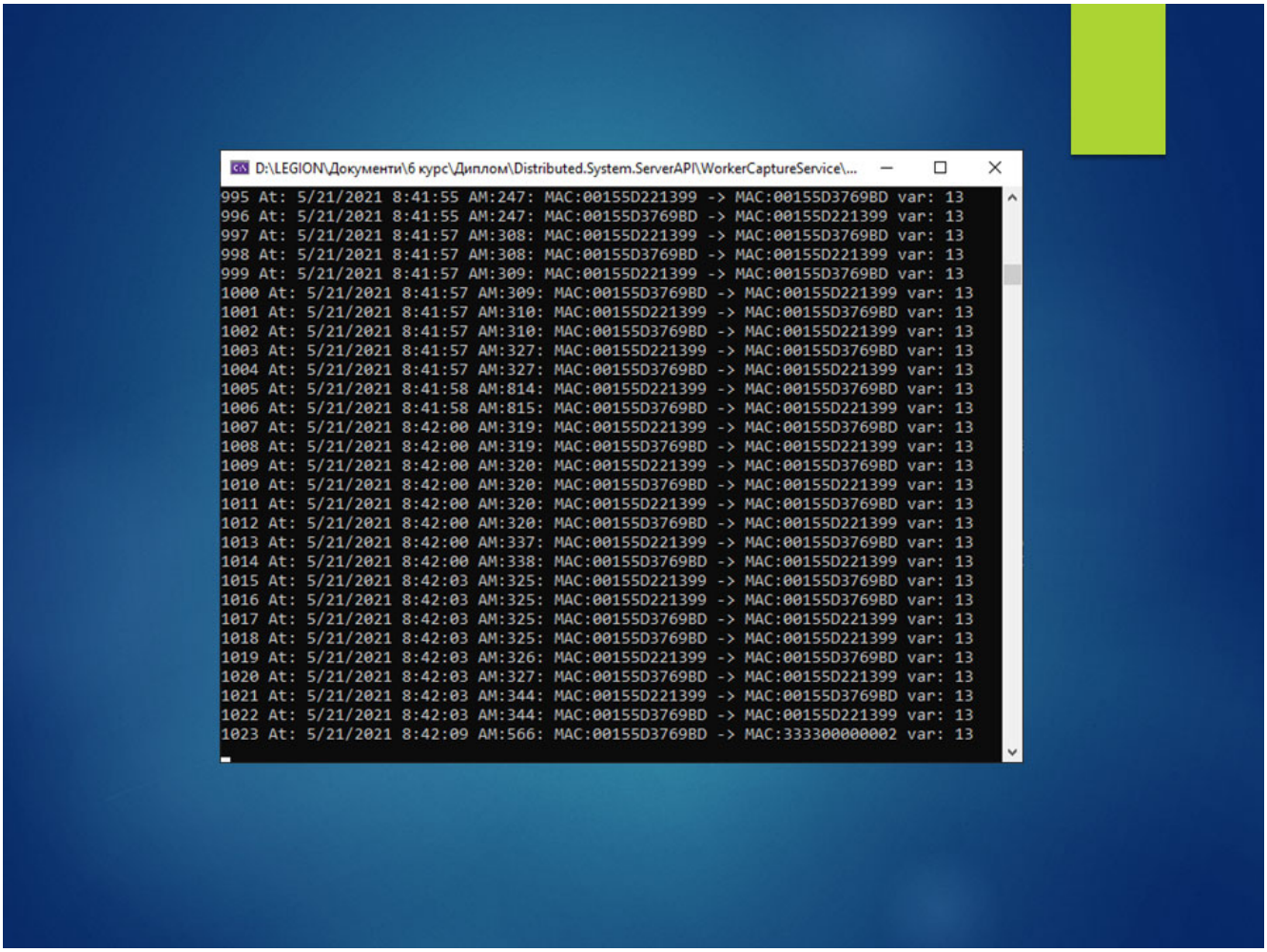
- Мультифракталами називають складні фрактали, що зустрічаються, як правило, в природі. Фактично мультифрактальний підхід означає, що деякий досліджуваний об'єкт можна розділити на частини, що мають власні характеристики подібності, відмінні від інших. Мережевий трафік є самоподібним на деяких часових проміжках. Тому для його аналізу буде використано метод максимумів модулів вейвлет-перетворення, що дозволяє визначити особливості сигналу. Вейвлет аналіз полягає в побудові коефіцієнтів, що використовуються при розподіленні вихідного сигналу на базисні функції. В якості сигналу може виступати інтенсивність мережевого трафіку чи дані кореляції кінцевих IP-адрес. Вейвлет-перетворення дозволяє перетворити найбільш вагомі дані в сигнал, що відповідає вказаній амплітуді коливання та відкинути менш корисну інформацію з малою амплітудою, класифікуючи її як шум.


Нові дані мережевого трафіку
Оновлення трафіку мережі
Оновлення вейвлет-коефіцієнтів
Оновлення статистики
Визначення крайньої точки в кожному масштабі
Крайні точки існують в багатьох масштабах одночасно?
Ні
Так
Оцінка показника Херста і швидкості його зміни
Адаптивні правила прийняття рішень
Визначення інтенсивності атаки

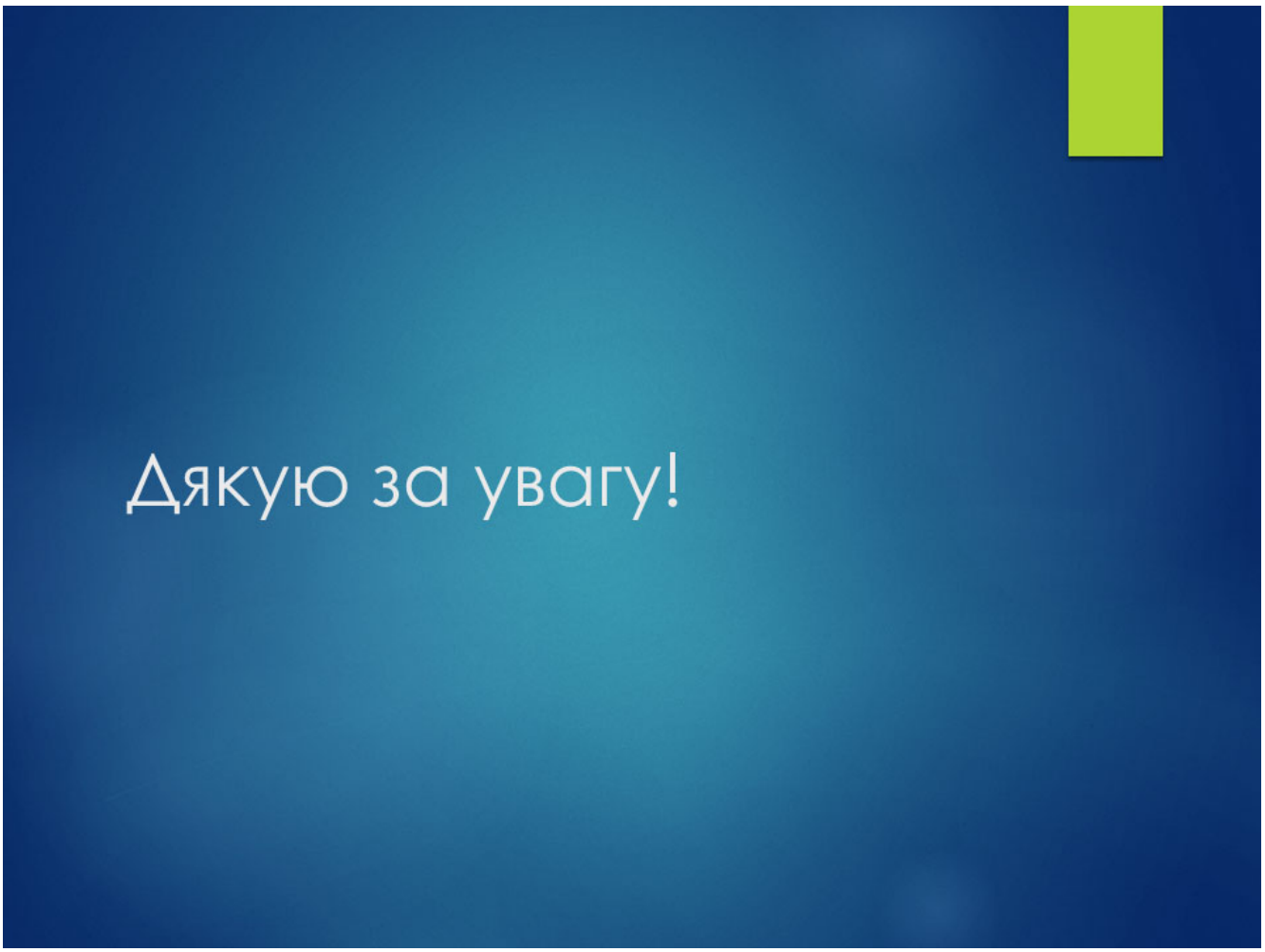

Початок
Збір мережевої інформації
Оригінальний трафік
Статистичний аналіз TCP, UDP, ARP
Індикатори трафіку
Оцінка показника Херста
Оновлення моделі
Показник Херста
Визначення аномалій
Еталонне порівняння
Модель нормального трафіку
Результати визначення аномалії
Оцінка безпеки
Кінець


Traffic type is normal.
File Edit View Insert Tools Desktop Window Help
packets
Time
×10^5
Traffic type is neptune.
File Edit View Insert Tools Desktop Window Help
packets
Time
×10^5


Multifractal Spectrum
File Edit View Insert Tools Desktop Window Help
back
ipsweep
neptune
nmap
normal
pod
portsweep
satan
smurf
teardrop
warezclient
D(h)
h
2
1.5
1
0.5
0
-0.5
-1
-1
0
1
2
3
4


D:\LEGION\Документи\6 курс\Диплом\Distributed.System.ServerAPI\WorkerCaptureService\...
995 At: 5/21/2021 8:41:55 AM:247: MAC:00155D221399 -> MAC:00155D3769BD var: 13
996 At: 5/21/2021 8:41:55 AM:247: MAC:00155D3769BD -> MAC:00155D221399 var: 13
997 At: 5/21/2021 8:41:57 AM:308: MAC:00155D221399 -> MAC:00155D3769BD var: 13
998 At: 5/21/2021 8:41:57 AM:308: MAC:00155D3769BD -> MAC:00155D221399 var: 13
999 At: 5/21/2021 8:41:57 AM:309: MAC:00155D221399 -> MAC:00155D3769BD var: 13
1000 At: 5/21/2021 8:41:57 AM:309: MAC:00155D3769BD -> MAC:00155D221399 var: 13
1001 At: 5/21/2021 8:41:57 AM:310: MAC:00155D221399 -> MAC:00155D3769BD var: 13
1002 At: 5/21/2021 8:41:57 AM:310: MAC:00155D3769BD -> MAC:00155D221399 var: 13
1003 At: 5/21/2021 8:41:57 AM:327: MAC:00155D221399 -> MAC:00155D3769BD var: 13
1004 At: 5/21/2021 8:41:57 AM:327: MAC:00155D3769BD -> MAC:00155D221399 var: 13
1005 At: 5/21/2021 8:41:58 AM:814: MAC:00155D221399 -> MAC:00155D3769BD var: 13
1006 At: 5/21/2021 8:41:58 AM:815: MAC:00155D3769BD -> MAC:00155D221399 var: 13
1007 At: 5/21/2021 8:42:00 AM:319: MAC:00155D221399 -> MAC:00155D3769BD var: 13
1008 At: 5/21/2021 8:42:00 AM:319: MAC:00155D3769BD -> MAC:00155D221399 var: 13
1009 At: 5/21/2021 8:42:00 AM:320: MAC:00155D221399 -> MAC:00155D3769BD var: 13
1010 At: 5/21/2021 8:42:00 AM:320: MAC:00155D3769BD -> MAC:00155D221399 var: 13
1011 At: 5/21/2021 8:42:00 AM:320: MAC:00155D221399 -> MAC:00155D3769BD var: 13
1012 At: 5/21/2021 8:42:00 AM:320: MAC:00155D3769BD -> MAC:00155D221399 var: 13
1013 At: 5/21/2021 8:42:00 AM:337: MAC:00155D221399 -> MAC:00155D3769BD var: 13
1014 At: 5/21/2021 8:42:00 AM:338: MAC:00155D3769BD -> MAC:00155D221399 var: 13
1015 At: 5/21/2021 8:42:03 AM:325: MAC:00155D221399 -> MAC:00155D3769BD var: 13
1016 At: 5/21/2021 8:42:03 AM:325: MAC:00155D221399 -> MAC:00155D3769BD var: 13
1017 At: 5/21/2021 8:42:03 AM:325: MAC:00155D3769BD -> MAC:00155D221399 var: 13
1018 At: 5/21/2021 8:42:03 AM:325: MAC:00155D3769BD -> MAC:00155D221399 var: 13
1019 At: 5/21/2021 8:42:03 AM:326: MAC:00155D221399 -> MAC:00155D3769BD var: 13
1020 At: 5/21/2021 8:42:03 AM:327: MAC:00155D3769BD -> MAC:00155D221399 var: 13
1021 At: 5/21/2021 8:42:03 AM:344: MAC:00155D221399 -> MAC:00155D3769BD var: 13
1022 At: 5/21/2021 8:42:03 AM:344: MAC:00155D3769BD -> MAC:00155D221399 var: 13
1023 At: 5/21/2021 8:42:09 AM:566: MAC:00155D3769BD -> MAC:333300000002 var: 13

Id	DateTime	DateTimeMiliseconds	DestinationHardwareAddress	SourceHardwareAddress	PacketType	ClientFK	ClientEntityId
2084	2021-05-21 08:41:57.3089430 +00:00	308	00155D221399	00155D3769BD	IPv4	0	13
2085	2021-05-21 08:41:57.3094060 +00:00	309	00155D3769BD	00155D221399	IPv4	0	13
2086	2021-05-21 08:41:57.3094830 +00:00	309	00155D221399	00155D3769BD	IPv4	0	13
2087	2021-05-21 08:41:57.3102680 +00:00	310	00155D3769BD	00155D221399	IPv4	0	13
2088	2021-05-21 08:41:57.3103830 +00:00	310	00155D221399	00155D3769BD	IPv4	0	13
2089	2021-05-21 08:41:57.3274730 +00:00	327	00155D3769BD	00155D221399	IPv4	0	13
2090	2021-05-21 08:41:57.3276130 +00:00	327	00155D221399	00155D3769BD	IPv4	0	13
2091	2021-05-21 08:41:58.8146470 +00:00	814	00155D3769BD	00155D221399	Arp	0	13
2092	2021-05-21 08:41:58.8151470 +00:00	815	00155D221399	00155D3769BD	Arp	0	13
2093	2021-05-21 08:42:00.3190200 +00:00	319	00155D3769BD	00155D221399	IPv4	0	13
2094	2021-05-21 08:42:00.3191700 +00:00	319	00155D221399	00155D3769BD	IPv4	0	13
2095	2021-05-21 08:42:00.3200490 +00:00	320	00155D3769BD	00155D221399	IPv4	0	13
2096	2021-05-21 08:42:00.3201390 +00:00	320	00155D221399	00155D3769BD	IPv4	0	13
2097	2021-05-21 08:42:00.3208970 +00:00	320	00155D3769BD	00155D221399	IPv4	0	13
2098	2021-05-21 08:42:00.3209840 +00:00	320	00155D221399	00155D3769BD	IPv4	0	13
2099	2021-05-21 08:42:00.3379560 +00:00	337	00155D3769BD	00155D221399	IPv4	0	13


Дякую за увагу!

UNICHECK

Ім'я користувача:
Кафедра КІ

ID перевірки:
1008054209

Дата перевірки:
27.05.2021 20:53:06 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
27.05.2021 20:53:48 EEST

ID користувача:
100005591

Назва документа: **Централізована розподілена система виявлення атак в корпоративних комп'ютерних мере...**

Кількість сторінок: **79** Кількість слів: **12357** Кількість символів: **98866** Розмір файлу: **1.96 MB** ID файлу: **1008141803**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.73%
Схожість

Найбільша схожість: **3.12%** з джерелом з Бібліотеки (ID файлу: **1007657363**)

6.24% Джерела з Інтернету	571	Сторінка 81
3.67% Джерела з Бібліотеки	113	Сторінка 85

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1
Підозріле форматування	17 сторінок

Thu May 27 20:07:33 EEST 2021, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 3.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 12%**

	Документ		Суммарное совпадение по Базе Данных	
ID: 91497 Название: Централізована розподілена система виявлення атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу Добавлено в БД: 2021-05-27 Авторы: Шагін В.Ю. Руководители: Савенко О.С. Консультанты: Опоненты:	Символы	Лексемы	Символы	Лексемы
	81880	756	3451 (4%)	38 (5%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

Дипломник: Шагін В.Ю

Тема: Централізована розподілена система виявлення атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг дипломної роботи:

Кількість листів креслень _0_; кількість сторінок записки 140

1. Короткий зміст роботи та прийнятих рішень

У магістерській роботі здійснено аналіз існуючих методів та засобів для виявлення мережевих атак в корпоративних комп'ютерних мережах; виявлено переваги та недоліки основних методів визначення; обґрунтований вдосконалений метод виявлення мережевих атак; обґрунтовані алгоритми та технологія реалізації централізованої розподіленої системи; створена комп'ютерна система з можливістю автоматичного опрацювання мережевого трафіку та зміни структури за потреби.

2. Висновок про відповідність роботи дипломному завданню

Магістерська робота виконана у відповідності до дипломного завдання.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи:

У першому розділі було проведено детальний аналіз предметної області, було розглянуто розподілені системи, їх переваги та недоліки, .особливості і вимоги при розробці, використання методів виявлення мережевих атак, згідно проведеного аналізу було сформовано актуальність роботи і визначено вимоги для створюваної системи. У другому розділі згідно досліджених джерел було розроблено архітектуру розподіленої системи, її апаратну та програмну складову, детально розглянуто всі компоненти системи і їх взаємодія, а також крім удосконалення її архітектури розроблено метод підтримки цілісності та відмовостійкості. У третьому розділі було вдосконалено метод виявлення мережевих атак в комп'ютерних мережах згідно методу мультифрактального аналізу та його використання у

розподілених мережах для виявлення мережевих атак. У четвертому розділі на основі проектування програмного забезпечення була створена централізована розподілена система виявлення мережевих атак на основі мультифрактального аналізу згідно запропонованих рішень щодо архітектури методу підтримки цілісності та відмовостійкості, описана розроблена розподілена система, наведено результати експериментальних досліджень та проведено порівняння ефективності з відомими рішеннями.

4. Позитивні сторони роботи:

До позитивних сторін роботи слід віднести актуальність напрямку дослідження, отримані наукові і практичні результати з предметної області з комп'ютерної інженерії, реалізацію запропонованих рішень та експериментальні дослідження з розробленою системою.

5. Негативні сторони роботи:

Недостатньо деталізовано представлення розподіленої системи

6. Оцінка графічного оформлення та пояснювальної записки роботи:

Пояснювальна записка та графічні матеріали оформлені у відповідності до вимог, що висуваються до такого роду документів.

7. Відгук про роботу в цілому:

Зміст представленої роботи в повній мірі розкриває обрану тему. Дослідження, проведені в матеріалах є достатньо аргументованими.

8. Інші зауваження: _____

_____

_____

9. Оцінка дипломної роботи:

Робота заслуговує оцінки «відмінно», а її автор – присвоєння кваліфікації «магістра з комп'ютерної інженерії»

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи)
Клюб Ю.П. Зав. кафедри КБКСМ, К.Т.Н, доцент

"28" 05 2020 р. _____(підпис)

Завідувачу кафедри КІСП
д-р.техн.наук, проф. Говорущенко Т. О.

Шагіна В'ячеслава Юрійовича

ПІБ здобувача вищої освіти

ФПКТС, 2 курсу, групи КІ2м-19-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіатоповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

25.05.2021

дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА СИСТЕМНОГО ПРОГРАМУВАННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Централізована розподілена система виявлення атак в корпоративних комп'ютерних мережах на основі мультифрактального аналізу»

Автор: Шагін В'ячеслав Юрійович

Спеціальність: 123 – Компютерна інженерія та програмування

Освітня програма: освітньо-наукова

Науковий керівник: Березький Олег Миколайович, д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) В тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень в бланках (титулка, бланк завдання, в структурі підрозділів ВСТУПУ) та в назвах публікацій джерел посилання;
2) в якості запозичень в окремих місцях системою зафіксовано послідовності вихідного коду, які є спільними для великої кількості задач і не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.
4) Збігів та ідентичності в тексті кваліфікаційної роботи немає, наявна лише схожість.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ідентичності/схожості, складає 0,61% і адресується до 571 першоджерела, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ О. М. Березький

Гарант ОП _____ О. С. Савенко

Завідувач кафедри КІСП _____ Т. О. Говорущенко