Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“___” _____ 20__ р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація інтелектуальної системи для виявлення DDoS-атак”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-22М-2
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Якименко О.А.
«____» _____ 20 р.

Керівник проекту
доктор технічних наук, професор
_____ Єлизавета МЕЛЕШКО
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань *12 "Інформаційні технології"*

Спеціальність *123 "Комп'ютерна інженерія"*

Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
« __ » _____ 20__ року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Якименку Олександру Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація інтелектуальної системи для виявлення DDoS-атак*

2. Керівник роботи *Мелешко Єлизавета Владиславівна, доктор техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу №__-__ від __.__.20__ року

3. Строк подання роботи до захисту __.__.*2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи: *Дослідження та програмна реалізація інтелектуальної системи для виявлення DDoS-атак*

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

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Наукова новизна
7. Економічна ефективність розробленої програми.
8. Заходи з охорони праці та техніки безпеки.
9. Висновки.

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

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Діаграма процесів	*1 аркуш*
Показники економічної ефективності	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В., к.т.н., доцент	25.10.2023	10.11.2023
Охорона праці	Оришака О.В., к.т.н., доцент	03.11.2023	21.11.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2023 р.	
2.	Постановка задачі, оформлення ТЗ	16.10.2023 р.	
3.	Розробка моделі компонента	20.10.2023 р.	
4.	Розробка структур даних	25.10.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2023 р.	
6.	Програмування алгоритмів	10.11.2023 р.	
7.	Розрахунок економічної ефективності	13.11.2023 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2023 р.	
9.	Оформлення ПЗ	17.11.2023 р.	
10.	Попередній захист роботи	10.12.2023 р.	

Дата видачі завдання
«__»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Якименко О.А. Дослідження та програмна реалізація інтелектуальної системи для виявлення DDoS-атак. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній магістерській роботі розроблено програмне забезпечення, яке призначено для реалізації інтелектуальної системи для виявлення DDoS-атак.

Метою розробки є дослідження та програмна реалізація інтелектуальної системи для виявлення DDoS-атак.

Об'єктом дослідження є процес виявлення DDoS-атак.

Предметом дослідження є методи аналізу часових рядів трафіку для виявлення інформаційних атак.

Методи дослідження базуються на теорії комп'ютерних мереж, теорії аналізу часових рядів, теорії математичної статистики, теорії алгоритмів і структур даних та методах розробки програмного забезпечення.

Результат роботи – програмна реалізація інтелектуальної системи для виявлення DDoS-атак.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено на мові програмування високого рівня Python.

Ключові слова: комп'ютерна інженерія, комп'ютерні мережі, кібербезпека, інформаційні атаки, DDoS-атаки.

ABSTRACT

Yakymenko O.A. Research and software implementation of an intelligent system for detecting DDoS attacks. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi 2023.

In this master's thesis, software designed for a simulation system of computer network and data transfer processes.

The purpose of the development is the research and program implementation of a simulation system for computer network and data transfer processes.

The object of the research is the process of detecting DDoS-attacks.

The subject of the research is the methods of analyzing time series of traffic to detect information attacks.

The research methods are based on the theory of computer networks, the theory of time series analysis, the theory of mathematical statistics, the theory of algorithms and data structures, and software development methods.

The result of the work is the software implementation of a simulation system for computer network and data transfer processes.

In the process of working on a software model an analysis of existing hardware and software was performed. All components of the software developed are fully described.

User-friendly user interface is developed. These are instructions for working with software.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the high-level programming language – Python.

Keywords: computer engineering, computer networks, cybersecurity, information attacks, DDoS-attacks.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

DDoS-атака (Distributed Denial of Service attack) – це вид кібератаки, в якому зловмисники намагаються перевантажити цільовий сервер або комп'ютерну мережу, надсилаючи велику кількість запитів з багатьох джерел одночасно, з метою призупинити нормальну роботу цільового ресурсу.

FTP (File Transfer Protocol) – це протокол для передачі файлів між комп'ютерами в мережі.

HTTP (Hypertext Transfer Protocol) – це протокол, використовуваний для передачі веб-сторінок та інших ресурсів в Інтернеті, що визначає спосіб, яким веб-браузери та веб-сервери взаємодіють між собою.

ICMP (Internet Control Message Protocol) – це мережевий протокол, що входить у стек протоколів TCP/IP та в основному використовується для передачі повідомлень про помилки та інші виняткові ситуації, що виникли при передачі даних.

ISP (Internet Service Provider) – це постачальник послуг Інтернету, який надає доступ до мережі Інтернет та інші Інтернет-послуги своїм клієнтам.

NetFlow – це інструментарій для моніторингу та аналізу даних мережевого трафіку, розроблений компанією Cisco.

R/S-аналіз – це метод аналізу часових рядів, який використовується для визначення ступеня самоподібності або фрактальної структури в даних.

SMTP (Simple Mail Transfer Protocol) – це протокол, використовуваний для надсилання та приймання електронної пошти в мережі Інтернет.

TCP (Transmission Control Protocol) – це протокол на транспортному рівні мережевої моделі OSI, який забезпечує надійну і послідовну доставку даних в мережі. Він забезпечує контроль над з'єднаннями та управлінням потоком даних.

UDP (User Datagram Protocol) – це протокол на транспортному рівні мережевої моделі OSI, який використовується для передачі даних в мережі без гарантії доставки та в порядку, в якому дані були відправлені.

XML (Extensible Markup Language) – це мова розмітки, яка використовується для представлення та обміну даними в текстовому форматі, вона дозволяє структурувати дані за допомогою тегів.

Індекс Херста – використовується для визначення структури часового ряду трафіку. Він знаходиться в межах від 0 до 1 і вказує на ступінь самоподібності в часовому ряді. Значення близьке до 0,5 вказує на випадковий розподіл, а значення близьке до 1 вказує на детермінований розподіл.

Інтенсивність трафіку – вимірює кількість подій, які відбуваються (наприклад, кількість переданих пакетів даних), протягом певного інтервалу часу. Зазвичай вимірюється в одиницях на одиницю часу (наприклад, байти на секунду або пакети на хвилину).

ОС – операційна система.

ПЗ – програмне забезпечення.

Часовий ряд трафіку – послідовність вимірів інтенсивності (або кількості подій) відносно часу, представляє динаміку трафіку в мережі протягом певного періоду.

ВСТУП

Актуальність теми. Виявлення DDoS-атак, розподілених атак з відмовою в обслуговуванні, в інформаційному суспільстві сьогодні важлива більше, ніж будь-коли. DDoS-атаки залишаються однією з найпоширеніших та найпагубніших загроз для онлайн-систем, бізнесу та інтернет-інфраструктури.

Зростання обсягів Інтернет-трафіку, поширення хмарних та розподілених систем, а також розвиток Інтернету речей призводять до збільшення різноманітних інформаційних атак. DDoS-атаки стають більш складними, динамічними та завдають значної шкоди бізнесу та організаціям, включаючи втрату прибутку, порушення доступності послуг, підриву репутації та безпеки.

Створення інтелектуальної системи для виявлення DDoS-атак є важливим завданням у боротьбі з цими загрозами. Традиційні методи виявлення DDoS атак можуть бути недостатньо ефективними через постійну модифікацію атак та великі масштаби зловмисних дій. Розвиток штучного інтелекту, машинного навчання та аналізу даних відкриває нові можливості для розробки інтелектуальних систем, які можуть вчасно виявляти та реагувати на DDoS-атаки. Такі системи здатні аналізувати великі обсяги мережевого трафіку в реальному часі, ідентифікувати аномальний трафік та виявляти атаки, які можуть залишитися непоміченими традиційними засобами захисту.

Отже, створення інтелектуальних систем для виявлення DDoS-атак має велике значення для безпеки та стабільності сучасного Інтернет-середовища та є актуальним завданням в інформаційній безпеці та кіберзахисті.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація інтелектуальної системи для виявлення DDoS-атак.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Дослідження існуючих систем для виявлення DDoS-атак.

– Розробка методів та алгоритмів для інтелектуальної системи виявлення DDoS-атак.

– Програмна реалізація інтелектуальної системи виявлення DDoS-атак.

Об'єктом дослідження є процес виявлення DDoS-атак.

Предметом дослідження є методи аналізу часових рядів трафіку для виявлення інформаційних атак.

Методи дослідження базуються на теорії комп'ютерних мереж, теорії аналізу часових рядів, теорії математичної статистики, теорії алгоритмів і структур даних та методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Запропоновано метод виявлення DDoS-атак на основі R/S-аналізу часових рядів мережевого трафіку.

2. Розроблено вітчизняний продукт виявлення DDoS-атак, який має більш широкі можливості, на відміну від існуючих аналогів та може бути використаний при розробці методів захисту веб-сайтів від інформаційних атак.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі виявлення DDoS-атак на веб-сайти.

Достовірність наукових результатів підтверджена теоретичними викладками, результатами імітаційного моделювання та результатами тестування розробленого програмного забезпечення.

Результати досліджень, які включені до кваліфікаційної магістерської роботи оприлюднено на 7-мій Міжнародній науково-практичній конференції "Інформаційна безпека та комп'ютерні технології", 1 листопада 2023 року, м. Кропивницький.

Тож, виходячи з вищеперерахованого, дослідження та програмна реалізація інтелектуальної системи для виявлення DDoS-атак, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній магістерській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Інтелектуальна система для виявлення DDoS-атак має на меті захищати комп'ютерні мережі та веб-додатки від атак, спрямованих на перевантаження ресурсів та блокування доступу користувачів до веб-систем.

Основні функції такої системи:

1. **Виявлення атак**. Головною функцією інтелектуальної системи для виявлення DDoS-атак є виявлення надмірного трафіку або аномалій у мережі, що можуть свідчити про атаку на відмову в обслуговуванні. Це може включати аналіз заголовків пакетів, обсягу трафіку, швидкості обробки запитів користувачів та інших параметрів.

2. **Реакція на атаки**. Після виявлення атаки інтелектуальна система може приймати різні заходи для зменшення впливу атаки. Це може включати блокування IP-адрес, які генерують атаку, або перенаправлення трафіку для відвідувачів через системи фільтрації.

3. **Моніторинг та аналіз**. Система проводить моніторинг мережі та трафіку в режимі реального часу і збирає дані для подальшого аналізу. Це допомагає виявляти нові атаки та адаптуватися до них.

4. **Сбір статистики**. Інтелектуальна система може збирати базу даних інформаційних атак, що дозволить у подальшому на основі зібраної статистики вдосконалювати існуючі методи виявлення атак.

5. **Оптимізація роботи комп’ютерної мережі**. Інтелектуальні системи можуть допомагати оптимізувати роботу мережі та використовувати ресурси більш ефективно, навіть під час атак та аномальних пікових навантаженнях.

6. **Логування та аудит**. Система може здійснювати логування подій і проводити аудит атак та заходів, прийнятих для їх припинення.

Інтелектуальні системи для виявлення DDoS-атак є надзвичайно важливими для забезпечення безпеки комп’ютерних мереж та веб-додатків, оскільки DDoS-атаки можуть призвести до довгих перебоїв в роботі веб-систем та втрати даних. Вони допомагають виявляти, запобігати та відновлювати веб-систему від таких атак, забезпечуючи надійну роботу інфраструктури.

1.2 Область застосування

Інтелектуальні системи для виявлення DDoS-атак знаходять широке застосування в різних галузях та сферах діяльності, де важлива надійність та доступність мережевих служб. Ось деякі області застосування цих систем:

1. **Інтернет-постачальники (ISP)**. ISP використовують інтелектуальні системи для виявлення DDoS-атак для захисту своїх мереж та клієнтів від масштабних атак, які можуть призвести до відключення Інтернет-послуг для багатьох користувачів.

2. **Електронна комерція**. Онлайн-торгові платформи та веб-магазини використовують інтелектуальні системи для виявлення DDoS-атак, оскільки вони можуть стати об'єктом атак у піковий час продажів або акцій.

3. **Фінансові установи**. Банки, фінансові компанії та платіжні системи застосовують захист від DDoS-атак для забезпечення безпеки фінансових транзакцій та обслуговування клієнтів у режимі 24/7.

4. **Організації з великими даними**. Компанії, які обробляють великі обсяги даних, такі як віртуальні соціальні мережі та хмарні платформи, використовують інтелектуальні системи виявлення DDoS-атак для захисту даних та послуг користувачів.

5. **Медіа та розваги**. Медіа-компанії та провайдери контенту можуть бути об'єктом DDoS-атак, оскільки доступність вмісту для глядачів є критичною. Інтелектуальні системи виявлення DDoS-атак допомагають забезпечити стійкість до атак, які можуть перервати трансляцію.

6. **Організації з обмеженим бюджетом**. Навіть невеликі компанії та організації можуть стати жертвами DDoS-атак. Вони можуть використовувати системи виявлення DDoS-атак для захисту своєї онлайн-присутності.

7. **Громадські ініціативи та активісти**. Інтернет-активісти та громадські організації, які працюють у сфері прав людини, можуть стати об'єктом атак через свою діяльність. Вони також можуть скористатися інтелектуальними системами для виявлення DDoS-атак.

8. **Державні органи**. Уряди та державні установи застосовують системи виявлення від DDoS-атак для захисту своїх веб-ресурсів та баз даних, які можуть містити важливі державні дані.

Інтелектуальні системи для виявлення DDoS-атак є критичними для забезпечення безпеки та доступності в мережах та онлайн-сервісах, і вони застосовуються в різних галузях з метою захисту від атак і забезпечення надійності інфраструктури.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми магістерської роботи

DDoS-атака – це цілеспрямована спроба порушити роботу веб-ресурсу. Хакери перевантажують системні ресурси жертви шкідливим трафіком, у результаті сайт завантажується повільно або зовсім перестає працювати.

DDoS-епідемія є серйозною проблемою для компаній у всьому світі. Хакери атакують і великі проекти, і маловідомі сайти. Причини різні: від випадковості, коли зловмисники тренують свої сили перед масштабним нападом до цілеспрямованого шкідництва з метою вивести з ладу конкретний веб-ресурс. Від DDoS ніхто не застрахований, проте, виконуючи низку рекомендацій, можна раніше виявити атаку та пом'якшити її наслідки.

Причини DDoS-атак:

– Конкуренція. Мабуть, саме це є головним приводом для реалізації хакерської атаки. Недобросовісні компанії-конкуренти вдаються до способу ddos атаки на сервери для усунення основних претендентів на лідируючі позиції.

– Політичне замовлення. Активісти, тобто опоненти, використовують DDoS для кібер-атак у разі сильних розбіжностей. Це своєрідний радикальний протест.

– Особиста неприязнь. Через особисті переконання та якісь ситуації зловмисники атакують великі компанії, у тому числі й урядові.

– Бажання збагатитися. Бувають ситуації, коли зломщики зв'язуються з представниками, а то й власниками великих корпорацій, щоб запросити викуп. В іншому випадку вони загрожують серйозним втручанням у функціонування веб-ресурсів у вигляді злому системи.

– Експеримент. Трапляється і таке, що новачки-хакери хочуть перевірити

свої можливості та на практиці випробувати набуті навички і тому вони здійснюють напад.

Загальна схема DDoS-атаки:

1. Зловмисник використовує ботнет, щоб відправити шкідливий трафік на сайт-жертву.

2. Ботнет генерує безліч запитів – від кількох тисяч до мільйонів.

3. Веб-сервер не витримує такого навантаження і виходить з ладу.

Ілюстрація цієї загальної схеми DDoS-атаки наведена на рис. 2.1.

Існує безліч типів DDoS-атак, вони різняться за потужністю та принципом впливу, але важливо пам'ятати, що будь-який з них може стати критичною загрозою веб-ресурсу.

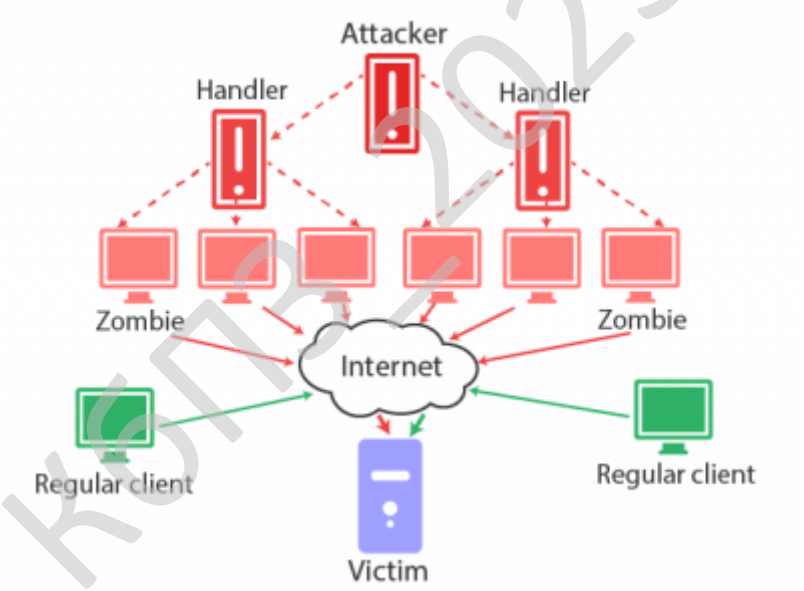

Рисунок 2.1 – Приклад схеми DDoS-атаки

Вивчаючи тему кібер-атак, значний акцент слід приділити їх видам. Ця інформація допоможе краще розібратися у питанні як захистити свій сайт від Ddos-атак та визначитися з алгоритмом подальших дій.

Класифікація DDOS-атак:

– протокольні;

– прикладні;

– атаки на програми.

Розглянемо всі види та методи ddos атак детальніше.

Протокольні. DDoS-атака націлена на мережевий рівень. Основна мета – спровокувати перезавантаження табличного простору на екрані з брандмауером у мережі. Її ще називають атакою транспортного рівня. Мережевий флуд прийнято вважати найпоширенішим методом цього виду. На різних рівнях запускається безліч запитів, з якими вузол не впорається. Звичайно, діє правило FIFO, коли обробка наступних запитів не починається, поки не закінчиться обробка першого. Але при кібернападі кількість запитів настільки зростає, що пристрою бракує ресурсів для того, щоб завершити роботу з вихідним запитом. Види Ddos-атак мережевого флуду:

– HTTP-флуд. Вузли забиваються величезною кількістю HTTP-повідомлень. Хост-машина перевантажується службовими запитами.

– SYN-флуд. Вплив виконується на базовий протокол передачі TCP.

– UDP флуд. Порти забиваються пакетами за протоколом UDP, через що перевантажується мережа.

– MAC-флуд. Порти мережного обладнання завалюються потоком пакетів із різними MAC-адресами.

Прикладні. Атаки на рівні інфраструктури, що застосовуються для того щоб вивести з робочого процесу апаратні ресурси та технології, процесор доводиться до перевантаження. Види:

– Сервер заповнюється лог-файлами за рахунок скрипта. Атака спрацює, якщо на сервері не встановлений ліміт.

– Відправляє габаритні пакети, з якими не справляється процесор.

– Система квотування. Якщо хакер має доступ до CGI, може написати скрипт для використання частини ресурсів.

– Атака 2-го роду. Помилкова сигналізація, що провокує закриття ресурсу.

Атаки на рівні додатків. Ця атака використовує недогляди у розробці

програмного коду, роблячи ПЗ вразливим. Сюди належить Ping of death. Але для атаки великих компаній, де досить складні системи, хакери пишуть експлойт-програму для виявлення вразливості та подальшої атаки.

Ознаки DDoS-атаки:

- збільшилося навантаження на мережу;
- збільшився обсяг трафіку на порти з'єднань;
- сайт повільно працює або видає помилки 502, 503, 504;
- різко зростає навантаження на процесор та оперативну пам'ять;
- збільшується кількість запитів до баз даних або до інших внутрішніх послуг;
- з'являються багаторазові звернення користувачів до тих самих файлів або сторінок сайту;
- звернення користувачів не відповідають тематиці веб-ресурсу.

Перераховані вище пункти не є стовідсотковим підтвердженням DDoS-атаки. Однак вони звернуть увагу власника сайту на проблему, щоб той, у свою чергу, встиг вжити своєчасних заходів щодо захисту веб-ресурсу.

Методи виявлення DDoS-атак

Один із ключових принципів боротьби з кібератаками – моніторинг трафіку. Регулярне спостереження та аналітика допоможе своєчасно виявити аномалії та зробити вирішальні кроки для захисту від шкідливої активності. Далі докладніше розглянемо кілька методів виявлення кібератак із описом функціоналу кожного. Основні методи виявлення DDoS-атак:

1. Систематичний аналіз трафіку веб-ресурсу. Аналіз можна виконувати двома способами: самостійно, за наявності технічних знань, або підключити автоматичні системи, такі як брандмауер. Міжмережевий екран здійснюватиме контроль та фільтрацію трафіку. Журнали міжмережевого екрану дозволяють виявити нетипові сплески трафіку та з'ясувати, чи йде атака на веб-ресурс.

2. Моніторинг часу відгуку. На ранніх етапах атаку досить складно виявити, оскільки уповільнення сайту настає ледь помітно. Тому рекомендовано

регулярно аналізувати стан сайту, щоб з'ясувати, який час є нормальним відгуком. Відхилення від цього показника можуть бути пов'язані з DDoS-атакою. Щоб виявити раннє гальмування веб-ресурсу та запобігти надмірному навантаженню на сервер, знадобиться сервіс з функцією моніторингу часу відгуку. На ринку представлено безліч послуг з цим функціоналом – вибір слід здійснювати, виходячи з завдань та бюджету.

3. Налаштування автоматичного оповіщення про атаку. Після того, як було визначено, який патерн трафіку є нормальним для досліджуваного веб-ресурсу, можна скористатися сторонніми сервісами для оповіщення про аномалії. Оповіщення можуть надходити у вигляді SMS, email, повідомлень у корпоративному месенджері та іншими способами.

4. Комплексний підхід до виявлення DDoS-атак. Найефективніше використовувати всі перелічені вище методи. Різні моделі моніторингу дозволять якомога раніше виявити підозрілу активність, а значить підвищити шанси впоратися з нею. Комбінуйте ручний та автоматичний моніторинг, підключіть систему оповіщення про атаки та постійно аналізуйте трафік веб-ресурсу.

Аналізатори та колектори NetFlow – це дуже корисний інструментарій для моніторингу та аналізу даних мережевого трафіку, який допомагає виявити можливі проблеми ще до того, як вони стануть реальною загрозою. Аналізатори NetFlow дозволять визначити ті машини та пристрої, які негативно впливають на пропускну здатність мережі, знайти вузькі місця у системі, а також зрештою підвищити загальну ефективність функціонування мережі.

Саме поняття NetFlow відноситься до мережевого протоколу, розробленого компанією Cisco для збору інформації про трафік, що проходить через різні пристрої в мережі. Існує кілька версій протоколу NetFlow (від 1 до 9), частина яких вже застаріла. На даний момент найбільш поширеними є NetFlow версії 5, 7 і 9. Зараз NetFlow є фактично промисловим стандартом і підтримується не тільки обладнанням Cisco, але і багатьма іншими пристроями від інших виробників.

Частина вендорів використовують власні версії NetFlow: наприклад,

компанія Juniper Networks називає свій протокол J-Flow, у компанії Huawei – це NetStream. Незважаючи на те, що ці Flow мають різні імена, всі вони працюють аналогічним NetFlow чином, надаючи, в основному, ту саму інформацію. Крім того, на основі NetFlow версії 9 також було розроблено відкритий універсальний стандарт IPFIX (Internet Protocol Flow Information eXport) для передачі інформації про IP-потоки.

Інформація, яку збирає з IP-трафіку NetFlow, включає:

- IP-адреса джерела;
- IP-адреса призначення;
- порт джерела для UDP та TCP;
- порт призначення для UDP та TCP;
- тип та код повідомлення для ICMP;
- номер Інтернет-протоколу транспортного рівня, інкапсульованого у протокол IP;
- тип обслуговування (Type of Service, ToS);
- мережевий інтерфейс.

Збираючи та аналізуючи цю інформацію, можна багато дізнатися про функціонування компєютерної мережі, а також використовувати її для різних цілей, включаючи моніторинг пропускної спроможності, усунення несправностей у роботі мережі та виявлення аномалій.

Коли в мережі реалізовано підтримку протоколу NetFlow, активується робота двох основних компонентів: *Flow Exporter* та *Flow Collector*.

Flow Exporter захоплює статистичну інформацію про потік та відправляє її в колектор. Він зазвичай налаштовується на пристрої, такому як маршрутизатор або комутатор, а в деяких випадках на одному пристрої використовується кілька Flow Exporter для моніторингу різних потоків.

Flow Collector отримує дані про потоки та зберігає їх. У багатьох сучасних рішеннях для аналізу мережного трафіку функціональність колектора та аналізатора поєднана в одному рішенні – статистична інформація про потоки не

тільки збирається та зберігається, але також обробляється та аналізується, щоб подати її користувачам у зручному для розуміння вигляді. У ряді випадків Flow Collector може використовуватися просто для отримання даних, при цьому аналіз здійснюється іншим додатком з функціональністю аналізатора.

Існує безліч програмних рішень на основі протоколу NetFlow, розглянемо 5 найкращих комерційних та безкоштовних аналізаторів та колекторів NetFlow. Більшість постачальників програмного забезпечення NetFlow, наведених нижче, мають інструкції щодо включення NetFlow на пристрої різних виробників. Крім того, ця інформація також повинна міститися в документації виробника Вашого пристрою.

Аналізатор трафіку Solarwinds NetFlow

NetFlow Traffic Analyzer (NTA) виробництва компанії Solarwinds – це програмний інструмент для аналізу мережевого трафіку та моніторингу пропускної спроможності, який підтримує роботу з різними протоколами, включаючи: NetFlow, J-Flow, sFlow, IPFIX та NetStream.

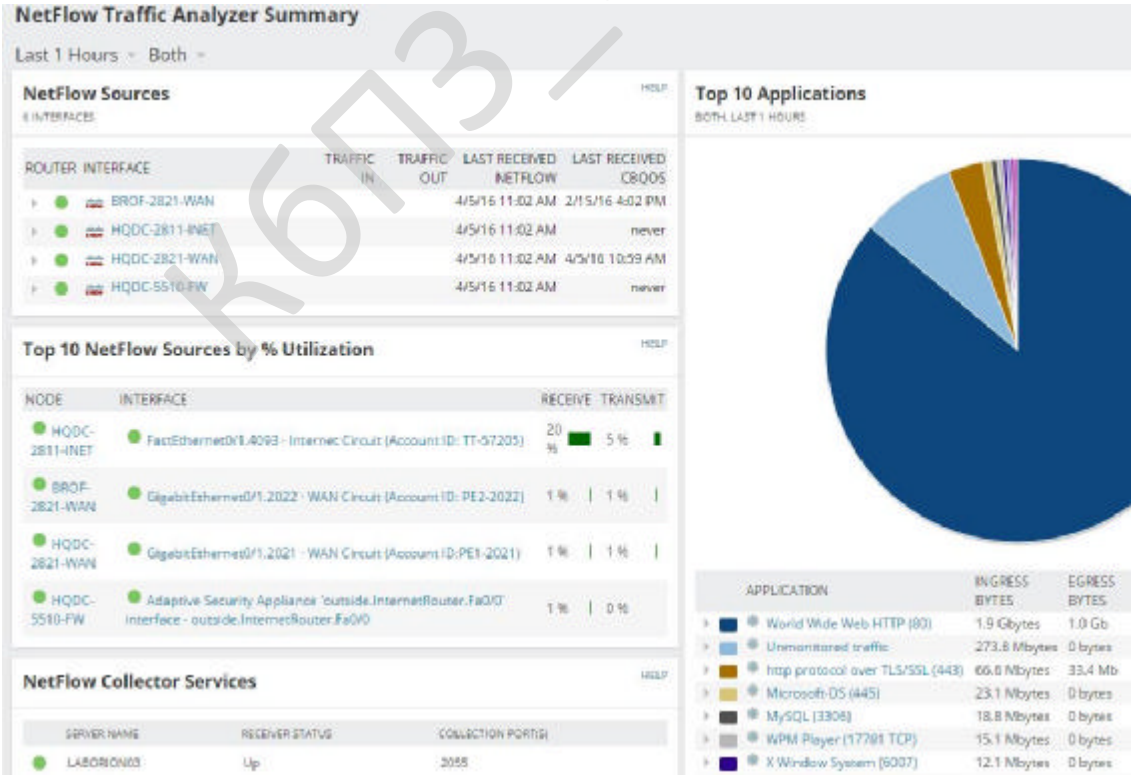

Рисунок 2.2 – Аналізатор трафіку Solarwinds NetFlow

Рішення Solarwinds NTA дозволить зрозуміти, як використовується пропускна спроможність мережі. Наприклад, ви можете дізнатися, яка IP-адреса або програма має максимальні обсяги споживання ресурсів пропускної спроможності в певний момент часу. Також ви зможете використовувати шаблони для аналізу трафіку, тим самим значно розширивши свої можливості щодо детального мережевого аналізу.

NetFlow Traffic Analyzer збирає дані трафіку, погоджує їх з форматами, що використовуються, і демонструє отриману інформацію користувачеві через веб-інтерфейс для моніторингу мережного трафіку. Ви можете аналізувати мережевий трафік за конкретний період: хвилину, годину, день або місяць.

Solarwinds NTA дозволить зрозуміти, чи політики QoS працюють так, як це було заплановано. Тобто, якщо ваш бізнес залежить від VoIP, електронної комерції або інших хмарних програм, моніторинг NetFlow допоможе підтвердити, що пріоритетний трафік проходить безперешкодно через мережу.

Ви можете планувати та проводити докладний аналіз мережевого трафіку, а також отримувати різні часові звіти про використання ресурсів пропускної спроможності за допомогою кількох кліків мишки. Накопичення даних дозволить визначати пікові періоди пропускної спроможності та коригувати політики для більшої керованості процесів у вашій мережі. У ряді випадків це дозволить заощадити на невиправданому підвищенні пропускної спроможності.

Solarwinds NTA – це комерційний продукт, ціна якого залежить від кількості елементів для моніторингу, проте для ознайомлення доступна 30-денна безкоштовна пробна версія. Також важливо те, що Solarwinds NTA повністю інтегрується з Solarwinds Network Performance Monitor (NPM) – рішенням для скорочення збоїв у роботі мережі та підвищення продуктивності.

Існує також спрощена безкоштовна версія NetFlow Traffic Analyzer від компанії SolarWinds – Real-Time NetFlow Analyzer. Цей інструмент дозволяє сортувати дані, представляти їх у вигляді графіків, а також відображати інформацію різними способами, які дозволяють візуалізувати та аналізувати

мережевий трафік. Продукт чудово підійде для таких завдань, як перегляд мережного трафіку в залежності від типу або конкретного часу, а також запуску різних тестів для перевірки того, скільки ресурсів мережі використовується різними програмами. Таким чином, за допомогою безкоштовного рішення Real-Time NetFlow Analyzer можна спростити пошук пристроїв, що обмежують пропускну здатність (в тому числі кінцевих точок), вивчати та усувати неполадки та спади у функціонуванні мережі, а також визначати користувачів, додатки та пристрої, які використовують більше всього ресурсів мережі. Обмеженням безкоштовної версії програми є доступність лише одного інтерфейсу NetFlow для моніторингу та зберігання лише 60 хвилин даних.

Монітор мережі PRTG

PRTG Network Monitor – це універсальне програмне рішення для моніторингу мережі, яке включає моніторинг продуктивності, відстеження обмежень пропускної спроможності, спостереження за роботою серверів, центральних процесорів, додатків, веб-сайтів, хмарних служб та багато іншого.

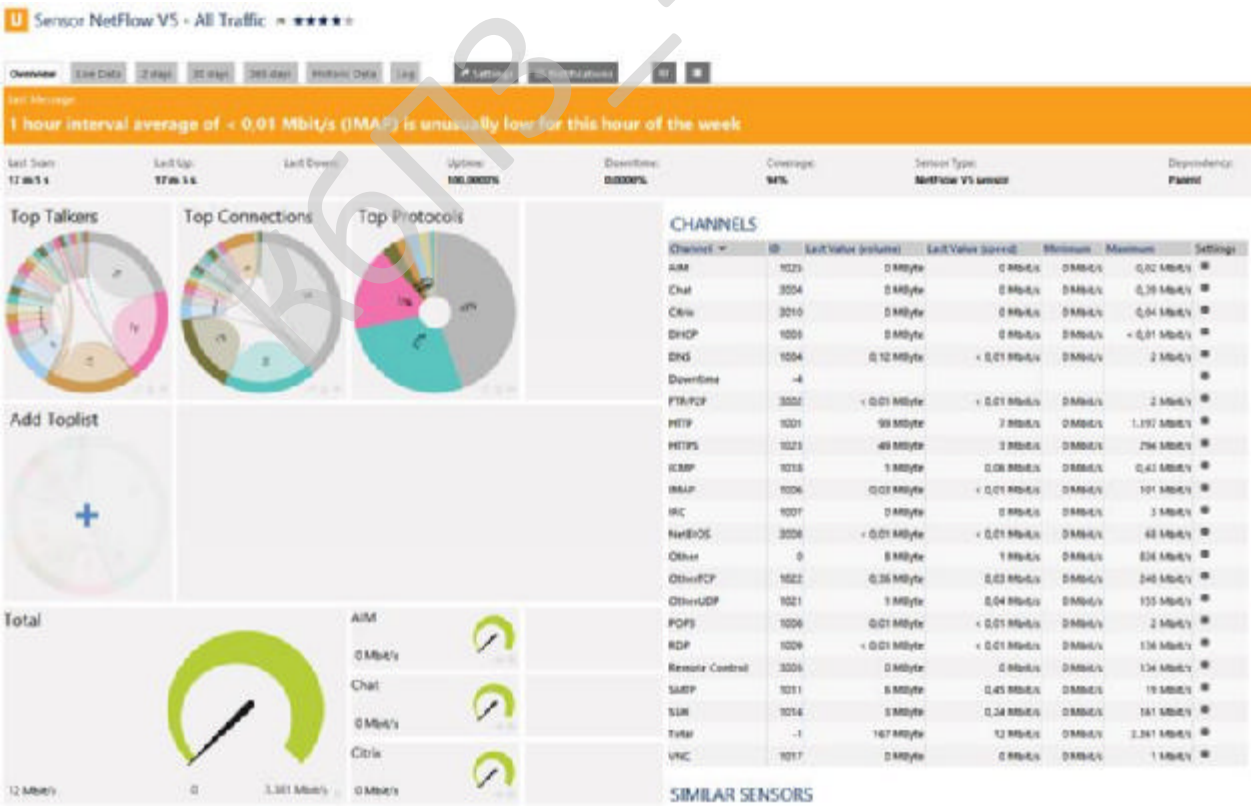

Рисунок 2.2 – Монітор мережі PRTG

Стандартне встановлення спочатку надає всі необхідні інструменти для моніторингу всього трафіку у комп’ютерній мережі. PRTG Network Monitor може використовувати для аналізу різні версії стандарту NetFlow (v5 і v9), відкритий універсальний стандарт IPFIX, а також інші технології обліку мережевого трафіку, такі як sFlow та J-Flow.

Однією з можливостей застосування моніторингу NetFlow, доступною в PRTG Network Monitor, є аналіз пропускної здатності. Звіти візуалізуються у вигляді рейтингових списків, що дозволяють відстежувати дані по різних категоріях, які налаштовується, що значно спростить вам завдання. Наприклад, ви можете відстежувати і відразу порівнювати отримані результати окремо для різних IP-адрес, для підключень, для протоколів, для додатків і т. д. Це дуже корисно при усуненні несправностей в роботі мережі.

Існує також невелика безкоштовна програма Paessler NetFlow Testers, яка просто збирає всі пакети (підтримуються версії 5, 9, а також IPFIX від прив'язаних до NetFlow маршрутизаторів. Для цього спочатку NetFlow необхідно включити і налаштувати на кожному маршрутизаторі, з якого ви хочете отримувати статистику, а також необхідно задати IP-адресу комп'ютера, на якій запущено NetFlow Testers, щоб сюди відправлялися UDP-пакети з даними NetFlow, це типова Flow Collector у класичному розумінні, яка не має нічого спільного з програмним забезпеченням для аналізу.

Програмне забезпечення PRTG Network Monitor включає до складу NetFlow Collector, який працює автоматично на будь-якому локальному або віддаленому комп'ютері з встановленим PRTG зондом. PRTG Network Monitor використовує зонди як платформу для здійснення моніторингу усієї вашої мережної інфраструктури. Усі об'єкти, налаштовані під конкретним зондом, будуть контролюватись через цей зонд.

Для цінової політики компанії Paessler важливе значення в ієрархії об'єктів програмного забезпечення PRTG має поняття сенсора. На кожному пристрої (практично будь-якому фізичному або віртуальному пристрої у вашій мережі, який

має власну IP-адресу) можна створити кілька сенсорів. Кожен сенсор контролює один аспект роботи пристрою. Наприклад, це може бути:

- один пристрій NetFlow;
- один мережевий сервіс, такий як SMTP, FTP, HTTP тощо;
- трафік одного порту мережного комутатора;
- завантаження центрального процесора пристрою;
- завантаження пам'яті пристрою;
- трафік на одній мережній карті;
- загальний стан системи на пристрої;
- різний контент (наприклад, використання бази даних, електронної пошти, HTTP, XML, файлів тощо).

Програмне забезпечення PRTG Network Monitor доступне у двох версіях: комерційної та безкоштовної. Безкоштовна редакція – це повністю функціональне рішення PRTG Network Monitor, що дозволяє контролювати до 100 сенсорів. Якщо потрібно здійснювати моніторинг понад 100 сенсорів, знадобиться вже комерційна ліцензія. Крім того, спочатку доступна безкоштовна пробна версія, яка в перші 30 днів не має жодних обмежень.

Scrutinizer

Програмне рішення Scrutinizer – це набагато більше, ніж просто аналізатор NetFlow. Програма є повноцінною системою реагування на інциденти, яка може використовуватися як для аналізу мережевого трафіку, так і для отримання звітів про інциденти безпеки. Scrutinizer може збирати та аналізувати дані з різних типів потоків, використовуючи NetFlow, J-Flow, NetStream та IPFIX. Іншими словами, це рішення буде повноцінно працювати з різними пристроями від різних постачальників.

Scrutinizer забезпечує видимість як у фізичному середовищі, так і віртуальному. Він також має швидку та просунуту функціональність щодо формування звітності, підтримує можливості колективної роботи та надзвичайно масштабуємо, оскільки побудований на принципах розподіленої архітектури.

Для Scrutinizer існує три варіанти розгортання: на базі апаратного забезпечення як віртуальна машина, а також як SaaS (Software as a Service, програмне забезпечення як послуга). Можна спробувати Scrutinizer безкоштовно протягом 30 днів, після чого функціональність продукту буде знижена до безкоштовної версії, яка має низку обмежень. Зокрема, обмежений колективний доступ, а також немає можливості зберігати або експортувати дані. Крім того, ви можете збирати дані з будь-якої кількості потоків, але ця інформація буде доступна лише протягом п'яти годин. Іншими словами, у вас не буде доступу до раніше зібраних даних і щоразу вам доведеться починати заново.

Рисунок 2.3 – ПЗ Scrutinizer

ManageEngine NetFlow Analyzer

Сервіс ManageEngine пропонує колектор і аналізатор NetFlow, який за своєю функціональністю дуже схожий на рішення, описані вище. Це потужний програмний продукт з повним спектром можливостей для збору даних про потоки у комп’ютерній мережі та аналізу цієї інформації. ManageEngine NetFlow Analyzer

підтримує різні технології збору інформації про потоки, такі як NetFlow, J-Flow і NetStream, і призначений для аналізу мережевого трафіку, а також моніторингу пропускної спроможності в реальному часі.

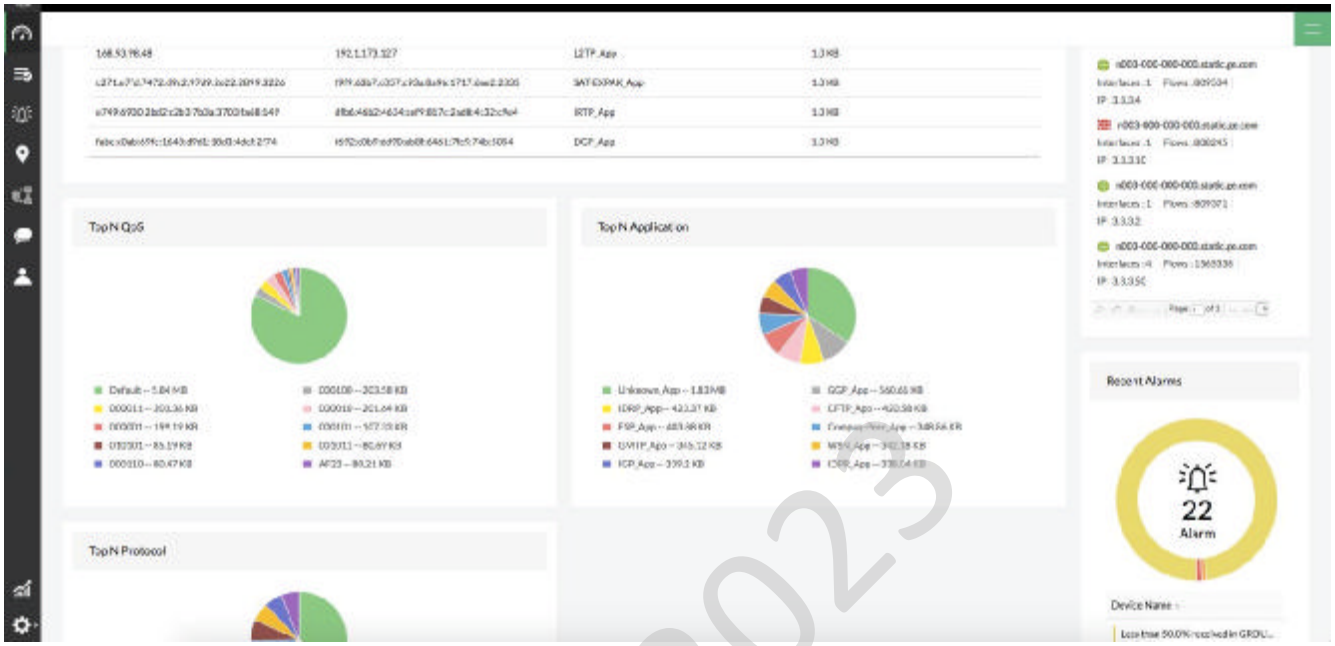

Рисунок 2.4 – ManageEngine NetFlow Analyzer

Окрім цього, ManageEngine NetFlow Analyzer має декілька цікавих можливостей, таких як:

- панель моніторингу, що настроюється для виведення зведених даних за додатками, протоколами та використовуваними ресурсами;
- вбудована система оповіщень під час досягнення порогових значень;
- додаток для iPhone, що дозволяє проводити моніторинг у будь-який час та з будь-якого місця;
- підтримка звітів Cisco Medianet та Cisco WAAS.

ManageEngine пропонує онлайн-демонстрацію свого аналізатора NetFlow, що дозволяє оцінити його можливості, перш ніж ухвалити рішення про повнофункціональне використання. ManageEngine NetFlow Analyzer доступний у двох версіях: Essential (для малих, середніх та великих компаній) та Distributed (для

корпорацій). Обидві версії можуть бути випробувані безкоштовно протягом 30 днів. Ціна версії Essential залежить від вибраної функціональності та кількості інтерфейсів, для яких проводиться моніторинг. Існує також можливість використовувати безкоштовну версію рішення без необхідності будь-якої ліцензії, яка може бути застосована для моніторингу не більше двох інтерфейсів.

nProbe і ntopng

Рішення ntopng – це веб-інструмент з відкритим кодом для моніторингу та аналізу мережного трафіку. Програма була написана для використання на мобільних пристроях під керуванням будь-якої з найпопулярніших платформ: Unix, MacOS та Windows.

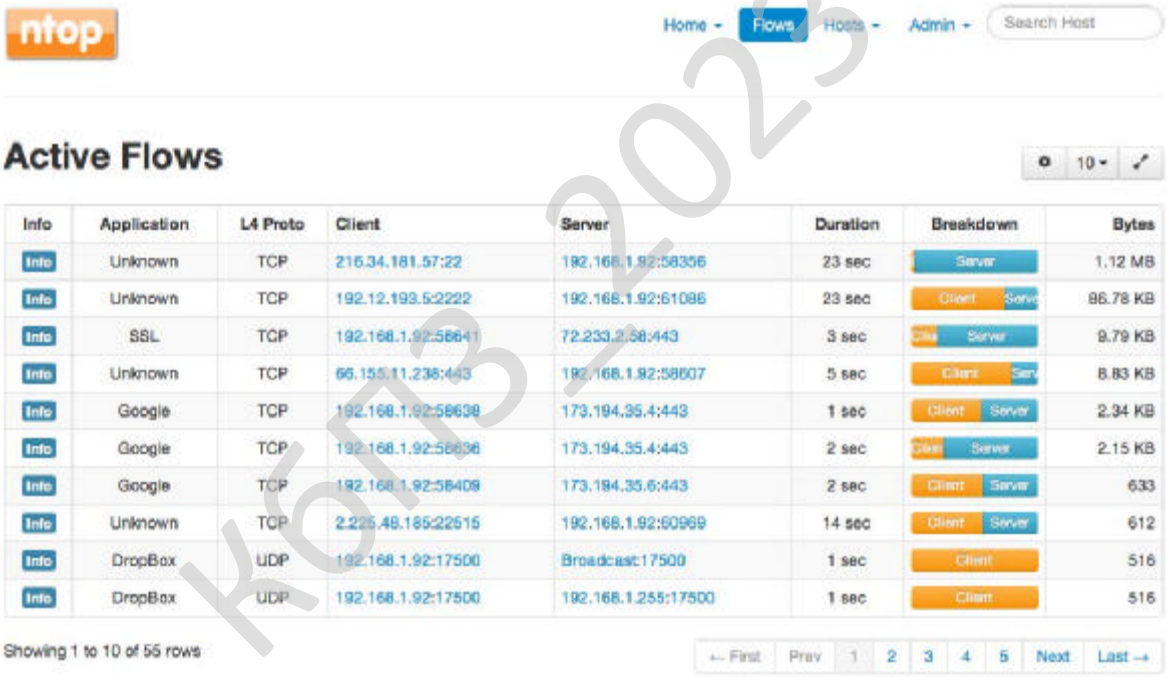

Рисунок 2.5 – nProbe і ntopng

Через зашифрований інтерфейс надходять дані про передані пакети. Програма аналізує їх та надає різноманітну корисну інформацію про функціонування вашої мережі. Наприклад, за допомогою ПЗ ntopng ви можете:

– сортувати інформацію про мережевий трафік за багатьма параметрами, включаючи IP-адреси, порти, протоколи, пропускну здатність тощо;

– в режимі реального часу відстежувати найбільш активні хости та додатки, що вимагають найбільше пропускної спроможності;

– відстежувати інформацію про передані байти, а також кількість відправлених, отриманих та втрачених пакетів;

– зберігати на диск історію мережевого трафіку для подальшого аналізу;

– визначати географічне розташування та будувати схеми з'єднання хостів на карті місцевості;

– а також багато іншого.

Програма ntopng може підключатися до nProbe, який є колектором NetFlow/IPFIX. У цьому тандемі ролі розподілені так: nProbe збирає дані про потоки і відправляє цю інформацію в ntopng, який, у свою чергу, аналізує її і представляє у зручному для сприйняття вигляді.

Хоча ntopng і доступний у безкоштовній версії (існують також розширені платні версії продукту), вам знадобиться ліцензія для використання nProbe (за винятком некомерційних організацій або навчальних закладів). nProbe поставляється у двох версіях: Standard і Pro з плагінами.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для розробки інтелектуальної системи для виявлення DDoS-атак було обрано мову програмування Python.

Мова програмування Python була обрана з наступних причин:

1. **Простота та читабельність коду**. Мова Python славиться своєю простотою та читабельністю коду. Це робить її ідеальним вибором для розробки складних систем, таких як системи виявлення DDoS-атак, де розуміння та підтримка коду важливі.

2. **Велика спільнота розробників**. Python має велику та активну спільноту розробників, це означає, що завжди є доступ до багатьох бібліотек та фреймворків,

які можна використовувати для розробки інтелектуальної системи.

3. **Багато бібліотек для машинного навчання та аналізу даних**. Для мови Python розроблено багато бібліотек, які дозволяють легко впроваджувати та навчати моделі машинного навчання для виявлення DDoS-атак.

4. **Портативність**. Python є портативною мовою, це означає, що програми можуть бути запущені на різних операційних системах без значних змін. Це важливо для забезпечення сумісності системи з різними середовищами.

5. **Швидкість розробки**. Python дозволяє швидко розробляти прототипи та тестувати ідеї, що особливо важливо для систем виявлення DDoS-атак, де час реакції на нові загрози може бути критичним.

6. **Безпека**. Python має багато бібліотек та інструментів для забезпечення безпеки програм, що можна використовувати для захисту розробленого програмного забезпечення.

7. **Доступність ресурсів і підтримка**. Про мову Python написано велику кількість книг, створено багато онлайн-курсів, є детальна і постійно оновлювана документація, що допомагає розробникам швидко вивчити мову та використовувати її для розробки програмного забезпечення.

З урахуванням цих переваг, Python стає привабливим вибором для розробки інтелектуальних систем для виявлення DDoS-атак, сприяючи швидкому розвитку, забезпеченню безпеки та ефективності системи.

2.3 Розгорнута постановка завдання

Сучасні мережеві інфраструктури піддаються зростаючому ризику DDoS-атак, які можуть викликати перебої у роботі мереж та сервісів, призводячи до серйозних фінансових збитків та втрати довіри користувачів. Щоб забезпечити захист мереж та служб від таких атак, потрібна система для виявлення DDoS-атак.

Метою даної роботи є розробка та програмна реалізація інтелектуальної системи, яка здатна виявляти DDoS-атаки в мережах та веб-додатках. Система

повинна бути здатною аналізувати трафік, виявляти аномалії та атаки і повідомляти про них у разі їх виявлення.

Для досягнення поставленої мети, потрібно виконати наступні завдання:

1. **Дослідження літературних джерел**. Провести огляд наукових робіт та публікацій, пов'язаних з DDoS-атаками та методами їх виявлення. Дослідити існуючі підходи та технології.

2. **Розробка моделі виявлення DDoS-атак**. Розробити математичну модель та алгоритми виявлення DDoS-атак на основі аналізу мережевого трафіку.

3. **Реалізація програмного забезпечення**. Реалізувати інтелектуальну систему для виявлення DDoS-атак на мові програмування Python.

4. **Тестування та оцінка продуктивності**. Провести тестування системи на симульованих атаках та реальних даних з відкритих датасетів для оцінки її продуктивності та ефективності виявлення атак.

5. **Розробка інтерфейсу користувача**. Створити зручний інтерфейс користувача для моніторингу стану комп’ютерної мережі або веб-сайту і відслідковування інформаційних атак.

6. **Документування результатів**. Підготувати документацію, включаючи технічний опис системи, інструкцію користувача та звіт про результати дослідження.

Очікувані результати

Очікується, що результатом даного дослідження буде інтелектуальна система для виявлення DDoS-атак, яка зможе захищати мережі та веб-додатки виявляючи інформаційні атаки, надмірний трафік та аномалії. Система буде відповідати вимогам надійності та ефективності, забезпечуючи стійкість до атак та надійну роботу мереж.

Ця постановка завдання визначає основні кроки та цілі дослідження і розробки інтелектуальної системи для виявлення DDoS-атак, і на основі ней й буде проведено дослідження.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Статистичні методи, такі як ентропія, кореляція та коваріація, вважаються найпоширенішими методами виявлення DDoS-атак, а також аналізу та виявлення аномалій у мережевому трафіку. Весь запис трафіку або лише окремі мережеві пакети можуть бути перетворені в параметри характеристики мережі, які далі використовуються як вхідні дані для статистичних методів.

Основним методом опису мережевого трафіку є використання процесу надходження $A\ (r, s)$, який представляє кількість пакетів в інтервалі (r, s). Інший метод – це використання змінних T_i, які описують тривалість проміжку часу між появою окремих пакетів, як показано на рис. 3.1.

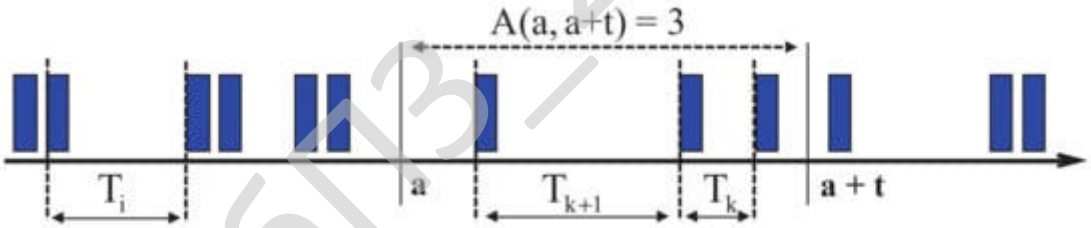

Рисунок 3.1 – Потік пакетів, описаний за допомогою випадкового процесу $A(t)$ і змінних T_i [28]

Загалом, припускається, що потік, який представляє IP-трафік, є випадковим, з $A(r, s)$ що представляє собою випадковий процес з відповідними випадковими величинами T_i. Ще одне спрощення може полягати у тому, щоб враховувати лише так звані стаціонарні випадкові процеси. Простіше кажучи, всі ймовірнісні характеристики стаціонарного процесу інваріантні відносно часу. Немає значення, в якому інтервалі часу починати спостерігати за потоком, це

залежить лише від тривалості спостереження, як показано на рис. 3.2.

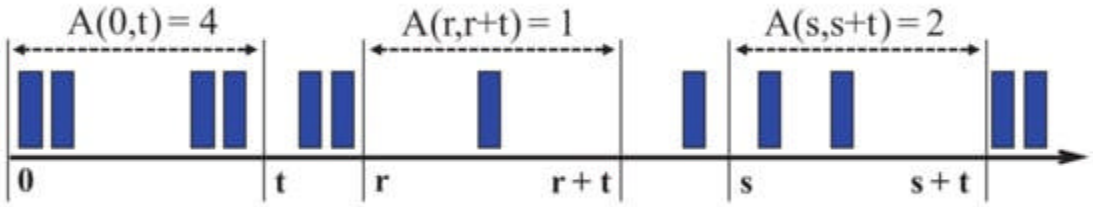

Рисунок 3.2 – Стаціонарний потік пакетів [28]

Для стаціонарного процесу, в якому не має значення, де в часі знаходиться спостережуване часове вікно, оскільки це залежить лише від довжини спостережуваного вікна, можна ввести спрощене позначення $A(t)$:

$$\forall a \in R^{+};\ A(a,a+t)=A(0,t)=A(t). \tag{3.1}$$

Стаціонарний процес не змінює своїх імовірнісних характеристик з часом. Отже, випадкові величини T_i, які описують *i*-ті проміжки в потоці пакетів, мають однаковий розподіл ймовірностей. Можна використовувати ту саму випадкову величину *T* для всіх розривів, яка має функцію розподілу $F(t)$, $T \sim F(t)$.

Якщо IP-трафік вимірюється в заданих часових інтервалах, процес $A(t)$ складається з так званих приростів $a(i)$, які записують появу пакетів у *i*-тих часових інтервалах. Таким чином, у нульовий момент часу пакет не виникає $A(0)=a(0)=0$:

$$A(t) = \sum_{i=1}^{t} a(i),\ a(t) = A(t)A(t-1). \tag{3.2}$$

Якщо ми розглядаємо часове вікно довжиною *n* часових інтервалів, то основними характеристиками IP-потоку є середня швидкість λ_{avg} і пікова ставка λ_{peak} , що мають форму:

$$\lambda_{\text{avg}} = EA(n), \lambda_{\text{peak}} = \max_{1 \le i \le n} a(i) = n. \tag{3.3}$$

Більшість регулярних складових часових рядів належить до двох класів: вони є *трендом*, або *сезонною складовою*.

Тренд – загальна систематична лінійна або нелінійна компонента, яка може змінюватися в часі.

Сезонна складова – це компонент, що періодично повторюється.

Обидва ці види регулярних компонентів часто присутні в ряді одночасно. Наприклад, трафік на веб-ресурсі може зростати кожен день, але він також містить сезонну складову (наприклад, 45% переглядів припадає на вечір і лише 7% на ранок).

Аналіз тренду у часовому ряді

Не існує "автоматичного" способу виявлення тренду в часовому ряді. Однак якщо тренд є монотонним (стійко зростає або стійко зменшується), аналізувати такий ряд зазвичай неважко. Якщо часові ряди містять значну помилку, першим кроком виділення тренда є згладжування.

Згладжування. Згладжування завжди включає певний спосіб локального усереднення даних, у якому несистематичні компоненти взаємно погашають одне одного. Найзагальніший метод згладжування - *ковзне середнє,* в якому кожен член ряду замінюється простим або зваженим середнім *n* сусідніх членів, де *n* - ширина "вікна". Замість середнього можна використовувати медіану значень, що потрапили у вікно. Основна перевага медіанного згладжування, у порівнянні зі згладжуванням ковзним середнім, полягає в тому, що результати стають більш стійкими до викидів (всередині вікна). Таким чином, якщо у даних є викиди (пов'язані, наприклад, з помилками вимірювань), то згладжування медіаною зазвичай призводить до більш гладких або, принаймні, більш "надійних" кривих, порівняно із ковзним середнім з тим же самим вікном. Основний недолік медіанного згладжування в тому, що за відсутності явних викидів, воно призводить до більш "зубчастих" кривих (ніж згладжування ковзним середнім) і не дозволяє використовувати ваги.

Щодо рідше, коли помилка виміру дуже велика, використовується метод *згладжування методом найменших квадратів, зважених щодо відстані* або метод *негативного експоненційно зваженого згладжування* . Всі ці методи відфільтровують шум і перетворюють дані на відносно гладку криву. Ряди з відносно невеликою кількістю спостережень та систематичним розташуванням точок можуть бути згладжені за допомогою *бікубічних сплайнів* .

Підбір функції. Багато монотонних часових рядів можна добре наблизити

лінійною функцією. Якщо є явна монотонна нелінійна компонента, то дані спочатку слід перетворити, щоб усунути нелінійність. Зазвичай для цього використовують логарифмічне, експонентне або (менш часто) поліноміальне перетворення даних.

Аналіз сезонності

Періодична та сезонна залежність (сезонність) є іншим загальним типом компонент часового ряду. Загалом періодична залежність може бути формально визначена як кореляційна залежність порядку *k* між кожним *i*-тим елементом ряду і *ik*-тим елементом. Її можна виміряти за допомогою автокореляції (тобто кореляції між самими членами ряду); *k* зазвичай називають *лагом* (іноді використовують еквівалентні терміни: зсув, запізнення). Якщо помилка виміру не надто велика, то сезонність можна визначити візуально, розглядаючи поведінку членів ряду через кожні *k* часових одиниць.

Автокореляційна корелограма. Сезонні складові часового ряду можуть бути знайдені за допомогою корелограми. Корелограма (автокорелограма) показує чисельно і графічно автокореляційну функцію, тобто коефіцієнти автокореляції (і їх стандартні помилки) для послідовності лагів з певного діапазону (наприклад, від 1 до 30). На корелограмі зазвичай відзначається діапазон у розмірі двох стандартних помилок на кожному лазі, проте зазвичай величина автокореляції більш цікава, ніж її надійність, тому що інтерес переважно представляють дуже сильні (а отже, високо значущі) автокореляції.

Дослідження корелограм. При вивченні корелограми слід пам'ятати, що автокореляції послідовних лагів формально залежні між собою. Розглянемо наступний приклад. Якщо перший член ряду тісно пов'язаний з другим, а другий з третім, перший елемент повинен також якимось чином залежати від третього і т.д. Це призводить до того, що періодична залежність може суттєво змінитись після видалення автокореляцій першого порядку, тобто після взяття різниці з лагом 1).

Для визначення наявності Ddos-атаки було вирішено аналізувати вхідний трафік на веб-сайт або комп’ютерну мережу методом R/S-аналізу. Це дозволить

виявляти тренди у трафіку, зокрема, тенденцію до продовження його збільшення. R/S-аналіз часових рядів трафіку дозволяє отримати показник Херста. Інтерпретація одержаних значень показника Херста може здійснюватися наступним чином:

– $H = 0.5$ – певного тренду у мережевого трафіку немає, вважаємо його повністю випадковим.

– $H > 0.5$ – мережевий трафік характеризується персистентністю – має тенденцію до збереження тренду; чим більше число, тим більш проявлене збереження тренду.

– $H < 0.5$ – мережевий трафік характеризується антиперсистентністю – будь-яку тенденцію прагне змінити протилежна; чим менше число, тим більш проявлене прагнення до зміни тренду.

Персистентний трафік (з показником $H > 0.5$) є зоною ризику, так як може свідчити про Ddos-атаку, або природнє підвищення інтересу до сайту, що також може призвести до відмови в обслуговуванні при надто великих і тривалих об'ємах вхідного трафіку.

Показник Херста H треба розглядати разом з поточною інтенсивністю трафіку λ. Адже тільки сукупність високого індексу Херста та високої інтенсивності трафіку може свідчити про можливу небезпеку перевантаження сервера та ймовірність відмови в обслуговуванні.

Метод R/S-аналізу для прогнозування трафіку складається з наступних етапів:

Етап 1. Завантажується часовий ряд мережевого трафіку S_t. Для нього розраховується логарифмічне відношення за наступною формулою:

$$N_t = ln \frac{S_{t+a+n}}{S_{t-1+a+n}}. \quad (3.4)$$

Етап 2. Часовий ряд трафіку N розділяється на A суміжних періодів довжиною n. Кожен період позначимо як I_a, де $a = 1, 2, \ldots, A$. Для кожного періоду I_a визначається середнє значення за формулою:

$$E(I_a) = \frac{1}{n} \sum_{t=1}^{n} N_{t,a}. \quad (3.5)$$

Етап 3. За наступною формулою розраховується відхилення від середнього значення для кожного періоду I_a:

$$X_{k,a} = \sum_{i=1}^{k} \left(N_{i,a} - E(I_a) \right) \quad (3.6)$$

Етап 4. Потім розраховується розмах в межах кожного періоду:

$$R_{I_a} = max\left(X_{k,a}\right) - min\left(X_{k,a}\right). \quad (3.7)$$

Етап 5. Далі обчислюється стандартне відхилення для кожного періоду I_a:

$$S_{I_a} = \sqrt{\frac{1}{n} \sum_{k=1}^{n} \left(N_{k,a} - E(I_a) \right)^2}. \quad (3.8)$$

Етап 6. Кожен R_{I_a} ділиться на S_{I_a}. Далі розраховується середнє значення *R/S* за наступною формулою:

$$R/S(n) = \frac{\sum_{a=1}^{A} R_{I_a}/S_{I_a}}{A}. \quad (3.9)$$

Етап 7. Потім *n* збільшується та етапи (2)-(6) повторюються доти, поки виконується умова $n \leq N/2$.

Етап 8. Далі необхідно побудувати графік залежності $log\left(R/S(n)\right)$ від $log(n)$ і за допомогою методу найменших квадратів знайти регресію виду: $log\left(R/S(n)\right) = H \cdot log(n) + c$, де коефіцієнт нахилу кривої *H* – показник Херста.

Етап 9. Отриманий результат перевіряється на значущість. Для цього необхідно перевірити гіпотеу про те, що часовий ряд трафіку є нормально розподіленим. Якщо *R/S* є нормально розподіленими випадковими змінними, можна припустити, що й *H* також розподілені нормально. Асимптотичною границею для незалежного процесу є показник Херста, рівний 0.5. Еніс і Ллойд, а також Петерс запропонували використовувати такі очікувані показники *R/S*:

$$E\left(R/S(n)\right) = \frac{n-0.5}{n} \cdot \left(n \cdot \frac{\pi}{2}\right)^{-0.5} \cdot \sum_{r=1}^{n-1} \sqrt{\frac{n-r}{r}} \quad (14)$$

Для *n* спостережень необхідно знайти очікуваний показник Херста *E*(*H*).

Етап 10. Розраховується очікувана дисперсія показника Херста за наступною формулою:

$$Variance(H) = \frac{1}{N} \quad (15)$$

де H – показник Херста; N – довжина часового ряду мережевого трафіку.

Етап 11. Перевіряється значущість отриманого індексу Херста шляхом оцінки кількості стандартних відхилень, на які H перевищує $E(H)$. Значущим вважається результат, при якому показник значущості по модулю більше 2.

Саме такий метод було використано для визначення індексу Херста у розробленому програмному забезпеченні.

Також необхідно розрахувати інтенсивність трафіку на основі його часового ряду. Для цього пропонується наступна послідовність дій.

Інтенсивність часового ряду трафіку, може розумітися як середнє число запитів за одиницю часу. Якщо ми припускаємо, що кожен елемент часового ряду представляє кількість запитів, що надійшли протягом певного інтервалу часу (наприклад, за секунду), тоді середнє значення цього ряду покаже середню інтенсивність запитів на обслуговування за цей час.

Математично інтенсивність трафіку можна виразити таким чином:

$$I = P / T.$$

де I – інтенсивність трафіку, тобто, кількість пакетів (або об'єм даних) на одиницю часу; P – кількість пакетів, які були передані за даний часовий інтервал; T – часовий інтервал, тобто, тривалість часу, протягом якого були передані ці пакети (в секундах).

Якщо трафік представлений у вигляді часового ряду, у якому кожен елемент – кількість пакетів за одиницю часу, наприклад за 1 секунду, то по-суті такий часовий ряд містить список інтенсивностей трафіку на однакових проміжках часу. В такому разі, будемо визначити загальну інтенсивність на заданому проміжку. Приклад програми на Python для визначення середньої інтенсивності трафіку:

```
import numpy as np

def calculate_intensity(time_series, time_start, time_stop):
    # Переконуємось, що time_series містить числа
    time_series = np.array(time_series[time_start: time_stop],
dtype=np.float64)
    # Розраховуємо інтенсивність
```

```
    intensity = np.sum(time_series) / len(time_series)

    return intensity

# Приклад використання функції
time_series_example = [20, 30, 25, 40, 35, 30, 45, 50, 60, 55]
time_start = 0
time_stop = len(time_series_example)
intensity = calculate_intensity(time_series_example, time_start, time_stop)
print(f"Середня інтенсивність: {intensity} запитів на одиницю часу.")
```

Цей приклад демонструє обчислення інтенсивності на основі кількості пакетів і часового інтервалу.

Інтенсивність, нормалізована до діапазону від 0 до 1, може бути отримана з застосуванням наступної формули:

$$I_{\text{нормалізована}} = (I - I_{min}) / (I_{max} - I_{min}).$$

Це перетворення змінить шкалу інтенсивності, так що максимальне спостереження буде еквівалентно 1, а всі інші значення будуть відповідно меншими, зберігаючи свої відносні пропорції.

Тоді приклад знаходження інтенсивності буде виглядати наступним чином:

```
import numpy as np

def calculate_intensity(time_series, time_start, time_stop):
    # Переконуємось, що time_series містить числа
    time_series = np.array(time_series[time_start: time_stop],
dtype=np.float64)

    # Розраховуємо інтенсивність
    intensity = np.sum(time_series) / len(time_series)
    intensity = (intensity - np.min(time_series)) / (np.max(time_series) -
np.min(time_series))


    return mean_intensity

# Приклад використання функції
time_series_example = [20, 30, 25, 40, 35, 30, 45, 50, 60, 55]
```

```
time_start = 0
time_stop = len(time_series_example)
intensity = calculate_intensity(time_series_example, time_start, time_stop)
print(f"Середня інтенсивність: {intensity} запитів на одиницю часу.")
```

У цих прикладах за допомогою змінних time_start та time_stop можна вказати, на якому саме інтервалі часового ряду слід визначити середню інтенсивність.

3.2 Розробка структурної схеми

Структурна схема розробленого програмного забезпечення зображена на рисунку 3.3.

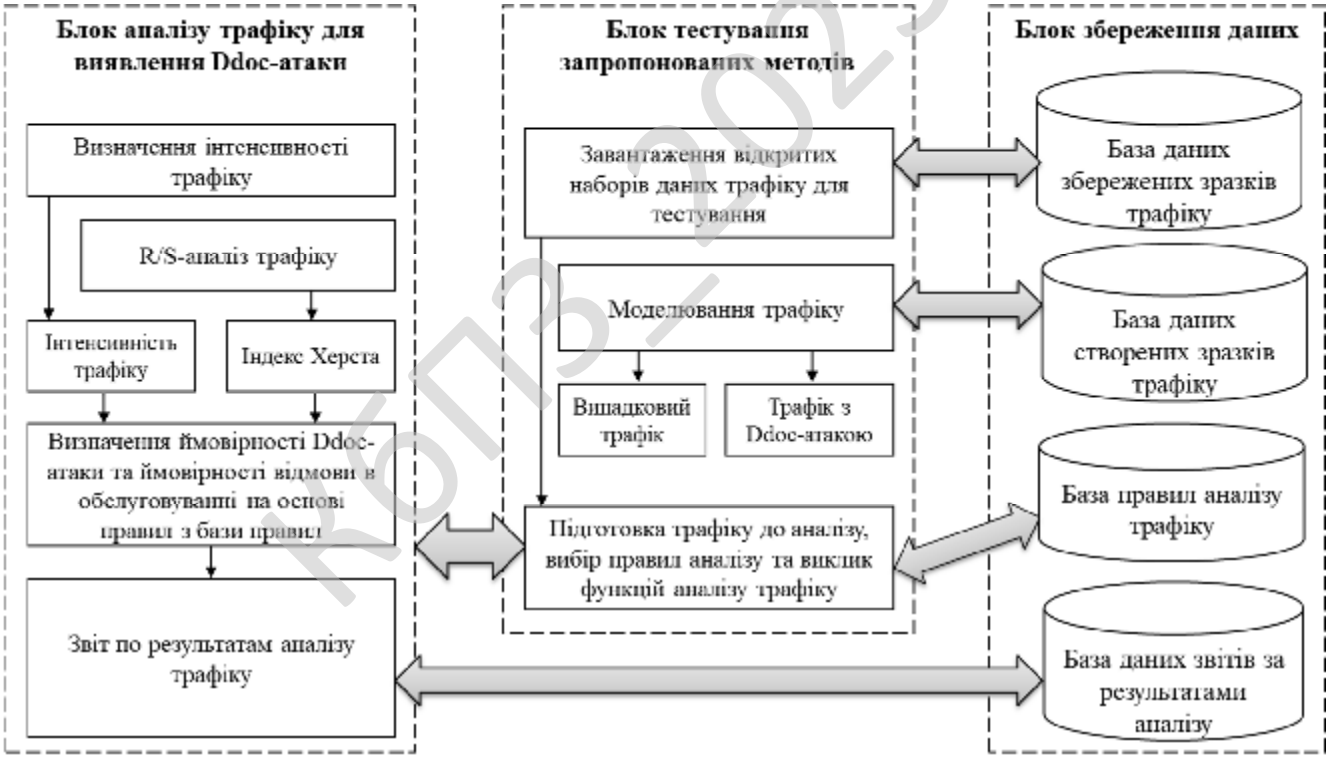

Рисунок 3.3 – Структурна схема системи

З рисунку видно, що система складається з наступних частин:

- Блок аналізу трафіку для виявлення Ddoc-атаки;
- Блок тестування запропонованих методів;

– Блок збереження даних.

Блок аналізу трафіку для виявлення Ddoc-атаки містить наступні модулі:

– Визначення інтенсивності трафіку.

– R/S-аналіз трафіку.

– Визначення ймовірності Ddoc-атаки та ймовірності відмови в обслуговуванні на основі інтенсивності трафіку та індексу Херста.

– Звіт по результатам аналізу трафіку.

Блок тестування запропонованих методів містить наступні модулі:

– Завантаження відкритих наборів даних трафіку для тестування.

– Моделювання трафіку: 1) випадкового, 2) такого, що містить Ddoc-атаки.

– Підготовка трафіку до аналізу, вибір правил аналізу та виклик функцій аналізу трафіку.

Блок збереження даних містить наступні бази даних:

– База даних збережених зразків трафіку.

– База даних створених зразків трафіку.

– База правил аналізу трафіку.

– База даних звітів за результатами аналізу.

3.3 Розробка функціональної схеми

Функціональна схема системи наведена на рис. 3.4.

Як видно зі схеми, розроблювана система складається з наступних модулів:

- Модуль нормалізації часового ряду.
- Модуль визначення інтенсивності трафіку.
- Модуль R/S-аналізу трафіку.
- Модуль аналізу трафіку для виявлення Ddoc-атаки.
- Модуль тестування розробленої системи.

- Модуль симуляції трафіку.
- База даних датасетів трафіку.
- Модуль збереження звітів.
- Модуль бази даних.

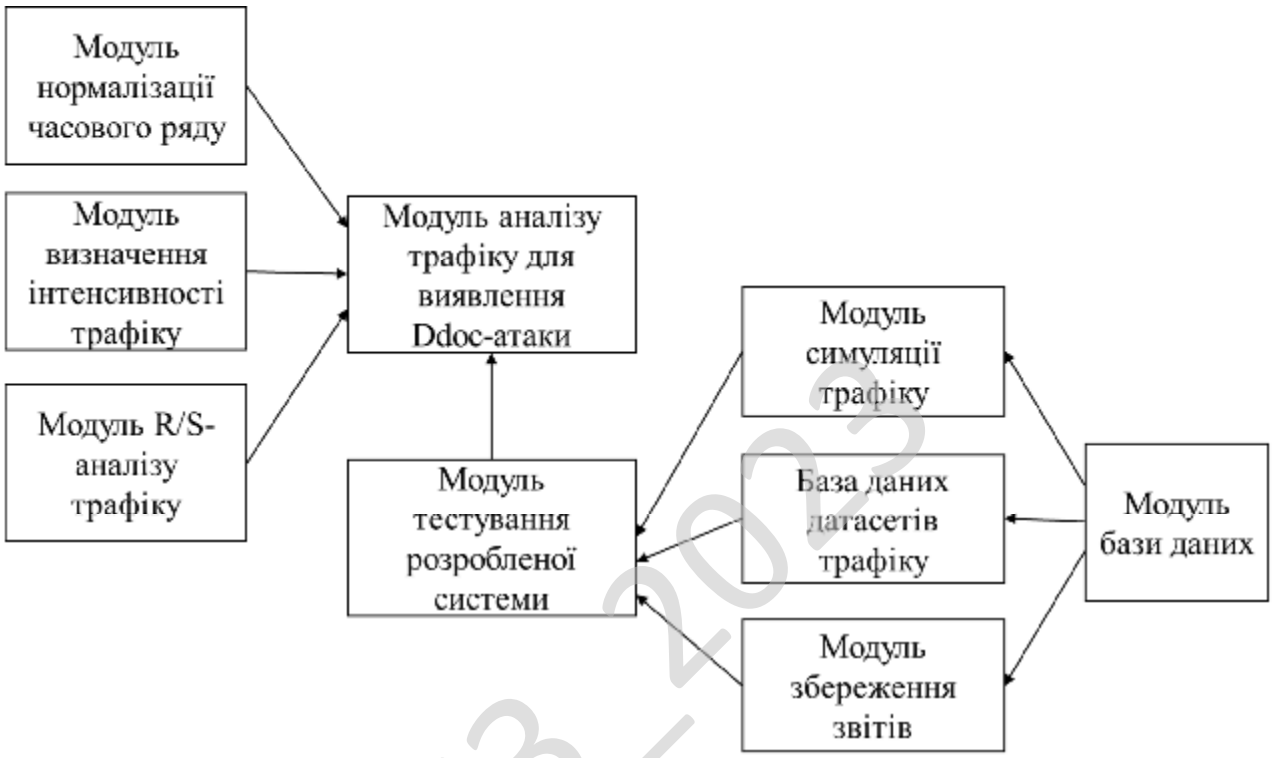

Рисунок 3.4 – Функціональна схема системи

Також функціональна схема ілюструє як ці модулі пов’язані і взаємодіють між собою.

3.4 Розробка діаграми процесів

На рис. 3.5 зображена діаграма процесів, що описує наявні у розроблюваній інтелектуальній системі виявлення Ddos-атак процеси та їх взаємодію.

Розроблена система містить наступні процеси:

- Головне вікно програми;

- Отримати трафік для аналізу;
- Моделювання часового ряду трафіку;
- Завантаження часового ряду трафіку з датасету;
- Аналіз трафіку;
- Визначення інтенсивності трафіку;
- R/S-аналіз трафіку;
- Визначення трендів;
- Визначення ймовірності Ddos-атаки;
- Визначення ймовірності відмови в обслуговуванні;
- Результати аналізу;
- Збереження результатів аналізу;
- Перегляд результатів попередніх аналізів трафіку.

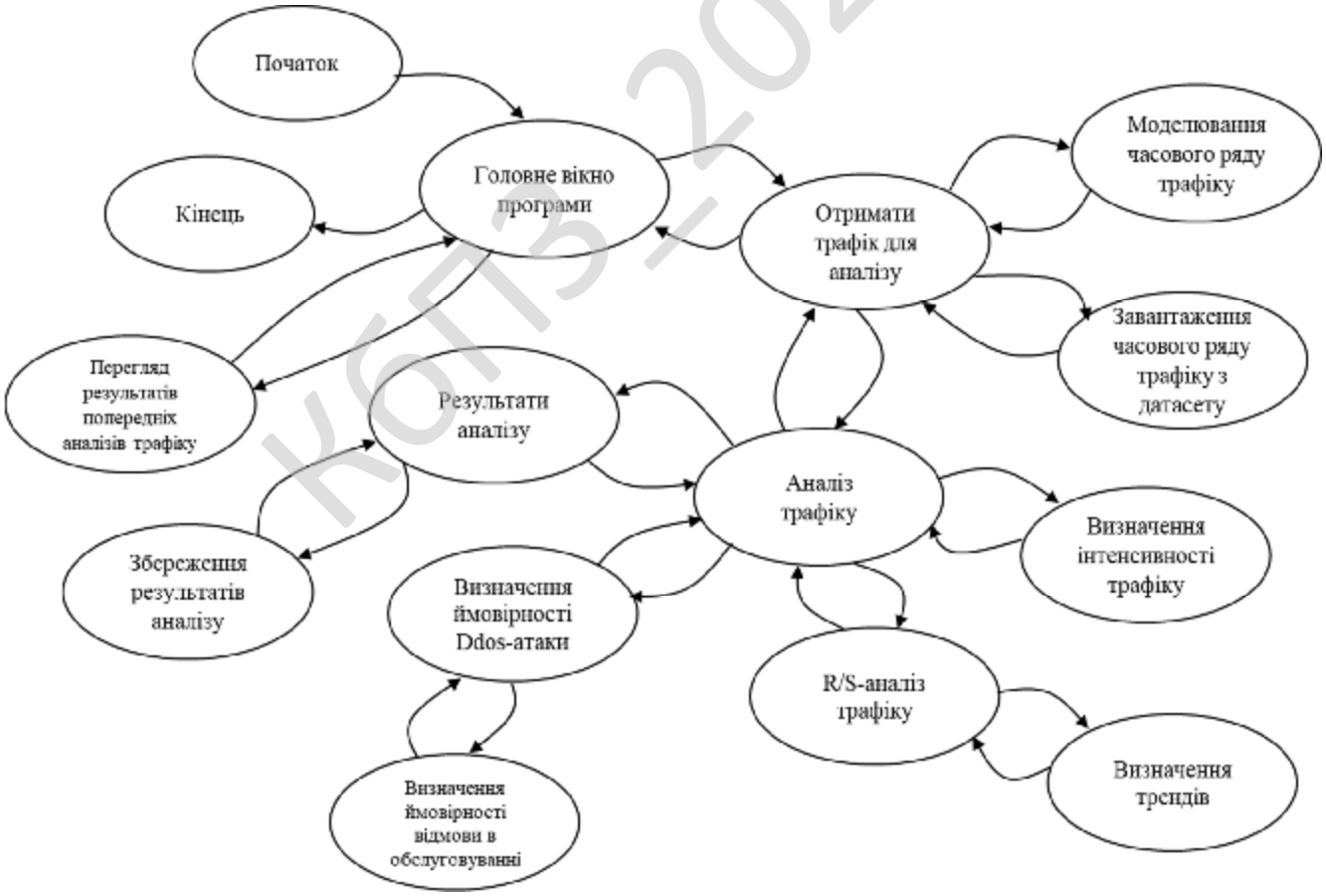

Рисунок 3.5 – Діаграма процесів системи

При запуску програми можна обрати завантаження трафіку для аналізу з наявних датасетів або отримати його шляхом моделювання. Також можна переглянути історію попередніх аналізів трафіку. Після завантаження трафіку відбувається аналіз, що полягає у визначенні його інтенсивності та обчисленні показника Херста з застосуванням R/S-аналізу. На основі інтенсивності і показника Херста для досліджуваного трафіку відбувається визначення ймовірності Ddos-атаки та ймовірності відмови в обслуговуванні. Результати аналізу трафіку виводяться на екран та зберігаються у базу даних.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Блок-схема основної програми зображена на рис. 4.1.

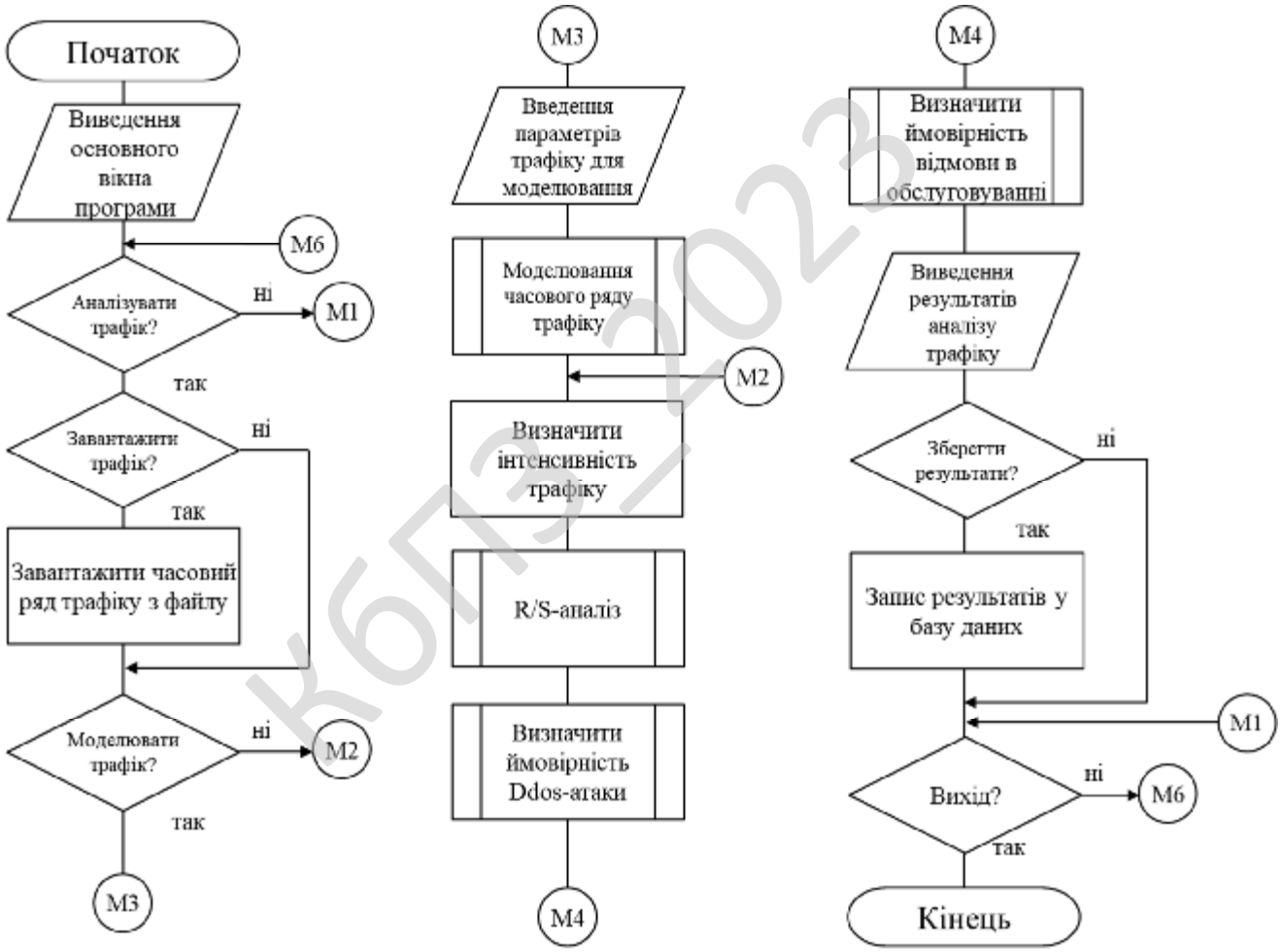

Рисунок 4.1 – Блок-схема основної програми

Алгоритм роботи основної програми полягає у наступному:

Крок 1. Виведення основного вікна програми.

Крок 2. Якщо користувач обрав аналізувати трафік, виконуються наступні

кроки, інакше програма переходить на крок 15.

Крок 3. Якщо користувач обрав завантажити трафік, виконується наступна дія, інакше програма переходить на крок 5.

Крок 4. Завантажити часовий ряд трафіку з файлу.

Крок 5. Якщо користувач обрав моделювати трафік, виконуються наступні дії, інакше програма переходить на крок 8.

Крок 6. Введення параметрів трафіку для моделювання.

Крок 7. Виконання підпрограми моделювання часового ряду трафіку.

Крок 8. Визначити інтенсивність трафіку

Крок 9. Виконання підпрограми R/S-аналізу.

Крок 10. Виконання підпрограми визначення ймовірності Ddos-атаки.

Крок 11. Виконання підпрограми визначення ймовірності відмови в обслуговуванні.

Крок 12. Виведення результатів аналізу трафіку.

Крок 13. Якщо користувач обрав зберегти результати аналізу, то виконання наступної дії, інакше перехід на крок 15.

Крок 14. Запис результатів у базу даних.

Крок 15. Якщо користувач обрав вихід з системи, то завершення роботи програми, інакше перехід на крок 2.

Для виявлення Ddoc-атаки здійснюється визначення інтенсивності та індексу Херста завантаженого мережевого трафіку.

Велика інтенсивність трафіку не завжди свідчить про атаку, тому для прийняття рішення треба визначити додаткові параметри. Моніторинг індексу Херста може допомогти виявити аномалії у фрактальній структурі часового ряду трафіку. Наприклад, значення індексу Херста близьке до 0.5 вказує на випадковий ряд без фрактальних властивостей, а значення, яке сильно відрізняється від 0.5, може свідчити про аномалії. Індекс Херста > 0.5 вказує на тривале збереження тренду. Тож, якщо інтенсивність трафіку висока і Індекс Херста > 0.5 – висока ймовірність Ddoc-атаки та відмови в обслуговуванні. І чим більші ці показники,

тим більші вказані ймовірності.

На рис. 4.2 наведено блок-схему алгоритму роботи підпрограми визначення показника Херста за допомогою R/S-аналізу.

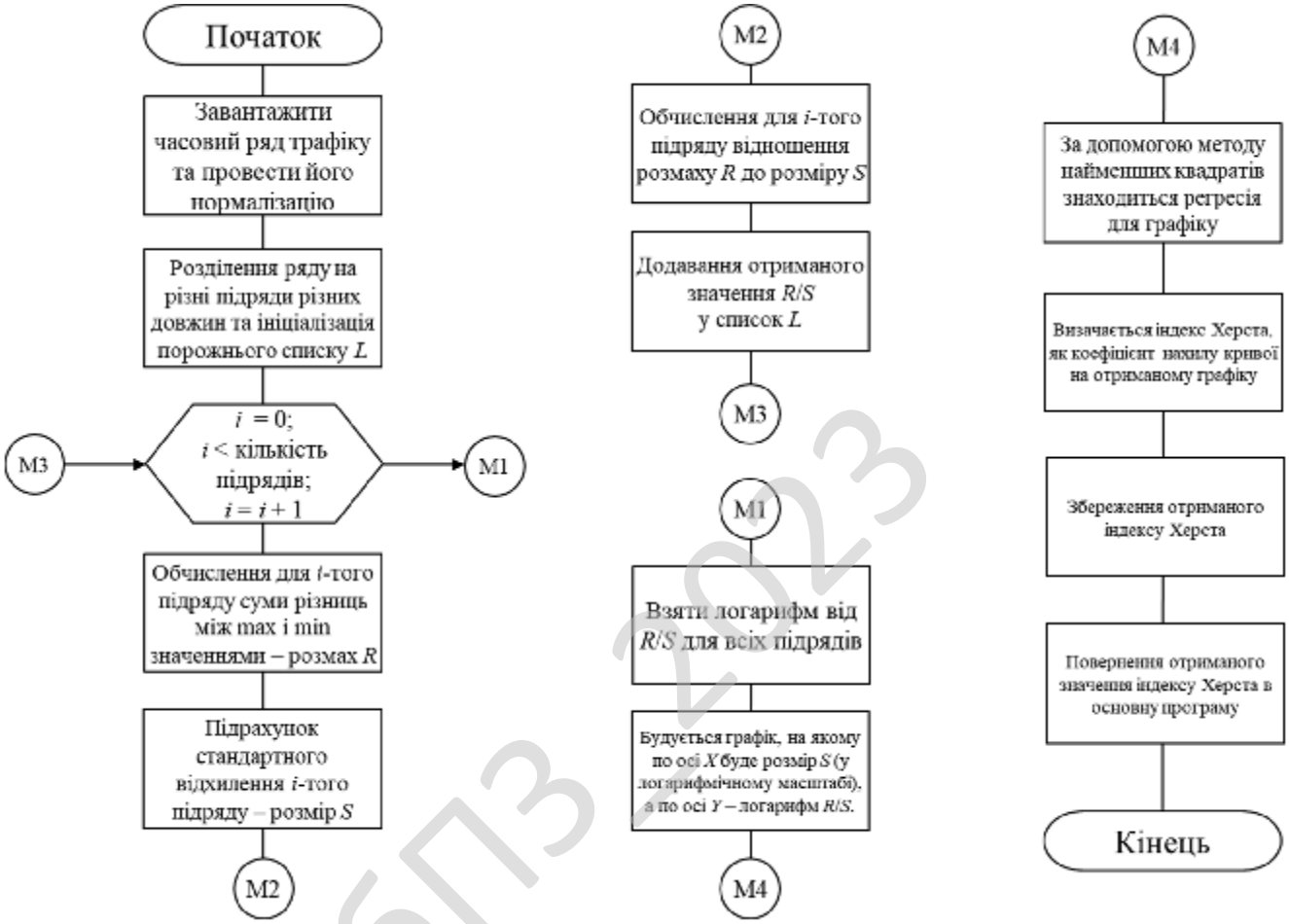

Рисунок 4.2 – Блок-схема алгоритму роботи підпрограми визначення показника Херста за допомогою R/S-аналізу

Визначення показника Херста за наведеним алгоритмом (рис. 4.2) відбувалося наступним чином:

1. **Підготовка даних**:

- Завантаження часового ряду трафіку.
- Нормалізація часового ряду.

2. **Розділення ряду**:

− Розділення нормалізованого часового ряду на різні підряди різних довжин (починаючи з деякого мінімального значення і збільшуючи його на кожному кроці).

− Для кожного підряду обчислюється сума різниць між максимальним і мінімальним значеннями (розмах) і підраховується стандартне відхилення цього підряду (розмір).

3. **Обчислення R/S**:

− Для кожного підряду обчислюється відношення розмаху R до розміру S.

− Отриманні значення R/S для всіх підрядів записуються у деякий список.

4. **Логарифмування**:

− Береться логарифм від R/S для всіх підрядів.

5. **Побудова графіка**:

− Будується графік, на якому по осі X буде розмір S (у логарифмічному масштабі), а по осі Y - логарифм R/S.

− Зазвичай результат буде схожим на лінійний графік, і його кут нахилу вказує на ступінь фрактальності. Графік з кутом нахилу близьким до 1 свідчить про самоподібність.

6. **Аналіз результатів**:

− Аналізується графік та кут нахилу, щоб визначити ступінь фрактальності часового ряду. Кут більше 1 вказує на позитивну кореляцію, а менше 1 – на негативну кореляцію.

− Індекс Херста обчислюється як нахил лінії на графіку. Це зазвичай виконується за допомогою методу найменших квадратів, щоб знайти коефіцієнт нахилу.

R/S-аналіз є одним із способів аналізу фрактальних часових рядів. Він допомагає визначити самоподібність, структуру кореляції в ряді та тренди, що може бути корисним для подальшого аналізу та прогнозування.

Індекс Херста H може бути меншим, більшим або дорівнювати 0.5, і це

вказує на різні ступені фрактальності:

- H < 0.5: Антікорельований (зворотно корельований) часовий ряд, тренд буде змінюватися.
- H = 0.5: Ряд білого шуму або випадковий ряд без кореляції.
- H > 0.5: Самоподібний або фрактальний часовий ряд, тренд буде тривати довго.

Розглянемо детальніше нормалізацію часового ряду перед R/S-аналізом.

Нормалізація часового ряду перед R/S-аналізом – це процес приведення часового ряду до стандартного вигляду, в якому середнє значення ряду дорівнює нулю, а стандартне відхилення – одиниці. Це важливо для того, щоб уникнути систематичних артефактів у результаті аналізу та зробити часовий ряд готовим для порівняння.

Основні кроки нормалізації часового ряду перед R/S-аналізом:

1. **Обчислення середнього значення**: Обчисліть середнє значення (середнє арифметичне) часового ряду. Для цього просто додайте всі значення ряду та поділіть на кількість точок у ряді.

$$X_{\text{середнє}} = \frac{1}{N}\sum_{i=1}^{N} X_i$$

де N – кількість точок у часовому ряду, а x_i – значення ряду у точці i.

2. **Віднімання середнього**: Від кожного значення ряду віднімiть середнє значення, щоб зробити середнє значення ряду рівним нулю.

$$x_{i,\text{нормалізований}} = x_i - X_{\text{середнє}}$$

3. **Обчислення стандартного відхилення**: Обчисліть стандартне відхилення нормалізованого ряду. Стандартне відхилення вимірює розкид значень відносно середнього. Для цього використовуйте наступну формулу:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_{i,\text{нормалізований}})^2}$$

де N – кількість точок у нормалізованому ряді, а $x_{i,\text{ нормалізований}}$ – нормалізоване

значення ряду у точці i.

4. **Нормалізація**: Поділіть кожне значення нормалізованого ряду на стандартне відхилення. Це зробить стандартне відхилення ряду рівним одиниці.

$$x_{i,\text{нормалізований}} = \frac{x_{i,\text{нормалізований}}}{\sigma}$$

R/S-аналіз у розробленому програмному забезпеченні реалізовано наступним чином:

```
import math, numpy, random
import pandas as pd


def normalize_traffic_dataset(file_name):
    """
    Зчитує датасет з файлу трафіку та нормалізує часовий ряд.

    :param file_name: Назва файлу з датасетом.
    :return: Нормалізований часовий ряд.
    """
    # Зчитуємо датасет з файлу
    dataset = pd.read_csv(file_name)

    # Визначаємо часовий інтервал між вимірами
    time_interval = dataset['time'].diff().mean()

    # Нормалізуємо часовий ряд
    normalized_traffic = dataset['packets'] / time_interval

    return normalized_traffic


def ArrayResize(mas, l, zap = 0):
    n = len(mas)
    if n < l:
        mas = mas + [zap]*(l-n)
    elif n < l:
        mas = mas[:l]
    return mas

LogReturns = []
```

```
N  = []
DevAccum = []
StdDevMas = []
pi = 3.14159265358979323846264338

exp = 1
close = [0]*exp
close[0] = normalize_traffic_dataset(file1.csv)

for i in range(len(close)):
    for j in range(len(close[i])):
        if close[i][j] == 5.0:
            close[i][j] = 2
        else:
            close[i][j] = 1

rs = []

#+------------------------------------------------------------------+
#| Функція старту                                                   |
#+------------------------------------------------------------------+

def OnStart(close, nn):
    global LogReturns, DevAccum, StdDevMas
    global rs

    num = [5, 10, len(close)]
    n_passes = len(num)
    #all_n = 1001
    all_n = len(close) + 1
    for_div = [int(all_n / num[i]) for i in range(0, n_passes)]

    rs = [[0] for i in range(0,n_passes)]
    close = ArrayResize(close, all_n, 1)


#+------------------------------------------------------------------+
#| Підготовка масивів                                               |
#+------------------------------------------------------------------+

    LogReturns = ArrayResize(LogReturns, all_n)
    DevAccum = ArrayResize(DevAccum, all_n)
```

```
        StdDevMas = ArrayResize(StdDevMas, all_n)

    #+------------------------------------------------------------------+
    #| Масив логарифмічних значень                                      |
    #+------------------------------------------------------------------+

        for i in range(1, all_n - 1):
            try:
                LogReturns[i] = numpy.log(close[i-1] / close[i])
            except:
                LogReturns[i] = 0

    #+------------------------------------------------------------------+
    #|                          R/S-аналіз                              |
    #+------------------------------------------------------------------+

    #--- Задамо кількість елементів у кожній підгрупі

        LogRS = [0] * n_passes



    #--- Розрахунок складових Log(R/S)

        for A in range(0, n_passes):
            RSsum = 0.0
            for j in range(1, for_div[A]+1):
                if (len(close)-1) % num[A] != 0 and j == for_div[A]:
                    rs[A].append( RSculc(num[A] * j - num[A] + 1 ,len(close) -
1, (len(close)-1) % num[A]) )
                else:
                    rs[A].append(RSculc(num[A] * j - num[A] + 1 , num[A] * j,
num[A]))
                RSsum = RSsum + rs[A][j]
            try:
                RS = RSsum / for_div[A]
            except:
                RS = 0
            LogRS[A] = numpy.log(RS)

    #+------------------------------------------------------------------+
    #|  Розрахунок коефіцієнта Херста                                   |
```

```
#+--------------------------------------------------------------------+

    H = RegCulc(LogRS)

#+--------------------------------------------------------------------+
#| Розрахунок очікуваних значень log(E(R/S))                          |
#+--------------------------------------------------------------------+

    E = []
    for i in range(n_passes):
        E.append(numpy.log(ERSculc(num[i])))

#+--------------------------------------------------------------------+
#|  Розрахунок бети очікуваних значень E(R/S)                         |
#+--------------------------------------------------------------------+

    betaE = RegCulc(E)
    print(nn, "H = " + str(H),",  E = " + str(betaE))
#    for ii in range(n_passes):
#        print(num[ii], LogRS[ii])


#+--------------------------------------------------------------------+
#|  Функція розрахунку R/S                                            |
#+--------------------------------------------------------------------+

def RSculc(bottom, top, barscount):
    global LogReturns, all_n
    MaxValue = 0.0
    MinValue = all_n - 1
    Sum = 0.0               # Початкове значення суми дорівнює нулю
    DevSum = 0.0            # Початкове значення суми накопичених
                            # відхилень дорівнює нулю
#--- Розрахунок суми накопичень
    for i in range(bottom, top+1):
        Sum = Sum + LogReturns[i]          #Накопичені суми

#--- Розрахунок середнього
    try:
        M = Sum / barscount
    except:
        M = 0
```

```
    #--- Розрахунок накопичених відхилень
        for i in range(bottom,top+1):
            DevAccum[i] = LogReturns[i] - M + DevAccum[i-1]
            StdDevMas[i] = math.pow((LogReturns[i] - M), 2)
            DevSum = DevSum + StdDevMas[i]               # Складова для
розрахунку відхилення
            if DevAccum[i] > MaxValue: #Якщо значення в масиві менше деякого
                MaxValue = DevAccum[i] #максимального, то присвоюємо
максимальному значенню значення елемента масиву DevAccum
            if DevAccum[i] < MinValue:                  #Логіка дій аналогічна
                MinValue = DevAccum[i]

    #--- Розрахунок розмаху R та відхилення S

        R = MaxValue - MinValue       #Розмах дорівнює різниці максимального і
        MaxValue = 0.0
        MinValue = 1000                 #мінімальних значень
        S1 = math.sqrt(DevSum / barscount)  #Розрахунок стандартного відхилення
    #--- Розрахунок показника R/S
        if S1 != 0:
            RS = R / S1                 #Виключаємо помилку поділу на "нуль"
    #    else Alert("Zero divide!")
        else:
            RS = 0
        return RS                       #Повертаємо значення RS-статистики

    #+------------------------------------------------------------------+
    #|  Калькулятор регресії                                            |
    #+------------------------------------------------------------------+

    def RegCulc(Y):
        global N
        global num, n_passes
        SumY = 0.0
        SumX = 0.0
        SumYX = 0.0
        SumXX = 0.0
        b = 0.0  #масив, в якому лежатимуть логарифми дільників масив дільників



    #--- Розрахунок коефіцієнтів N
        N = ArrayResize(N, n_passes)
```

```
    for i in range(0, n_passes):
        N[i] = numpy.log(num[i])
        SumX = SumX + N[i]
        SumXX = SumXX + N[i] * N[i]

    for i in range(len(Y)):
        SumY += Y[i]
        SumYX += Y[i]*N[i]
#--- Розрахунок коефіцієнта Beta регресії або шуканого показника Херста
    try:
        b = (n_passes*SumYX-SumY*SumX)/(n_passes*SumXX-SumX*SumX)
    except:
        b = 0
    return b

#+------------------------------------------------------------------+
#| Функція розрахунку очікуваних значень E(R/S)                     |
#+------------------------------------------------------------------+

def ERSculc(m):                 #m - дільники 1000
    nSum = 0.0
    part = 0.0
    for i in range(1, m):
        part = math.pow(((m-i)/i), 0.5)
        nSum = nSum + part
    e = math.pow((m*pi/2),-0.5)*nSum
    return e


for i in range(exp):
    OnStart(close[i], i)
```

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення ліцензією є важливою складовою для забезпечення легального використання та контролю над продуктом. Ліцензія встановлює правила та умови, за якими користувачі можуть використовувати програмне забезпечення, а також визначає їхні обов'язки та

обмеження. Ось деякі кроки та рекомендації щодо захисту програмного забезпечення ліцензією:

1. **Вибір типу ліцензії**. По-перше, треба вибрати тип ліцензії, який найкраще відповідає потребам власника ПЗ. Декілька поширених типів ліцензій включають в себе відкриті ліцензії (наприклад, MIT, GPL), пропрієтарні ліцензії, комерційні ліцензії та ліцензії з відкритим джерелом. Кожен тип має свої умови та вимоги.

2. **Написання тексту ліцензії**. Далі треба ретельно сформулювати текст ліцензії, в якому визначаються умови використання, обов'язки користувачів, права та обмеження. Текст ліцензії повинен бути чітким та зрозумілим.

3. **Реєстрація ліцензії**. Деякі ліцензії вимагають реєстрації від автора або розробника. В цьому випадку, слід виконати процедуру реєстрації та отримати відповідний номер ліцензії.

4. **Розповсюдження ліцензії**. Слід повідомити користувачів про умови ліцензії та забезпечити їм доступ до тексту ліцензії разом з програмним забезпеченням.

5. **Механізми перевірки ліцензій**. Якщо необхідно обмежити використання програмного забезпечення лише тим, хто має ліцензію, то слід розглянути можливості використання механізмів перевірки ліцензій, таких як генерація ключів або перевірка через Інтернет.

6. **Оновлення ліцензії**. При необхідності треба оновлювати текст ліцензії, щоб вона відповідала зміненим умовам або вимогам. Оновлені версії ліцензії повинні бути доступні користувачам.

7. **Документація та інструкції**. Треба включити до поставки програмного забезпечення документацію та інструкції з використання, щоб користувачі були повністю ознайомлені з умовами ліцензії.

8. **Використання підпису електронного документа**. Підпис електронного документа може бути використаний для забезпечення цілісності та автентичності ліцензії.

Захист програмного забезпечення ліцензією допоможе контролювати його використання та забезпечити дотримання умов використання. Важливо ретельно розробити та документувати ліцензію, а також вжити заходів для забезпечення її дотримання.

Для розробленого програмного забезпечення було обрано ліцензію на вільне використання GNU General Public License.

Текст ліцензії можна підписати цифровим підписом RSA для подальшого підтвердження його достовірності.

Для реалізації електронного підпису ліцензії алгоритмом RSA, знадобляться наступні кроки:

1. **Генерація ключів RSA**. Спершу потрібно згенерувати пару ключів RSA, яка складається з приватного ключа та публічного ключа. Приватний ключ служитиме для підпису, а публічний – для перевірки підпису. Скористаємося бібліотекою **cryptography** для цієї мети. Ось приклад коду для генерації ключів:

```
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa

# Генерація ключів
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
)

# Збереження приватного ключа в файл
private_pem = private_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.PKCS8,
    encryption_algorithm=serialization.NoEncryption()
)
with open('private_key.pem', 'wb') as f:
    f.write(private_pem)

# Отримання публічного ключа
public_key = private_key.public_key()
public_pem = public_key.public_bytes(
```

```
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
)
with open('public_key.pem', 'wb') as f:
    f.write(public_pem)
```

2. **Підпис ліцензії**. Підпишемо текст ліцензії приватним ключем RSA. Ось приклад підпису:

```
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding

# Завантаження приватного ключа
with open('private_key.pem', 'rb') as f:
    private_key = serialization.load_pem_private_key(
        f.read(),
        password=None
    )

# Текст ліцензії, який потрібно підписати
license_text = "Текст вашої ліцензії"

# Підпис
signature = private_key.sign(
    license_text.encode('utf-8'),
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)

# Збереження підпису
with open('license_signature.bin', 'wb') as f:
    f.write(signature)
```

3. **Перевірка підпису**. Щоб інші користувачі могли перевірити підпис, потрібно надати їм ваш публічний ключ та підпис. Ось приклад перевірки:

```
from cryptography.hazmat.primitives import serialization
```

```
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes

# Завантаження публічного ключа
with open('public_key.pem', 'rb') as f:
    public_key = serialization.load_pem_public_key(f.read())

# Завантаження підпису
with open('license_signature.bin', 'rb') as f:
    signature = f.read()

# Текст ліцензії, який потрібно перевірити
license_text = "...Текст ліцензії..."

try:
    public_key.verify(
        signature,
        license_text.encode('utf-8'),
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    print("Підпис валідний. Ліцензія дійсна.")
except Exception:
    print("Підпис не валідний. Ліцензія недійсна.")
```

Цей приклад демонструє, як створити, поставити та перевірити цифровий підпис за допомогою алгоритму RSA.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Для використання розробленого програмного забезпечення необхідно встановити наступні бібліотеки для мови Python:

− **pandas** – це бібліотека для обробки та аналізу даних в Python. Вона надає структури даних, такі як DataFrame (таблиці з даними) та Series (одномірні масиви даних), які дозволяють зручно працювати з даними, виконувати фільтрацію, обчислення та візуалізацію. Вона дозволяє зчитувати та записувати дані з і до різних джерел, таких як CSV-файли, Excel-файли, бази даних і багато інших. Зручна для операцій з даними, такими як сортування, фільтрація, групування, агрегація і об'єднання даних. Широко використовується в аналітиці даних, машинному навчанні, фінансах і багатьох інших областях.

− **sqlite3** - це модуль Python для роботи з базами даних SQLite. Завдяки бібліотеці sqlite3, можна створювати, зчитувати, оновлювати та видаляти дані з бази даних SQLite без необхідності встановлювати окремий сервер баз даних. Він дозволяє виконувати SQL-запити до бази даних, здійснювати транзакції та керувати структурою бази даних. Модуль sqlite3 зручно використовувати для збереження даних на локальному рівні, зокрема, збереження конфігураційних параметрів, журналів або інших локальних даних.

Впровадження розробленої системи виявлення DDoS-атак у промислову експлуатацію вимагає уважного планування та виконання кількох етапів. Нижче подано загальний опис методики впровадження розробленої системи:

1. **Аналіз потреб**. Перед початком роботи над системою важливо провести аналіз потреб організації або клієнта. Визначити типи DDoS-атак, які потрібно виявляти, та рівень захисту, який необхідно забезпечити.

2. **Аналіз розробленої системи**. Слід переконатися, що розроблена система для виявлення DDoS-атак відповідає потребам організації. Вона повинна бути

налаштована для виявлення конкретних видів атак і відповідати потужністю мережі.

3. **Тестування**. Перш ніж впроваджувати систему в реальне виробництво, необхідно провести відповідне тестування. Систему слід перевірити на тестових серверах та імітувати різні сценарії атак для переконання в її ефективності.

4. **Інтеграція**. Треба інтегрувати систему для виявлення DDoS-атак у існуючу мережеву інфраструктуру організації. Це може включати налаштування мережевого обладнання та інші необхідні дії для взаємодії з системою.

5. **Моніторинг і налаштування**. Слід постійно моніторити роботу системи та налаштовувати її параметри відповідно до змін у мережевому трафіку і структурі атак. Для цього можна використовувати системи моніторингу мережі.

6. **Навчання персоналу**. Необхідно організувати навчання для персоналу, який відповідає за моніторинг та управління системою виявлення DDoS-атак. Вони повинні знати, як реагувати на атаки та аналізувати результати системи.

7. **Тестова експлуатація**. Перед повним впровадженням рекомендується провести тестовий період експлуатації, під час якого система фактично використовується на обмеженій кількості серверів або мережевих сегментах.

8. **Повне впровадження**. Після успішного тестового періоду можна перейти до повного впровадження системи на всій мережі або середах, які потребують захисту.

9. **Постійний моніторинг і оновлення**. Після впровадження систему слід постійно моніторити, оновлювати та адаптувати до нових загроз. DDoS-атаки постійно еволюціонують, тому систему слід підтримувати в актуальному стані.

10. **Резервне копіювання і відновлення**. Рекомендовано розробити процедури резервного копіювання та відновлення даних системи.

Вхідними даними до програми є часовий ряд трафіку. Вихідними – інтенсивність трафіку, індекс Херста часового ряду трафіку та ймовірність наявності DDoS-атаки («низька», «середня», «висока»). Приклади вхідних даних, згенерованих в процесі симуляції трафіку, та вихідні дані наведені на рис. 5.1-5.3.

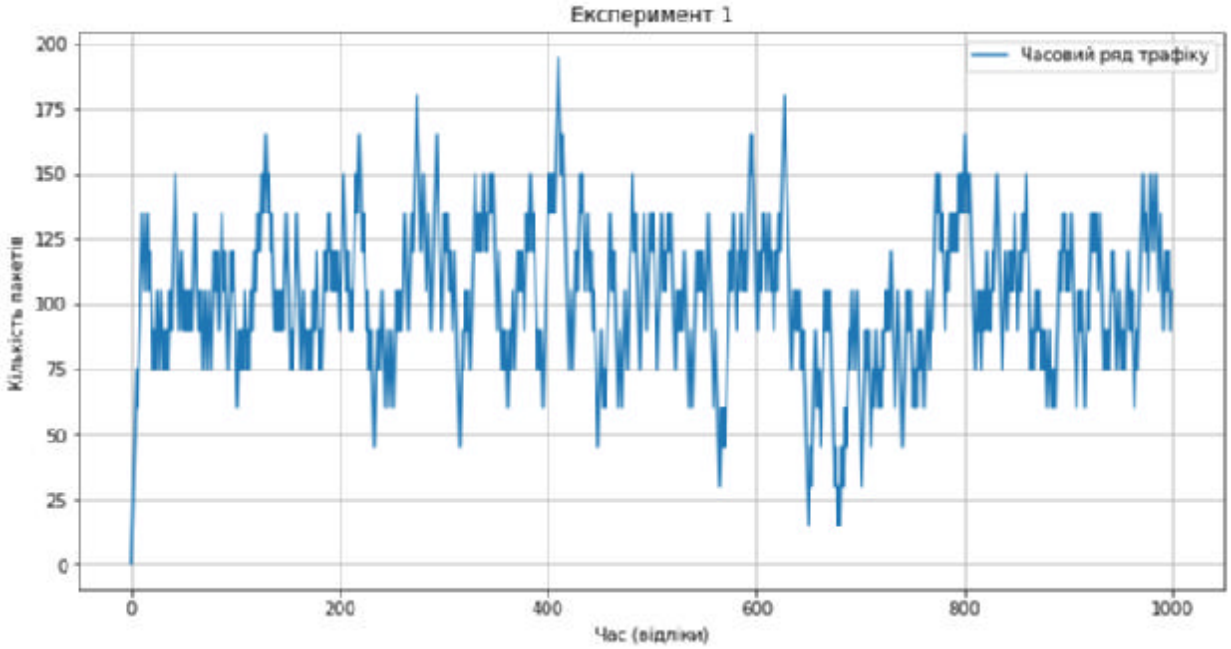

Рисунок 5.1 – Згенерований часового ряду трафіку: експеримент №1

Результати аналізу трафіку в експерименті №1:

– Інтенсивність трафіку 0.5239.

– Індекс Херста 0.6083.

– Ймовірність Ddos-атаки «низька».

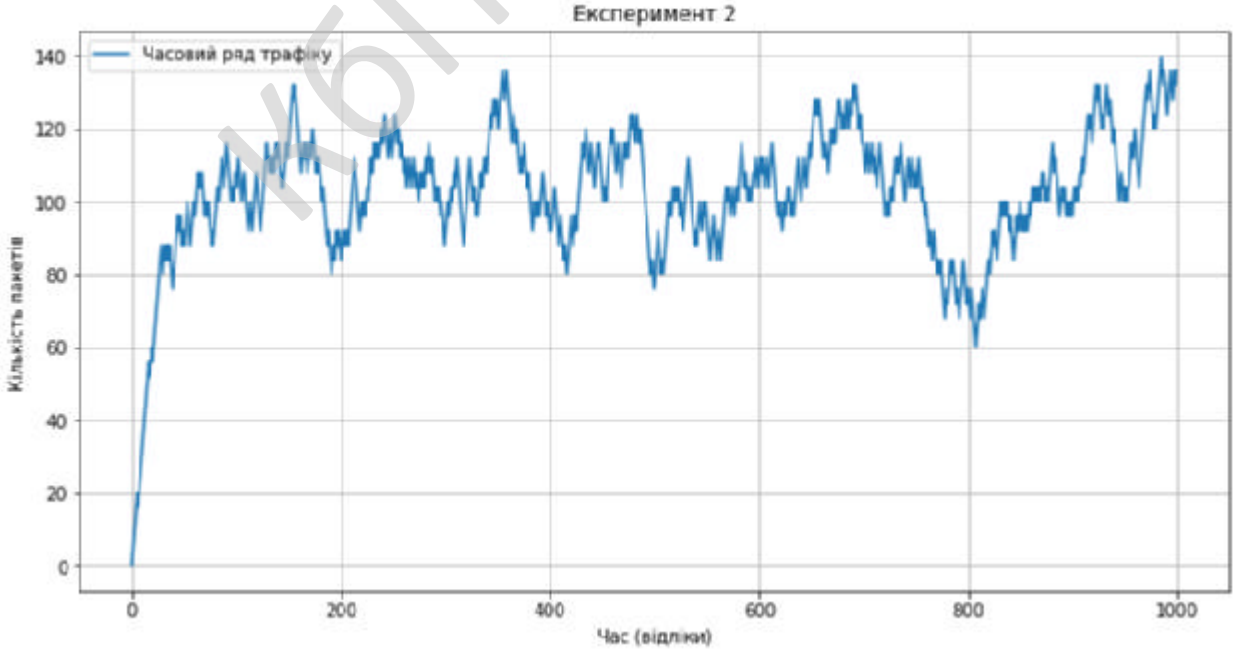

Рисунок 5.2 – Згенерований часового ряду трафіку: експеримент №2

Результати аналізу трафіку в експерименті №2:

– Інтенсивність трафіку 0.7328.

– Індекс Херста 1.

– Ймовірність Ddos-атаки «висока».

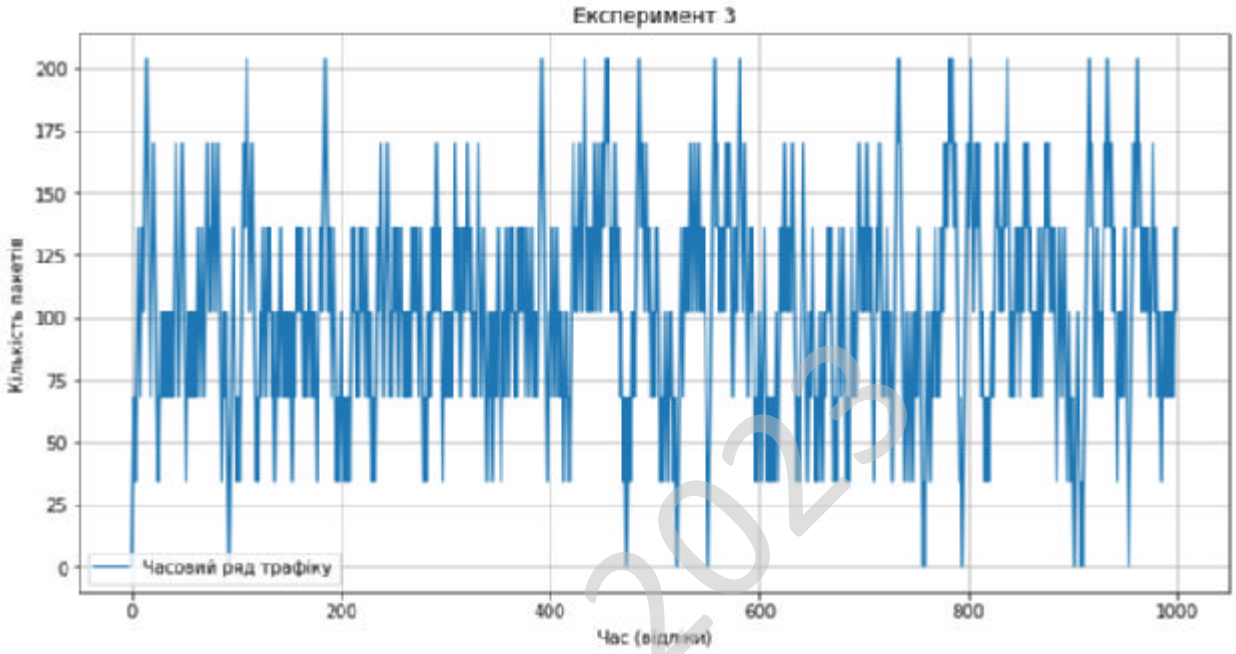

Рисунок 5.3 – Згенерований часового ряду трафіку: експеримент №3

Результати аналізу трафіку в експерименті №3:

– Інтенсивність трафіку 0.5013.

– Індекс Херста 0.512.

– Ймовірність Ddos-атаки «низька».

6 НАУКОВА НОВИЗНА

Метою роботи є дослідження та програмна реалізація інтелектуальної системи для виявлення DDoS-атак.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Дослідження існуючих систем для виявлення DDoS-атак.

– Розробка методів та алгоритмів для інтелектуальної системи виявлення DDoS-атак.

– Програмна реалізація інтелектуальної системи виявлення DDoS-атак.

Об'єктом дослідження є процес виявлення DDoS-атак.

Предметом дослідження є методи аналізу часових рядів трафіку для виявлення інформаційних атак.

Методи дослідження базуються на теорії комп'ютерних мереж, теорії аналізу часових рядів, теорії математичної статистики, теорії алгоритмів і структур даних та методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Запропоновано метод виявлення DDoS-атак на основі R/S-аналізу часових рядів мережевого трафіку.

2. Розроблено вітчизняний продукт виявлення DDoS-атак, який має більш широкі можливості, на відміну від існуючих аналогів та може бути використаний при розробці методів захисту веб-сайтів від інформаційних атак.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі виявлення DDoS-атак на веб-сайти.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми магістерської роботи

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 48 днів (два місяці).

В магістерській роботі проведене дослідження та виконана програмна реалізація інтелектуальної системи для виявлення DDoS-атак.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність

Таблиця 7.1 – Початкові данні

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт	N	1
2. Кількість екземплярів програм, шт	Ne	57
3. Запланований термін розробки, днів	Fpq	48 (2 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2
7. Кількість макетів вхідної інформації	–	3

Продовження таблиці 7.1

1	2	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	3
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	_	2
13. Рівень спрацьованості колективу (1-6)	_	2
14. Ступінь вимірності процесів (1-6)	_	3
15. Необхідна надійність програмного забезпечення (1-6)	_	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	_	2
17. Складність кінцевого програмного продукту (1-6)	_	2
18. Необхідний рівень забезпечення повторного використання (1-6)	_	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	_	2
20. Вимоги до швидкодії ПП (1-6)	_	2
21. Обмеження на розміри основного сховища даних (1-6)	_	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	_	2
23. Професійний рівень аналітиків (1-6)	_	2
24. Професійний рівень програмістів (1-6)	_	2
25. Постійність складу команди розробників (1-6)	_	2
26. Досвід розробки додатків (1-6)	_	2
27. Досвід роботи з обчислювальною платформою (1-6)	_	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПО для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн	–	60000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	50
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A\ \text{Size}^B \qquad (7.1)$$

де А – коефіцієнт Боема, А=2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням

$$B = 1{,}01 + 0{,}001 \sum W_i \quad (7.2)$$

де Wi – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B=1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}026$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн}=T_{ном}\, ПВ_j, \quad (7.3)$$

де $ПВ$j – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн}= 6{,}78 \cdot (0{,}88 \cdot 0{,}93 \cdot 0{,}88 \cdot 0{,}91 \cdot 0{,}95 \cdot 1 \cdot 1 \cdot 0{,}87 \cdot 1{,}22 \cdot 1{,}16 \cdot 1{,}1 \cdot 1{,}1 \cdot$$

$$\cdot 1{,}12 \cdot 1{,}1 \cdot 1 \cdot 1{,}1 \cdot 1{,}1) = 9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+\ ,2(B-1{,}01)} S, \quad (7.4)$$

де C – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); S - коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПО згідно встановленим вимогам. Вибираємо в межах (25…350)%

$$T_{РП}=0{,}3 \cdot 2{,}85 \cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)} \cdot 55 = 99 \text{ люд/день}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	99	Ф 7.1-7.4
Впровадження	13	Д13
Всього	140	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою

$$Ч = \frac{T_{пз}N}{F_{pq} - H_{ев}}, \quad (7.5)$$

де F_{pq} – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{1\ 0{\cdot}1}{48\text{-}5} = 3{,}25 \text{ ставки}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	6	540	9
Монітор	60	6	360	6
Клавіатура	30	6	180	3
Маніпулятор «мишка»	30	6	180	3
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор– маршрутизатор	30	1	30	0,5
Кабельні господарства ЛВС на 1 м. п.	2,5	140	350	5,83
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	32,99

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{mic}}{1,2} \quad (7.6)$$

$$Ф_{др}^{с} = \frac{33 \cdot}{1,2} = 82,5 \text{ год}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф_{др}^{с}}{F_{др} \cdot T_{зм}} \quad (7.7)$$

$Ч_{ел}$=82,5/(48·8)=0,2 ставки

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів–електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2019, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	0,8	0,2
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,2	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,2	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	0,4	
Всього		1,6	

Продовження таблиці 7.4

Посада	Вид роботи	Час	Кількість штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	2	0,5
	Підтримка постійних клієнтів	1	
	Оформлення договорів, ведення тендерів	0,5	
	Контроль взаєморозрахунків з постачальниками	0,5	
Всього		4	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	0,5	0,2
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,3	
	Розміщення графіки і контенту на Інтернет сторінках	0,3	
Всього		1,6	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,2
	Верстка друкованих видань	0,2	
	Додрукова підготовка макетів	0,2	
	Розміщення графіки і контенту на Інтернет сторінках	0,2	
Всього		1,6	

Складемо штатний розклад виконавців у таблицю 7.5.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньо-місячний оклад, грн.	Всього за період роз-робки, грн.
Керівник (IT-менеджер)	1	22604	45208
Продакт-менеджер	0,5	18000	18000
Інженер-програміст	3,25	20000	130000
Інженер-електронщик	0,2	15000	6000
Інженер-системотехнік	0,2	15000	6000
Адміністратор мережі	0,2	14000	5600
Системний програміст	0,2	14000	5600
Дизайнер WEB	0,2	14000	5600
Інженер-верстальник	0,2	14000	5600
Бухгалтер-економіст	0,2	14000	5600
Всього за період розробки	$R_{сп}$=6,15	-	$Ф_{роб}$=233208

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_c \ F_{pq}}, \tag{7.8}$$

де $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{2\ 3\ 0}{6{,}15 \cdot\ 8} = 790\ грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі і розраховується за формулою (7.9).

$$Б_{уд} = R_{сп}^{1} S_у Ц_{пл}, \quad (7.9)$$

де $R_{сп}^{1}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць.

$S_у$ – питома площа на одне робоче місце, $м^2$,

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$.

Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно $8м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн на одне робоче місце. Тобто

$$I_{нв} = R_{сп}^{1} \cdot Ц_{м}, \quad (7.10)$$

де Цм – ціна меблів для одного робочого місця, грн.

$$I_{нв} = 8 \cdot 3500 = 28000 \text{ грн}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7. Дані по оптовій ціні на обладнання та комплектуючі вибирались за комерційною пропозицією фірми Brain за 14.10.23 – джерело http://brain.com.ua/

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		11771
Системний блок		7771
Процесор	INTEL Core™ i3 10105 (BX8070110105) 1200, 4 ядра, 8 потоків, 3.7 GHz, 4.4 GHz, TDP - 65 Вт, 14nm, 6 MB Intel Smart Cache, 8 GT/s, BOX	-
Системна плата	GIGABYTE H470M H сокет - 1200, DDR4, 3200 MHz, LAN - 1 Гбит/с, D-Sub (VGA), HDMI, 1 x M.2 2280, 4 x Sata 6.0 Gb/s	-
Жорсткий диск	SSD 2.5" 256GB Mibrand (MI2.5SSD/CA256GBST) 256 GB, 3D TLC NAND, 2.5", SATA III (6Gb/s)	-
Оперативна пам’ять	DDR4 8GB 2666 MHz eXceleram (E408266A)	-
Корпус	Vinga CS105B-450W, ATX, PSU - 450 Вт	-
Кардрідер внутрішній	Transcend TS-RDF8K USB 3.0	-
інше	Клавіатура, мишка	-
Монітор	Монітор BenQ GL2450HM Black	2600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Сканер	Epson Perfection V37	2800
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Пристрій безперебійного живлення	Powercom BNT-600AP USB	1400

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 - Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	8	11771	9416,8	103584,8
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	1	2800	280	3080
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	125216,3

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400

Продовження таблиці 7.8

1	2	3	4
Група 4			
3. Обчислювальна техніка	125216	-	-
Всього по групі	125216	50	62608
Група 5,6			
4. Вимірювальні пристрої	5190	-	-
5. Транспортні засоби	143000	-	
6. Господарський інвентар	28000	-	-
Всього по групі	176190	20	35238
7. Нематеріальні активи	60000	10	6000
Разом	$К_р$= 1769406		$А_р$= 174246

Примітка: вартість автомобіля Chevrolet Aveo 2008 взята за даними сайту «Авто-Ріа», джерело https://auto.ria.com/uk/auto_chevrolet_aveo_32795647.html, складає 143000 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де $N_е$ – Кількість екземплярів програм, шт.

$$З_о = 790 \cdot 140/57 = 1940 \text{ грн}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де $Н_q$ – норматив додаткової зарплати, %

$$З_д = 1940 \cdot 10 \cdot 0{,}01 = 194 \text{ грн}$$

Відрахування на соціальні потреби за нормативом Нс=22% від суми основної та додаткової зарплати

$$С_{оц} = 0{,}01 \cdot Н_с(З_о + З_д), \quad (7.13)$$

де $Н_с$ – відрахування на соціальні потреби, %

$$С_{оц} = 0{,}01 \cdot 22(1940+194) = 470 \text{ грн}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом *Нг*=15% від основної зарплати

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де $Н_г$ – загальногосподарські витрати, %

$$Г_{осп} = 1940 \cdot 15 \cdot 0{,}01 = 291 \text{ грн}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де $З_{М1}$ – вартість паперу, грн., $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн., $З_{М3}$ – вартість фарби, картриджів, тонеру, грн., $N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 0,4 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=200 грн., визначаємо вартість паперу за період розробки:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.16)$$

$$З_{М1} = 200 \cdot 0{,}4 = 80 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 10):

$$З_{М2} = \sum Ц_д, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 23,6 грн./шт., DVD-R box – 46,6 грн./шт.

$$З_{М2} = 46{,}6 \cdot 10 = 466 \text{ грн.}$$

Згідно норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{МЗ} = \sum Ц_з., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{МЗ} = 574+558+570=1702 \text{ грн.}$$

$$З_М = (80 + 466 + 1702)/57=39 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \quad (7.19)$$

де $Н_п$ - норматив витрат на освоєння нових мов програмування, %

$$О_п = 1940 \cdot 15 \cdot 0{,}01=291 \text{ грн}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 57 прим.)

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \quad (7.20)$$

де $А_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 174246 \cdot 2/(57 \cdot 12)=509 \text{ грн}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \quad (7.21)$$

$$С_п = 1940+194+470+291+39+291+509=3734 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності (Рп) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 50%

$$П_р = 0{,}01 \cdot \_n \cdot \_n, \quad (7.22)$$

де $Р_с$ – рівень рентабельності, %

$$П_р = 0{,}01 \cdot 50 \cdot 3734 = 1867 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
1. Основна зарплата виконавців	$З_о$	1940
2. Додаткова зарплата виконавців	$З_д$	194
3. Відрахування на соціальні потреби	$С_{оц}$	470
4. Загальногосподарські витрати	$Г_{осп}$	291
5. Витрати на матеріали	$З_М$	39
6. Освоєння нових операційних систем, мов програмування	$О_п$	291
7. Амортизація основних фондів	$А_м$	509
8. Повна собівартість програмного забезпечення	$С_п$	3734
9. Плановий прибуток	$П_р$	1867
10.Ціна підприємства $Ц_п=С_п+П_р$	$Ц_п$	5601
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	1120,2
12.Відпускна ціна програмної продукції Ц = Цп + ПДВ	$Ц$	6721,2

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 – Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн	
	Базовий	Новий
Вартість програмної продукції	–	6721
Всього капітальних витрат	–	6721

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 - Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позна-чення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування	$З_р$	40260	26840
2. Витрати на електроенергію	$З_{ел}$	4608	4224
3. Витрати на амортизацію	$З_{ам}$	0	3361
Всього витрат за рік	$І$	44868	34425

Витрати на технічне обслуговування:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_с), \quad (7.23)$$

де $Т_р$ – кількість годин обслуговування системи за рік, *год*.,

$З_г$ – заробітна плата обслуговуючого персоналу, $грн/год$

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 300 годин на рік до 200 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\ баз} = 300 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 40260\ грн.$$

до:

$$З_{р\ нов} = 200 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 26840\ грн.$$

Витрати на електроенергію визначаються з урахуванням спожитої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$).

$$З_{ел}=П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

Зел баз=0,2·7200·3,2=4608 грн

Зел нов=0,2·6600·3,2=4224 грн

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 - Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	50	–	6721	–	3360,5
Всього відрахувань	-	–	6721	–	3360,5

Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (5601 - 3734) \cdot 57 - (0{,}05 \cdot 1408000 + 0{,}5 \cdot 125216 +$$

$$+0{,}2 \cdot 176190 + 0{,}1 \cdot 60000) \frac{\cdot 2}{12} = 77378 \ грн.$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_p^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: K_p^* – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх строк служби на порядок більший ніж період розробки ПЗ.

$$T_в = \frac{36\ 4\ 6}{(5\ 01 - 734) \cdot 5\ \cdot 12/2} = 0{,}6 \ років$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно, $K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються

$$E_{сп} = (44868\text{-}34425)\text{-}0{,}5 \cdot 6721 = 7083 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н} \quad (7.28)$$

$$T_{сп} = \frac{67\ \ 1}{44868\ \ 344\ \ 5} = 0{,}64\ років$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	57
2. Повна собівартість розробленої програми	Грн.	3774
3. Ціна розробленої програми	Грн.	5601
4. Плановий прибуток від реалізації розробленої програми	Грн.	1827
5. Рентабельність програмної продукції	%	50
6. Об’єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1769406
7. Загальний прибуток від реалізації програмної продукції	Грн.	104139
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	77378
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	0,6
10.Об’єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	6721
11.Величина економічного ефекту у користувача програмної продукції	Грн.	7083
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,64

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1. Вступ

Охорона праці – система збереження життя і здоров'я працівників у процесі трудової діяльності, що включає правові, соціально-економічні, організаційні, технічні, санітарно-гігієнічні, лікувально-профілактичні, реабілітаційні та інші заходи.

Згідно закону України “Про охорону праці” кожна компанія впроваджує заходи з охорони праці. Реалізується трудові відносини з вживанням необхідних засобів з охорони праці та розробки відповідних документів:

- Інструкцій з охорони праці по кожній професії і загальні;
- Положення про охорону праці;
- Накази з охорони праці;
- Журнали реєстрації та інструктажу.

Роботодавець створює відділ який працює відповідно до типового положення, яку затверджується центральним органом виконавчої влади і забезпечує виконання вимог державної політики у сфері охорони праці.

За недотриманням вимог, керівники ІТ компаній можуть бути притягнуті до відповідальності, яка виглядає у виді накладання штрафу. Якщо в результаті порушення умов охорони праці є постраждалі працівники то керівні особи ІТ компаній притягуються до кримінальної відповідальності.

Законом України “Про охорону праці” регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями»,

яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби польців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робитників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

Науково-технічний прогрес вніс серйозні зміни в умови виробничої діяльності робітників розумової діяльності. Їх праця стала більш інтенсивною, напруженою і вимагає значних витрат розумової, емоційної і фізичної енергії. Це призвело до необхідності у знаходженні комплексного рішення проблем ергономіки, гігієни і організації праці, регламентації режимів праці та відпочинку. Охорона здоров'я робітників, забезпечення безпеки умов праці, ліквідація та профілактика професійних захворювань і виробничого травматизму складає одну з головних турбот людського суспільства.

Пожежна безпека

Вимоги до пожежної безпеки на підприємстві неухильно повинен дотримуватися кожен співробітник, а організаційна складова при цьому покладається на посадових осіб за відповідним рішенням керівництва і прописується в посадових інструкціях і положеннях по структурним підрозділам.

Зокрема, вказуються конкретні території, ділянки, зони, об’єкти, цілі будівлі і їх частини, поверхи, на яких відповідального співробітника повинне проводити такі організаційні роботи.

Відповідальні особи зобов'язуються розробити, впровадити та підтримувати в певному інструкцією і положенням на ввірених їм об'єктах протипожежний режим і інструкції відповідно до вимог, викладених в нормативних актах.

Передбачено також створення підрозділу добровільної пожежної охорони та пожежно-рятувальної команди в його складі.

Встановлений режим включає порядки з описом місць спеціального призначення та правила їх користування та утримання, наприклад:

- евакуаційних шляхів;
- так званих «курилок»;
- місць складування продукції та сировини;
- стоянки транспорту.

Також встановлюється порядок роботи та технічного обслуговування:

- вентиляційного устаткування;
- засобів пожежогасіння і захисту від загорянь;
- нагрівальних приладів;
- електрообладнання.

Розробляються і впроваджуються правила роботи з відкритим вогнем і горючими матеріалами. Створюються графіки проходження інструктажів з пожежної безпеки співробітників, а також порядок і терміни перевірок знань пожежно-технічного мінімуму, в тому числі, тих працівників, які відповідальні за цю ділянку роботи на підприємстві. При цьому можуть передбачатися внутрішні лекції, семінари, тренінги та практичні заняття на підприємстві, а також зовнішні – на базі спеціалізованих навчальних центрів з професійними викладачами.

Важливою складовою протипожежного режиму на будь-якому об'єкті є розробка і впровадження порядку дій при виникненні пожежі. Неодмінно має бути план евакуації, описано, як повинні відключатися електроустановки, що і в якій послідовності необхідно робити співробітникам.

Відповідно, для кожного об'єкта, кожного приміщення (крім коридорів,

санвузлів, басейнів і подібних приміщень), окремих видів робіт складаються інструкції, за якими повинен працювати персонал, залучений на певних ділянках і в виконанні окремих видів робіт. За інструкціями проводиться навчання (інструктаж) персоналу з подальшим контролем знань.

Детально про те, як розробити протипожежний режим, прописати порядки та інструкції, пояснюють на тематичних курсах і семінарах.

8.3. Характеристика умов праці програміста

В приміщенні, в якому проводиться розробка і дослідження програмного продукту, відсутні умови, які можуть створювати підвищену або особливо підвищену небезпеку, тому воно відноситься до класу звичайних приміщень згідно Правил улаштування електроустановок (ПУЕ). Джерелом живлення є трифазна мережа напруги 380/220 В з глухо заземленою нейтралі, з частотою 50 Гц згідно За пожежо-вибухонебезпекою приміщення відноситься до класу В. В таблиці 8.2 наведена загальна характеристика приміщення щодо вибухопожеженобезпеки та важкістю робіт.

Таблиця 8.2 – Загальна характеристика приміщення щодо вибухопожежо-небезпеки та важкістю робіт

Характеристика приміщень за вибухопожежною категорією та класом зони	**Загальна характеристика приміщення**	**Категорія за важкістю робіт згідно ГН 3.3.5-8.6.6.1 -2002**
В – пожежонебезпечне клас П – II	Звичайне без ознак хімічного забруднення та нормальної вологості і за санітарними нормами	1а…….до 139 Вт/м^2 1б…….до 140-174 Вт/м^2 Клас умов праці – оптимальний

Температура повітря в приміщенні визначається температурою зовнішнього повітря і тепловою енергією, що виділяється всередині приміщення. Джерелами теплоти в даному приміщенні є люди, електроустаткування, а також освітлювальні прилади в темний час доби. Зовнішнім джерелом надлишкового тепла є сонячна радіація у світлий час доби. Робота, виконувана в даному приміщенні, відноситься до категорії I-а. Людиною в цьому випадку виділяється до 120 ккал теплової енергії в годину. Вологість повітря в приміщенні визначається вологістю атмосферного і видихуваного людьми повітря, а також випарами з поверхні шкіри.

У приміщенні немає виділення шкідливих газів. Тому що в ньому не проводиться монтажних робіт, пайки чи інших робіт, при яких виділяються шкідливі гази. Для нормалізації параметрів повітряного середовища також періодично здійснюється провітрювання приміщення і вологе прибирання. У всьому будинку діє встановлена загально обмінна витяжна вентиляція.

Раціональне освітлення приміщення сприяє кращому виконанню виробничого завдання і забезпеченню комфорту при роботі. Для забезпечення нормального освітлення застосовуються природне, однобічне, бічне і штучне освітлення, а також сполучене, нормуються зідно ДБН В.2.5-28-2006 Природне і штучне освітлення. За результатами виміру освітленості величина освітленості від системи загального штучного освітлення дорівнює 310 лк, що відповідає вимогам, які пред'являються до даного приміщення.

Основними джерелами шуму на робочих місцях, обладнаних відео дисплейними терміналами, є принтер, сканер факс і обладнання для кондиціювання повітря, в самих відео дисплейних терміналів – вентилятори систем охолодження і трансформатори. Згідно ДСанПіН 3.3.2.007-98 допустимий еквівалентний рівень шуму для робочого місця програміста складає 50 дБА (акустичних децибела).

8.4. Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

- розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;

- мікроклімат відповідає нормативному значенню;

- акустичні умови роботи не перевищують нормативних значеннь;

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга) [64].

8.5. Розрахункова частина

В приміщенні (де відсутні джерела виділення шкідливих речовин) працює одна людина. Робота пов'язана з використанням ПЕОМ. Розміри приміщення: А = 4 м, В = 3,5 м, Н = 2,8 м, устаткування займає 15% об'єму. Визначити найменшу необхідну кількість повітря для вентиляції.

Для приміщень, в яких відсутні виділення шкідливих речовин у повітрі, розрахунок вентиляції здійснюється залежно від кількості працюючих.

Необхідна кількість повітря ($м^3$ /год.), яка забезпечує відповідність параметрів повітря робочої зони нормованим значенням, визначається за наступною формулою:

$$L = L' \cdot N, \tag{8.1}$$

де L' - нормативна кількість повітря на одного працюючого, яка залежить від питомого об'єму приміщення, $м^3$ /(год.-люд.); N - кількість працюючих.

Питомий об’єм приміщення $V_п$, ($м^3$ /люд.), визначається за формулою:

$$V_n = V/N, \quad (8.2)$$

де V - об’єм приміщення, $м^3$.

Визначаємо вільний об’єм приміщення

$$V = A \cdot B \cdot H \cdot 0{,}85 = 4 \cdot 3{,}5 \cdot 2{,}8 \cdot 0{,}85 = 33{,}3 \text{ м}^3.$$

Питомий вільний об’єм складає

$$V' = V / N = 33{,}2 / 1 = 33{,}2 \text{ м}^3 \text{ /люд.} < 20 \text{ м}^3\text{/люд.}$$

Нормована кількість повітря на одну людину при $V' < 20$ $м^3$ /люд. становить 30 $м^3$ /(год. • люд.).

Висновки

У даному розділі магістерської роботи проведено аналіз умов працівника робота якого пов’язана з комп’ютерною технікою. Проведено аналіз основних санітарно – гігієнічних показників в заданому приміщені , де працівник зайнятий постійною роботою за комп’ютером.. Створені умови повинні забезпечувати комфортну роботу. На підставі вивченої літератури з даної проблеми, були зазначені оптимальні параметри мікроклімату, освітлення, допустимі рівні шуму та іонізуючого випромінювання при роботі з ПЕОМ, а також розраховано найменшу необхідну кількість повітря для вентиляції.

Дотримання умов, що визначають оптимальну організацію робочих місць працівників, дозволить зберегти гарну працездатність протягом усього робочого дня, підвищить як в кількісному, так і в якісному відносинах продуктивність їх праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання магістерської роботи, призначено для реалізації інтелектуальної системи для виявлення DDoS-атак. Воно дозволяє вчасно виявляти та реагувати на спроби атаки та захищати мережеві ресурси від переривань обслуговування.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У магістерській роботі наведені теоретичне узагальнення й рішення наукового завдання дослідження методів виявлення DDoS-атак.

Рішення даного завдання полягало у вирішенні наступних задач:

– Дослідження існуючих систем для виявлення DDoS-атак.

– Розробка методів та алгоритмів для інтелектуальної системи виявлення DDoS-атак.

– Програмна реалізація інтелектуальної системи виявлення DDoS-атак.

Розроблені під час виконання магістерської роботи алгоритми дозволяють успішно вирішувати завдання виявлення DDoS-атак на комп'ютерні мережі та веб-ресурси.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє проводити аналіз великих обсягів даних, має багато бібліотек для статистичного аналізу. Це дозволило полегшити процес створення програмного забезпечення, і, як наслідок, зменшити витрати на розробку.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення та впровадження його в промислову експлуатацію.

Для захисту програмного забезпечення був обраний метод асиметричного шифрування даних RSA, яким пропонується шифрувати ліцензію на програмне забезпечення.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 7083 грн. З урахуванням вартості розробки програми та обладнання, строк окупності становить 0,64 роки.

Загалом, розроблена система для виявлення DDoS-атак є важливим інструментом для забезпечення кібербезпеки організацій та мереж. Її ефективність полягає у здатності вчасно повідомляти про загрози та забезпечувати стабільну роботу мережі, що вкрай важливо в сучасному цифровому світі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A fractal analysis of a Markov chain based self-similar traffic generator / H. M. Drieieva, O. A. Smirnov, O. M. Drieiev, T. V. Smirnova. Central Ukrainian Scientific Bulletin. Engineering sciences. 2019. Vol. 2, no 33. P. 161–172.

2. A method of determining the fractal dimension of network traffic by its probabilistic properties and experimental research of the quality of this method / H. Drieieva, O. Drieiev, Ye. Meleshko et al. Computational Linguistics and Intelligent Systems (COLINS 2022): CEUR-WS, Gliwice, Poland / Vasyl Lytvyn (Ed.). 2022. Vol. 3154, P. 1694–1707 (ISSN 1613-0073). DOI: 10.5281/zenodo.7892006. URL: https://ceur-ws.org/Vol-3171/paper120.pdf.

3. Accurate recovery of internet traffic data under dynamic measurements / K. Xie, C. Peng, X. Wang et al. IEEE INFOCOM 2017: IEEE Conf. on Computer Communications. 2017. P. 1–9.

4. Ali M. Time Series Forecasting Tutorial. 2022. URL: https://www.datacamp.com/tutorial/tutorial-time-series-forecasting#rdl.

5. An empirical comparison of generators for self similar simulated traffic / G. Horn, A. Kvalbein, J. Blomskøld, E. Nilsen. Performance Evaluation. 2007. Vol. 64, no. 2. P. 162–190.

6. Analysis of the queueing-inventory system with impatient customers and mixed sales / Y. Zhang, D. Yue, L. Sun, J. Zuo. Discrete Dynamics in Nature and Society. 2022. Vol. 2022. DOI: 10.1155/2022/2333965.

7. Anis A. A., Lloyd E. H. The expected value of the adjusted rescaled Hurst range of independent normal summands. Biometrica. Vol. 63, no. 1. P. 283–298. DOI: 10.1093/BIOMET/63.1.111.

8. Areström E., Carlsson N. Early online classification of encrypted traffic streams using multi-fractal features. IEEE INFOCOM 2019 - IEEE Conf. on Computer Communications Workshops (INFOCOM WKSHPS). 2019. P. 84–89. DOI:

10.1109/INFCOMW.2019.8845127.

9. Aweya J. IP Routing Protocols: Link-State and Path-Vector Routing Protocols. 1st ed. CRC Press, 2021. 438 p.

10. Barabási A.-L. Network Science. Cambridge University Press, 2018. 475 p. URL: http://networksciencebook.com/.

11. Barabási A.-L., Albert R. Emergence of scaling in random networks. Science. 1999. Vol. 286, no. 5439 P. 509–512. DOI: 10.1126/science.286.5439.509.

12. Bassingthwaighte J. B., Raymond G. M. Evaluating rescaled range analysis for time series. Annals of Biomedical Engineering. 1994. Vol. 22. P. 432–444. DOI: 10.1007/BF02368250. URL: https://link.springer.com/article/10.1007/BF02368250#citeas.

13. Bulakh V., Kirichenko L., Radivilova T. Time series classification based on fractal properties. Int. Conf. on Data Stream Mining & Processing (DSMP): Proc. of the 2018 IEEE Second, (Lviv, Ukraine, Aug. 21–25, 2018). P. 198–201. DOI: 10.1109/DSMP.2018.8478532.

14. Characterisation of wireless network traffic: Fractality and stationarity / S. Mukherjee, R. Ray, M. H. Khondekar et al. Third Int. Conf. on Research in Computational Intelligence and Communication Networks (ICRCICN). Kolkata, India, 2017. P. 79–83. DOI: 10.1109/ICRCICN.2017.8234485.

15. Cisco. Dynamic routing protocols. Cisco Press. 2001. URL: https://www.ciscopress.com/articles/article.asp?p=24090&seqNum=4.

16. Cisco. IP Routed protocols. Technology Support. 2022. URL: https://www.cisco.com/c/en/us/tech/ip/ip-routed-protocols/index.html.

17. Cisco. Understand open shortest path first (OSPF) – design guide. Technology Support. 2022. URL: https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/7039-1.html.

18. Czarkowski M., Kaczmarek S., Wolff M. Influence of self -similar traffic type on performance of QoS routing algorithms. INTL Journal of electronics and telecommunications. 2016. Vol. 62, no. 1. P. 81–87. DOI: 10.1515/eletel-2016-0011.

19. Daradkeh Y. I., Kirichenko L., Radivilova T. Development of QoS methods in the information networks with fractal traffic. International Journal of Electronics and Telecommunications. 2018. Vol. 64, no. 1. P. 27–32. DOI: 10.24425/118142.

20. Developing a model of the dynamics of states of a recommendation system under conditions of profile injection attacks / Ye. Meleshko, O. Drieiev, M. Yakymenko, D. Lysytsia. Eastern-European Journal of Enterprise Technologies. 2020. Vol. 4, no. 2 (106). P. 14–24. URL: https://www.scopus.com/record/display.uri?eid=2-s2.0-85096707995&origin=resultslist.

21. Dimitrakos T. D., Kyriakidis E. G. A semi-Markov decision algorithm for the maintenance of a production system with buffer capacity and continuous repair times. International Journal of Production Economics. 2008. Vol. 111, no. 2. P. 752–762. DOI: 10.1016/j.ijpe.2007.03.010.

22. Dymora P., Mazurek M. Influence of model and traffic pattern on determining the self-similarity in IP networks. Applied Sciences. 2021. Vol. 11, no. 1. 190 p. DOI: 10.3390/app11010190. URL: https://www.mdpi.com/2076-3417/11/1/190.

23. Farahani A., Shoja A., Tohidi H. Markov and semi-Markov models in system reliability. Engineering Reliability and Risk Assessment. Elsevier, 2023. P. 91–130. DOI: 10.1016/B978-0-323-91943-2.00010-1.

24. Fast low-rank matrix approximation with locality sensitive hashing for quick anomaly detection / G. Xie, K. Xie, J. Huang et al. IEEE INFOCOM 2017: IEEE Conf. on Computer Communications. 2017. P. 1–9.

25. Feldmann A., Gilbert A. C., Willinger W. Data networks as cascades: Investigating the multifractal nature of Internet WAN traffic. Proc. of the ACM SIGCOMM ‘98 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM ‘98). Association for Computing Machinery, New York, NY, USA, 1998. P. 42–55. DOI: 10.1145/285237.285256.

26. Fractal modeling of big data networks / M. Barat, Z. Joveini, J. Sadri, H. A. Khoushhal. Int. Conf. on Pattern Recognition and Artificial Intelligence (ICPRAI 2018). Canada, Montreal: Concordia University, 2018. P. 1–4.

27. Hae-Duck Joshua Jeong. Modelling of self-similar teletraffic for simulation: D. Ph. thesis. University of Canterbury. New Zealand: Christchurch, 2002. P. 270.

28. Hajtmanek, R.; Kontšek, M.; Smieško, J.; Uramová, J. One-Parameter Statistical Methods to Recognize DDoS Attacks. Symmetry 2022, 14, 2388. https://doi.org/10.3390/sym14112388

29. Ivanisenko I., Kirichenko L., Radivilova T. Investigation of self-similar properties of additive data traffic. X Int. Sci. and Tech. Conf. "Computer Sciences and Information Technologies" (CSIT), IEEE. 2015. P. 169–171. URL: https://arxiv.org/ftp/arxiv/papers/1904/1904.05925.pdf.

30. Jain N., Payal A. Comparison between IPv4 and IPv6 using OSPF and OSPFv3 on riverbed modeler. IEEE Int. Conf. on Advanced Networks and Telecommunications Systems (ANTS). Goa, India, 2019. P. 1–7. DOI: 10.1109/ANTS47819.2019.9118101.

31. Jiang D., Huo L., Li Y. Fine-granularity inference and estimations to network traffic for SDN. PLoS ONE. / Zhihan Lv (Ed.). London, UK: University College, 2018. Vol. 13, no. 5. DOI: 10.1371/journal.pone.0194302.

32. Kirichenko L., Radivilova T., Bulakh V. Machine learning in classification time series with fractal properties. IEEE Second Int. Conf. on Data Stream Mining & Processing (DSMP), (Lviv, Ukraine, Aug. 21–25, 2018). 2018. Vol. 4, no. 1. DOI: 10.3390/data4010005.

33. Lakhmi Priya Das, Sanjay Kumar Patra, Sarojananda Mishra. Impact of hurst parameter value in self-similarity behaviour of network traffic. International Journal of Research in Computer and Communication Technology. 2016. Vol. 5, no. 12. P. 631–633.

34. Li Q.-L., Lui J. C. S. Block-structured supermarket models. Discrete Event Dynamic Systems. 2014. Vol. 26, no. 2. P. 147–182. DOI: 10.1007/s10626-014-0199-1.

35. Lysytsia D. O., Semenov S. G., Lysytsia A. O. Gert-model of processes of active analysis of the system resource management and implementation in the computer system. Středoevropský věstník pro vědu a výzkum. 2018. Vol. 6, no. 50.

36. Ma C., Dai G., Zhou J. Short-term traffic flow prediction for urban road sections based on time series analysis and LSTM_BILSTM method. IEEE Transactions on Intelligent Transportation Systems. 2021. Vol. 23, no. 6. P. 5615–5624. DOI: 10.1109/TITS.2021.3055258. URL: https://ieeexplore.ieee.org/document/9364926.

37. Millán G. Traffic flows analysis in high-speed computer networks using time series. arXiv preprint arXiv:2103.03984. 2021. DOI: 10.48550/arXiv.2103.03984. URL: https://arxiv.org/abs/2103.03984.

38. Millána G., Lefranc G. A fast multifractal model for self-similar traffic flows in high-speed computer networks. Information Technology and Quantitative Management (ITQM2013) Procedia Computer Science. 2013. Vol. 17. P. 420–425.

39. Mirchandani P. B., Zou N. Queuing models for analysis of traffic adaptive signal control. IEEE Transactions on Intelligent Transportation Systems. 2007. Vol. 8, no. 1. P. 50–59. DOI: 10.1109/TITS.2006.888619.

40. Moy J. T. OSPF: Anatomy of an Internet Routing Protocol. Addison-Wesley Professional, 1998.

41. Peters E. Fractal Market Analysis: Applying Chaos Theory to Investment and Economics. John Wiley & Sons, 1994. Vol. 24. 336 p.

42. Phinyomark A., Larracy R., Scheme E. Fractal analysis of human gait variability via stride interval time series. Front Physiol. 2020. Vol. 11. DOI: 10.3389/fphys.2020.00333. URL: https://pubmed.ncbi.nlm.nih.gov/32351405/.

43. Raaijmakers Y., Albrecher H., Boxma O. The single server queue with mixing dependencies. Methodology and Computing in Applied Probability. 2019. Vol. 21. P. 1023–1044. URL: http://www.hec.unil.ch/halbrech_files/QueueMixing.pdf.

44. Ribeiro V. J., Zhang Z.-L Christophe Diot imall-time scaling behavior of Internet backbone traffic. Computer Networks. 2005. Vol. 48, no. 3. P. 315–334. DOI: 10.1016/j.comnet.2004.11.012.

45. Robert S., Le Boudec J. Y. New models for pseudo self-similar traffic. Performance Evaluation. 1997. Vol. 30, no. 1–2. P. 57–68.

46. Sobh T., Elleithy K., Mahmood A. Novel Algorithms and Techniques in

Telecommunications and Networking. Springer, 2010. P. 41–46.

47. Tadimety P. R. OSPF messages. OSPF: A Network Routing Protocol. Berkeley, CA: Apress, 2015. DOI: 10.1007/978-1-4842-1410-7_18.

48. Tian, Yu-Chu, Zu-Guo Yu, Colin Fidge. Multifractal nature of network induced time delay in networked control systems. Physics Letters A. 2007. Vol. 361, no. 1–2. P. 103–107. DOI: 10.1016/j.physleta.2006.09.046 (date of access: 11.06.2006).

49. Traag V. A. Algorithms and dynamical models for communities and reputation in social networks. Springer International Publishing. 2014. P. 229. URL: https://doi.org/10.1007/978-3-319-06391-1.

50. Vassiliou P.-C. G., Georgiou A. C. Markov and semi-Markov chains, processes, systems and emerging related fields. Mathematics. 2021. Vol. 9, no. 19. 294 p. DOI: 10.3390/math9192490.

51. Verma A., Bhardwaj N. A review on routing information protocol (RIP) and open shortest path first (OSPF) routing protocol. International Journal of Future Generation Communication and Networking. 2016. Vol. 9, no. 4. P. 161–170.

52. Wang C., Maguluri S. T., Javidi T. Heavy traffic queue length behavior in switches with reconfiguration delay. IEEE INFOCOM 2017: IEEE Conf. on Computer Communications. 2017. P. 1–9.

53. Watts D. J., Strogatz S. H. Collective dynamics of “small-world” networks. Nature. 1998. Vol. 393, no. 6684. P. 440–442. URL: https://www.nature.com/articles/30918.

54. Yu B. OSPF-based network engineering design and implementation. Informatics and Management Science VI / W. Du (Ed.). London: Springer, 2013. Vol. 209. P. 131–138. DOI: 10.1007/978-1-4471-4805-0_16.

55. Державні будівельні норми України: ДБН В.2.5-28:2018. - Режим доступу до ресурсу: https://goo.su/9AkQ (дата звернення 19.10.22).

56. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98.

57. Дрєєва Г. М., Мелешко Є. В., Міхав В. В. Програмна імітаційна модель комп’ютерної мережі для тестування алгоритмів маршрутизації трафіку. Автоматика, комп’ютерно-інтегровані технології та проблеми енергоефективності в промисловості і сільському господарстві: матеріали Міжнар. наук.-техн. конф. (Кропивницький, 10–11 листоп. 2022 р.) / М-во освіти і науки України, Центральноукр. нац. техн. ун-т. Кропивницький: Ексклюзив-Систем, 2022. С. 44–45.

58. Кучук Г. А., Можаєв О. О., Воробйов О. В. Аналіз та моделі самоподібного трафіка. Авиационно-космическая техннка и технология. 2006. Вип. 9, № 35. С. 173–180. URL: http://nbuv.gov.ua/UJRN/aktit_2006_9_35.

59. Кучук Г. А., Можаєв О. О., Воробйов О. В. Метод прогнозування фрактального трафіка. Радіоелектронні і комп’ютерні системи. 2006. Вип. 6. С. 181–188. URL: http://nbuv.gov.ua/UJRN/recs_2006_6_34.

60. Кучук Г. А., Можаєв О. О., Воробйов О. В. Прогнозування трафіка для управління перенавантаженнями інтегрованої телекомунікаційної мережі. Радіоелектронні та комп’ютерні системи. 2007. Вип. 8. С. 261–271. URL: http://nbuv.gov.ua/UJRN/recs_2007_8_48.

61. Методичні рекомендації до виконання розділу "Заходи з охорони праці та техніки безпеки" випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти для здобувачів вищої освіти спеціальностей 123 "Комп’ютерна інженерія" та 122 “Комп’ютерні науки” / М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т, каф. кібербезпеки та програм. забезпечення; [укл. О.В. Оришака, К.М. Марченко]. - Кропивницький: ЦНТУ, 2022. — 19 с. [Електронний ресурс]. – Режим доступу : http://dspace.kntu.kr.ua/jspui/handle/123456789/12240 (дата звернення 19.10.22).

62. Моделі та процедури класифікації і прогнозування недетермінованих процесів за показниками хаотичної динаміки / В. В. Скалозуб, В. М. Горячкін, І. В. Клименко, Д. О Шаповал. System technologies. 2022. Vol. 3, no. 140. DOI: 10.34185/1562-9945-3-140-2022-10.

63. Наказ Міністерства внутрішніх справ України 30.12.2014 №1417 «Про затвердження Правил пожежної безпеки України» - Режим доступу до ресурсу https://zakon.rada.gov.ua/laws/show/z0252-15#Text (дата звернення 19.10.22).

64. Наказ Міністерства освіти і науки України 15.08.2016 № 974 «Про затвердження Правил пожежної безпеки для навчальних закладів та установ системи освіти України» - Режим доступу до ресурсу https://zakon.rada.gov.ua/laws/show/z1229-16#Text (дата звернення 19.10.22).

65. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508 (дата звернення 19.10.22).

66. Пасічник В. В., Іванущак Н. М. Дослідження та моделювання складних мереж. Східно-Європейський журнал передових технологій. 2010. Вип. 2, № 3 (44). С. 43–48.

67. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99 - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99 (дата звернення 19.10.22).

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію інтелектуальної системи для виявлення DDoS-атак.

2 Підстава для розробки

Підставою для розробки служить завдання на магістерську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. №__-__ від __.__.20__ року).

3 Мета та призначення розробки

Метою магістерської роботи є дослідження та програмна реалізація інтелектуальної системи для виявлення DDoS-атак.

4 Джерела розробки

Джерелом цієї магістерської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки

програмного забезпечення;

– аналіз умов праці розробників програмного забезпечення;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– виявлення DDoS-атак на веб-сайти;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Застосунок, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Мова програмування високого рівня Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД.

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 1 вересня 2023 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи повинні бути розглянуті умови праці програмістів під час розробки програмного забезпечення.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуші.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 97 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі магістерської роботи. Постановка задачі на виконання магістерської роботи (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень магістерської роботи.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання магістерської роботи на попередній захист __.__.2023 р.

11.2 Подання магістерської роботи на захист __.__.12.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ
Керівник випускної кваліфікаційної роботи
за другим (магістерським) рівнем вищої освіти
_____ Є.В. Мелешко

Дослідження та програмна реалізація інтелектуальної системи для виявлення DDoS-атак

Лістинг програми

Код документу 12

Носій: DVD-диск

Загальна кількість аркушів: 15

Літера: РП

Кропивницький – 2023 року

rs-analysis.py – модуль R/S-аналізу часових рядів трафіку

```
import math, numpy, random
import pandas as pd


def normalize_traffic_dataset(file_name):
    """
    Зчитує датасет з файлу трафіку та нормалізує часовий ряд.

    :param file_name: Назва файлу з датасетом.
    :return: Нормалізований часовий ряд.
    """
    # Зчитуємо датасет з файлу
    dataset = pd.read_csv(file_name)

    # Визначаємо часовий інтервал між вимірами
    time_interval = dataset['time'].diff().mean()

    # Нормалізуємо часовий ряд
    normalized_traffic = dataset['packets'] / time_interval

    return normalized_traffic


def ArrayResize(mas, l, zap = 0):
    n = len(mas)
    if n < l:
        mas = mas + [zap]*(l-n)
    elif n < l:
        mas = mas[:l]
    return mas

LogReturns = []
N  = []
DevAccum = []
StdDevMas = []
pi = 3.14159265358979323846264338

exp = 1
close = [0]*exp
close[0] = normalize_traffic_dataset(file1.csv)

for i in range(len(close)):
    for j in range(len(close[i])):
        if close[i][j] == 5.0:
            close[i][j] = 2
        else:
            close[i][j] = 1

rs = []

#+------------------------------------------------------------------+
#| Функція старту                                                   |
#+------------------------------------------------------------------+

def OnStart(close, nn):
    global LogReturns, DevAccum, StdDevMas
    global rs

    num = [5, 10, len(close)]
    n_passes = len(num)
    #all_n = 1001
    all_n = len(close) + 1
    for_div = [int(all_n / num[i]) for i in range(0, n_passes)]

    rs = [[0] for i in range(0,n_passes)]
```

```
            close = ArrayResize(close, all_n, 1)


        #+------------------------------------------------------------------+
        #| Підготовка масивів                                               |
        #+------------------------------------------------------------------+

            LogReturns = ArrayResize(LogReturns, all_n)
            DevAccum = ArrayResize(DevAccum, all_n)
            StdDevMas = ArrayResize(StdDevMas, all_n)

        #+------------------------------------------------------------------+
        #| Масив логарифмічних значень                                      |
        #+------------------------------------------------------------------+

            for i in range(1, all_n - 1):
                try:
                    LogReturns[i] = numpy.log(close[i-1] / close[i])
                except:
                    LogReturns[i] = 0

        #+------------------------------------------------------------------+
        #|                           R/S-аналіз                             |
        #+------------------------------------------------------------------+

        #--- Задамо кількість елементів у кожній підгрупі

            LogRS = [0] * n_passes



        #--- Розрахунок складових Log(R/S)

            for A in range(0, n_passes):
                RSsum = 0.0
                for j in range(1, for_div[A]+1):
                    if (len(close)-1) % num[A] != 0 and j == for_div[A]:
                        rs[A].append( RSculc(num[A] * j - num[A] + 1 ,len(close)
- 1, (len(close)-1) % num[A]) )
                    else:
                        rs[A].append(RSculc(num[A] * j - num[A] + 1 , num[A] *
j, num[A]))
                    RSsum = RSsum + rs[A][j]
                try:
                    RS = RSsum / for_div[A]
                except:
                    RS = 0
                LogRS[A] = numpy.log(RS)

        #+------------------------------------------------------------------+
        #|  Розрахунок коефіцієнта Херста                                   |
        #+------------------------------------------------------------------+

            H = RegCulc(LogRS)

        #+------------------------------------------------------------------+
        #| Розрахунок очікуваних значень log(E(R/S))                        |
        #+------------------------------------------------------------------+

            E = []
            for i in range(n_passes):
                E.append(numpy.log(ERSculc(num[i])))

        #+------------------------------------------------------------------+
        #|  Розрахунок бети очікуваних значень E(R/S)                       |
        #+------------------------------------------------------------------+

            betaE = RegCulc(E)
            print(nn, "H = " + str(H),",  E = " + str(betaE))
```

```
        #    for ii in range(n_passes):
        #        print(num[ii], LogRS[ii])


        #+----------------------------------------------------------------------+
        #|  Функція розрахунку R/S                                              |
        #+----------------------------------------------------------------------+

        def RSculc(bottom, top, barscount):
            global LogReturns, all_n
            MaxValue = 0.0
            MinValue = all_n - 1
            Sum = 0.0               # Початкове значення суми дорівнює нулю
            DevSum = 0.0            # Початкове значення суми накопичених
                                    # відхилень дорівнює нулю
        #--- Розрахунок суми накопичень
            for i in range(bottom, top+1):
                Sum = Sum + LogReturns[i]          #Накопичені суми

        #--- Розрахунок середнього
            try:
                M = Sum / barscount
            except:
                M = 0
        #--- Розрахунок накопичених відхилень
            for i in range(bottom,top+1):
                DevAccum[i] = LogReturns[i] - M + DevAccum[i-1]
                StdDevMas[i] = math.pow((LogReturns[i] - M), 2)
                DevSum = DevSum + StdDevMas[i]                    # Складова для
розрахунку відхилення
                if DevAccum[i] > MaxValue: #Якщо значення в масиві менше деякого
                    MaxValue = DevAccum[i]  #максимального, то присвоюємо
максимальному значенню значення елемента масиву DevAccum
                if DevAccum[i] < MinValue:                          #Логіка дій
аналогічна
                    MinValue = DevAccum[i]

        #--- Розрахунок розмаху R та відхилення S

            R = MaxValue - MinValue       #Розмах дорівнює різниці максимального
і
            MaxValue = 0.0
            MinValue = 1000                 #мінімальних значень
            S1 = math.sqrt(DevSum / barscount)    #Розрахунок стандартного
відхилення
        #--- Розрахунок показника R/S
            if S1 != 0:
                RS = R / S1                       #Виключаємо помилку поділу на
"нуль"
        #    else Alert("Zero divide!")
            else:
                RS = 0
            return RS                    #Повертаємо значення RS-статистики

        #+----------------------------------------------------------------------+
        #|  Калькулятор регресії                                                |
        #+----------------------------------------------------------------------+

        def RegCulc(Y):
            global N
            global num, n_passes
            SumY = 0.0
            SumX = 0.0
            SumYX = 0.0
            SumXX = 0.0
            b = 0.0   #масив, в якому лежатимуть логарифми дільників масив
дільників
```

```
#--- Розрахунок коефіцієнтів N
    N = ArrayResize(N, n_passes)

    for i in range(0, n_passes):
        N[i] = numpy.log(num[i])
        SumX = SumX + N[i]
        SumXX = SumXX + N[i] * N[i]

    for i in range(len(Y)):
        SumY += Y[i]
        SumYX += Y[i]*N[i]
#--- Розрахунок коефіцієнта Beta регресії або шуканого показника Херста
    try:
        b = (n_passes*SumYX-SumY*SumX)/(n_passes*SumXX-SumX*SumX)
    except:
        b = 0
    return b

#+------------------------------------------------------------------------+
#| Функція розрахунку очікуваних значень E(R/S)                           |
#+------------------------------------------------------------------------+

def ERSculc(m):                 #m - дільники 1000
    nSum = 0.0
    part = 0.0
    for i in range(1, m):
        part = math.pow(((m-i)/i), 0.5)
        nSum = nSum + part
    e = math.pow((m*pi/2),-0.5)*nSum
    return e


for i in range(exp):
    OnStart(close[i], i)
```

intensity.py – модуль визначення інтенсивності трафіку

```
import pandas as pd
import numpy as np

def calculate_traffic_intensity(dataset):
    """
    Обчислює інтенсивність трафіку на основі датасету.
    :param dataset: Датасет з колонками 'time' і 'packets'.
    :return: Датасет з додатковою колонкою 'intensity' з обчисленою
інтенсивністю.
    """

time_series = dataset['packets']

def calculate_intensity(time_series):

    time_series = np.array(time_series, dtype=np.float64)

    # Розраховуємо інтенсивність

    intensity = np.sum(time_series) / len(time_series)

    intensity = (intensity - np.min(time_series)) / (np.max(time_series) -
np.min(time_series))

    return intensity



# Зчитуємо датасет
dataset = pd.read_csv('your_dataset.csv')  # шлях до датасету

# Викликаємо функцію для обчислення інтенсивності трафіку
result_dataset = calculate_traffic_intensity(dataset)

# Виводимо результат
print(result_dataset[['time', 'packets', 'intensity']])
```

trafficSimulation.py – модуль симуляції трафіку, модель 1

```
def generate_traffic(length, intensity):
    """
    Генерує самоподібний часовий ряд трафіку.

    :param length: Довжина часового ряду.
    :param intensity: Інтенсивність трафіку (кількість пакетів).
    :return: Масив згенерованого часового ряду.
    """
    np.random.seed(0)
    increments = np.random.randn(length)
    cumsum = np.cumsum(increments)
    scaled = cumsum
    scaled -= np.mean(scaled)

    traffic = np.zeros(length)
    for i in range(intensity):
        start = np.random.randint(0, length)
        traffic[start::intensity] += scaled[:length - start:intensity]

    traffic = traffic.astype(int)
    traffic = np.abs(traffic)

    return traffic

def save_traffic_dataset(filename, traffic):
    """
    Зберігає часовий ряд трафіку у форматі датасету.

    :param filename: Назва файлу для збереження датасету.
    :param traffic: Часовий ряд трафіку (масив).
    """
    df = pd.DataFrame({'Час': range(len(traffic)), 'Кількість пакетів':
traffic})
    df.to_csv(filename, index=False)

# Приклади використання функцій симуляції трафіку
length = 10000
intensity = 50



traffic = generate_traffic(length, intensity)

print(traffic)


# Приклад використання функції для збереження трафіку
save_traffic_dataset('traffic_dataset1.csv', traffic)
```

trafficSimulation.py – модуль симуляції трафіку, модель 2

```
import numpy as np
import matplotlib.pyplot as plt

def generate_traffic(n, intens = 1.0):
    lst = np.zeros(n, dtype=int)

    for n in range(1, n):

        # Генеруємо випадкову змінну
        x = np.random.randint(0, 200)  # Випадкове ціле число з інтервалу

        r = np.sign(x - lst [n - 1])
        lst [n] = ряд[n - 1] + r * intens
    return lst

# Параметри генерації трафіку
n = 1000
intens = 34  # Збільште це значення для більшої інтенсивності
персистентність = 0.9  # Збільште це значення для більшої персистентності

# Генерувати трафік
tr = generate_traffic (n, intens)
print(tr)

# Візуалізація часового ряду
plt.figure(figsize=(12, 6))
t = np.arange(n)
plt.plot(t, tr, label='Часовий ряд трафіку')
plt.xlabel('Час (відліки)')
plt.ylabel('Кількість пакетів')
plt.legend()
plt.title('Експеримент 1')
plt.grid(True)
plt.show()
```

trafficRead.py – читання часового ряду трафіку з датасету

```
import pandas as pd

def extract_time_series_and_attack_presence(dataset_path):
    try:
        # Завантаження даних з CSV-файлу
        df = pd.read_csv(dataset_path)

        # Вибір потрібних стовпців ('Timestamp' - це час, а 'Attack' - це
наявність атаки)
        selected_columns = ['Timestamp', 'Attack']
        df = df[selected_columns]

        # Формування часового ряду
        time_series = df.set_index('Timestamp')['Attack'].tolist()

        # Визначення наявності атаки
        attack_presence = any(time_series)

        return time_series, attack_presence
    except Exception as e:
        print(f"Помилка при обробці даних: {e}")
        return None, False

# Використання функції для отримання часового ряду та наявності атаки
dataset_path = 'dataset1.csv'
time_series, attack_presence =
extract_time_series_and_attack_presence(dataset_path)

n = 10
# Друкуємо перші n значень часового ряду та наявність атаки
print("Часовий ряд трафіку (перші n значень):", time_series[:n])
print("Наявність атаки:", attack_presence)
```

res.py – модуль визначення ймовірності Ddos-атаки

```
import pandas as pd
import sqlite3

# Функція для аналізу даних і визначення ймовірності атаки
def analyze_traffic_and_ddos_probability(dataset_path, knowledge_base_path):

    # Завантажуємо дані
    data = pd.read_csv(dataset_path)

    # Підключаємося до бази знань
    conn = sqlite3.connect(knowledge_base_path)
    cursor = conn.cursor()

    # Використовуємо SQL-запит для отримання правил визначення ймовірності атаки
з бази знань
    cursor.execute("SELECT threshold_1, threshold_2, threshold_3, threshold_4
FROM ddos_rules")
    thresholds = cursor.fetchone()

    # Отримуємо інтенсивність трафіку та індекс Херста з даних датасету (вони є
стовпцями у файлі)
    traffic_intensity = data['Traffic Intensity'].mean()
    hurst_index = data['Hurst Index'].mean()

    # Визначаємо ймовірність атаки на основі отриманих даних та правил з бази
знань
    ddos_probability = calculate_ddos_probability(traffic_intensity,
hurst_index, thresholds)

    # Закриваємо підключення до бази знань
    conn.close()

    # Повертаємо результат аналізу
    return ddos_probability

# Шляхи до датасету та бази знань
dataset_path = 'dataset.csv'
knowledge_base_path = 'knowledge_base.db'

# Аналізуємо дані та виводимо результат
result = analyze_traffic_and_ddos_probability(dataset_path, knowledge_base_path)
print(f'Ймовірність DDoS-атаки: {result}')
```

knowledgeBase.py – модуль роботи з базою правил визначення ddos-атаки

```
import sqlite3

# Функція для створення бази даних (якщо її ще не існує)
def create_knowledge_base(database_path):
    try:
        conn = sqlite3.connect(database_path)
        cursor = conn.cursor()

        # Створення таблиці для правил визначення DDoS-атак
        cursor.execute('''CREATE TABLE IF NOT EXISTS ddos_detection_rules (
                            id INTEGER PRIMARY KEY AUTOINCREMENT,
                            intensity_threshold REAL,
                            hurst_index_threshold REAL
                          )''')

        conn.commit()
        conn.close()
    except sqlite3.Error as e:
        print(f"Помилка бази даних: {e}")

# Функція для додавання правила в базу знань
def add_ddos_detection_rule(database_path, intensity_threshold,
hurst_index_threshold):
    try:
        conn = sqlite3.connect(database_path)
        cursor = conn.cursor()

        # Вставка нового правила в таблицю ddos_detection_rules
        cursor.execute("INSERT INTO ddos_detection_rules (intensity_threshold,
hurst_index_threshold) VALUES (?, ?)",
                       (intensity_threshold, hurst_index_threshold))

        conn.commit()
        conn.close()
    except sqlite3.Error as e:
        print(f"Помилка бази даних: {e}")

# Функція для виведення всіх правил в базі знань
def view_ddos_detection_rules(database_path):
    try:
        conn = sqlite3.connect(database_path)
        cursor = conn.cursor()

        # Вибір всіх правил з таблиці ddos_detection_rules
        cursor.execute("SELECT * FROM ddos_detection_rules")
        rules = cursor.fetchall()

        # Виведення правил
        print("ID | Intensity Threshold | Hurst Index Threshold")
        for rule in rules:
            rule_id, intensity_threshold, hurst_index_threshold = rule
            print(f"{rule_id} | {intensity_threshold} |
{hurst_index_threshold}")

        conn.close()
    except sqlite3.Error as e:
        print(f"Помилка бази даних: {e}")
```

bdRead.py – модуль роботи з базою даних

```
import sqlite3

# Функція для виведення даних про датасети трафіку
def view_traffic_datasets(database_path):
    try:
        # Підключаємося до бази даних
        conn = sqlite3.connect(database_path)
        cursor = conn.cursor()

        # Виконуємо SQL-запит для отримання даних про датасети трафіку
        cursor.execute("SELECT * FROM traffic_datasets")
        datasets = cursor.fetchall()

        # Виводимо заголовок
        print("ID | Dataset Name | Start Date | End Date")

        # Виводимо дані про датасети
        for dataset in datasets:
            dataset_id, dataset_name, start_date, end_date = dataset
            print(f"{dataset_id} | {dataset_name} | {start_date} | {end_date}")

        # Закриваємо підключення до бази даних
        conn.close()
    except sqlite3.Error as e:
        print(f"Помилка бази даних: {e}")

# Функція для виведення даних про інтенсивність та індекс Херста для часових
рядів трафіку
def view_traffic_metrics(database_path):
    try:
        # Підключаємося до бази даних
        conn = sqlite3.connect(database_path)
        cursor = conn.cursor()

        # Виконуємо SQL-запит для отримання даних про інтенсивність та індекс
Херста
        cursor.execute("SELECT * FROM traffic_metrics")
        metrics = cursor.fetchall()

        # Виводимо заголовок
        print("ID | Dataset ID | Traffic Intensity | Hurst Index")

        # Виводимо дані про інтенсивність та індекс Херста
        for metric in metrics:
            metric_id, dataset_id, traffic_intensity, hurst_index = metric
            print(f"{metric_id} | {dataset_id} | {traffic_intensity} |
{hurst_index}")

        # Закриваємо підключення до бази даних
        conn.close()
    except sqlite3.Error as e:
        print(f"Помилка бази даних: {e}")

# Функція для виведення даних про ймовірності DDoS-атак
def view_ddos_probabilities(database_path):
    try:
        # Підключаємося до бази даних
        conn = sqlite3.connect(database_path)
        cursor = conn.cursor()

        # Виконуємо SQL-запит для отримання даних про ймовірності DDoS-атак
        cursor.execute("SELECT * FROM ddos_probabilities")
        probabilities = cursor.fetchall()

        # Виводимо заголовок
        print("ID | Dataset ID | DDoS Probability")
```

```
        # Виводимо дані про ймовірності DDoS-атак
        for probability in probabilities:
            probability_id, dataset_id, ddos_probability = probability
            print(f"{probability_id} | {dataset_id} | {ddos_probability}")

        # Закриваємо підключення до бази даних
        conn.close()
    except sqlite3.Error as e:
        print(f"Помилка бази даних: {e}")

# Шлях до бази даних
database_path = 'traffic_database.db'

# Виводимо дані про датасети трафіку
print("Дані про датасети трафіку:")
view_traffic_datasets(database_path)

# Виводимо дані про інтенсивність та індекс Херста
print("\nДані про інтенсивність та індекс Херста:")
view_traffic_metrics(database_path)

# Виводимо дані про ймовірності DDoS-атак
print("\nДані про ймовірності DDoS-атак:")
view_ddos_probabilities(database_path)
```

RSA.py – модуль захисту розробленого програмного забезпечення

```
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding


# Генерація ключів
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
)

# Збереження приватного ключа в файл
private_pem = private_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.PKCS8,
    encryption_algorithm=serialization.NoEncryption()
)
with open('private_key.pem', 'wb') as f:
    f.write(private_pem)

# Отримання публічного ключа
public_key = private_key.public_key()
public_pem = public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
)
with open('public_key.pem', 'wb') as f:
    f.write(public_pem)


# Завантаження приватного ключа
with open('private_key.pem', 'rb') as f:
    private_key = serialization.load_pem_private_key(
        f.read(),
        password=None
    )

# Текст ліцензії, який потрібно підписати
license_text = "Текст вашої ліцензії"

# Підпис
signature = private_key.sign(
```

```
    license_text.encode('utf-8'),
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)

# Збереження підпису
with open('license_signature.bin', 'wb') as f:
    f.write(signature)


# Завантаження публічного ключа
with open('public_key.pem', 'rb') as f:
    public_key = serialization.load_pem_public_key(f.read())

# Завантаження підпису
with open('license_signature.bin', 'rb') as f:
    signature = f.read()

# Текст ліцензії, який потрібно перевірити
license_text = "...Текст ліцензії..."

try:
    public_key.verify(
        signature,
        license_text.encode('utf-8'),
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    print("Підпис валідний. Ліцензія дійсна.")
except Exception:
    print("Підпис не валідний. Ліцензія недійсна.")
```