Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій
(повне найменування інституту, назва факультету)

Кафедра інформаційної безпеки та наноелектроніки
(повна назва кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр
(ступінь вищої освіти)

на тему Проєктування та розробка програмного застосунку деанонімізації користувачів у одноранговому протоколі BitTorrent в рамках OSINT-розвідки

Design and development of a software application for user deanonymization in the BitTorrent peer-to-peer protocol as part of OSINT intelligence

Виконав(ла): студент(ка) 6 курсу, групи БК-812м
Спеціальності 125 Кібербезпека
(код і найменування спеціальності)
Освітня програма (спеціалізація)
Безпека інформаційних і комунікаційних мереж
ОРЛОВСЬКИЙ Д.І.
(ПРІЗВИЩЕ та ініціали)

Керівник НЕЛАСА Г.В.
(ПРІЗВИЩЕ та ініціали)

Рецензент НІКУЛІЩЕВ Г.І.
(ПРІЗВИЩЕ та ініціали)

2023

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет інформаціїної безпеки та електроних комунікації
Кафедра інформаційної безпеки та наноелектроніки
Ступінь вищої освіти магістр
Спеціальність 125 Кібербезпека
(код і найменування)
Освітня програма (спеціалізація) Безпека інформаційних і комунікаційних мереж
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри ІБтаН
к.ф. -м.н. Андрій КОРОТУН
«____» _____2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ОРЛОВСЬКОГО Данила Ігоровича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Проєктування та розробка програмного застосунку деанонімізації користувачів у одноранговому протоколі BitTorrent в рамках OSINT-розвідки

Design and development of a software application for user deanonymization in the BitTorrent peer-to-peer protocol as part of OSINT intelligence

Керівник проєкту (роботи) к.т.н.,доцент НЕЛАСА Ганна Вікторівна ,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові,)
затверджені наказом закладу вищої освіти від " 28 " листопада 2023 року № 475

2. Строк подання студентом проєкту (роботи) 11.12.2023

3. Вихідні дані до проєкту (роботи) створити аналітичну систему на базі протоколу BitTorrent, за допомогою якого можна буде відшуковувати підозрілу активність

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) протокол BitTorrent, модифікований BitTorrent клієнт, аналітична система, система штучного інтелекту з класифікації torrent завантажень

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) презентація (15)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-8	НЕЛАСА Г.В., к.т.н. доцент каф. ІБтаН		
Нормоконтроль	КОРОЛЬКОВ Р.Ю., к.т.н. доцент каф ІБтаН		

7. Дата видачі завдання «04» вересня 2023року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1.	Постановка завдання роботи.	04.09.23 – 10.09.23	Виконано
2.	Аналіз принципів роботи однорангових мереж.	11.09.23 – 17.09.23	Виконано
3.	Аналіз існуючих методів деанонімізації в однорангових мережвх.	18.09.23 – 20.09.23	Виконано
4.	Аналіз юридичних аспектів щодо деанонімізації у мережі BitTorrent.	21.09.23 – 24.09.23	Виконано
5.	Проєктування та розробка програмного застосунку з деанонімізації.	25.09.23 – 01.10.23	Виконано
6.	Розробка підробленого BitTorrent-клієнту.	02.10.23 – 15.10.23	Виконано
7.	Розробка аналітичного сервера деанонімізації у P2P мережах.	16.10.23 – 05.11.23	Виконано
8.	Розробка штучного інтелекту класифікації BitTorrent.	06.10.23 – 12.11.23	Виконано
9.	Розробка візуального інтерфейсу аналітичної системи.	13.11.23 – 26.11.23	Виконано
10.	Оформлення матеріалів магістерської роботи.	27.11.23 – 10.12.23	Виконано

Студент(ка) _____ Данило ОРЛОВСЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Ганна НЕЛАСА
(підпис) (Ім'я ПРІЗВИЩЕ)

АНОТАЦІЯ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 120 с., 13 табл., 21 рис., 6 дод., 28 джерел.

ДЕАНОНІМІЗАЦІЯ, КІБЕРБЕЗПЕКА, ОДНОРАНГОВА МЕРЕЖА, ЦИФРОВА РОЗВІДКА, BITTORRENT, P2P, NODEJS

Питання можливості деанонімізації користувачів є актуальним з точки захисту анонімності та приватності під час користування мережі Інтернет. Не меньш актуальним воно є для представників правоохороних органів, робота яких є займатись превенцією та розслідуванням кіберзлочинів. Злоумисники часто використовують протокол BitTorrent у корисливих цілях, тому питання їх деанонімізації є актуальним

Об’єкт дослідження – деанонімізація користувачів у одноранговому протоколі BitTorrent.

Предмет дослідження – одноранговий протокол BitTorrent.

Мета роботи – створити аналітичну систему на основі протоколу BitTorrent, за допомогою якого можна буде проаналізувати завантаження торентів з окремої IP-адреси та провести їх класифікацію по типу. Продемонструвати вразливості анонімності під час використання протоколу BitTorrent

Для реалізації програмної частини використовувалась мова програмування JavaScript, програмна платформа NodeJS, середовище розробки Visual Studio Code.

У результаті ми отримали працюючий прототип аналітичної системи, яка може бути використана правоохоронними органами з метою пошуку підозрілої/незаконної активності, прототип власноруч зробленого шпигунського BitTorrent-клієнту. Система реалізована із використанням мови

програмування JavaScript, бази даних PostgreSQL та Redis, брокеру повідомлень BullMQ, бібліотеки штучного інтелекту TensorFlow, системи контейнеризації Docker та клієнтського сайту, написаного із використанням фреймворку NextJS (React)

Апробація результатів: Орловський Д.І. Деанонімізація у однорангових мережах: аналіз способів і приклади на базі протоколу BitTorrent // Тиждень науки: щорічна наук.-практ. конф. викладачів, науковців, молодих учених, аспірантів та студентів, 24-28 квітня 2023 р.: тези доповідей. – Запоріжжя, 2023.

Орловський Д.І. Розробка та програмна реалізація методики цифрової розвідки на основі відкритих джерел/ Д.І. Орловський, Г.В. Неласа // Всеукраїнський конкурс на кращу студентську наукову роботу 2020/2021 навчального року – Львів, 2020.

Орловський Д.І. Техніка розвідки за відкритими джерелами інформації. Технології OSINT. Вивчення протоколу BitTorrent засобами OSINT»/ Д.І. Орловський, Г.В. Неласа // Тиждень науки: щорічна наук.-практ. конф. викладачів, науковців, молодих учених, аспірантів та студентів, 13-17 квітня 2020 р.: тези доповідей. – Запоріжжя, 2020.

Орловський Д.І. Техніка розвідки за відкритими джерелами інформації (OSINT). DATA SCRAPING як засіб OSINT / Орловський Д.І., Неласа Г.В. // Інформаційні технології: теорія і практика: III Всеукр. Інтернет-конф. здобувачів вищої освіти і молодих учених, 2020 р.: тези доповідей. – Харків, 2020. – С. 77-78.

ABSTRACT

Explanatory note to the master's thesis: 120 p., 13 tables, 21 figure, 6 appendix, 28 sources.

BITTORRENT, CYBERSECURITY, DEANONYMALIZATION, DIGITAL INTELLIGENCE, NODEJS, P2P, PEER-TO-PEER NETWORK,

The issue of the possibility of de-anonymization of users is relevant from the point of view of protecting anonymity and privacy while using the Internet. It is no less relevant for representatives of law enforcement agencies, whose job is to deal with the prevention and investigation of cybercrimes. Criminals often use the BitTorrent protocol for selfish purposes, so the issue of their deanonymization is relevant

The object of research – is the deanonymization of users in the BitTorrent peer-to-peer protocol.

The subject of research - is the BitTorrent peer-to-peer protocol

The goal of the work – is to create an analytical system based on the

The goal of the work – is to create an analytical system based on the BitTorrent protocol, with which it will be possible to analyze torrent downloads from a separate IP address and classify them by type. Demonstrate anonymity vulnerabilities of usage the BitTorrent protocol

The programming part was implemented using the JavaScript programming language, the NodeJS software platform, and the Visual Studio Code development environment.

As a result, we received a working prototype of an analytical system that can be used by law enforcement agencies to search for suspicious/illegal activity, a prototype of a self-made spy BitTorrent client. The system is implemented using the JavaScript programming language, the PostgreSQL and Redis databases, the

BullMQ message broker, the TensorFlow artificial intelligence library, the Docker containerization system, and a client site written using the NextJS (React) framework.

Approbation of the results: Orlovsky D.I. Deanonymization in peer-to-peer networks: analysis of methods and examples based on the BitTorrent protocol // Week of science: annual science-practice. conf. of teachers, scientists, young scientists, graduate students and students, April 24-28, 2023: abstracts of reports. – Zaporizhzhia, 2023.

Orlovsky D.I. Development and software implementation of digital techniques intelligence based on open sources/ D.I. Orlovskyi, A.V. Nelasa // All-Ukrainian competition for the best student scientific work of the 2020/2021 academic year - Lviv, 2020.

Orlovsky D.I. Intelligence technique for open sources of information. OSINT technologies. Study of the BitTorrent protocol by means of OSINT"/ D.I. Orlovskyi, A.V. Nelasa // Week of science: annual science-practice. conf. of teachers, scientists, young scientists, graduate students and students, April 13-17, 2020: abstracts of reports. – Zaporizhzhia, 2020.

Orlovsky D.I. Technique of intelligence on open sources of information (OSINT). DATA SCRAPING as a means of OSINT / Orlovskyi D.I., Nelasa G.V. // Information technologies: theory and practice: III Vseukr. Internet Conf. applicants of higher education and young scientists, 2020: abstracts of reports. - Kharkiv, 2020. - C. 77-78.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

BEP (BitTorrent Enhancement Proposal) – пропозиції покращення bittorrent

DHT (Distributed Hash Table) – розподілена геш-таблиця;

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста;

IP (Internet Protocol) – інтернет протокол;

P2P (Peer to peer) – однорангова мережа;

RPC (Remote Procedure Call) – виклик віддалених процедур

SHA (Secure Hash Algorithm) – алгоритм безпечного хешування

TCP (Transmission Control Protocol) – протокол керування передачею;

UDP (User Datagram Protocol) – протокол датаграм користувача;

VPN (Virtual Private Network) – віртуальна приватна мережа;

ВСТУП

Упродовж останніх років відбувся значний розвиток інформаційного простору, який став більш відкритим і доступним. Це сприяло популяризації однорангових мереж (P2P) та протоколу BitTorrent. Ці технології забезпечують ефективний обмін даними між користувачами. Водночас, їх широка доступність відкриває шляхи для розповсюдження контенту, як легального, так і нелегального. Використання протоколу BitTorrent часто фігурує у кримінальних справах, пов'язаних з дистрибуцією неліцензійного програмного забезпечення, поширенням контенту жорстокого характеру та інших порушень. Така ситуація поставила перед правоохоронними органами завдання розробки методів деанонімізації користувачів P2P-мереж для ідентифікації осіб, що порушують закон.

Основною метою цієї дипломної роботи є дослідження та розробка методів деанонімізації користувачів в однорангових мережах, зокрема, з використанням протоколу BitTorrent. Робота передбачає створення програмного забезпечення для асистування правоохоронним органам у виявленні потенційних правопорушників та формуванні запитів до інтернет-провайдерів.

Актуальність теми обумовлена відсутністю в Україні власного програмного забезпечення для деанонімізації та аналізу трафіку в мережі BitTorrent без підтримки з боку іноземних держав-партнерів.

Для досягнення мети цієї роботи необхідно виконати наступні завдання:

– ознайомлення з принципами роботи однорангових мереж та протоколу BitTorrent;

– аналіз існуючих методів деанонімізації користувачів P2P-мереж, включаючи юридичну практику щодо деанонімізації у правоохоронних органах;

– розробка клієнтського програмного застосунку для моніторингу трафіку у мережі BitTorrent;

– створення сервера аналітичної системи завантажень в мережі BitTorrent;

– розробка серверу класифікації контенту за допомогою бібліотеки штучного інтелекту TensorFlow.

Методологія дослідження охоплює аналітичний огляд наукової літератури, нормативних актів, теоретичний аналіз протоколу BitTorrent та практичну реалізацію запропонованої системи.

Структура дипломної роботи буде включати такі розділи: вступ, принцип роботи однорангових мереж та протоколу bittorrent, методи деанонімізації в однорангових мережах, юридичні аспекти щодо користування мережею bittorrent, проєктування та розробка програмного застосунку з деанонімізації, розробка підробленого bittorrent-клієнту, розробка аналітичного сервера деанонімізації у p2p мережі, розробка штучного інтелекту класифікації bittorrent, розробка візуального інтерфейсу аналітичної системи, висновки.

Результатом цієї роботи стане реалізація аналітичної системи, що надасть правоохоронним органам візуальний веб-інтерфейс для відстеження завантажень за конкретними IP-адресами та формування запитів до інтернет-провайдерів. Це спростить процес аналізу та генерації юридичних запитів, ефективно сприяючи роботі правоохоронних органів.

1 ПРИНЦИПИ РОБОТИ ОДНОРАНГОВИХ МЕРЕЖ ТА ПРОТОКОЛУ BITTORRENT

1.1 Основні поняття однорангових мереж (P2P)

У рамках сучасного розвитку програмного забезпечення існує два основних напрямки у створенні архітектури комп’ютерних мереж, а саме: клієнт-серверна модель та модель однорангових мереж. Велика частина програмних застосунків надає перевагу клієнт-серверній моделі, що ілюструється на рисунку 1.1:

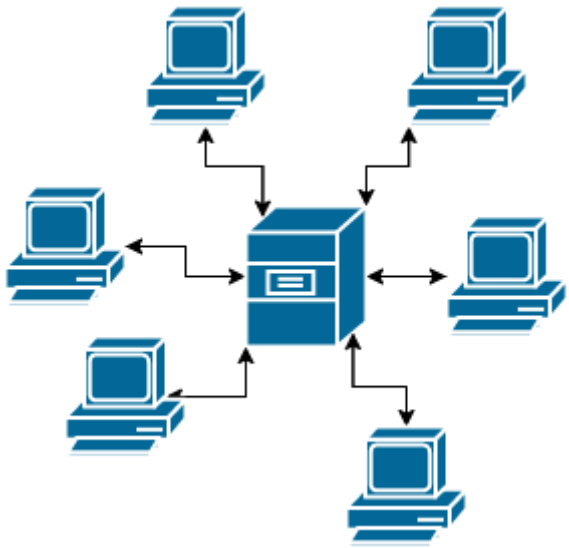


Рисунок 1.1 – Схема «клієнт-серверної» архітектури

Ця архітектура характеризується простотою впровадження завдяки наявності одного «надійного» джерела даних, з яким взаємодіє «клієнт». Серверна частина мережі може включати об'єднання серверів різних типів (поштового, веб, термінального) та баз даних. Однак, ця архітектура має деякі обмеження, зокрема, потребує регулярного технічного обслуговування, масштабування, що включає додавання або заміну серверних компонентів, та наявність кваліфікованих системних адміністраторів. При зростанні чисельності користувачів мережі та обсягів генерованої ними інформації, ці

проблеми стають більш вираженими та можуть призвести до збільшення витрат на утримання серверної частини компанії [2].

В контексті пошуку альтернативних рішень зазначених проблем варто звернути увагу на однорангові мережі (P2P, англ. Peer-to-Peer), що представлено на рисунку 1.2:

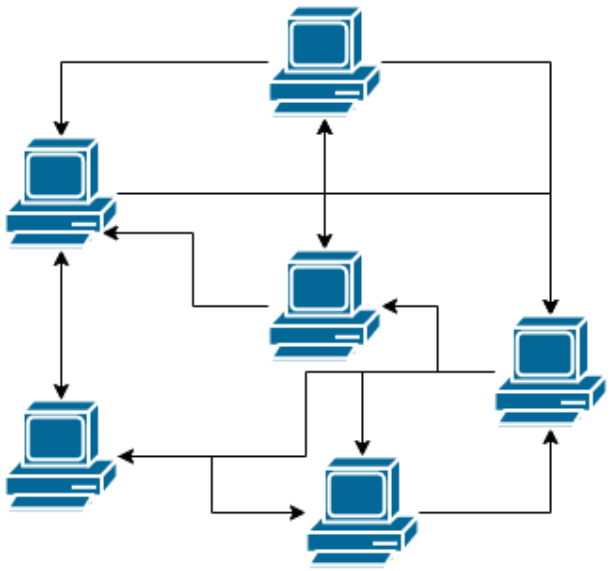


Рисунок 1.2 – Схематичне зображення однорангової архітектури

Однорангові мережі (P2P, від англ. Peer-to-Peer) – це тип мережевої структури, що забезпечує безпосередній (прямий) обмін даними між комп'ютерами-учасниками мережі без необхідності використання централізованих серверів. У таких мережах кожен комп'ютер відіграє роль як клієнта, так і сервера одночасно, забезпечуючи зберігання, обробку та передачу даних між учасниками.

Основний принцип роботи однорангових мереж полягає у відсутності фіксованих центральних серверів та ієрархічної структури. Учасники мережі здійснюють обмін ресурсами та даними безпосередньо між собою, використовуючи спеціалізовані серверні протоколи. Кожен член подібної мережі не гарантує своєї присутності на постійній основі: він має можливість як з’являтись, так і зникати у будь-який час, тому серверна машина повинна ще виконувати адміністративні функції та підтримувати актуальність

інформації о клієнтах. Подібна організація дозволяє зберегти повну працездатність мережі за будь-якою кількістю вузлів та різною апаратною конфігурацією.

Питання прикладного дослідження якості однорангових мереж порівняно з клієнт-серверними опрацьовано у дослідженні «Ghareeb, M., Rouibia, S., Parrein, B., Raad, M., & Thareau, C. (2013). P2PWeb: A Client/Server and P2P hybrid architecture for content delivery over internet. 2013 Third International Conference on Communications and Information Technology (ICCIT)» [3].

У контексті поглибленого вивчення архітектур комп'ютерних мереж, дослідницька група здійснила практичне випробування за допомогою створення прототипів файлових серверів згідно двух ключових моделей: «клієнт-серверною» та «одноранговою». Обидва сервери були налаштовані на пропускний канал зі швидкістю 2 Мбіт/с. Ці експерименти мали на меті вимірювання швидкості передачі файлів до клієнтів, кількість яких поступово збільшувалась до 23. Як показано на рисунку 1.3, одноранговий сервер спочатку обслуговував перші вісім клієнтів, а потім зменшив пропускну здатність до 300 Кбіт/с, виконуючи лише функцію передачі списку активних користувачів мережі. Таким чином, як це видно з графіків на рисунку 1.3, у клієнт-серверній моделі було зафіксовано пряму залежність затримки в передачі файлів від кількості клієнтів, що одночасно завантажували файли (навантажують сервер) [2,3].

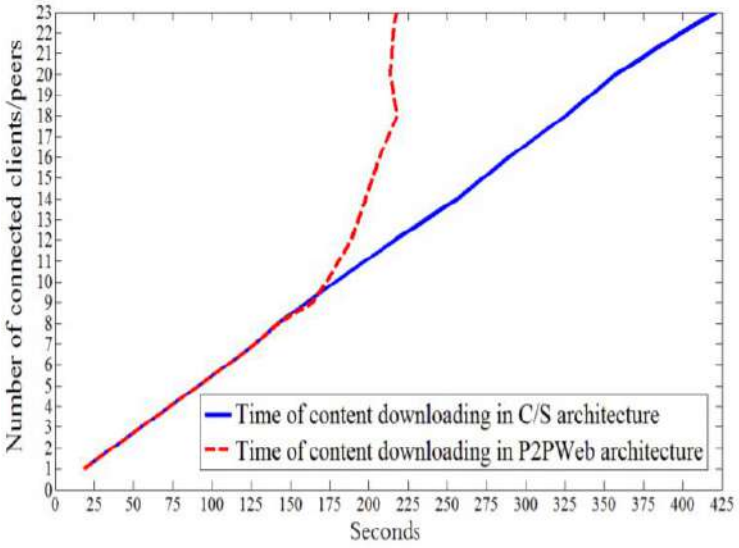


Рисунок 1.3 – Ілюстрація порівняння клієнт-серверної та однорангової архітектури за часом передачі інформації, де синій графік відображає час завантаження контенту з клієнт-серверного серверу, а червоний - з однорангового [3].

Однорангові мережі можуть бути класифіковані за різними параметрами, які відображають їх структуру, рівень анонімності, топологію та інші характеристики. Розглянемо основні види класифікацій однорангових мереж:

За структурою:

– чисті однорангові мережі (Pure P2P) – мережі, в яких відсутні централізовані сервери та усі учасники мають однакові права та можливості [4];

– гібридні однорангові мережі (Hybrid P2P) – мережі, що поєднують властивості чистих P2P-мереж та клієнт-серверних мереж. Вони можуть використовувати централізовані сервери для індексації, пошуку ресурсів або контролю за деякими аспектами мережі, в той же час забезпечуючи прямий обмін даними між учасниками [5].

За рівнем анонімності:

– неанонімні однорангові мережі – мережі, в яких учасники можуть вільно переглядати IP-адреси та інші ідентифікаційні дані інших учасників, що може спричинити проблеми з приватністю та анонімністю;

– анонімні однорангові мережі – мережі, в яких застосовуються технології шифрування, аутентифікації та інші методи, що забезпечують анонімність та конфіденційність даних учасників мережі.

За топологією:

– централізовані P2P-мережі – мережі, що використовують центральний сервер для управління ресурсами, індексації та пошуку, але передача даних відбувається безпосередньо між учасниками мережі. Прикладом такої мережі є Napster - мережа для обміну аудіофайлів;

– децентралізовані структуровані P2P-мережі – мережі, які використовують структуровані протоколи для розподілу та зберігання ресурсів та інформації. Зазвичай такі мережі використовують алгоритми DHT (розподілені хеш-таблиці) для забезпечення масштабованості та стабільності. Приклади структурованих P2P-мереж включають Kademlia, Chord, BitTorrent та Pastry;

– децентралізовані неструктуровані P2P-мережі – мережі, в яких немає чіткої структури для зберігання та розподілу ресурсів. Вони працюють на основі «випадкового пошуку» ресурсів серед учасників мережі. Прикладами неструктурованих P2P-мереж є Gnutella та eDonkey.

Слід зазначити, що в сучасному цифровому світі P2P-мережі знаходять застосування у багатьох областях, включаючи системи потокової передачі відео, файлообмінні системи, та мають особливе значення у сфері інтернет-фінансів. Однорангові мережі лягли в основу криптовалют, як-то Bitcoin, що забезпечують анонімність та безпеку фінансових операцій для їх клієнтів. Також ці мережі відіграють ключову роль у створенні децентралізованих фінансових систем (DeFi), які гарантують надійність та прозорість транзакцій.

1.2 Принципи роботи протоколу BitTorrent

BitTorrent – це протокол розподіленого обміну файлами, що використовує однорангову мережу для ефективної передачі даних. Завдяки своїй масштабованості та стабільності, протокол BitTorrent став одним з найпопулярніших способів обміну файлами в Інтернеті. Зокрема, у звіті компанії SandVine під назвою «The Global Internet Phenomena Report» надається факт, що більше 9.78 відсотків upsteam-трафіку (трафіку-заватнаження), що становить 1 місце у світі [6]. Також, 2.91 відсотків трафіку у всії мережі (9 місце у світі) належать BitTorrent [6]. Наведенна вище інформація сприяє необхідності формалізування та вивчення принципів роботи протоколу BitTorrent.

BitTorrent працює на основі трьох основних компонентів:

– торрент-файл (торрент) - метадані, що містять інформацію про розподілені файли, такі як розмір, структура та хеш-суми для перевірки цілісності [7];

– трекер - сервер, який відповідає за координацію обміну файлами між учасниками мережі (пірами) [7];

– піри - учасники мережі BitTorrent, які обмінюються частинами файлів між собою та продовжують завантажувати контент torrent-файлу [7].

BitTorrent використовує так званий принцип «роздавай тому, хто роздає іншим» (англ. tit-for-tat), що сприяє поліпшення співпраці (себто децентралізації мережі) між учасниками мережі [8].

Механізм роботи протоколу можна детально описати, використовуючи алгоритм, що складається з декількох послідовних кроків. Початковий етап процесу передбачає, що користувач виконує завантаження торрент-файлу, який містить важливі метадані та адресу трекера. Цей етап є фундаментальним для подальшого розвитку процесу обміну даними.

Надалі, клієнт протоколу BitTorrent ініціює процедуру підключення до трекера. Під час цього з'єднання, клієнт має пройти процес ініціалізації на трекері, передаючи свою IP-адресу та хеш-суму файлів, що запитуються. Цей етап, відомий як «анонс», є ключовим для наступної фази, під час якої клієнт отримує список пірів, які володіють необхідними частинами файлу або на даний момент займаються їх завантаженням [7].

Третій крок алгоритму полягає в тому, що клієнт здійснює підключення до вищезазначених пірів та починає процес обміну даними, що представляють собою частини файлу. Особливістю цього процесу є те, що завантаження файлу відбувається не в строго визначеному порядку, а у випадковій послідовності, що в свою чергу забезпечує максимально ефективне використання пропускної здатності мережі.

Додатково слід відзначити, що частини файлу, які вже були завантажені, передаються іншим учасникам мережі. Цей процес сприяє розподілу навантаження між користувачами та покращенню загальної пропускної здатності мережі.

Завершальний етап алгоритму полягає у продовженні процесу завантаження та передачі частин файлу до того моменту, поки користувач або не отримає весь файл повністю, або не прийме рішення про вихід з процесу роздачі. Цей крок є логічним завершенням описаного процесу обміну даними в рамках протоколу BitTorrent.Після завершення завантаження користувач може продовжувати роздачу файлів, приймаючи статус «сідом» (seed) мережі, тобто джерелом повного файлу з вищим пріоритетом на запити для інших пірів [7].

Одним з ключових аспектів роботи протоколу BitTorrent є оптимізація пропускної здатності та розподілу навантаження між учасниками мережі. Для досягнення цього протокол використовує такі механізми:

– режим ендшпілю (англ. endgame mode) - режим, який спрацьовує коли користувачу залишається завантажити невелику кількість частин файлу. Для нього, клієнт BitTorrent запитує інформацію щодо файлів від усіх

доступних пірів одночасно. Це дозволяє швидко завершити процес завантаження та зменшує вірогідність затримки через мережеві проблеми [9];

– суперсідування (англ. super-seeding) - режим роботи для учасників мережі з обмеженою пропускною здатністю, що дозволяє ефективно віддавати файли, передаючи кожну частину тільки один раз. Це забезпечує оптимальне використання пропускної здатності та покращує швидкість роздачі [10];

– система кредитів (англ. credit system) - механізм реєструє та враховує обсяг даних, переданих між пірами, і на основі цієї інформації визначає пріоритетність доступу до ресурсів мережі, ґрунтуючись на рівні попередньої активності учасників. Такий підхід стимулює користувачів до активного обміну файлами, забезпечуючи більш ефективний розподіл ресурсів у мережі [11].

1.3 Використання DHT у протоколі BitTorrent

Під час вивчення протоколу BitTorrent, однією з ключових характеристик, на які слід звернути увагу, є залежність від центрального серверу, відомого як трекер, в контексті процесу «анонсування» у мережі. Ця залежність може впливати на пропускну спроможність торрент-мережі, особливо у випадку коливань пропускної здатності Інтернет-каналу самого трекера. Не менш важливим фактором, що потребує особливої уваги, є можливий потенційний вплив третіх сторін на стабільність і коректність функціонування мережі, що може бути спричинене такими факторами, як DDoS-атаки на трекер чи його повне закриття власником. Такі виклики спонукають до пошуку шляхів усунення залежності від центрального трекера в BitTorrent-мережах.

Шляхом розв'язання цих проблем є концепція розподіленої хеш-таблиці (DHT), що являє собою структуру даних для зберігання пар ключ-значення в

рамках однорангової мережі. DHT забезпечує ефективний пошук і вилучення значень за відповідними ключами, використовуючи хеш-функції для перетворення ключів у індекси таблиці, що дозволяє точно визначати місце зберігання даних у мережі [12].

Серед визначальних особливостей DHT варто виокремити децентралізацію, що забезпечує високу стійкість мережі до відмов та сприяє її масштабуванню. Крім того, DHT характеризується масштабованістю, що дозволяє легко додавати або вилучати вузли без суттєвих затрат на реорганізацію мережі, та ефективністю пошуку, забезпечуючи швидкий доступ до даних навіть у великих мережах.

До найвідоміших реалізацій DHT належать Chord, Kademlia та Pastry, кожна з яких має свої унікальні особливості та застосування в різних системах:

– chord - одна з перших DHT, розроблена в MIT, використовує простий та ефективний алгоритм пошуку та вирішує проблеми завантаження вузлів та відмови [13];

– kademlia - алгоритм DHT, який використовується в багатьох однорангових системах, таких як BitTorrent. Kademlia відрізняється використанням XOR-метрики для визначення відстані між ключами та оптимізацією процесу пошуку [13];

– pastry - цей DHT алгоритм базується на структурі простору ключів та використовує ієрархічну адресацію для забезпечення ефективного пошуку та вставки [13].

З появою версії 4.2.0 офіційного клієнта BitTorrent, відбулося впровадження безтрекерової роботи на основі протоколу Kademlia, а також розширення для спілкування між клієнтами - PEX (Peer Exchange). Такі нововведення сприяють підвищенню автономності та ефективності однорангових мереж [13].

Комунікаційний механізм, що застосовується у мережах з використанням DHT-таблиць, базується на використанні протоколу UDP. Важливою складовою цього процесу є необхідність клієнтів слухати порт,

який працює з UDP, оскільки саме через нього вони приймають вхідні з'єднання.

У контексті DHT-мережі кожен пір функціонує як незалежний вузол, що має свій унікальний ідентифікатор (ID), який обирається випадково з 160-бітного простору хешу, що відповідає торенту. Цей аспект є фундаментальним для правильної роботи мережі. Кожен вузол утримує маршрутизаційну таблицю, що включає контактну інформацію про інші вузли, які є «найближчими» до нього. Важливо окремо зауважати, що поняття «близькості» у цьому контексті визначається за схожістю ID завдяки XOR операції і не корелюється з географічним розташуванням піра.

Коли вузол прагне віднайти окремий ресурс у BitTorrent-мережі, він здійснює порівняння хешу роздачі з ID вузлів, що знаходяться в його DHT-таблиці. По завершенню цього етапу порівняння, вузол знаходить інший вузол з найбільш схожим ID і відправляє до нього запит. Вузол, який отримує запит, у свою чергу, порівнює через XOR алгоритм запитаний ID з наявними у власній DHT-таблиці та направляє запитувачу контактні дані іншого піра, чий ID ще більше наближається до потрібного хешу.

У результаті цього алгоритму, вузол-шукач послідовно отримує контакти вузлів, чий хеш найбільше відповідає шуканому торенту. Кожен вузол в цьому ланцюжку запитів записує у свою DHT-таблицю інформацію про запитувача та результати запиту, тим самим збагачуючи таблицю актуальною інформацією та розширюючи мережу контактів [2,12].

Однак, важливо враховувати певні аспекти інформаційної безпеки, пов'язані з DHT-алгоритмом. Кожний клієнт, приєднуючись до мережі, ділиться власною IP-адресою та унікальним хешем, який ідентифікує його у мережі. Цей факт може бути проблематичним, оскільки користувачі (володільці IP-адреси) можуть бути одночасно учасниками інших мереж, де вони також діляться своїми даними (IP-адресою зокерма).

Це стає особливо значущим, враховуючи потенційні ризики, які можуть виникати з публічним розголошенням IP-адрес та унікальних хешів

користувачів у межах DHT-алгоритму. Ці дані можуть бути використані сторонніми особами або організаціями для виявлення патернів поведінки користувачів, а також для встановлення зв'язку між активністю користувачів у різних торент-роздачах.

Таким чином, можна сказати що протокол BitTorrent є потужним інструментом для розподіленого обміну файлами, що забезпечує високу швидкість передачі та масштабованість. Його основні принципи роботи та оптимізаційні механізми сприяють створенню стабільних та ефективних одноранговых мереж. Однак, для розробки та використання таких систем важливо враховувати обмеження та проблеми, пов'язані з аспектом приватності та деанонімізації.

2 МЕТОДИ ДЕАНОНІМІЗАЦІЇ В ОДНОРАНГОВИХ МЕРЕЖАХ

2.1 Загальний перелік популярних методів деанонімізації

Однорангові мережі у цілому, та BitTorrent зокерма, хоча і надають певний рівень анонімності для користувачів, не є повністю захищеними від спроб деанонімізації. Зловмисники чи правоохоронні органи можуть використовувати різні методи для виявлення ідентичності користувачів та аналізу їх діяльності. У цьому розділі ми розглянемо основні методи деанонімізації в однорангових мережах та способи протидії таким атакам.

Загалом, методи деанонімізації в P2P-мережах можна поділити на 2 основні способи (вектори атак):

- атака «Man in the Middle» (MiTM);
- атака Sybil.

2.2 Атака «Man in the Middle»

Man in the Middle - зловмисник розташовує себе між двома вузлами, що здійснюють обмін даними, тим самим перехоплюючи або модифікуючи передавані дані без відома легітимних учасників [14]. Зловмисник може виступати в ролі активного або пасивного спостерігача, в залежності від його цілей та можливостей [14].

У контексті мереж P2P, де кожен вузол одночасно виступає як клієнт і сервер, MitM атака набуває особливої актуальності [14]. Через відсутність централізованого контролю над розміщенням вузлів, зловмисник має можливість вільно інтегруватися у мережу, ускладнюючи виявлення такої атаки.

Реалізація атаки MitM у мережах P2P здійснюється через різні методи та може включати наступні кроки:

– перехоплення - зловмисник перехоплює канал зв'язку між двома одноранговими вузлами та використовує різні методи, щоб позиціонуватися в середині цього каналу [14]. Зазвичай використовуються такі методи, як отруєння кешу ARP (ARP-cache poisoning) у середовищах, що не є P2P, і імітація звичайного вузла в мережах P2P;

– прослуховування - після створення вразливого каналу, зловмисник пасивно прослуховує обмін даними [14]. Це дозволяє збирати конфіденційну інформацію, що передається між вузлами, наприклад особисті дані та інші конфіденційні матеріали.

Реалізація атаки MiTM може бути ускладнена різними захисними механізмами, такими як шифрування, імплементація методів аутентифікації користувачів. Однак, вразливості у конфігурації мережі та існуючому програмному забезпеченні можуть створювати точки дотику, які дозволяють нападникам успішно реалізувати атаку.

2.3 Атака Sybil

Атака Sybil - створення великої кількості підроблених або фальшивих вузлів з метою змінення поведінки мережі або здійснення деанонімізації користувачів. Атака дозволяє зловмиснику контролювати значну частину мережі та впливати на її роботу [1,15].

Атака Sybil зазвичай починається зі створення великої кількості фальшивих вузлів зловмисником. Ці вузли можуть бути реалізовані як окремі програмні екземпляри на одному комп'ютері або як окремі віртуальні пристрої, розподілені по різних мережевих вузлах. З метою збільшення ефективності атаки, фальшиві вузли можуть намагатися імітувати поведінку

реальних користувачів, включаючи обмін даними, відповідь на запити та інші типові дії.

Одним з основних цілей атаки Sybil є зміна характеристик роботи мережі на користь зловмисника. Це може включати зміну процесу вибору релейних вузлів, порушення протоколів консенсусу, забезпечення неконтрольованого доступу до даних користувачів та інші дії, які можуть підривати безпеку та стабільність мережі . В результаті атаки Сивіли зловмисник може здійснювати деанонімізацію користувачів, перехоплювати комунікації, блокувати транзакції або ресурси, а також проводити інші зловмисні дії [1,15].

Атака Сивіли є серйозною загрозою для P2P-мереж, оскільки вона може призвести до порушення приватності, перешкоджання обміну ресурсами, впливу на консенсус та підриву довіри до мережі.

2.4 Деанонімізація у BitTorrent-мережі

У контексті деанонімізації у протоколі BitTorrent, можливим є використання усіх перелічених вище алгоритмів. Вибір необхідного залежить від мети деанонімізації. Наприклад:

- якщо мета деанонімізації є основною або необхідно дізнатись інформацію щодо користувачів, які завантажують конкретно обрану Torrent роздачу - доцільним є обрати атаку атаку Sybil;
- якщо мета деанонімізації є другорядною, а пріорітетнішим є ускладнити чи навіть унеможливити доступ користувачів до специфічної Torrent роздачі - доцільним є обрати атаку Man in the Middle.

Викладений алгоритм роботи DHT у розділі 1.3 на основі DHT має суттєвий недолік зі сфери інформаційної безпеки: кожний клієнт при підключенні до рою віддає усій мережі власну IP-адресу та свій унікальний хеш який ідентифікує його у рою. Кожний клієнт, у цей же час, може бути

учасником інших мереж, у яких він так само надає власний хеш та IP-адресу іншим учасникам мережі.

Таким чином, подібна проблема відкритості даних надає великі можливості для експлуатації цієї вразливості. Для аналітика спеціальних служб необхідно розробити власний торент-клієнт основний функціонал якого збір «анонс»-запитів та їх подальший аналіз. Прикладом подібного сервіса може бути: програмне забезпечення, яке допомагає правовласникам відшуковувати порушників авторського права; розробка статистичної платформи для аналізу інтересів користувачів BitTorrent протоколу; розробка моніторингової системи, яка дозволяє аналізувати активність у потенційно «небезпечних» мережах через які розповсюджується заборонений контент.

Питання вивчення існуючих технічних реалізації таких зондів було проаналізовано нами у науковій роботі на Всеукраїнський конкурс на кращу студентську наукову роботу 2020/2021 навчального року з теми «Розробка та програмна реалізація методики цифрової розвідки на основі відкритих джерел» [2]. Згідно з висновків цієї роботи, можна сказати що на даний момент існують програмні рішення, які використовують вразливості DHT-таблиці для онлайн-пошуку за IP-адресою. Ці рішення дають можливість користувачам отримувати інформацію про файл, його вміст, час та дату завантаження, тип файлу тощо. Різні спеціалісти з різних сфер можуть використовувати ці пошукові сервіси для своїх завдань. Наприклад, правоохоронні органи можуть користуватись програмним комплексом «cps.gridcop» (США), а маркетологи та рекламні агенти - «iknowwhatyoudownload.com» (Німеччина).

Деанонімізація користувачів в однорангових мережах, зокрема BitTorrent, є актуальною проблемою, пов'язаною з приватністю та анонімністю. Для правоохоронних органів, вразливості які існують у мережі BitTorrent можуть бути використані з метою пошуку злочинців, що займаються розповсюдженням незаконно отриманого (піратського) або незаконного контенту.

3 ЮРИДИЧНІ АСПЕКТИ ЩОДО КОРИСТУВАННЯ МЕРЕЖЕЮ BITTORRENT

3.1 Юридичний аспект використання мереж BitTorrent

Законодавчі акти, які можуть регулювати питання використання мережі BitTorrent:

– Закон України «Про авторське право і суміжні права» (від 23.12.1993 № 3792-XII) встановлює правові засади захисту авторських прав на твори, які розповсюджуються через мережу інтернет;

– Закон України «Про захист персональних даних» (від 29.07.2022 № 2494-IX). Згідно з цим законом, персональні дані - це будь-яка інформація, яка вказує на конкретну особу або може бути ідентифікована як така.

Виходячи з практики використанням користувачами мережі BitTorrent, можна зробити невтішний висновок, що велика частина завантажень у мережі, є так званий «піратський» контент. Таким чином, користувачі, які незаконно завантажують та розповсюджують авторські твори через мережу BitTorrent, можуть бути відповідальні за порушення авторських прав. Згідно із Законом України «Про авторське право і суміжні права», порушення можуть призвести до цивільної, адміністративної чи кримінальної відповідальності, залежно від обсягу та характеру порушення.

Виходячи з наявної вище інформації, чином, учасники мережі BitTorrent можуть нести різні види юридичної відповідальності згідно з українським законодавством:

– адміністративна відповідальність - згідно з Кодексом України про адміністративні правопорушення, користувачі можуть нести адміністративну відповідальність за порушення правил користування мережею, наприклад, за розповсюдження нелегального контенту;

– цивільна відповідальність - учасники мережі BitTorrent можуть нести цивільну відповідальність за порушення авторських прав, що може призвести до відшкодування збитків власникам авторських прав;

– кримінальна відповідальність - у деяких випадках, розповсюдження нелегального контенту через мережу BitTorrent може призвести до кримінальної відповідальності, особливо якщо це стосується розповсюдження контенту, що містить матеріали, що пропагують насильство, контент пов'язаний з торгівлею людей або інші.

Українське законодавство передбачає посилення відповідальності за порушення, пов'язані з розповсюдженням контенту без дозволу авторів. Згідно цього, органи влади, провайдери інтернет-послуг та розробники програмного забезпечення повинні забезпечувати відповідний рівень захисту інформації та дотримання авторських прав, а також боротися з незаконним доступом до інформації. Таким чином, актуальність пошуку інформації щодо можливих порушників у мережі є актуальним для правоохоронних органів.

3.2 Юридичний аспект деанонімізації у мережі BitTorrent

Згідно норм українського законодавства, передбачена відповідальність за порушення захисту персональних даних користувачів. IP-адреса може виступати як приклад «персональних даних» (у деяких випадках). Відповідно до цього, якщо будь-які користувачі (зокрема правоохоронні органи) будуть збирати персональну інформацію про користувачів BitTorrent без відповідних дозволів або на підставі нечітких підстав, то це може розглядатися як порушення закону.

Законодавство також передбачає можливість звернення до суду у випадку порушення захисту персональних даних користувача у мережі Інтернет. Якщо користувач може довести, що його персональні дані були

розкриті без дозволу суду або на підставі нечітких підстав, то він може вимагати відшкодування заподіяної собі шкоди від цього витоку.

Однак, у цих правил можна помітити деякі особливі винятки:

– анонімізовані адреси - якщо IP-адреса була анонімізована, тобто знеособлена від прив'язки до конкретної особи (персони), то вона не є персональними даними. Наприклад, якщо веб-сайт збирає інформацію про відвідувачів, використовуючи анонімізовані IP-адреси, то ці дані не можуть бути пов'язані з конкретною людиною;

– динамічні IP-адреси - У деяких випадках провайдери Інтернету надають динамічні IP-адреси, тобто адреси, які змінюються з часом. У таких випадках IP-адреса не може бути пов'язана з конкретною особою, якщо провайдер не зберігає додаткової інформації, яка дозволить ідентифікувати користувача за його IP-адресою;

– правоохоронні органи - якщо працівники правоохоронних органів мають підставу на перевірку IP-адреси користувача, то вони можуть виконувати процесуальні дії щодо цього адресу. Також, загальновідомим є той факт, що правоохоронні органи України використовують системи по пошуку та ідентифікації учасників мережі BitTorrent з метою затримання злочинців (переважно у справах повє’язаних з розповсюдженням контенту пов’язаним з злочинами щодо експлуатації дітей). Яскравим прикладом можуть бути кримінальні провадження №12018040150000125 [16], №12022244000000060 [17] та багато інших. У цих матеріалах зображена процедура ідентифікації користувача у мережі BitTorrent за допомгою запиту на спеціалізовані платформи. Приклад з судового рішення: «14 вересня 2018 року Слідчий суддя Вільногірського міського суду Дніпропетровської області ОСОБА_1 , за участю секретаря ОСОБА_2 , розглянувши у судовому засіданні клопотання слідчого СВ Вільногірського ВП Жовтоводського ВП ГУНП в Дніпропетровській області ОСОБА_3 про проведення експертизи в кримінальному провадженні № 12018040150000125 … Слідчий СВ Вільногірського ВП Жовтоводського ВПГУНП в Дніпропетровської області

ОСОБА_3 12.09.2018 року звернулася до суду з клопотанням про призначення комплексної комп`ютерно-технічної експертизи та судово-мистецтвознавчої експертизи, вказуючи, що 04.04.2018 року до Вільногірського ВП надійшов рапорт співробітників УБЗПТЛ ГУНП в Дніпропетровській області про те, що в рамках вжиття заходів з виявлення кримінальних правопорушень було виявлено IP-адреси: «НОМЕР_1», «НОМЕР_2», з яких з території м.Вільногірська за допомогою програмного комплексу «Emule» через пірингову мережу протягом лютого-березня 2018 року здійснювалося розповсюдження продукції …. Відповідно до інформації, яка на даний час збережена в базі сервісу "ICACChildOnlineProtectiveServices" встановлено, що шляхом використання IP адреси НОМЕР_1 , у період часу з 27 грудня 2021 року 04:12 год по 03 січня 2022 року 05:27 год, здійснювалось закачування та розповсюдження іншим особам відео файлів, контрольні суми яких (унікальний цифровий підпис) відповідають таким, що містять ознаки сексуальної експлуатації дітей та неповнолітніх.» [16]. Згідно матеріалів цього ж судового рішення, правоохоронні органи України використовують програмні засоби «надані представниками правоохоронних органів США та Великобританії» [16,17].

Таким чином, можна зробити такі висновки:

– для користувача, що не є представником правоохоронних органів, збір інформації щодо завантаження контенту за окремою IP-адресою не є розкриттям персональних данних, так як звичайний користувач не зможе викрити конкретну особу якій належить ця адреса, тому цю адресу можна назвати «анонімізованою»;

– згідно з наявної у Україні судової практики, можна зробити висновок щодо легальності використання засобів деанонімізації з метою пошуку злочинців для правоохоронних органів без/та з використанням судових рішень (у контексті передбачення кримінального правопорушення).

4 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАТОСУНКУ ДЕАНОНІМІЗАЦІЇ У P2P МЕРЕЖАХ

4.1 Постановка задачі дослідження

За результатами отриманої під час дослідження інформації, автор поставив собі мету розробки програмного забезпечення, яке може автоматизувати процес пошуку та аналізу інформації, що була зібрана з однорангових мереж.

Згідно з отриманою у першій та другій частині роботи інформацією, однорангові мережі (зокрема BitTorrent) мають певний архітектурний недолік. Ці недоліки, при їх ефективному використанні, відкривають можливість виявлення детальної інформації про користувачів. Ця інформація включає, але не обмежується, IP-адресами користувачів, даними про завантажені ними торент-файли (як-от інфомрація щодо назви цих файлів, точний час їхнього завантаження, розширення файлів тощо).

Отримане програмне забезпечення має бути пристосоване для використання різноманітною авдиторією. Також, окрім інформації щодо зафіксованих завантажень торент-файлів за окремою IP-адресою, воно повинно аналізувати інформацію щодо можливого типу контента, який міг завантажити користувач: музика, програмне забезпечення, відео-ігри, фільми, контент для дорослих, тощо.

Правоохоронці можуть використовувати це ПЗ для моніторингу та забезпечення дотримання законодавства про цифровий контент, особливо в контексті боротьби з нелегальним чи забороненим контентом.

Автор бачить своєю основною аудиторію – представників правоохоронних структур. Таким чином, це ПЗ має мати функціонал генерації юридично оформлених документів-запитів до провайдерів щодо надання інформації о користувачах запитуваної IP-адреси.

4.2 Вибір мови програмування для розробки ПЗ

Перед тим, як почати розроблювати ПЗ, необхідно обрати коректний інструмент. У аспекті розробки ПЗ, основним інструментом розробки є «мова програмування». Згідно статистики порталу DOU, у 2023 році відсотком розробників що послугуються мовами програмування такими як Python (73.8% розробників), Golang (83.6% розробників) та JavaScript/TypeScript (78.6% розробників) [18].

Python це високорівнева, інтерпретована мова програмування з динамічною типізацією. Її головна перевага - простота синтаксису та велика стандартна бібліотека, що прискорює розробку. Python, як мова програмування, використовується у галузі наукових дослідженнь, під час розробці веб-додатків та у контексті автоматизації. Проте, Python має відносно низьку продуктивність у порівнянні з компільованими мовами програмування, через що він не завжди підходить для високонавантажених систем.

Golang (Go) - статична типізована, компільована мова, що створена компанією Google. Ця мова зсікала свою популярність та нішу своєю ефективністю, простотою та швидкістю виконання програмного коду. Golang зручний для розробки мережевих додатків, створення мікросервісів. Однак не дивлячись на такі плюси, синтаксис Go може здатися менш гнучким у порівнянні з Python, а бібліотеки ще не такі розвинені та екосистема мови ще не є сформованою.

JavaScript, на противагу, є мовою високого рівня, яка первісно була розроблена для веб-браузерів, але з часом її сфера застосування значно розширилася до багатьох: розробка серверних застосунків, мобільна розробка, розробка під персональні комп’ютери. Проте, переважно JavaScript використовується саме для розробки клієнтської частини застосунків та є невід’ємною частиною веб-розробки. Основні переваги цієї мови програмування є:

– універсальність - JavaScript може використовуватися як на клієнтській, так і на серверній частині програмного застосунку (Node.js), що дозволяє розробникам використовувати одну мову для розробки різних частин застосунку;

– масштабованість - завдяки асинхронному програмуванню та подієвій архітектурі (event based architecture) JavaScript може ефективно оброблювати велику кількість запитів. В результаті, це робить JavaScript ідеальним для створення високонавантажених застосунків;

– велика екосистема: - JavaScript має величезну кількість бібліотек та фреймворків для розробки різних частин програмного застосунку: для клієнтської частини (React, Angular, Vue), для серверної (Koa, Express, Fastify, NestJS), для мобільної розробки (Ionic, React Native), для розробки задля персональних комп'ютерів (ElectronJS). В результаті, це робить розробку швидшою та ефективнішою.

Враховуючи ці аспекти, JavaScript постає як мова, що поєднує гнучкість, масштабованість та широкі можливості для веб-розробки. Хоча Python та Golang мають свої власні сильні сторони, JavaScript є більш універсальним рішенням для розробки бажаного ПЗ. Таким чином, для реалізації програмного забезпечення для аналітики завантажень у одноранговій мережі BitTorrent будемо використовувати JavaScript.

4.3 Діаграма архітектури програмного забезпечення

У цьому розділі ми розглянемо архітектуру нашого програмного застосунку у цілому та зобразимо систему мікро сервісів які існують для його функціонування. На рисунку 4.1 зображено цю архітектуру:

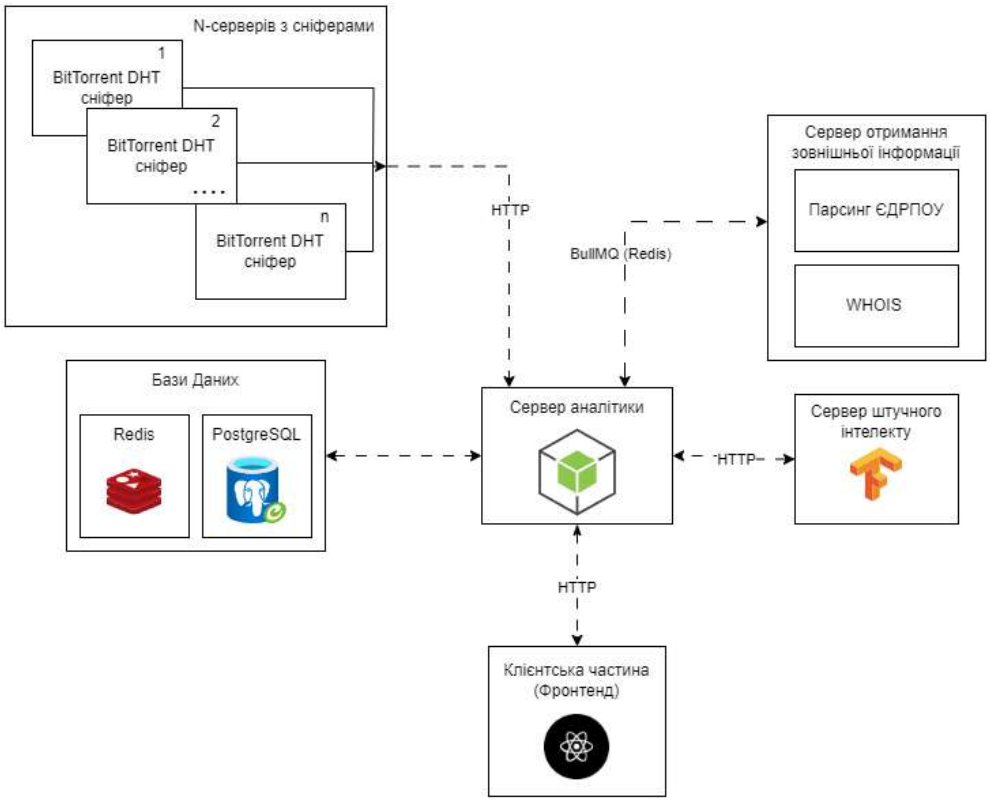


Рисунок 4.1 – Схематичне зображення архітектури програмного застосунку

Ось основні елементи, що виділяються, та про які буде розписано у відповідному модулі:

– набор серверів-сніферів - це кластер серверів з розгорнутими програмами для моніторінгу трафіку (які називаються «сніферами»). Ці сніфери підключаються до BitTorrent мережі за допомогою концепції розподіленої хеш-таблиці (DHT);

– бази даних – у цьому ПЗ використовуються дві системи управління базами даних: Redis та PostgreSQL;

– сервер аналітики – це сервер основної системи, який займається менеджментом даних з усіх джерел інформації (інших серверів). Цей сервер займається обробкою запитів та комунікацію з головними базами даних;

– клієнтська частина (Фронтенд) – це окрема платформа, на якій зберігається користувацький інтерфейс, що взаємодіє з сервером аналітики (основним сервером) через протокол HTTP;

– сервер отримання зовнішньої інформації – це сервер, що збирає інформацію з зовнішніх джерел. Цей сервер має підключення до механізму ініціації WHOIS запитів та до парсеру інформації щодо провайдерів за допомогою Єдиного державного реєстру підприємств та організацій України). Цей сервер займається обробкою інформації з різних зовнішніх серверів для отримання інформації щодо IP-адрес та ідентифікації реєстраційних даних цього провайдеру;

– сервер штучного інтелекту – це окремий сервер, який займається обробкою інформації за допомогою створеного штучного інтелекту.

Усі сервери між собою поєднані через мережу Internet завдяки HTTP протоколу чи через механізм RPC-повідомлень. У результаті, вся система у цілому виглядає як взаємопов'язана мережа серверів для збору, обробки та аналізу даних з різноманітних джерел.

5 РОЗРОБКА ПІДРОБЛЕНОГО BITTORRENT-КЛІЄНТУ

5.1 Вимоги до DHT-таблиці

Користуючись інформацією з розділу 1 (інформації щодо шляхів використання DHT у протоколі BitTorrent) та розділу 2 (інформація щодо способів деанонімізації у BitTorrent-мережі) можемо почати планувати розробку ПЗ, яке буде займатись безпосередньо прослуховуванням трафіку у одноранговій мережі BitTorrent.

Для того, щоб отриманий нами клієнт мав можливість коректно ідентифіковуватись у мережі іншими учасниками та отримувати від них інформацію, необхідно реалізувати у коді повноцінний клієнт BitTorrent з модифікацією DHT таблиці. Стандарт, яким послуговується переважна частина такого ПЗ це BEP005 «DHT Protocol», який регламентує процес пошуку, зберігання та вставку node у BitTorrent протоколі [19]. Дану специфікацію можемо вважати нашим технічним завданням.

Згідно з розділу «Routing Table» цієї специфікації, створимо відповідну таблицю технічних вимог цієї таблиці маршрутизаці:

Таблиця 5.1 – Вимоги до таблиці маршрутизації

Назва	Опис	Вимоги
Таблиця маршрутизації	Кожен вузол підтримує таблицю маршрутизації відомих вузлів. Вузли в таблиці маршрутизації використовуються як вихідні точки для запитів у DHT.	Всі вузли в таблиці маршрутизації мають бути відомими хорошими вузлами.
Визначення "хороших" вузлів	Хорошим вузлом є той, який відповів на один із наших запитів протягом останніх 15 хвилин.	Вузли, що не відповідають на запити протягом 15 хвилин, стають сумнівними.

Продовження таблиці 5.1

Розподіл на "відра"	Таблиця маршрутизації покриває весь простір ідентифікаторів вузлів від 0 до 2160. Вона поділена на "відра", кожне з яких покриває частину цього простору. Порожня таблиця має одне відро з діапазоном ідентифікаторів від min=0 до max=2160.	Кожне відро може містити лише K вузлів перед тим, як воно стане "повним".
Обробка повних відер	Коли відро заповнене хорошими вузлами, нові вузли відкидаються." Якщо відро містить поганих вузлів, один з них замінюється новим вузлом.	Якщо відро містить сумнівні вузли, відбувається перевірка («ping») найменш активного вузла.
Наявність мітки «остання зміна»	Кожне відро повинно підтримувати властивість «остання зміна», щоб вказувати, наскільки «свіжий» є його вміст.	Оновлення відбувається шляхом пошуку вузлів за допомогою випадкового ідентифікатора в діапазоні «відра».
Збереження таблиці маршрутизації	Після вставлення першого вузла в таблицю маршрутизації і при кожному наступному запуску, вузол повинен намагатися знайти найближчі вузли в DHT до себе.	Таблиця маршрутизації повинна зберігатися між викликами клієнтського програмного забезпечення.

Завдяки масштабованості платформи NodeJS існує стабільне рішення для реалізації подібної таблиці – модуль «k-bucket» [20]. Цей модуль зберігає об’єкти називаємі «contact». Ці об’єкти репрезентують адресу ноди у мережі BitTorrent та конструкцію IP-адреса:порт. Ця бібліотека працює через обробку бінарних даних (Buffer) що з’являються під час обробки запитів у протоколі BitTorrent. Ця бібліотека має дві основні функції:

– arbiter – функція що вирішує виникаючі конфлікти між двома об’єктами «contact» з однаковими ідентифікаторами у DHT [20]. Ця функція приймає 2 аргументи: перший, це «contact» що на даний момент зберігається у таблиці та другий, це «contact» який бажають додати до таблиці. Результатом виконання цієї функції буде новий «contact», у якому будуть вирішені, під час зберігання інформації, конфлікти;

– distance – функція для визначення відстані між двума нодами завдяки XOR алгоритму [19, 20]. Результатом виконання цієї функції буде повернуте натуральне число, яке буде відповідати правилу: чим менше число тим більш ближча ця нода є у таблиці поміж порівнюваними.

Під час створення сутності класу KBucket, її конструктор приймає такі аргументи як: arbiter (опціональний параметр), distance (опціональний параметр), localNodeId (опціональний параметр, який репрезентує ідентифікатор локальної ноди), metadata (опціональний параметр, який зберігає додаткову інформацію що буде переслана до нод), numberOfNodesPerKBucket (опціональний параметр, який визначає максимальну вмістимось бакету), numberOfNodesToPing (опціональний параметр, який визначає кількість нод для перевірки за одну ітерацію). За замовченням, параметр numberOfNodesPerKBucket дорівнює 20, а параметр numberOfNodesToPing дорівнює 3.

Для підтримки функціонування бакету клас KBucket має також набір «сервісних» методів, перелік яких наведемо у таблиці 5.2:

Таблиця 5.2 – Перелік сервісних методів класу KBucket з їх описом

Метод	Опис
add(contact)	Додає contact до k-bucket
closest(id, [n])	Повертає n найближчих contact до заданого id
count()	Повертає кількість contact у дереві
get(id)	Отримує contact за заданим id
remove(id)	Видаляє contact за заданим id

Продовження таблиці 5.2

toArray()	Повертає всі contact у вигляді масиву
toIterable()	Повертає всі contact як ітерабельний об'єкт

З метою створення коректної комунікації поміж сутності класу KBucket та класу DHT протоколу, розробниками KBucket створено систему «подій». Перелік цих подій зображено у таблиці 5.3.

Таблиця 5.3 – Перелік подій генеруємих KBucket з їх описом

Метод	Опис
'added'	Видається при додаванні нового contact
'ping'	Видається, коли додається contact, що перевищує місткість k-bucket
'removed'	Видається при видаленні contact
'updated'	Видається при оновленні contact

Таким чином, беручи до уваги цей перелік подій та сервісних методів можемо зробити висновок, що цей модуль повністю покриває наші потреби.

5.2 Що таке bencode? Як працює K-RPC протокол

5.2.1 Bencode

Для DHT протоколу існує спеціальний метод кодування контактної інформації з пірів та вузлів в мережі. Ця методика є ключовою для ефективної організації мережевих взаємодій, особливо в контексті розподілених систем і технологій P2P [19].

«Контактна» інформація щодо пірів кодується у якості 6-байтового рядку. Ця інформація відома як «компактна інформація IP-адреси/порту». Подібний формат включає IP-адресу зберігающуюся у 4-байтах із приєднаним

на кінці 2-байтовим портом. Такий підхід дозволяє оптимізувати процес передачі IP-адреси та порту в мінімалізованому форматі, що є важливим для зниження мережевого трафіку [19].

«Контактна» інформація щодо вузлів кодується як 26-байтовий рядок, відомий як «компактна інформація вузла». У такому форматі, 20-байтовий ідентифікатор вузла у мережевому порядку байтів доповнюється компактною інформацією IP-адреси/порту. Таке кодування ідентифікатора вузла разом з IP-адресою та портом забезпечує необхідну точність і ефективність при ідентифікації та з'єднанні поміж вузлами у BitTorrent мережі [19].

5.2.2 K-RPC протокол

Протокол K-RPC – це механізм RPC (віддаленого виклику процедур), який використовує бенкодовані словники, передані через UDP. У протоколі передбачено відправлення одного запиту і відповідне отримання однієї відповіді (з підтримкою того, що ця відповідь буде саме унікальна – не прийшла на інший запит). Механізм KRPC підтримує три типи повідомлень: запит, відповідь та помилка. У рамках протоколу DHT використовуються чотири типи запитів: ping (пінг), find_node (знайти вузол), get_peers (отримати піри) і announce_peer (оголосити піра) [19].

Кожне повідомлення KRPC представляє собою окрему структуру з трьома ключами, які є спільними для всіх повідомлень, та додатковими ключами, залежно від типу повідомлення.

У кожному повідомленні є ключ «t» зі значенням у вигляді рядка, який представляє ідентифікатор транзакції [19]. Цей ідентифікатор транзакції генерується самим вузлом, який робить запит, і буде надіслано знов у відповіді, тому відповіді можуть бути віднесені до кількох запитів до одного і того ж вузла. Ідентифікатор транзакції має бути закодований як короткий

рядок бінарних чисел, зазвичай достатньо 2 символів, оскільки вони можуть покривати 2^16 незавершених запитів [19].

Кожне повідомлення також містить ключ «y» з одним символом, що описує тип повідомлення [19]. Значення ключа «y» може бути одним із наступних: «q» для запиту, «r» для відповіді або «e» для помилки. У кожному повідомленні слід включати ключ «v» з рядком версії клієнта [19]. Цей рядок повинен містити двосимвольний ідентифікатор клієнта, зареєстрований у BEP 20, за яким слідує двосимвольний ідентифікатор версії. Однак не всі реалізації включають ключ "v", тому клієнти не повинні припускати його наявність [19].

5.3 Реалізація DHT запитів

5.3.1 Основний перелік DHT запитів

Для того, щоб робити коректний запит у мережі, необхідно закодувати цей запит у формат bencode. Кожен запит у DHT має містити ключ «id» та значення, що включає ідентифікатор вузла, який виконує запит [19]. Аналогічно, кожна відповідь також містить ключ «id», що вказує ідентифікатор вузла, який відповідає на цей запит.

Наведемо у таблиці 5.4 загальний перелік основних запитів у DHT:

Таблиця 5.4 – Перелік основних запитів у DHT таблиці

Назва	Опис	Запит/Відповідь	
ping	Запит, який використовується для перевірки активності (актуальності) інших вузлів.	Запит: `{"t":"aa", "y":"q", "q":"ping", "a":{"id":"abcdefghij0123456789"}}`	Відповідь: `{"t":"aa", "y":"r", "r": {"id":"mnopqrstuvwxyz12345g hfgdh gfdh fgd hgfdhgfd hgfd hgdfh gfdh6"}}`

Продовження таблиці 5.4

find_node	Запит, що використовується для знаходження контактної інформації щодо вузла за його ID серед інших учасників BitTorrent мережі.	Запит: `{"t":"aa", "y":"q", "q":"find_node", "a": {"id":"abcdefghij0123456789", "target":"mnopqrstuv”}}`	Відповідь: `{"t":"aa", "y":"r", "r": {"id":"0123456789abcdefghij", "nodes": "def456..."}}`

get_peers	Запит для отримання учасників мережі, асоційованих з інфохешем торрента.	Запит:
		`{"t":"aa", "y":"q", "q":"get_peers", "a": {"id":"abcdefghij0123456789", "info_hash":"mnopqrstuvwxyz123456"}}`
			Відповідь з наявними вузлами:
			`{"t":"aa", "y":"r", "r": {"id":"abcdefghij0123456789", "token":"aoeusnth", "values": ["axje.u", "idhtnm"]}}`
			Відповідь із найближчими вузлами:
			`{"t":"aa", "y":"r", "r": {"id":"abcdefghij0123456789", "token":"aoeusnth", "nodes": "def456..."}}`

announce_peer	Запит, що маркує пір який контролюється запитуючим вузлом, дійсно завантажує торрент у мережі.	Запит: `{"t":"aa", "y":"q", "q":"announce_peer", "a": {"id":"abcdefghij0123456789", "implied_port": 1, "info_hash":"mnopqrstuvwxyz123456", "port": 6881, "token": "aoeusnth"}}`	Відповідь: `{"t":"aa", "y":"r", "r": {"id":"mnopqrstuvwxyz123456"}}`

Кожен тип запиту та відповідь у цій таблиці відображає специфічну функціональність та формат даних у протоколі DHT, що є важливим для забезпечення правильної взаємодії між вузлами в мережі.

5.3.2 Реалізація BitTorrent DHT на платформі NodeJS

З метою створення таких запитів на платформі NodeJS існує готовий модуль, розроблений командою webtorrent, що реалізує запити у таблиці 5.4 – це «bittorrent-dht» [21].

Модуль «bittorrent-dht» використовується як частина веб-клієнту відомого Torrent-трекеру і має велику кількість завантажень (6500 за тиждень). Цей модуль також використовує KBucket, означений у розділах 5.2. Основна його мета це використання у якості підтримки «безфайлового» пошуку торенту, за допомогою magnet-посилання [21]. Власне, цей функціонал і наведений нижче:

```
import DHT from 'bittorrent-dht';
import magnet from 'magnet-uri';
const uri =
'magnet:?xt=urn:btih:e3811b9539cacff680e418124272177c47477157';
const parsed = magnet(uri);
const dht = new DHT();
dht.listen(20000, function () {
console.log('now listening');
});
dht.on('peer', function (peer, infoHash, from) {
console.log('found potential peer ' + peer.host + ':' + peer.port
+ ' through ' + from.address + ':' + from.port);
});
dht.lookup(parsed.infoHash);
```

У даному програмному коді зображено отримання інформації щодо вузлів, які завантажують вказане magnet-посилання. У нашому випадку, це список людей які завантажували офіційний дистрибутив ОС Ubuntu 18.

Наведемо перелік методів, які використовує ця бібліотека:

Таблиця 5.5 – Перелік основних методів у модулі «bittorrent-dht»

Метод	Опис
dht = new DHT([opts])	Створює новий екземпляр DHT. Можливо переназначення параметрів за допомогою opts

Продовження таблиці 5.5

dht.lookup(infoHash, [callback])	Виконує рекурсивний пошук однолітків для переданого infoHash
dht.listen([port], [address], [onlistening])	Монутє DHT на прослуховування заданого порту
dht.address()	Повертає інформацію щодо адреси, яку прослуховують
dht.announce(infoHash, [port], [callback])	Оголошує, що пір, який контролюється запитувальним вузлом, завантажує торрент на вказаному порті
arr = dht.toJSON()	Повертає поточний стан DHT, включно з вузлами DHT
dht.addNode(node)	Ручне додавання вузла до маршрутизаційної таблиці DHT
dht.destroy([callback])	Очіщує вміст DHT таблиці та закриває сокет з’єднання
dht.put(opts, callback)	Записує довільне «навантаження» у DHT таблицю
dht.get(hash, opts, callback)	Читає запис даних з DHT

Таким чином, можемо сказати що ця бібліотека повністю задовольняє нашу потребу у функціоналі протоколу DHT засобами NodeJS.

5.4 Розробка функціоналу сніферу

5.4.1 Огляд залежностій у класі імпорту

Згідно з отриманою інформацією у попередніх розділах можемо приступити до розробки класу, який буде прослуховувати BitTorrent мережу з метою отримання інформації щодо завантажень. Отриманий нами клас повинен мати змогу працювати у режимі «багатопоточності» задля пришвидшення роботи та оптимізації процесів. На рисунку 5.1 зображена діаграма створеного нами класу.

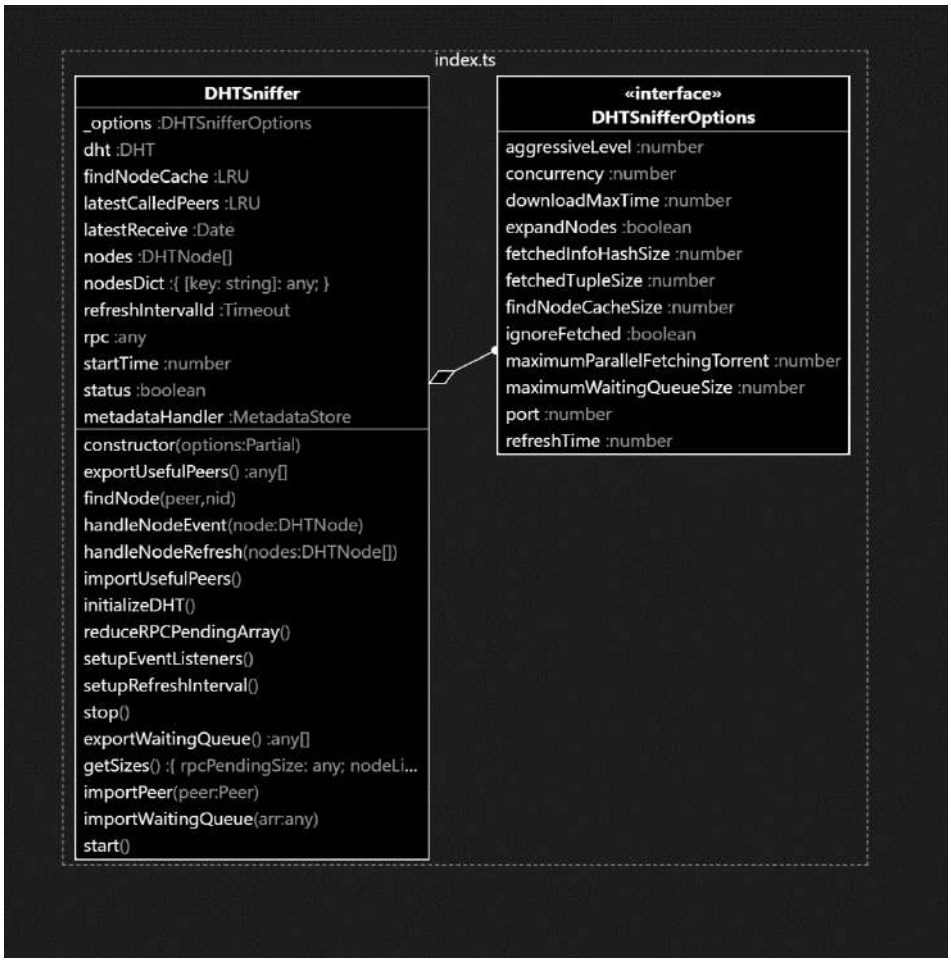


Рисунок 5.1 – Схематичне зображення архітектури програмного застосунку

Для того, щоб почати розробляти цей клас нам необхідно створити відповідний файл та додати залежності:

```
import * as crypto from "crypto";
import { EventEmitter } from "events";
import * as LRU from "lru-cache";
import DHT from 'bittorrent-dht'
import { parseMetaData } from "../metadata-helper";
import * as utils from "../utils";
import { MetadataStore } from "./MetadataStore";
import { DHTNode, Peer } from "./type.t";
```

Оголошуємо необхідні імпорти для опрацювання класу. Основний імпорт для нас, це вбудований у NodeJS модуль «EventEmitter». Цей модуль є нативною реалізацією паттерна проєктування з назвою «Підписник» у середовищі NodeJS. Ключова ідея цього паттерну полягає у наданні можливості повідомляти про різні події у будь-якій частині програмного

додатку. Цей підхід дозволяє забезпечити взаємодію між компонентами системи через механізм підписки на події.

Модуль crypto у NodeJS являє собою вбудовану бібліотеку, призначену для виконання криптографічних операцій, таких як шифрування, дешифрування, генерація цифрових підписів та перевірка їх справжності.

Модуль «lru-cache» на платформі Node.js реалізує алгоритм кешування Least Recently Used (LRU). Він використовується для оптимізації процесів зберігання та доступу до даних у ПЗ. Основна ідея LRU-алгоритму полягає в підтримці порядку елементів у кеші залежно від частоти їх використання. Це означає, що коли кеш досягає своєї максимальної ємності, найменш використані елементи будуть вилучені для звільнення місця для нових елементів.

З модулю «metadata-helper» імпортуємо створену нами функцію для парсингу метадати отримуваної з BitTorrent та перетворення у формат JS Object.

З модулю «utils» імпортуємо усі функції, що створені для опрацювання bencode формату та опрацювання вузлів BitTorrent.

5.4.2 Огляд методів та конструктору класу

Маючи на увазі інформацію щодо існуючих імпортів, створимо базовий прототип класу:

```
class DHTSniffer extends EventEmitter {
    private settings: DHTSnifferOptions;
    private dht: DHT = null;
    private startTime: number;
    private latestReceive: Date;
    private refreshIntervalId: NodeJS.Timeout; private rpc: any;
    private status: boolean = false;
    private findNodeCache: LRU<string, any>;
    private latestCalledPeers: LRU<string, any>;
```

```
    private nodes: Array<DHTNode> = [];
    private nodesDict: { [key: string]: any } = {};
    metadataHandler: MetadataStore;
    … }
```

Ці методи є базовими для повноцінного функціонування цього класу. Більш докладно має сенс зупинитись на полях `settings` та `metadataHandler`. У полі `metadataHandler` збергіається сутність обробника метаданих мережі BitTorrent, які використовуються коли ми будемо отримувати інформацію щодо контенту завантажуваного за вказаним id у BitTorrent. У полі `settings` зберігається інформація отримана у конструкторі класу та передані туди як параметри налаштуванн. Інтерфейс поля `settings` виглядає так:

```
interface DHTSnifferOptions {
    port: number;
    refreshTime: number;
    maximumParallelFetchingTorrent: number;
    maximumWaitingQueueSize: number;
    downloadMaxTime: number;
    expandNodes: boolean;
    ignoreFetched: boolean;
    concurrency: number;
    fetchedTupleSize: number;
    fetchedInfoHashSize: number;
    findNodeCacheSize: number;
    aggressiveLevel: number;
}
```

Також, інтерфейс зображений вище буде переданий у якості аргументу до конструктора. Повний код конструктору класу сніфера виглядає так:

```
constructor(options: Partial<DHTSnifferOptions>) {
        super();
        const defaultOptions: DHTSnifferOptions = {
            port: 6881,
            refreshTime: 1 * 60 * 1000,
            maximumParallelFetchingTorrent: 16,
            maximumWaitingQueueSize: -1,
            downloadMaxTime: 30000,
            expandNodes: false,
            ignoreFetched: true,
            concurrency: 16,
            fetchedTupleSize: 4000,
            fetchedInfoHashSize: 1000,
            findNodeCacheSize: 4000,
            aggressiveLevel: 0,
        };
```

```
      this.settings = { ...defaultOptions, ...options };
      this.metadataHandler = new MetadataStore(this.dht, this,
this.settings);
      this.findNodeCache = new LRU({
                  max: this.settings.findNodeCacheSize,
                  ttl: 24 * 60 * 60 * 1000,
                  updateAgeOnHas: true,
              });
      this.latestCalledPeers = new LRU({ max: 1000, ttl: 5 * 60 * 1000
});
          }
```

У коді конструктора ми ініціалізуємо наші поля відповідними значеннями. Особливої уваги мусимо звернути на код metadataHandler, у якому зберігається сутність іншого класу `MetadataStore`, про який ми сказали раніше. Також, нами було створено LRU-кеш, для зберігання останніх запитів find_node у протоколі BitTorrent з часом життя інформації щодо запиту протягом 24 годин. Останнім заповненим нами полем є LRU-кеш для зберігання інформації щодо вже колись запитуваних пірів іншими учасниками мережі з часом життя цієї інформації приблизно 5 хвилин.

Проте, факт створення сутності нашого класу не є гарантією старту процесу пошуку у мережі. Для цього нам треба запустити декілька асинхроних процесів, які ми створимо у основному тілі цього класу.

5.4.3 Огляд загальних методів класу сніферу

5.4.3.1 Метод старту сніферу

Для того, щоб клас почав свою роботу нам потрібно створити стартовий метод для цього. У нашому випадку це буде метод `start`:

```
  start() {
        if (this.status) {
            console.log("The sniffer is already working");
            return;
        }
        this.initializeDHT();this.setupEventListeners();
```

```
            this.setupRefreshInterval();
            this.status = true;}
```

У цьому публічному методі, ми викликаємо 3 методи, що виконують ініціалізацію усіх слухачів подій та ініціалізацію інформації щодо вже відомих нод, які були збережені попередньо у зовнішній базі даних.

Перший використаний метод це метод ініціалізації (initializeDHT). Основний функціонал цього методу полягає у тому, щоб ініціалізацію загального DHT класу (про нього було сказано у розділі 5.3). Також, означений поточний час як час старту процедури пошуку. Остання операція що виконується у цьому методі це ініціалізація старту прослуховування таблиці DHT і надання інформації про це через подію start. Загальний код цього методу ви можете побачити нижче:

```
        private initializeDHT() {
            this.dht = new DHT({ concurrency:
this.settings.concurrency || 16 });
            this.rpc = this.dht._rpc;
            this.latestReceive = new Date();
            this.startTime = Date.now();

    this.dht.listen(this.settings.port, () => {
    console.log(`DHT init: now listening:${this.settings.port}`);
    this.emit("start", { startTime: this.startTime, ...this.settings
});
            });
        }
```

Другий використаний нами метод, це метод ініціалізації слухачів (setupEventListeners). У нашому випадку початок слухання таких подій, як: warning, error, get_peers, node, announce, peer. Частина цих подій співпадає з основними запитами з таблиці 5.4 та ідентифіккують про цю активність (get_peers, announce), а ті що відсутні відповідають за індикацію помилок (error, warning) чи зміни статусу наявного вузла (node) чи знаходження нового активного вузла у мережі (peer).

```
        private setupEventListeners() {
            this.dht.on("warning", (err) => this.emit("warning",
err));
            this.dht.on("error", (err) => this.emit("error", err));
            this.dht.on("get_peers", (data) => {
```

```
                this.importPeer(data.peer);
                this.emit("infoHash", data.infoHash, data.peer);
            });
     this.dht.on("node", (node: DHTNode) =>
this.handleNodeEvent(node));
     this.dht.on("announce", (peer: Peer, infoHash, from) => {
     console.log("announce: ", peer, infoHash.toString("hex"),
from);});
     this.dht.on("peer", (peer: Peer, infoHash, from) => {
     this.importPeer(peer);
     this.metadataHandler.addQueuingMetadata(infoHash, peer, true);
     });}
```

Третій використаний нами метод, це метод для підтримки актуальності роботи нашої мережі (setupRefreshInterval). У цьому випадку він виконує ряд ключових операцій, керуючись часовими інтервалами, заданими в налаштуваннях мережі, для підтримки стабільності та реактивності системи. Цей метод використовує конструкцію JavaScript – setInterval для задання періодичність виконання визначених дій. У цьому інтервалі, метод виконує збір масиву вузлів DHT з подальшою переробкою їх у відповідний об’єкт (словник), де ключ це текстова конструкція «ip:port», а значення це число 1(як індекатор наявності цієї пари. Далі робиться процедура змішування вузлів з метою їх подальшого швидкого доступу, про результат чого сигналізується методом handleNodeRefresh. Якщо кількість вузлів менше 3 то ініціалізується процес пошуку нових вузлів, а якщо кількість незавершених запитів більше 1000 то кількість нод зменшується (і відповідно їх запитів також). Далі робляться сервісні методи, що роблять прискорення отримання інформації шляхом обмеження кількості паралельних процесів. Остаточний код методу це:

```
       private setupRefreshInterval() {
            this.refreshIntervalId = setInterval(() => {
                const nodes = this.dht._rpc.nodes.toArray() as
DHTNode[];
                this.nodes = nodes;
                this.nodesDict = nodes.reduce( prev, curr) => ({
...prev, [utils.getPeerKey(curr)]: 1,}), {});
                utils.shuffle(this.nodes);
                this.handleNodeRefresh(nodes);
                if (nodes.length <= 3) this.dht._bootstrap(true);
```

```
                if (this.rpc.pending.length > 1000)
this.reduceRPCPendingArray();
                this.metadataHandler.shuffleQueue();
                this.metadataHandler.boostMetadataFetching();
                this.importUsefulPeers();
            }, this.settings.refreshTime);
        }
```

Реалізація та впровадження цих трьох методів у класі є вирішальним для запуску та підтримки ефективної роботи у DHT сніфері.

5.4.3.2 Методи обробки подій DHT таблиці

У попередньому розділі ми використовували декілька сервісних методів обробки подій, а саме – handleNodeEvent, handleNodeRefresh, findNode.

Метод handleNodeEvent використовується для обробки подій, коли стає відомо що окрема нода змінила свій статус. Першою дією у цьому методі є встановлення поточної дати і часу як останнього моменту отримання даних. Далі, інформація що нода модіфікована сигналізується усім користувачам класу-сніфера. Останньою дією є перевірка за особливою умовою щодо необхідності робити новий запит на пошук нової ноди (якщо вузол не був оброблений та якщо навантаження на піри не велике та якщо кількість нод меньше 400).

```
      private handleNodeEvent(node: DHTNode) {
            this.latestReceive = new Date();
            this.emit("node", node);
            const nodeKey = `${node.host}:${node}`;
            const shouldFindNode =!this.findNodeCache.get(nodeKey) &&
this.latestCalledPeers.get(nodeKey) && Math.random() > 0.1 +
this.rpc.pending.length / 10 && this.nodes.length < 400;
            if (shouldFindNode) this.findNode(node, node.id);
        }
```

Метод handleNodeRefresh призначений для оновлення та підтримки актуальності інформації про вузли в мережі DHT. Цей процес ініціюється за двома основними умовами: коли налаштування системи передбачають

розширення списку вузлів чи коли минув визначений часовий інтервал з моменту останнього отримання даних (new Date().getTime() - this.latestReceive.getTime() > this.settings.refreshTime). Для кожного вузла у списку nodes формується унікальний ключ nodeKey і перевіряється на виконання умова щодо виклику методу findNode, що ідентична до коду з попереднього методу. Цей метод є важливий для результуючої стабільності мережі. Код цього методу наведений нижче:

```
     private handleNodeRefresh(nodes: DHTNode[]) {
     if (this.settings.expandNodes || new Date().getTime() -
this.latestReceive.getTime() > this.settings.refreshTime)
     nodes.forEach((node) => {
            const nodeKey = `${node.host}:${node.port}`;
     const shouldFindNode = this.nodes.length < 5 ||
(!this.latestCalledPeers.get(nodeKey) && this.nodes.length < 400
&&.random() > this.rpc.pending.length / 12);
     if (shouldFindNode) {.findNode(node, this.rpc.id);}});
     }
```

Останній необхідний метод для працювання сніферу, це метод findNode. Цей метод використовується для повторного пошуку інформацію щодо вузлів у мережі. Цей метод ініціює запит до вказаного вузла (peer) з метою знаходження інших вузлів. Ключовий аспект цього методу - використання ідентифікатора вузла (nid), який визначає, чи слід шукати сусідні вузли чи використовувати ідентифікатор поточного вузла (this.dht.nodeId). Після формування запиту відбувається відправлення запиту через DHT, після чого аналізуються отримані відповіді. У випадку помилки вузол може бути видалений з мережі, а отримані дані про інші вузли імпортуються для подальшої обробки.

```
     private findNode(peer: DHTNode, nid: Buffer) {
     const nodeKey = utils.getPeerKey(peer);
     this.findNodeCache.set(nodeKey, 1);
     this.latestCalledPeers.set(nodeKey, 1);
     const id = nid ? utils.getNeighborId(nid, this.dht.nodeId):
this.dht.nodeId;
     const message = {t: crypto.randomBytes(4), y: "q",q: "find_node",
a: {id,target: crypto.randomBytes(20), },};
     this.dht._rpc.query(peer, message, (err, reply) => {
     try {
```

```
     if(peer?.id&&this.rpc.nodes.get(peer?.id) &&
utils.isNodeId(peer?.id, 20)) {
     if (err && ["EUNEXPECTEDNODE", "ETIMEDOUT"].includes(err.code))
{this.rpc.remove(peer.id); }}} catch (e) { // do nothing}
     if (reply?.r?.nodes)
{utils.parseNodes(reply.r.nodes,20).filter((node) =>
utils.isNodeId(node.id, 20)).forEach((node) => this.importPeer(node));
}});}
```

Таким чином, ці методи разом створюють комплексний механізм для ефективного моніторингу, аналізу та управління структурою мережі DHT, сприяючи її адаптивності та розширенню можливостей.

5.5 Розробка функції запуску сніферу. Реалізація з’єднання до серверу

В процесі розробки ПЗ, фігурує поняття як «метод ініціалізації». Це метод, який часто називають «стартовою функцією» буде здійснювати первинне налаштування та підготовку середовища для подальшої роботи.

З метою реалізації такої функції необхідно забезпечити кілька важливих аспектів. Необхідним є визначити та ініціалізувати відповідні мережеві порти та параметри навантаження, які визначатимуть межі та умови роботи сніфера. На останньому етапі, слід інтегрувати підключення до аналітичної платформи, яка буде використовуватися для збору, обробки та надсилання отриманої інформації. Це включає у себе налаштування API-з'єднань за допомогою механізму WebSocket.

Для цього необхідно сформувати функціонал підключення до WebSocket загального серверу. Цей сокет працює за допомогою програмної платформи socket.io. Для реалізації з'єднання з WebSocket, слід ініціалізувати клієнтський та серверний компоненти socket.io. Серверний компонент, який управляється аналітичним сервером, відповідає за обробку вхідних підключень та передачу даних, в той час як клієнтський компонент, яким є наш сніфер, забезпечує інтеграцію з кінцевими користувацькими додатками.

Серверний компонент потребує авторизації за допомогою веб-заголовків, а саме передача відповідного словника «ключ-значення». Цей метод полягає у передачі відповідного словника, де ключем є рядок «Api-DHTSpider-Key». Значенням для цього ключа є унікальний ідентифікатор. В результаті, можемо сформувати графічну схему бажаного нами ПЗ для запуска сніфера. Цю схему можемо зобразити на рисунку 5.2:

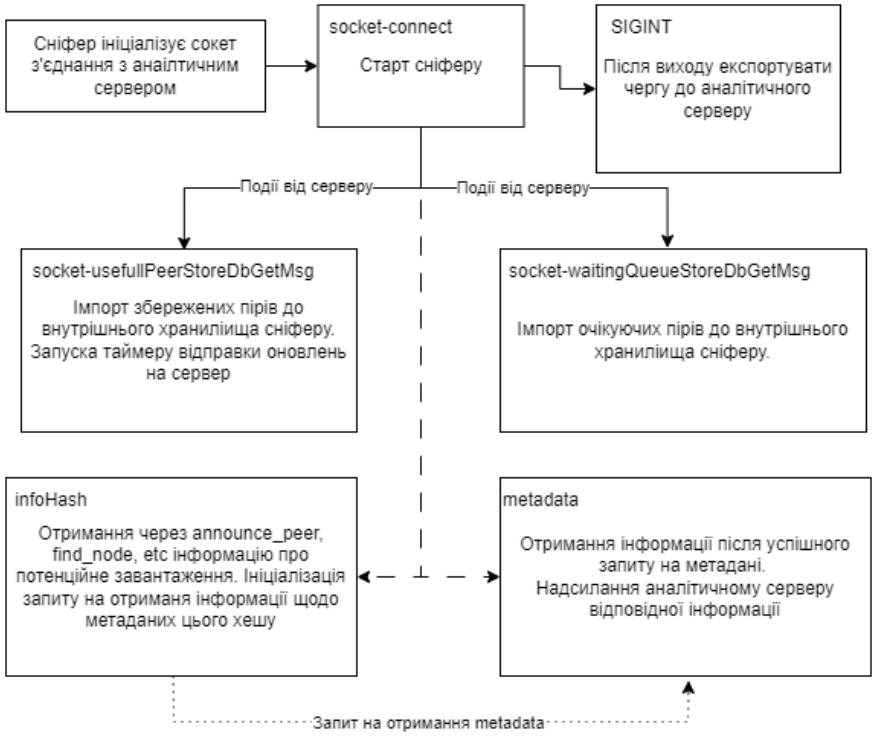


Рисунок 5.2 – Схематичне зображення архітектури роботи методу ініцілізації сніфера

З метою ініціалізації сокет з’єднання необхідним є створення функції-одиночки(singleton function). Для цього в прикладі вище використовується популярна бібліотека socket.io-client, яка надає потужні та гнучкі засоби для роботи з WebSockets. Функція getSocket() виконує роль фабрики для створення об’єкта сокета. Патерн «Singletone», використаний тут, гарантує, що об'єкт сокету буде створений тільки один раз, незалежно від кількості його викликів у програмі. Код для цього наведемо нижче:

```
import { io, Socket } from "socket.io-client";
let socket: Socket;
export function getIncidentSocket () {
    if (!socket) {
        socket = io("http://localhost:1337", {
            extraHeaders: {
                "Api-DHTSpider-Key": ключ,
            },
        });
    }
    return socket;
}
```

Отриману функцію ми імпортуємо у майбутню створену нами головну функцію. Ця функція, названа первинною, буде використана для запуску у вузловій мережі з метою збору даних про піри та метадані торрентів. Первинна функція main() ініціює сокетне з'єднання за допомогою означеною нами передньо функцію socket-з’єднань. Після цього створюється екземпляр DHTSniffer з налаштуваннями, які визначають параметри збору даних, такі як порт, максимальна кількість одночасних запитів, інтервал оновлення тощо.

```
  const socket = getIncidentSocket();
  const sniffer = new DHTSniffer({
        port: 6881, //порт завантажень
        maximumParallelFetchingTorrent: 40,
        maximumWaitingQueueSize: -1,
        concurrency: 32, //кількість потоків
        refreshTime: 30000, //час на оновлення бакетів
        downloadMaxTime: 20000, //таймаут на завантаження
        fetchedTupleSize: 100000,
        expandNodes: true, //чи откриті до перетирання пірів
        ignoreFetched: true, //чи додавати у список вже додані
        fetchedInfoHashSize: 100000,
        findNodeCacheSize: 100000,
        aggressiveLevel: 1, //чи ініціалізувати «хард» запити
  });
```

Слухач події socket.on("connect", () => {}) реагує на подію підключення до серверного сокета, ініціюючи роботу сніфера (sniffer.start();). В цьому ж блоку коду визначено обробку події SIGINT, яка викликається при виході з програми, зберігаючи поточний стан черги очікування.

```
socket.on("connect", () => {
        sniffer.start();
        process.on("SIGINT", () => {
```

```
            const arr = sniffer.exportWaitingQueue();
            const json = JSON.stringify(arr);
            if (arr.length > 0) {
socket.emit(SpiderEvents.waitingQueueStoreDbSetMsg, json);
            }
            process.exit();
        });
    });
```

У тілі нашої первинної функції створимо окремі допоміжні функції: importPeersFromStorage, updatePeerStorage, cleanUpPeerStorage.

Функція importPeersFromStorage відповідає за імпорт даних про піри з видаленого серверу. Це дозволяє підтримувати актуальну базу даних пірів у DHT.

```
    function importPeersFromStorage(storage: any) {
        Object.values(storage).forEach((peer: Peer) =>
            sniffer.importPeer(peer)
        );
    }
```

У функції updatePeerStorage оновлюється сховище пірів, додаючи нові дані про піри, які були корисними. Ця функція також оновлює час останнього спостереження пірів, що дозволяє відстежувати активність у мережі.

```
 function updatePeerStorage(sniffer: any, storage: any): any {
   const usefulPeers = sniffer.exportUsefulPeers();
   const newStorage = structuredClone(storage);
   const now = Date.now();
   usefulPeers.forEach((peer) => {
      const peerKey = `${peer.host}:${peer.port}`;
      if (!newStorage[peerKey]) {
      newStorage[peerKey] = {
        host: peer.host, port: peer.port,
        value: 1, lastSeen: now,
        };} else if (newStorage[peerKey].lastSeen !== now) {
                newStorage[peerKey].value += 1;
                newStorage[peerKey].lastSeen = now;
       }});
        return newStorage;
    }
```

Функція cleanUpPeerStorage відповідає за видалення застарілих або малоактивних пірів зі сховища, що допомагає підтримувати його актуальність та ефективність.

```
function cleanUpPeerStorage(storage: any) {
```

```
            const now = Date.now();
            Object.keys(storage).forEach((key) => {
                if (storage![key]?.value == 1 && now -
storage![key]?.lastSeen > 60 * 86400 * 1000) {
                    Reflect.deleteProperty(storage, key);
                }}); }
```

Слухачі подій socket.on реагують на специфічні повідомлення, пов'язані з управлінням даними пірів та черги очікування, що включає імпорт, оновлення та видалення даних.

Подія «spider:useful-peer-store:db-get» відповідає за прослуховування надісланих сервером подій до усіх сніферів. Після отримання цієї події (разом з відповідним навантаженням) сніфер імпортує отримані піри у внутрішнє хранилище та ініціалізує відправку кожної хвилини на сервер актуального знімку наявних пірів. Код цієї події навадений нижче

```
        socket.on(SpiderEvents.usefulPeersStoreDbGetMsg,
(usefulPeersStorage) => {
            importPeersFromStorage(usefulPeersStorage);
            setInterval(() => {
                const newPeerStorage = updatePeerStorage(
                    sniffer,
                    usefulPeersStorage
                );
                cleanUpPeerStorage(newPeerStorage);
                socket.emit(SpiderEvents.usefulPeersStoreDbSetMsg,
newPeerStorage);
            }, 60 * 1000);
        });
```

Також, необхідним є подія «spider:waiting-queue-store:db-get» відповідає за очікування запиту щодо готовності до відвантаження черги метадати. Отримання цієї події стає сигналом до імпорту черги.

```
socket.on(SpiderEvents.waitingQueueStoreDbGetMsg,
(waitingQueueStorage: Record<string, any>[]) => {
  const usefulPeerDict = waitingQueueStorage;
  sniffer.importWaitingQueue(usefulPeerDict); });
```

У подіях sniffer.on("start", ...) та sniffer.on("infoHash", ...) реалізовано логування та обробку основних дій сніфера, таких як старт роботи та отримання інфохешу від піра.

```
sniffer.on("start", (infos) => {console.log(infos); });
```

Подія sniffer.on("metadata", ...) відповідає за обробку отриманих метаданих, включаючи їх парсинг та відправлення через сокет. Це ключова частина системи для збору та аналізу метаданих торрентів. У нашому випадку, це надсилання на аналітичний сервер інформації, що конкретна IP-адреса завантажує відповідну торрент роздачу.

```
sniffer.on("metadata", (infoHash, metadata, peer) => {
   socket.emit(SpiderEvents.presetMetadata, {
      metadata: parseMetaData(metadata),
      ip: peer?.host,
      infoHash: infoHash?.toString("hex"),});
});
```

Таким чином, в результаті, ми отримали фінальну функцію, мета якої збір інформації про піри та метадані торентів з метою аналізу мережевої активності.

5.6 Розробка допоміжного функціоналу сніфера

Окрім зазначених вище методів у цьому класі повинно бути створено декілька допоміжних методів.

З метою оптимізації техніки пошуку необхідних пірів існує необхідність їх поступового вивантаження у БД списку активних нод. Для цього необхідно створити 4 методи - importUsefulPeers, importPeer, exportUsefulPeers, exportWaitingQueue, importWaitingQueue:

– метод importUsefulPeers призначений для імпорту потенційно корисних пірів з попередньо визначеного списку. З метою досягання рівномірності під час обробки, метод починає з перемішування списку пірів. Після цього, метод встановлює порогове значення (threshold), яке залежить від кількості очікуваних запитів та загальної кількості вузлів у мережі. Цей поріг визначає, як часто піри будуть імпортовані в систему. Кожен пір зі списку

обробляється і, якщо випадково генероване число перевищує порогове, пір імпортується у систему:

– метод importPeer використовується для додавання нового піра до мережі. Метод спочатку генерує унікальний ключ для кожного піра і перевіряє, чи вже існує такий пір у мережі. Якщо пір є новим, він додається до мережі DHT. Це дозволяє системі розширювати своє знання про мережу, інтегруючи нові вузли;

– метод exportUsefulPeers працює для видачі списку корисних пірів;

– методи exportWaitingQueue та importWaitingQueue використовуються для управління очікуваними запитами в системі. exportWaitingQueue отримує список очікуваних запитів з обробника метаданих, який може використовуватися для аналізу або збереження стану системи. З іншого боку, importWaitingQueue використовується для імпорту списку очікуваних запитів, що дозволяє системі відновити свій попередній стан або інтегрувати дані з зовнішніх джерел.

Повний код наведений у лістингу А.1.

5.7 Висновок щодо розробки підробленого BitTorrent-клієнту

Таким чином, після розробки підробленого BitTorrent-клієнту нами було отримана програмна платформа на мові JavaScript, яка дозволяє у автономному режимі опрацьовувати інформацію стосовно завантажень у BitTorrent мережі. Ця програма працює як звичайний BitTorrent-клієнт та відпрвіляє отриману інформацію на аналітичний сервер.

Для створення цієї платформи було повністю реалізовано механізм під’єднання до мережі BitTorrent, таблиця DHT, функціонал старту та запуску відповідної програми.

6 РОЗРОБКА АНАЛІТИЧНОГО СЕРВЕРА ДЕАНОНІМІЗАЦІЇ У P2P МЕРЕЖАХ

6.1 Вибір фреймворка для розробки

Маючи на меті розробити окремий аналітичний сервіс використовуючи програмну платформу NodeJS, нам необхідно вибрати відповідний фреймворк для цього. Розглянемо найбільш популярних фреймворків Node.js.

Платформа Express.js - це мінімалістичний та гнучкий фреймворк, який надає набір основних інструментів для розробки веб-додатків. Express підтримує численні плагіни та мідлвари, що робить його дуже гнучким та адаптивним до різних потреб розробників. Переважно Express.js використовуються у якості кістяка до інших фреймворків.

Фреймворк Koa.js - ще одне популярне, створений командою, яка розробила Express.js. Koa використовує генератори ES6 для спрощення асинхронного коду та забезпечує більш потужний та гнучкий механізм обробки запитів.

Платформа Meteor.js – це рішення яке об'єднує фронтенд, бекенд та базу даних у єдиному JavaScript-додатку. Meteor забезпечує швидку розробку та легкість інтеграції з різними базами даних.

Особливе місце у цьому переліку займає NestJS. Nest - це фреймворк для створення ефективних та надійних серверних додатків. Побудований на основі TypeScript, NestJS надає розробникам інтегрований набір інструментів для створення масштабованих і легко підтримуваних додатків [22].

Однією з ключових переваг NestJS є його модульна архітектура, що дозволяє легко розширювати та масштабувати додатки. Під час розробки аналітичної платформи це має особливе значення, оскільки обробка BitTorrent-трафіку зі сніферу вимагає великої кількості функціональних модулів: бази даних, система черг, кешування. Важливою перевагою NestJS є те, що він дозволяє легко інтегрувати ці модулі в єдину систему [22].

NestJS побудований з урахуванням того, що більша частина розробників буде використовувати TypeScript. У результаті, NestJS має переваги статичної типізації, що є важливим для створення надійних та безпечних серверних додатків з забезпеченням правильного використання даних, які обробляються та передаються у системі.

Під час розробки аналітичної платформи важливою є можливість зберігання та ефективної обробки великих обсягів даних. NestJS пропонує гнучку підтримку різноманітних систем управління базами даних, включаючи SQL та NoSQL, що дозволяє розробникам вибирати найбільш підходящу технологію для конкретних потреб проекту. Для безпечної та швидкої роботи з базами даних, NestJS підтримує велику кількість ORM-систем: TypeORM, Mongoose, Sequelize, тощо, що забезпечують не лише зручність у взаємодії з базами даних, але й дозволяють застосовувати принципи об'єктно-орієнтованого програмування для роботи з даними [22]. Це значно спрощує розробку, особливо при роботі з складними структурами даних, які часто зустрічаються в аналітичних платформах.

Таким чином, маючи на увазі усю раніше викладену інформацію, можна стверджувати, що використання NestJS є ідеальним варіантом у нашому випадку. Його модульна архітектура, підтримка сучасних патернів проєктування, інтеграція з різними базами даних та можливості для створення комплексних API роблять NestJS ідеальним вибором для розробки сучасних аналітичних платформ, здатних ефективно обробляти великі обсяги даних та забезпечувати гнучке управління мережевими ресурсами.

6.2 Огляд необхідних модулів системи

6.2.1 Загальний огляд

Таким чином, маючи фреймворк-основу для розробки, нарисуємо схематичне зображення можливих модулів у нашій системі. На рисунку 6.1 зображена така схема:

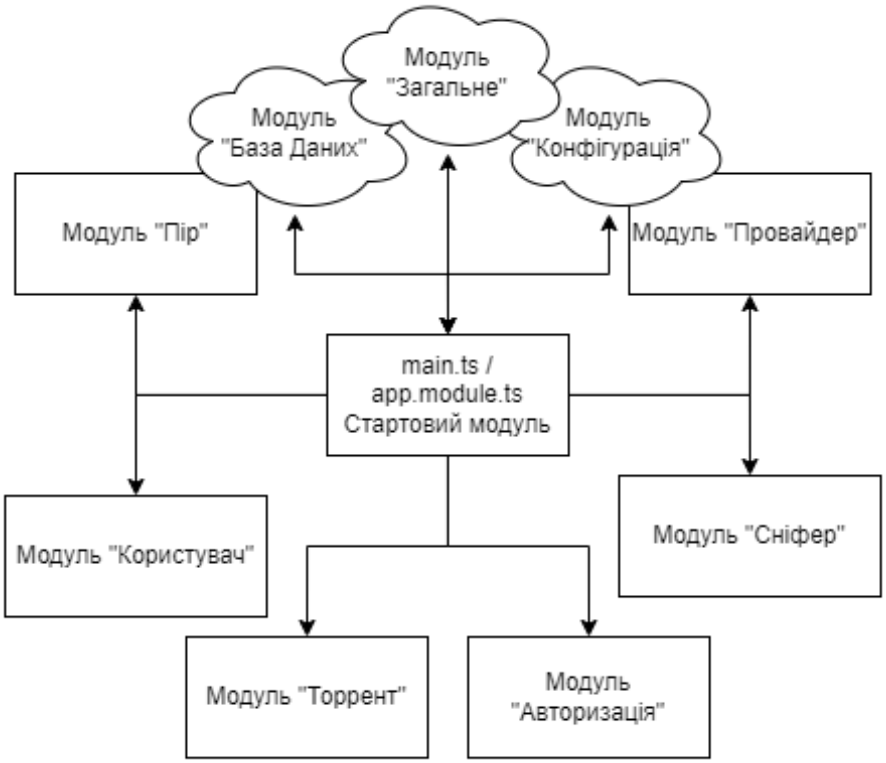


Рисунок 6.1 – Схематичне зображення
модульної архітектури серверу аналітики

Згідно документації фреймворку NestJS, ми розуміємо що основою серверу є так званий «головний» контейнер/стартовий модуль. Він інкапсулює у себе усі інші модулі, що відповідають за різні аспекти системи. Цей модуль виступає як центральний, з якого розгалужуються зв'язки до інших, більш предметних, компонентів, таких як: «Користувач», «Пір», «Торрент», «Авторизація», «Сніфер», «Провайдер», «Конфігурація» та «База Даних». Ці

модулі забезпечують логіку і функціонал конкретних бізнес-сутностей, і кожен з них може бути розроблений незалежно та інтегрований з головним модулем через декларовані інтерфейси.

Першими до розробки є модулі «Користувач» та «Авторизація». Ці модулі відповідають за обробку даних користувачів системи, збереження профілів, авторизації та управління доступами. Наступним до розробки є модуль сутності «Пір», який займається збором та аналізом інформації щодо пірів у мережі BitTorrent отриманих від сніферу. Модуль «Торрент» відповідає за збереження інформації щодо завантажуємих торентів користувачами. «Сніфер» — це модуль, який виконує процеси верифікації запитів від сніфіеру до сервера аналітики та збору інформації щодо їх статистики. Модуль «Провайдер» включає логіку отримання та збереження інформації щодо реєстрації українського інтернет провайдеру та користувача який ним користується.

Окреме місце займають загальні модулі, якими можуть користуватись інші модулі як своїми залежностями. Прикладом такого модуля є «Загальне» який відповідальний за конфігурацію загальних сервісів та утиліт, які можуть використовуватися різними частинами системи, наприклад, кешування, налаштування крон-системи, налаштуваня системи черг. «Конфігурація» використовується для налаштування поведінки сервера у різному оточенні. Нарешті, модуль «База Даних» виконує всю роботу з підключення до бази даних.

Важливим аспектом при розробці модульної системи є її внутрішня координація та здатність до комунікації між модулями. В NestJS це досягається за допомогою введення сервісів, які можуть використовуватися модулями для виконання специфічних завдань. Кожен сервіс може бути впроваджений у будь-який модуль, що забезпечує низький рівень зв'язності та високий рівень замінності компонентів.

6.2.2 Модуль «Користувачів». Модуль «Авторизація»

Основний модуль для розробки нашої системи є модуль користувачів. Розробником передбачено те, що основним користувачем системи є «представник правоохоронних органів». Таким чином, модулі «Користувач» та «Авторизація» розроблени з розрахунком на це. Для початку розробки створена відповідна таблиця у базі даних, що відображає сутність «Користувач». Наведемо її копію у таблиці 6.1.

Таблиця 6.1 – Значення таблиці «Користувач»

Поля	Тип	Опис
id	uuid	Унікальний ідентифікатор користувача
name	string	Ім'я користувача
image	string	URL зображення аватакри користувача
emailVerified	timetz	Дата підтвердження електронної пошти
password	string	Хешований пароль
currentHashedRefreshToken	string	Поточний хешований токен оновлення
role	Relation<Role>	Роль користувача

В архітектурі модуля автентифікації, реалізовано механізм ролевої моделі, який передбачає диференційований доступ для адміністраторів та звичайних користувачів. Концепція ролі користувача закріплена у вигляді окремої таблиці у базі даних, де кожному типу користувача присвоюється певний енумерований тип (наприклад, admin, user). За умовчанням, ініційований перший обліковий запис в системі отримує привілеї адміністратора. Натомість, усі наступні облікові записи, зареєстровані в системі, присвоюються до ролі «user», а наділення роллю адміністратора можливе лише через ручну дію існуючого адміністратора.

Процес автентифікації здійснюється за допомогою пароля та JSON Web Token (JWT), який міститься у HTTP-куках. В рамках проекту запроваджено багаторівневу систему аутентифікації, що включає токен авторизації та токен відновлення. Токен авторизації має час життя 12 годин і є основним засобом для підтвердження особи користувача. Токен відновлення, який має термін дії 24 години, використовується для забезпечення доступу до системи без необхідності повторного вводу пароля. Важливим аспектом безпеки є той факт, що токен відновлення може бути згенерованим виключно серверною частиною системи, а будь-які маніпуляції з цим токеном можливі лише через серверні процедури. У базі даних зберігається лише хешована версія токена відновлення, що забезпечує додатковий шар безпеки.

Токен відновлення є унікальним для кожного користувача і асоційований з усіма токенами авторизації, що видаються користувачу. У ситуації, коли токен авторизації втрачено, але наявний токен відновлення, останній використовується для генерації нового токена авторизації. Якщо ж токен відновлення втрачено, для подальшої роботи в системі необхідно пройти процедуру повторної автентифікації.

Для більшої наочності створених нами кінцевих точок, необхідно створити відповідну таблицю з переліком запитів та їх типу. Відповідний перелік наведений у таблиці 6.2.

Таблиця 6.2 – Таблиця запитів модуля «Користувач» та «Авторизація»

Посилання	Тип	Опис
/auth	GET	Отримати інформацію про поточного користувача
/auth/register	POST	Ім'я користувача
/auth/log-in	POST	URL зображення аватакри користувача
/auth/log-out	POST	Дата підтвердження електронної пошти
/users	GET	Хешований пароль

6.2.3 Модуль «Пір»

6.2.3.1 Модуль «Пір» у основному застосунку

Модуль пір створений з метою агрегації інформації щодо відповідних пірів (IP-адрес користувачів мережі BitTorrent). Робота цього модулю є опрацювання цих IP-адрес. Для початку процесу розробки створена необхідна таблиця у базі даних, що відображає сутність «Пір». Копія цієї моделі наведена у таблиці 6.3.

Таблиця 6.3 – Значення таблиці «Пір»

Поля	Тип	Опис
id	uuid	Унікальний ідентифікатор піра
ip	string	IP-адреса користувача
founded	number	Кількість раз цей пір був зафіксований
country	string	Назва країни походження IP-адрси
city	string	Назва міста походження IP-адрси
spiders	Relation<Spider[]>	Сніфери, що зафіксували цю адресу
torrents	Relation<Torrent[]>	Торренти, що завантажені цим піром
provider	Relation<ISProvider>	Інтернет провайдер цієї IP-адреси

Перш за все, головна роль цього модулю це створити відповідну систему з’єднання з окремим видаленим сервером отримання інформації щодо цієї IP адреси. Сервіс IsproviderService у поєднанні з модулем обробки черг QueueEventsHost з фреймворку NestJS створює розширену інфраструктуру для автоматизації цих процесів. Ключовим аспектом є інтеграція сервісу з розробленим нами віддаленим сервером, який здійснює збір інформації через WHOIS-запити для визначення країни походження IP-адреси та, у випадку асоціації з Україною, подальшого пошуку юридичної адреси провайдера через реєстри ЄДРПОУ.

З урахуванням високої вартості ресурсів при виконанні WHOIS-запитів, було прийнято рішення делегувати цей функціонал до окремого застосунку, який взаємодіє з аналітичним сервером через механізм черг. Для цього застосовано технологію BullMQ та базу даних Redis, які сприяють ефективній роботі з асинхронними задачами та їхньому масштабуванню [23].

Створений нами клас PeerParseProviderInfoQueueEvents виконує слухання подій черги ParseProviderInfo, реагуючи на завершення задачі «отримання інформації з ЄДРПОУ», з чого відповідно виконується створення запису в базі даних за допомогою методу create сервісу IsproviderService. Також, після отримання інформації виконуються необхідні перетворення назв провайдерів до єдиного формату, що спрощує подальші операції порівняння та пошуку.

Так само, створений клас PeerWhoiseInfoQueueEvents слухає події черги ParseWhoiseInfo та обробляє завершення задачі отримання WHOIS інформації. У випадку визначення країни як Україна, починається процес пошуку провайдера. Використання FlowProducer з BullMQ дозволяє організувати управління потоками задач та їхнє виконання, забезпечуючи реалізацію розподіленої обробки та збереження даних у базі. Програмна розробка окремого процесору пошуку наведена у розділі 6.2.3.2.

Окрім цього, функціонал зазначеного модуля спрямований на реалізацію процесів, необхідних для адекватного зберігання сутності піра у мережевому просторі, виконання агрегування даних, а також здійснення пошуку за унікальним ідентифікатором.

У таблиці 6.4 наведено перелік запитів з модуля «Пір».

Таблиця 6.4 – Таблиця запитів модуля «Пір»

Посилання	Тип	Опис
/peer	POST	Створити запис о пірі у мережі
/peer	GET	Отримати інформації о усіх пірах
/peer/{id}	GET	Отримати інформацію щодо піру за ідентифікатором

Продовження таблиці 6.4

/peer/{id}	PATCH	Оновити інформацію про пір за ідентифікатором
/peer/{id}	DELETE	Видалити пір за ідентифікатором
/peer/ip/{ip}	GET	Знайти інформацію про пір за IP-адресою

6.2.3.2 Сервер отримання зовнішньої інформації

На початковому етапі розробки серверного програмного забезпечення, яке відповідає за збір та аналіз даних з BitTorrent-мереж, нагальним є визначення архітектури взаємодії з аналітичним сервером. Це передбачає створення системи черг з використанням бази даних Redis, яка дозволить організувати асинхронний обмін інформацією між серверами. Для реалізації такої взаємодії застосовується інструмент BullMQ, що надає інтерфейс для управління робочими процесами та розподілу завдань у чергах.

Ключовим моментом щодо коректної роботи цього серверу є розуміння необхідності парсингу даних, який виконується за допомогою зовнішніх серверів. У зв'язку з цим, постає завдання забезпечення надійного механізму проксування запитів, для чого планується використання «Tor Proxy». Це дасть можливість анонімного та безпечного звернення до зовнішніх джерел інформації без прямого виявлення IP-адреси нашого серверу.

Процес парсингу веб-сторінок опосередковується програмним забезпеченням «puppeteer», що імітує дії користувача в інтернет-браузері Google Chrome [24]. Це програмне забезпечення дозволяє здійснювати автоматизований вхід на веб-сайти та збір необхідних даних, імітуючи поведінку користувача. Перевага «puppeteer» полягає у його здатності до легкої інтеграції з Tor Proxy, що робить можливим анонімний парсинг даних.

Для того щоб почати відповідну розробку, необхідно означити що цей сервер має з’єднуватись з відповідним аналітичним сервером через систему

«черг» та БД Redis. Для цього необхідно реалізовувати відповідне підключення завдяки BullMQ [24].

Також, необхідно зрозуміти, що парсинг виконується за допомогою зовнішніх серверів, тому необхідним є забезпечити механізм проксування запитів з нашого серверу до серверів-джерел інформації. З цією метою будемо використовувати так звані Tor Proxy

Власне, сам механізм парсингу працює завдяки системі автоматичного тестування ПЗ «puppeteer». Це ПЗ дозволяє автоматизувати процес заходу на веб-сторінку у інтернеті симулюючи дії «від особи користувача» через спеціально налаштований браузер Google Chrome. Також, не менш важливим є його простота інтеграції з Tor Proxy.

Маючи на меті створити 2 модулі, та відповідний клас який має вибирати яким з них оброблювати ці данні, було вибрано побудувати архітектуру цього додатку шляхом реалізації паттерна «Стратегія». Для цього, було сформована відповідний базовий клас, який має запускати відповідні модулі у залежності від джерела запиту. Програмний код сформованого модуля наведений у лістингу Г.1.

Таким чином, наведений у лістинг Г.1 нами нами клас «Scraper» виконує ініціацію об'єкту браузера Puppeteer, який є інструментом для автоматизованого веб-скрапінгу, імітуючи дії користувача в браузері. У коді, метод «init» класу «Scraper» відповідає за ініціацію браузера з конфігурацією для запуску в контейнері Docker та/або використання Tor Proxy. Метод «randomElementOfArray» використовується для вибору випадкового порту з наданих, який буде використовуватися для налаштування проксі-сервера. Такий підхід дозволяє розподілити навантаження та підвищити анонімність обробки запитів. Метод «setStrategy» призначений для встановлення конкретної стратегії скрапінгу, заснованої на базовому класі «BaseStrategy». Це надає можливість динамічної зміни поведінки скрапера в залежності від потреби обробки різних джерел даних. Метод run слугує для запуску алгоритму.

У контексті прикладу реалізації парсингу, взято за основу стратегію парсингу веб-ресурсу opendatabot.ua, яка імплементована у класі EdrpoParserStrategy, що є похідним від базового класу BaseStrategy [25]. Цей клас реалізує алгоритм визначення та збору даних про юридичні особи на основі коду ЄДРПОУ. Інтерфейс EdrpoReturnType визначає структуру даних, яка очікується як результат парсингу. Вона включає такі поля, як час витягу з ЄДР, повна назва компанії, назва англійською мовою, адреса, телефон, дата заснування, ім'я директора, стан компанії, код ЄДРПОУ, статутний капітал, основний та інші види діяльності. Основний метод doAlgorithm реалізує логіку пошуку інформації на сайті opendatabot.ua з використанням Google-запиту. Пошук проводиться за допомогою спеціалізованого запиту, що включає назву компанії та коди діяльності за ЄДРПОУ. Після виявлення відповідного результату, відбувається перехід на сторінку з результатами та очікування завантаження необхідних елементів.

З метою парсингу WHOIS-інформації, використовуються запит за допомогою команди «whois» у терміналі. Засобами NodeJS робиться відповідний запит до IP-адреси та повертається результат.

У результаті, нами сформовано відповідне ПЗ, яке може бути легко зібрано через систему контейнеризації Docker та працювати як окремий мікросервіс нашого загального додатку

6.2.4 Модуль «Провайдер»

Модуль «Провайдер» був розроблений з метою збору та інтеграції даних, які стосуються віднайдених українських провайдерів інтернет-послуг. Наявність цього провайдеру у системі є гарантією того, що клієнт цього провайдера був учасником BitTorrent мережі. На початковому етапі розробки

було створено таблицю в базі даних, яка представляє сутність «Провайдер», структура якої представлена в таблиці 6.5.

Таблиця 6.5 – Значення таблиці «Провайдер»

Поля	Тип	Опис
id	uuid	Унікальний ідентифікатор провайдера
fullName	string	Повна юридична назва провайдера
worldWideName	string	Ім’я провайдера з WHOIS
email	string	Електрона пошта для контактів провайдера
website	string	Адреса веб-сайта провайдера
phone	string	Телефонний номер провайдера
edrpoCode	string	Код ЄДРПОУ
peers	Relation<Peer[]>	Піри що користуються цим провайдером

Опрацюванням даних таблиці «Провайдер» ведеться спеціалізованим сервісом. Для інтеракції з цим сервісом використовуються визначені кінцеві точки (ендпоінти). У таблиці 6.6 представлений деталізований опис кінцевих точок, що належать до модуля «Провайдер».

Таблиця 6.6 – Таблиця запитів модуля «Провайдер»

Посилання	Тип	Опис
/isprovider	GET	Отримати інформацію о усіх провайдерах
/isprovider/{id}	GET	Отримати інформацію щодо провайдера за ідентифікатором

6.2.5 Модуль «Торрент»

Модуль «Торрент» інтегрує в себе можливості збереження даних, пов'язаних із торрентами, включаючи метадані та механізм їх класифікації за допомогою систем штучного інтелекту. Адекватна робота даного модуля

передбачає реалізацію таблиці в базі даних, яка репрезентує сутність «Торрент». Структуру цієї моделі можна знайти у Таблиці 6.7.

Таблиця 6.7 – Значення таблиці «Торрент»

Поля	Тип	Опис
id	uuid	Унікальний ідентифікатор торренту
name	string	Назва торрент роздачі
size	number	Розмір роздачі у байтах
torrentType	(multiple|single)	Тип роздачі за кількості файлів
infoHash	string	Адрес BitTorrent
filePaths	string[]	Назви файлів у роздачі
metadata	jsonb	Репрезентація торрента у форматі буферу
classification	(Application | Abuse | Games | Movies | Music | XXX |Other)	Класифікація торренту за типом
spiders	Relation<Spider[]>	Сніфери, що зафіксували цей торрент
peers	Relation<Peer[]>	Піри, що завантажували цей торрент

Головна роль цього модулю, окрім агрегації даних та їх збереження, є відправка на класифікацію раз на визначений час.

Для класифікації нами створений сервіс «TorrentClassifierService». Він визначає декілька основних категорій, таких як «Додатки», «Ігри», «Фільми», «Музика» та «Дорослий контент», кожна з яких відповідає певному індексу в масиві передбачення. На класифікацію надсилаються лише ті торенти, що відповідає необхідним критеріям (тільки Англійська мова), та відправляє запити на класифікацію тих торентів, які ще не були класифіковані до цього часу.

Для цього модулю використовуються відповідні кінцеві точки, зміст яких ви можете побачити у таблиці 6.8.

Таблиця 6.8 – Таблиця запитів модуля «Торрент»

Посилання	Тип	Опис
/torrent	POST	Створення записи о торренті
/torrent	GET	Отримання інформації о усіх торентах
/torrent/{id}	GET	Отримати інформацію щодо торенту за ідентифікатором
/torrent/{id}	PATCH	Оновити торрент за ідентифікатором
/torrent/{id}	DELETE	Видалити торрент за ідентифікатором

6.2.6 Модуль «Сніфер»

Модуль, що здійснює обробку даних, надісланих від сніферів, виконує важливу функцію реєстрації та асоціації цих даних у системі.

З метою означення інформації, яка зберігається о сніферах, вмістимо у таблиці 6.9 систематизований огляд даних у базі даних.

Таблиця 6.9 – Значення таблиці «Сніфер»

Поля	Тип	Опис
id	uuid	Унікальний ідентифікатор торренту
name	string	Назва торрент роздачі
size	number	Розмір роздачі у байтах
torrentType	(multiple|single)	Тип роздачі за кількості файлів
infoHash	string	Адрес BitTorrent
filePaths	string[]	Назви файлів у роздачі
metadata	jsonb	Репрезентація торрента у форматі буферу
classification	(Application | Abuse | Games | Movies | Music | XXX |Other)	Класифікація торренту за типом
spiders	Relation<Spider[]>	Сніфери, що зафіксували цей торрент
peers	Relation<Peer[]>	Піри, що завантажували цей торрент

В контексті модуля сніферів застосовується спеціалізована авторизаційна система, що оперує API ключами. Ця система авторизації, яка застосовується під час встановлення з'єднань між «сервером аналітики» та «сервером снiфінгу», не має за ціль аутентифікацію сніфера, а скоріше його авторизацію у системі для асоціації надходжувальних даних з існуючим записом щодо сніфера у БД. API ключі, які є унікальними ідентифікаторами сніферів у базі даних, генеруються в момент створення запису про сніфер адміністратором. Ця авторизація ефективно працює для WebSocket з'єднань та для HTTP запитів. Зокрема, за допомогою системі guard у NestJS відбувається перевірка наявності відповідного заголовку (Api-DHTSpider-Key) та ключа. Така авторизація, передусім, є перевіркою на наявність переданого ключа, який асоційований з конкретним сніфером у базі даних. У випадку коректної відповідності ключа, запити сніфера приймаються, в іншому разі — відхиляються

Крім реалізації функціоналу авторизації, зазначений модуль відіграє ключову роль у управлінні окремим сокет сервером для всіх наявних торрентів. Згідно з інформацією, представленою у розділі 5, сніфер, який нам потрібен, повинен підтримувати можливість багатопотоковості та кластеризації на кількох серверах. Саме для цих цілей і використовується згаданий сервер – він функціонує як менеджер з'єднань між усіма сніферами та центральним сервером класифікації.

Відповідно до функціоналу авторизації, модуль також відповідає за обробку стандартних REST запитів, перелік яких представлено у таблиці 6.10. Зазначена архітектура модуля демонструє його важливість у контексті забезпечення інтеграції, безпеки та координації між компонентами системи.

Таблиця 6.10 – Таблиця запитів модуля «Сніфер»

Посилання	Тип	Опис
/spider/me	GET	Отримати інформацію про себе (для сніфера)
/spider	GET	Оримати список усіх сніферів

Продовження таблиці 6.10

/spider	POST	Створити запис о новім сніфері
/spider/{id}	GET	Отримати інформацію щодо конкретного сніфера

6.2.7 Висновок щодо розробки аналітичного серверу деанонімізації

Відповідно до результатів розробки аналітичного сервера, його архітектура передбачає наявність головного модуля, який координує роботу підмодулів, таких як «Користувач», «Пір», «Торрент», «Авторизація», «Сніфер», «Провайдер» та «База Даних». Кожен з цих модулів зосереджений на обробці певного аспекту системи і може бути розроблений незалежно. Особливістю є модульна авторизація сніферів через API ключі, що забезпечує безпеку і автентичність мережевих запитів.

Система черг, реалізована за допомогою BullMQ та Redis, забезпечує ефективну асинхронну обробку даних, а використання Tor Proxy і Puppeteer сприяє безпечному та анонімному збору інформації з зовнішніх джерел. Розроблений аналітичний сервер демонструє здатність до швидкого масштабування та інтеграції, що є критично важливим для систем, які аналізують трафік у BitTorrent мережах.

7 РОЗРОБКА ШТУЧНОГО ІНТЕЛЕКТУ КЛАСИФІКАЦІЇ BITTORRENT

7.1 Мета розробки штучного інтелекту

У фінальній стадії розробки програмного забезпечення, спрямованого на аналітику даних з BitTorrent-мережі, постає можливість обробки значних масивів інформації, пов'язаної з користувацькими завантаженнями. Цей контент є багатожанровим та може містити широкий спектр категорій. Аналітичний огляд статистики завантажень на платформі «The Pirate Bay» свідчить, що домінуючими категоріями завантажень є «фільми» (33%), «програмне забезпечення» (14%), «телепрограми» (12%), «музика» (12%), «матеріали для дорослих» (11%), «ігри» (5%) [26]. Здійснення класифікації завантажень за категоріями є значущим завданням, але варто відмітити, що у специфікації BitTorrent Enhancement Proposal (BEP) відсутній механізм категоризації торрентів з використанням тегів. Тому категоризація проводиться з використанням альтернативних методів, а саме:

– використання штучного інтелекту (ШІ) у якості прогресивного інструмент класифікації: ШІ може класифікувати торренти на основі їх назви, використовуючи навчені моделі для розпізнавання ключових слів та фраз, що відображають зміст матеріалу. Більш складні системи ШІ можуть аналізувати різноманітні атрибути, такі як назва роздачі, розмір файлу, специфічні назви файлів у роздачі та інші метадані. Це дозволяє виконувати багатовимірну класифікацію, значно підвищуючи точність і вірогідність результатів;

– застосування регулярних виразів для пошуку збігів у назвах файлів, що є менш гнучким, але простішим методом класифікації. Цей підхід може бути ефективним для швидкого виявлення очевидних категорій;

– аналіз змісту завантажень після їх отримання, що передбачає безпосереднє дослідження вмісту файлів. Цей метод є найбільш ретельним,

проте вимагає значних обчислювальних ресурсів і може бути пов'язаний з юридичними та етичними питаннями.

Так як поміж цими опціями найбільш коректним та точним є вибір класифікації завдяки штучного інтелекту, то доцільним є зупинитись на саме цій опції.

7.2 Вибір типу штучного інтелекту

Сучасні методи класифікації штучним інтелектом, зокрема байєсівські алгоритми, які описані в дослідженні у статті «Bayesian Torrent Classification by File Name and Size Only» підкреслюють значення статистичних моделей у розробці інтелектуальних систем [27].

Такі моделі використовують принципи баєсової статистики для визначення ймовірності приналежності конкретного файлу до певної категорії на основі його атрибутів [27]. Баєсові класифікатори є надзвичайно потужними, коли питання стоїть про роботу з великими даними та нечіткими умовами, оскільки вони здатні робити інформовані припущення навіть на основі обмеженої кількості інформації, такої як назва та розмір файлу [27].

Алгоритми, подібні цьому, можуть виявляти складні взаємозв'язки поміж різними атрибутами файлів, навіть якщо ці атрибути не здаються відразу очевидними для класифікації [27]. Наприклад, певні фрази або терміни у назвах файлів можуть вказувати на конкретний жанр фільмів або музики. Розмір файлу також може бути індикатором певного типу контенту, особливо коли йдеться про відео або програмне забезпечення, де великі файли часто асоціюються з високою якістю зображення або більшою функціональністю [27].

Втім, для ефективної реалізації байєсових класифікаторів необхідна велика тренувальна вибірка для «навчання» цієї моделі, що вимагає чіткої

підготовки даних та їх попередньої обробки. Після навчання модель здатна виконувати класифікацію в режимі реального часу, швидко оцінюючи вхідні дані та призначаючи категорію. Такий підхід до класифікації, який використовує штучний інтелект на основі різноманітних атрибутів, значною мірою покращує точність та надійність системи обробки даних, забезпечуючи глибоке розуміння контенту, який розповсюджується через BitTorrent-мережі.

Виходячи з того, що у поточному випадку у автора відсутня велика БД з BitTorrent-мережі, а ця робота має на меті демонструвати можливості подібного аналізу, автор для створення моделі класифікації користується вже існуючими торрент-трекерами, збираючи з них дані. Таким чином, для вивчення власної моделі автор може використовувати лише назви торрент файлів та вже класифіковані торрент-трекерами завантаження.

Виходячи з цього, у нашому випадку, буде використана модель штучного інтелекту, що навчається лише на імені торрент роздачі.

7.3 Вибір платформи для розробки ШІ

Для розробки моделей штучного інтелекту з класифікації BitTorrent-трафіку здійснено вибір на користь мови програмування Python та бібліотеки Tensorflow. Підбір цих інструментів ґрунтується на їхніх технічних характеристиках та перевагах, що відповідають потребам поставленої задачі.

Тензорні обчислення є фундаментальною частиною багатьох алгоритмів машинного навчання, і саме тут бібліотека TensorFlow виявляється особливо корисною. Ця відкрита бібліотека дозволяє розробникам створювати складні моделі глибокого навчання, які мають здатність самонавчання на основі великих масивів даних [28]. Подібна можливість є критично важливою у задачах класифікації, де необхідно виявляти нюансові та складні закономірності в масивах інформації.

Основною перевагою Tensorflow є його гнучкість та масштабованість, що дозволяє розгортати навчені моделі на різноманітних платформах, від мобільних пристроїв до розподілених обчислювальних середовищ. Особливістю є вбудована підтримка платформи Docker для швидкого розгортання серверу штучного інтелекту у відокремленому контейнері. Не менш важливим є те, що Tensorflow підтримує автоматичну диференціацію, що є незамінною функцією при оптимізації алгоритмів навчання [28].

У сукупності, Python та Tensorflow надають потужні інструменти для створення глибоких нейронних мереж, які здатні ефективно класифікувати BitTorrent-трафік за назвою торрент-файів.

7.4 Розробка штучного інтелекту

З метою створення окремої моделі необхідно ініціалізувати Jupyter Notebook. Для цього необхідно розглянути кожен сформований нами блок окремо.

Встановлення необхідних бібліотек:

```
!pip install -U "tensorflow-text==2.13.*"
!pip install "tf-models-official==2.13.*"
```

У цьому рядку використовуємо команду pip для встановлення пакету tensorflow-text версії 2.13. Це робиться для того, щоб використовувати спеціалізовані функції TensorFlow для обробки тексту. Та встановлюємо пакет tf-models-official, який містить офіційні моделі та утиліти TensorFlow, що використовуються у проекті.

Імпорт бібліотек та налаштування середовища:

```
import os
import shutil
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text
```

```
from official.nlp import optimization
import pandas as pd
import matplotlib.pyplot as plt
tf.get_logger().setLevel('ERROR')
gpus = tf.config.list_physical_devices('GPU')
if len(gpus) > 0:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
       pass
```

У цьому блоку коду ми імпортуємо необхідні бібліотеки, такі як TensorFlow, TensorFlow Hub (для завантаження попередньо навчених моделей), TensorFlow Text (для обробки тексту) та інші. Також ми налаштовуємо використання GPU, що є важливим для прискорення навчання моделі.

Завантаження та аналіз даних:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv('./data.csv')
print(df.groupby('type').describe())
label_encoder = LabelEncoder()
counts_of_data = df['type'].value_counts()
```

У цьому блоку коду ми використовуємо бібліотеку Pandas для завантаження та аналізу даних о імені торрент-файлу з документу data.csv. Ми групуєте дані за типом та демонструємо описову статистику для кожного типу. Також ви визначаєте частоту кожного типу даних.

Розділення даних за типами:

```
df_movie = df[df['type']=='Movies']
df_xxx = df[df['type']=='XXX']
df_applications = df[df['type']=='Applications']
df_games = df[df['type']=='Games']
df_music = df[df['type']=='Music']
```

У цьому блоку ми створюємо окремі підмножини даних для кожного типу. Це може бути корисним для подальшої обробки даних, таких як балансування класів або специфічна обробка для кожного типу.

Балансування даних:

```
df_movie_downsampled = df_movie.sample(downsampling_count)
df_xxx_downsampled = df_xxx.sample(downsampling_count)
```

```
    ...
    df_balanced = pd.concat([df_movie_downsampled,
df_xxx_downsampled, ...])
    print(df_balanced.shape)
    print(df_balanced.head(4))
```

У цьому блоку ми виконуємо балансування класів шляхом відбору випадкової підмножини даних з кожного типу, щоб кількість записів у кожному типі була однаковою. Після цього, ми об'єднуємо ці підмножини в один збалансований набір даних.

Підготовка даних для навчання моделі:

```
    from sklearn.preprocessing import LabelEncoder
    from sklearn.model_selection import train_test_split
    from tensorflow.keras.utils import to_categorical
    label_encoder = LabelEncoder()
    integer_encoded =
label_encoder.fit_transform(df_balanced['type'])
    categorical_labels = to_categorical(integer_encoded)
    X_train, X_test, y_train, y_test =
train_test_split(df_balanced['title'], categorical_labels,
test_size=0.2, random_state=42)
```

У цьому блоку коду ми кодуємо категоріальні мітки (типи даних) та ділимо дані на тренувальний і тестовий набори. Функція train_test_split використовується для розділення даних на тренувальні та тестові набори з випадковим вибором зразків.

Векторизація тексту:

```
    max_features = 10000
    sequence_length = 250
    AUTOTUNE = tf.data.AUTOTUNE
    batch_size = 32
    seed = 42

    def custom_standardization(input_text):
        lowercase_text = tf.strings.lower(input_text)
        stripped_text = tf.strings.regex_replace(lowercase_text,
'[^a-zA-Z0-9\s]', ' ')
        return stripped_text
    vectorize_layer = tf.keras.layers.TextVectorization(
        max_tokens=max_features,
        output_mode='int',
        output_sequence_length=sequence_length,
        standardize=custom_standardization)
    vectorize_layer.adapt(X_train.values)
```

Тут ми створюємо шар векторизації тексту, який перетворює текстові дані в числові токени. Цей шар адаптується під тренувальні дані та використовується для подальшої обробки тексту перед подачею в нейронну мережу.

Створення моделі нейронної мережі:

```
from tensorflow.keras.models import Sequential
num_classes = len(label_encoder.classes_)
print(num_classes)
def build_classifier_model():
    model = Sequential([
        vectorize_layer,
        tf.keras.layers.Embedding(max_features, 64,
mask_zero=True),
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='softmax')
    ])
    return model
```

В цьому коді ми створюємо модель нейронної мережі, яка включає шари для обробки тексту, шар вбудовування (Embedding), двосторонній LSTM (для обробки послідовностей тексту) та вихідний шар з активацією softmax для класифікації тексту на кілька класів.

Ініціалізація моделі:

```
classifier_model = build_classifier_model()
```

У цьому рядку ми ініціалізуємо модель, яку ми раніше визначили у функції build_classifier_model. Це створює екземпляр моделі з усіма визначеними шарами та налаштуваннями.

Візуалізація структури моделі:

```
tf.keras.utils.plot_model(classifier_model)
```

Цей рядок коду використовується для генерації візуального представлення архітектури нашої моделі. Він показує різні шари та їхні зв'язки всередині моделі.

Вивід резюме моделі:

```
print(classifier_model.summary())
```

Команда summary() надає детальний огляд моделі, включаючи кількість параметрів, форму вхідних і вихідних даних для кожного шару. Це корисно для розуміння розміру та складності моделі. Ці кроки демонструють, як створюється та аналізується нейронна мережа для класифікації тексту. Отриманий огляд моделі зображений на рисунок 7.1

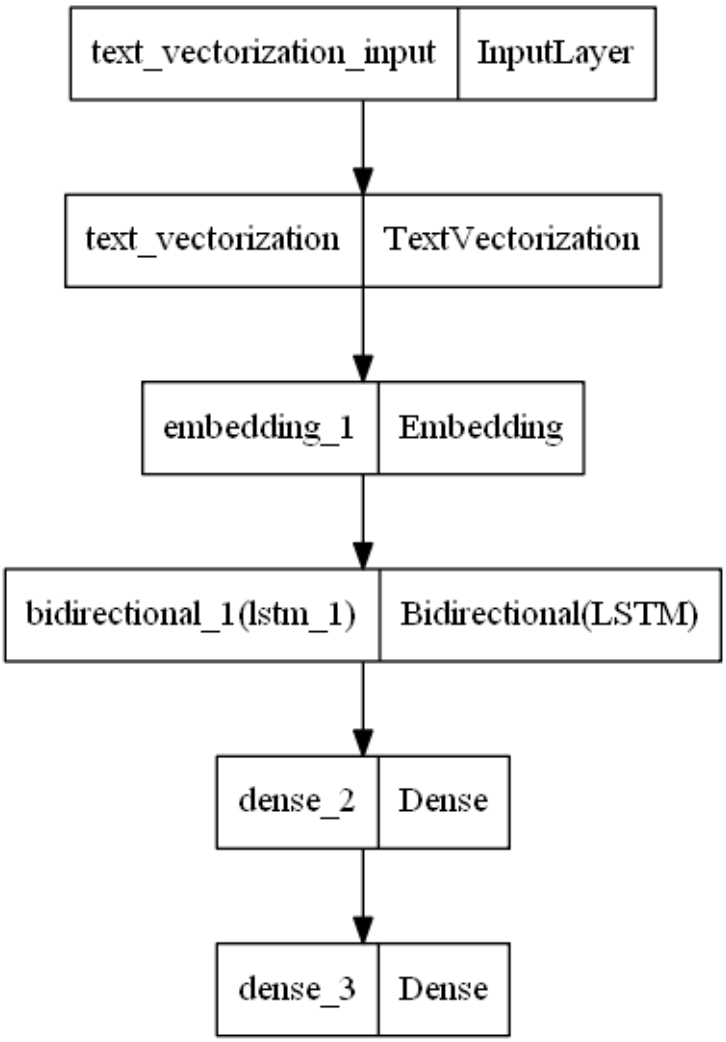


Рисунок 7.1 – Схематичне зображення нейронної мережі

Визначення функції втрат та метрик для моделі:

```
loss = 'categorical_crossentropy'
metrics = [
    "accuracy",
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
```

У цьому фрагменті ми визначаємо функцію втрат categorical_crossentropy, яка є стандартною для задач багатокласової класифікації. Також ми визначаємо метрики для оцінки моделі, включаючи точність (accuracy), точність (precision) та повноту (recall). Ці метрики допомагають оцінити якість моделі під час тренування та тестування.

Створення TensorFlow Dataset:

```
    train_dataset =
tf.data.Dataset.from_tensor_slices((X_train,y_train))
    test_dataset =
tf.data.Dataset.from_tensor_slices((X_test,y_test))
```

Тут ми створюємо об'єкти tf.data.Dataset для тренувальних та тестових даних. tf.data.Dataset є ефективним способом обробки великих наборів даних і дозволяє оптимізувати використання пам'яті та швидкість обробки даних.

Перемішування та пакування даних:

```
    train_dataset = train_dataset
    .shuffle(len(X_train))
    .batch(batch_size)
    test_dataset = test_dataset.batch(batch_size)
```

В цьому кроці ми налаштовуємо тренувальний набір даних для перемішування (щоб запобігти перенавчанню) та пакування в пакети заданого розміру (batch_size). Пакетна обробка даних допомагає прискорити процес тренування, оскільки модель обробляє кілька зразків одночасно.

Налаштування оптимізатора для тренування моделі:

```
    epochs = 3
    steps_per_epoch =
tf.data.experimental.cardinality(train_dataset).numpy()
    num_train_steps = steps_per_epoch * epochs
    num_warmup_steps = int(0.1*num_train_steps)

    init_lr = 3e-5
    optimizer = optimization.create_optimizer(init_lr=init_lr,

num_train_steps=num_train_steps, num_warmup_steps=num_warmup_steps,
optimizer_type='adamw')
```

У цьому блоку ми встановлюємо параметри для тренування моделі, включаючи кількість епох (epochs), кількість кроків тренування та

ініціалізаційну швидкість навчання (init_lr). Також нами використовується оптимізатор AdamW, який є модифікацією стандартного алгоритму Adam і часто використовується в задачах обробки природної мови.

Компіляція та тренування моделі:

```
    classifier_model.compile(optimizer='adam', loss=loss,
metrics=metrics)
    history = classifier_model.fit(train_dataset,
validation_data=test_dataset, epochs=epochs)
```

Тут ми компілюємо модель, вказуючи оптимізатор, функцію втрат і метрики, які були визначені раніше. Після цього відбувається тренування моделі на тренувальному наборі даних, з використанням тестового набору як валідаційних даних. Функція fit автоматично обробляє тренувальні дані, налаштовує ваги моделі для мінімізації втрат та відстежує метрики протягом заданої кількості епох.

Оцінка моделі:

```
evalResults = classifier_model.evaluate(test_dataset)
print(f'Results: {evalResults}')
```

Після тренування моделі ми виконуємо її оцінку на тестовому наборі даних за допомогою методу evaluate. Це надає метрики, такі як точність, втрати, точність і повноту на даних, які модель раніше не бачила, що допомагає зрозуміти, наскільки добре модель узагалі здатна узагальнювати. Процес аналізу навчання демонструє такі параметри:

- Похибка - 0.00864;
- Точність - 0.99797.

Статистику нашого навчання ми можемо вивести на окремому графіку. Цей графік наведемо на рисунку 7.2.

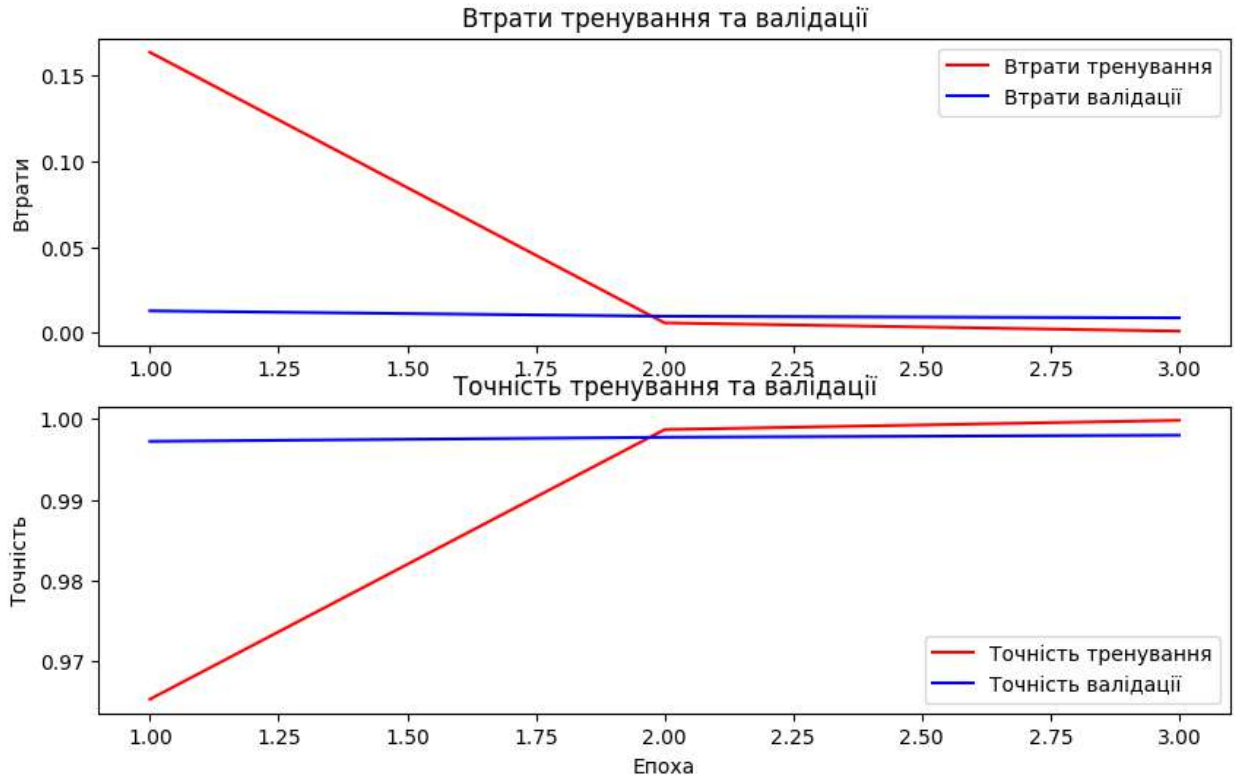
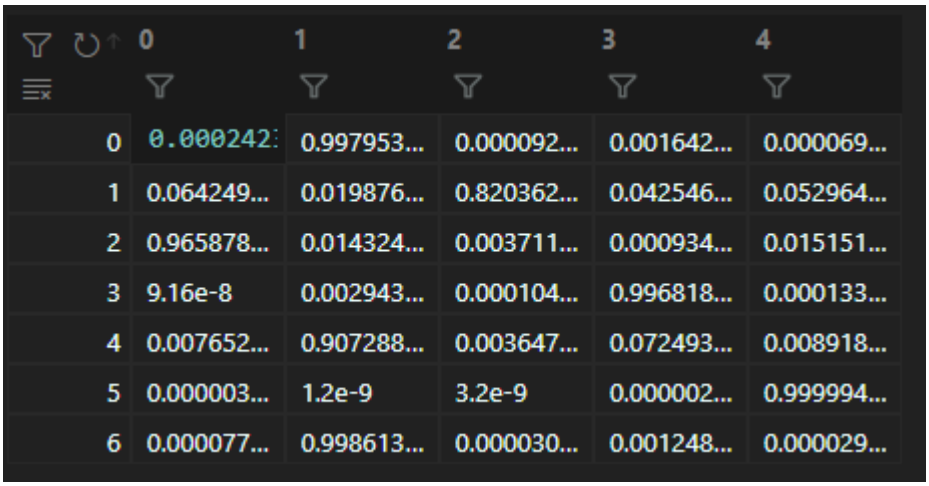


Рисунок 7.2 – Графік навчання штучного інтелекту

Отриману нами модель, протестуємо на працеспроможність на реальних даних. У блоці нижче задамо перелік з відповідними назвами

```
examples = [
    'fist dsg Repack By', //Game
    'Oppenheimer mkv HDRip', //Movie
    'photoshop iso', //Application
    'queen - show must go on (1975) mp3', //Music
    'assassins creed black flag', //Game
    'OnlyTarts - Eva Elfie - Eva Goes Cheating (18.08.2023)
rq.mp4', //XXX
    'Fding.Afternoon.RePack.by.Chovka', //Game
    ]
```

Результати отримані під час виконання роботи штучного інтелекту наведені на рисунку 7.3. Отримані результати треба інтерпритувати так, що на вертикальній осі порядковий номер текстового рядка (нумерація від 0), а на горизонтальній осі наведена відповідна категорія (0 – це Application, 1 – Game, 2 – Movie, 3 – Music, 4 – XXX)



	0	1	2	3	4
0	0.000242:	0.997953...	0.000092...	0.001642...	0.000069...
1	0.064249...	0.019876...	0.820362...	0.042546...	0.052964...
2	0.965878...	0.014324...	0.003711...	0.000934...	0.015151...
3	9.16e-8	0.002943...	0.000104...	0.996818...	0.000133...
4	0.007652...	0.907288...	0.003647...	0.072493...	0.008918...
5	0.000003...	1.2e-9	3.2e-9	0.000002...	0.999994...
6	0.000077...	0.998613...	0.000030...	0.001248...	0.000029...

Рисунок 7.3 – Демонстрація результатів опрацювання штучного інтелекту

7.5 Висновок щодо розробки штучного інтелекту

Таким чином, у процесі розробки програмного забезпечення для класифікації даних BitTorrent мережі, було створено систему штучного інтелекту (ШІ), здатну ефективно обробляти великі масиви інформації та класифікувати завантаження за категоріями. Повний код цього штучного інтелекту наведений у лістингу В.1.

Вибір типу ШІ ґрунтувався на використанні статистичних баєсових моделей, які забезпечують високу точність і надійність при класифікації в умовах великих даних та нечітких умов. Для реалізації цих моделей було вибрано мову програмування Python та бібліотеку TensorFlow, які разом створюють потужну платформу для розробки глибоких нейронних мереж, здатних виконувати складні завдання класифікації.

Процес розробки включав підготовку даних, балансування класів, векторизацію тексту, створення та тренування моделі нейронної мережі. Результати тренування підтвердили ефективність обраного підходу, демонструючи високу точність та інші позитивні метрики.

8 РОЗРОБКА ВІЗУАЛЬНОГО ІНТЕРФЕЙСУ АНАЛІТИЧНОЇ СИСТЕМИ

8.1 Мета та засоби розробки візуального інтерфейсу

Для розробки аналітичної системи необхідною, для кінцевого користувача, є можливість користуватись системою за допомогою зручного веб-інтерфейсу.

Для розробки цього веб-додатку було вирішено використовувати комбінацію технологій, що забезпечують ефективність – NextJS, Typescript, RTKQuery. Основою цього набору став фреймворк React, який є одним з провідних інструментів для створення інтерактивних користувацьких інтерфейсів.

З метою забезпечення підсилення функціональності React, було обрано окремий фреймворк на його основі - Next.JS. Ця система розширює можливості React, додаючи підтримку server-side rendering (SSR), static site generation (SSG) та інші оптимізації для підвищення продуктивності на персональний комп’ютерах користувача. Next.js спрощує процес розгортання додатку, забезпечуючи автоматичне розділення коду, оптимізоване завантаження сторінок та інтегроване керування маршрутизацією.

Так як отриманий проект має великою кодову базу, то використання типізації також для клієнтського застосунку є необхідним. Ця типізація була реалізована за допомогою Typescript. Ця мова програмування додає строгу статичну типізацію до JavaScript, що підвищує надійність програмного коду, уможливлює виявлення помилок на етапі компіляції та покращує рефакторинг і підтримку коду.

З метою ефективної роботи з даними, що приходять з аналітичного серверу, в контексті Redux архітектури було обрано використовувати RTK Query. RTK Query надає потужні засоби для створення API запитів, кешування

відповідей та автоматичного управління станом даних, що суттєво зменшує необхідність ручного керування станом додатку.

Таким чином, маючи ці основні елементи клієнтської системи, можемо продемонструвати, як вони працюють у контексті бажаної користувацької поведінки.

8.2 Огляд користувацьких сценаріїв

8.2.1 Авторизація та реєстрація

Для авторизації на клієнтській стороні було створена окрема відповідна сторінка. Інтерфейс містить дві основні кнопки для навігації — "Увійти" та "Зареєструватись", що відповідають за вхід у систему та реєстрацію нового користувача відповідно.

У нижній частині інтерфейсу знаходиться форма для вводу логіна та пароля. При цьому є підказка, яка вказує на приклад даних для входу: електронна адреса та пароль. Цим паролем, що зображений на рисунку, є пароль адміністратора. Демонстрація цієї сторінки наведена на рисунку 8.1:

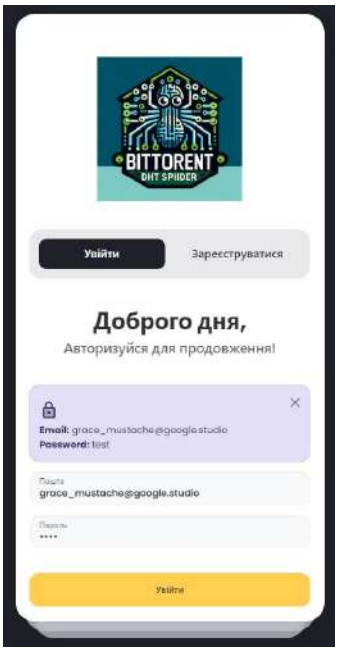


Рисунок 8.1 – Демонстрація сторінки авторизації

Для того щоб користувач мав можливість зареєструватись на платформі, користувачу необхідно зв’язатись з адміністратором платформи. Адміністратор платформи зробить реєстрацію профіля у ручному форматі (методом заносу інформації у базу даних). З метою інформування про цю безпекову особливість платформи, необхідно на клієнтській стороні було повідомити користувача щодо необхідності контактування з адміністратором. Приклад сторінки для такого користувача зображений на рисунку 8.2:

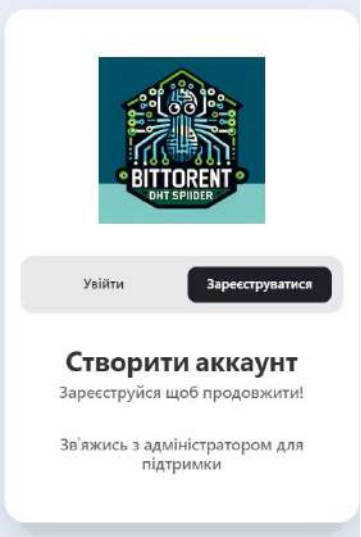


Рисунок 8.2 – Демонстрація сторінки реєстрації

8.2.2 Інформаційна сторінка «Адреса»

З точки зору працівника правоохоронних органів, необхідним є існування окремої сторінки, де він зможе подивитись статистику завантажень за окремою IP адресою. Цей інструментальний ресурс надає інформацію про розташування (Placement Info), зокрема країну та місто, асоційовані з IP-адресою, а також про інтернет-провайдера (ISP Info), включаючи назву компанії, останнє оновлення даних та контактну інформацію. Приклад подібного інформаційного блоку для IP адреси 245.24.139.59 зображений на рисунку 8.3:

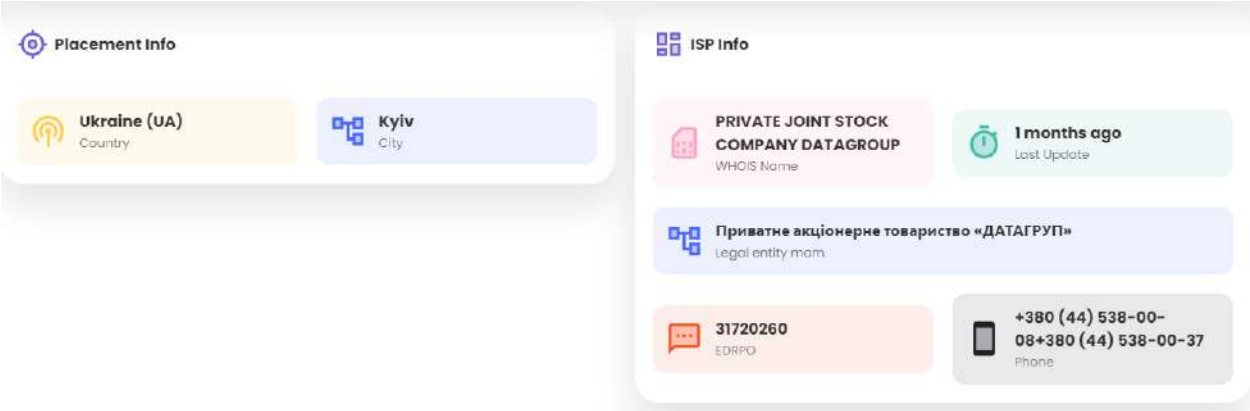


Рисунок 8.3 – Демонстрація інформаційного блоку
про провайдера та локацію адреси

У таблиці зі списком торрентів відображено назви файлів, які були завантажені через BitTorrent IP адресою «245.24.139.59», класифікацію (наприклад, ігри чи програми), розмір файлу, дату створення та оновлення запису. Ця інформація може використовуватися для виявлення потенційно нелегальної діяльності та слідкування за розповсюдженням контенту, що охороняється авторським правом. Приклад такого блоку зображений на рисунку 8.4:

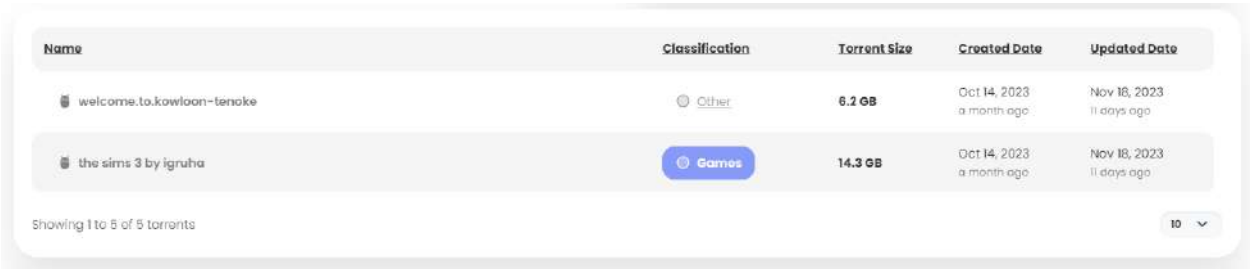


Рисунок 8.4 – Демонстрація інформаційного блоку
про провайдера та локацію адреси

Фінальний відображення цієї інформаційної сторінки наведено на рисунку 8.5:

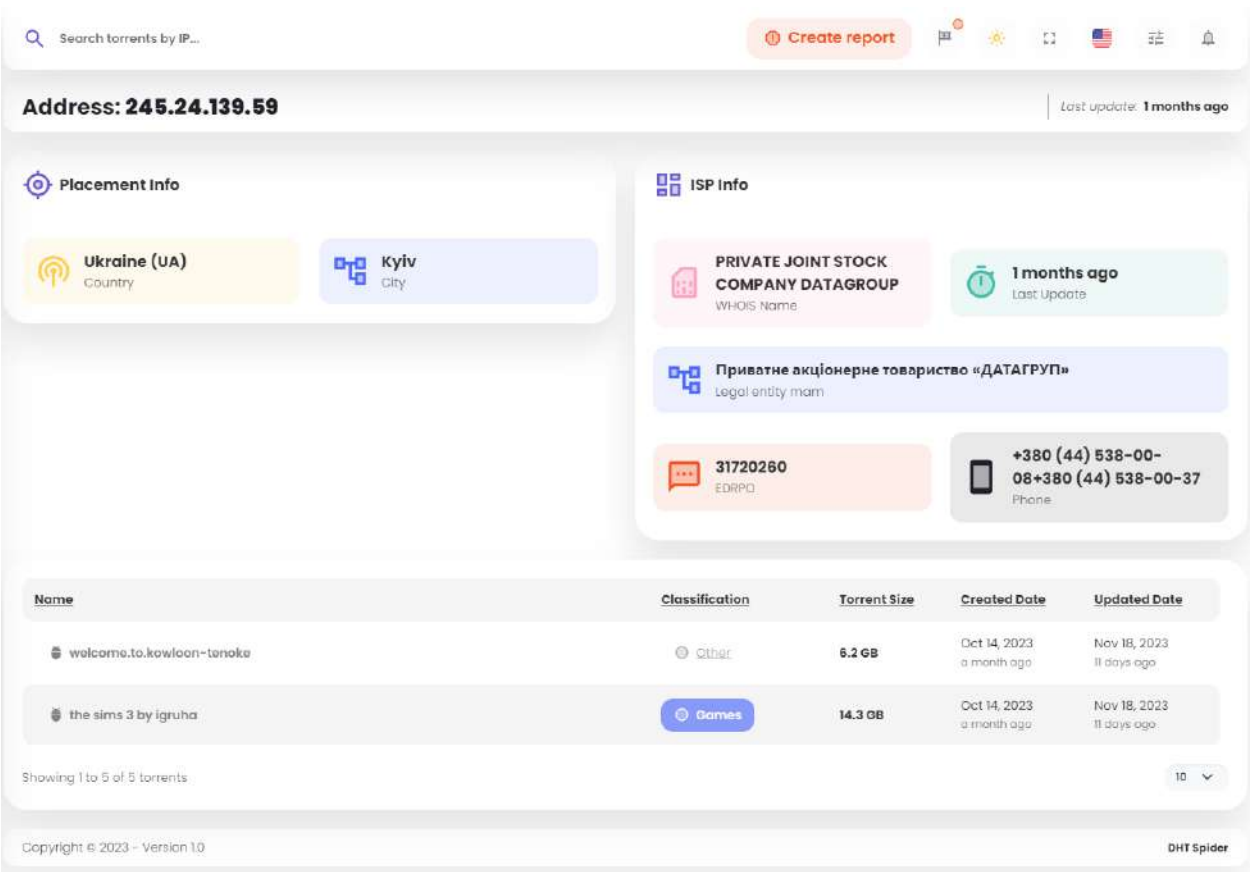


Рисунок 8.5 – Демонстрація інформаційної сторінки «Адреса»

З метою створення функціоналу для генерації звіту за окремою IP адресою кожна інформаційна сторінка має відповідну кнопку «Create report». Відповідна кнопка наведена на рисунку 8.6:

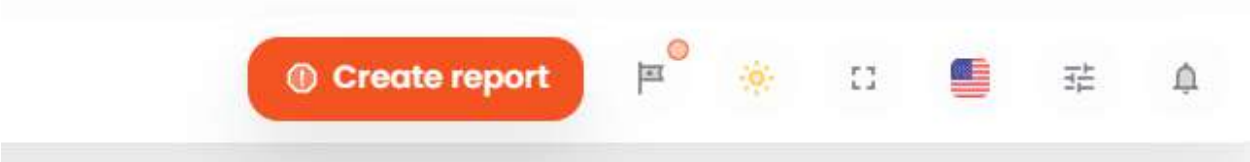


Рисунок 8.6 – Демонстрація кнопки «Create report»

Після активації цієї кнопки користувачу виводиться окреме модальне вікно з заздалегідь підготовленим шаблоном запиту на інформацію, у якому вказано юридичну назву організації-провайдера, суть вимоги, а також юридичну основу для її пред'явлення. Зазначений документ містить офіційне звернення до інтернет-провайдера з проханням надати інформацію, пов'язану з IP-адресою, яка може бути необхідною для ведення досудового розслідування. Такий звіт може бути завантажений у форматі сумісному з комп'ютерами користувача – PDF. Рисунок цієї модалки наведений на рисунку 8.7

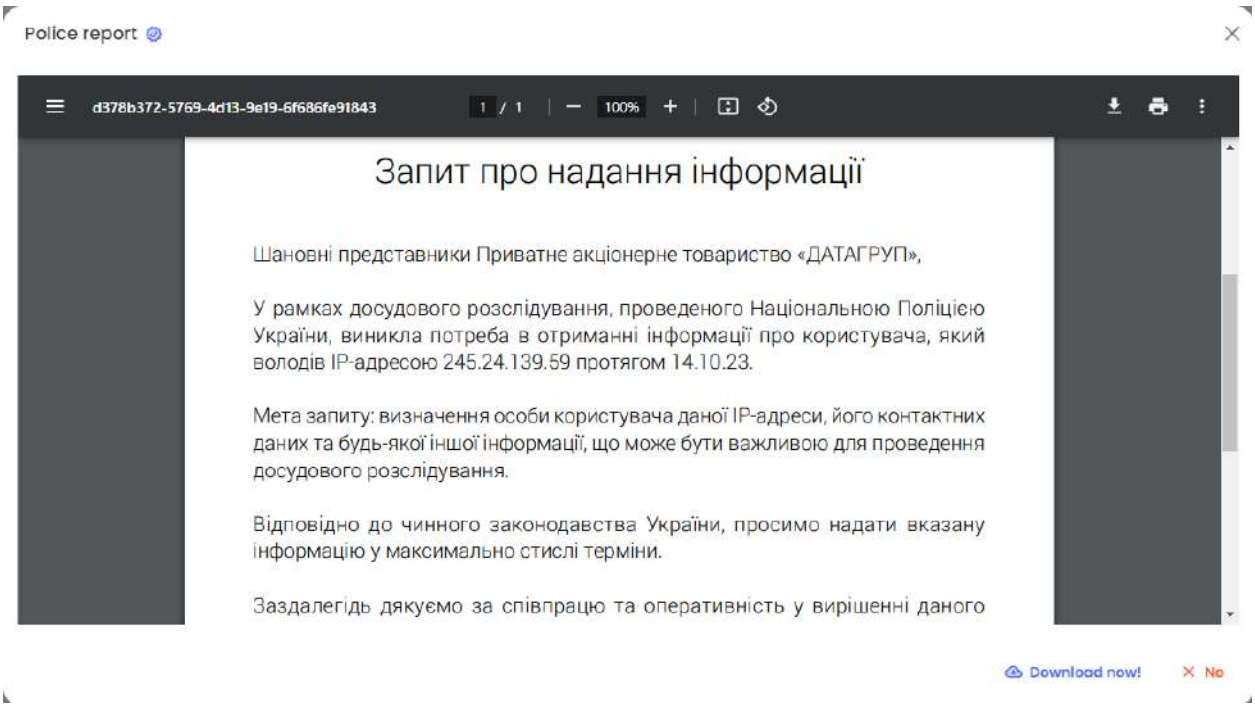


Рисунок 8.7 – Демонстрація модального вікна з відповідним запитом

Аналіз мережевих активностей вимагає точного введення IP-адреси, яка є предметом дослідження. Для цієї мети на інтерфейсі веб-сайту реалізовано

спеціалізоване поле вводу, що містить маску формату IPv4, запобігаючи тим самим введенню користувачем некоректних даних. Така маска забезпечує валідацію введених значень відповідно до стандартів IP-адресації.

Візуалізація зазначеного поля ілюструється на рисунку 8.8, де представлено поле з вже вписаною адресою «141.101.22.239».

Рисунок 8.8 – Демонстрація кнопки «Create report»

Активація пошукового запиту за допомогою клавіші «Enter» ініціює процес відправки запиту на сервер, в результаті чого користувач отримує доступ до детальної сторінки з інформацією про вказану IP-адресу. Візуальне представлення цієї сторінки демонструється на рисунку 8.9, відображаючи результати запиту.

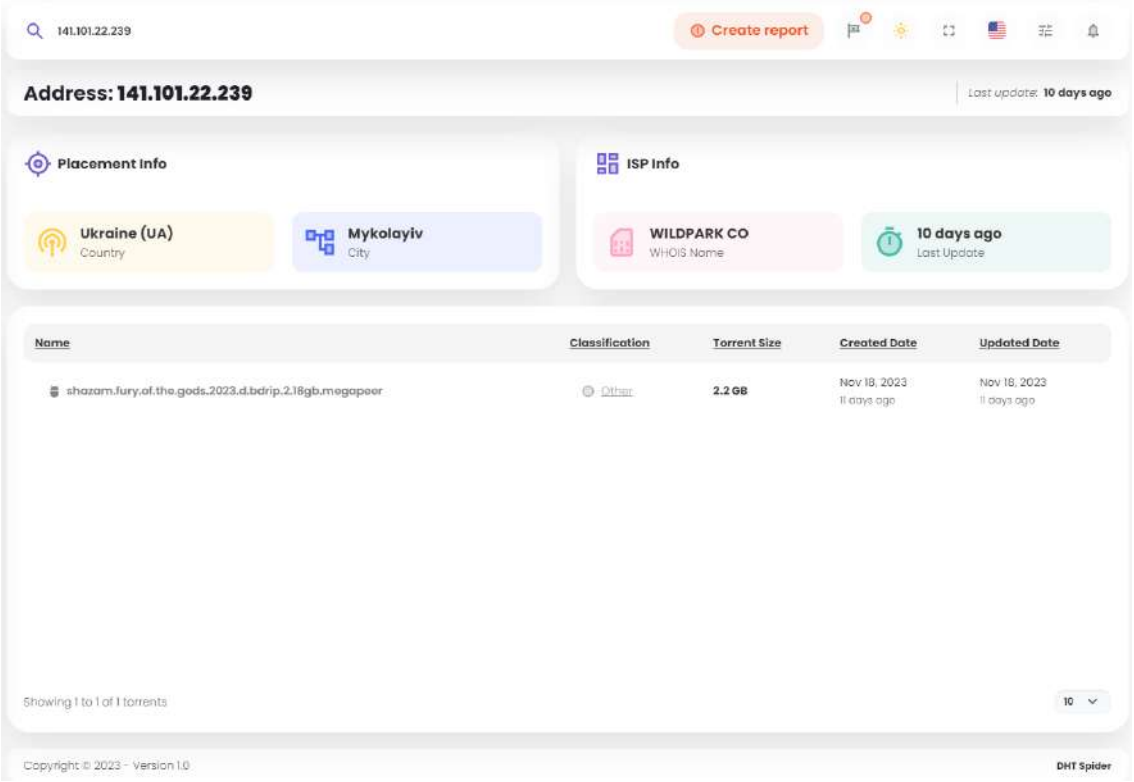


Рисунок 8.9 – Сторінка «Адреса» після відправки бажаної IP-адреси

Таким чином, у нас є повноцінна система для пошуку та аналізу інформації щодо завантажень за відповідною IP-адресою. Ця система дозволяє представнику правоохоронних органів згенерувати відповідний звіт та надіслати його бажаному провайдеру з метою проведення досудового розслідування.

8.2.3 Інформаційна сторінка «Список сніферів»

Для адміністратору аналітичної платформи важливим є аналіз інформації стосовно того, чи працює відповідний сніфер та як. Для цього адміністратор може переглянути відповідну сторінку «Список сніферів».

На цій сторінці передбачена відповідна таблиця, з такими полями як: «name», «online status», «created date», «updated date». Приклад цієї сторінка та таблиці наведений на рисунку 8.10.

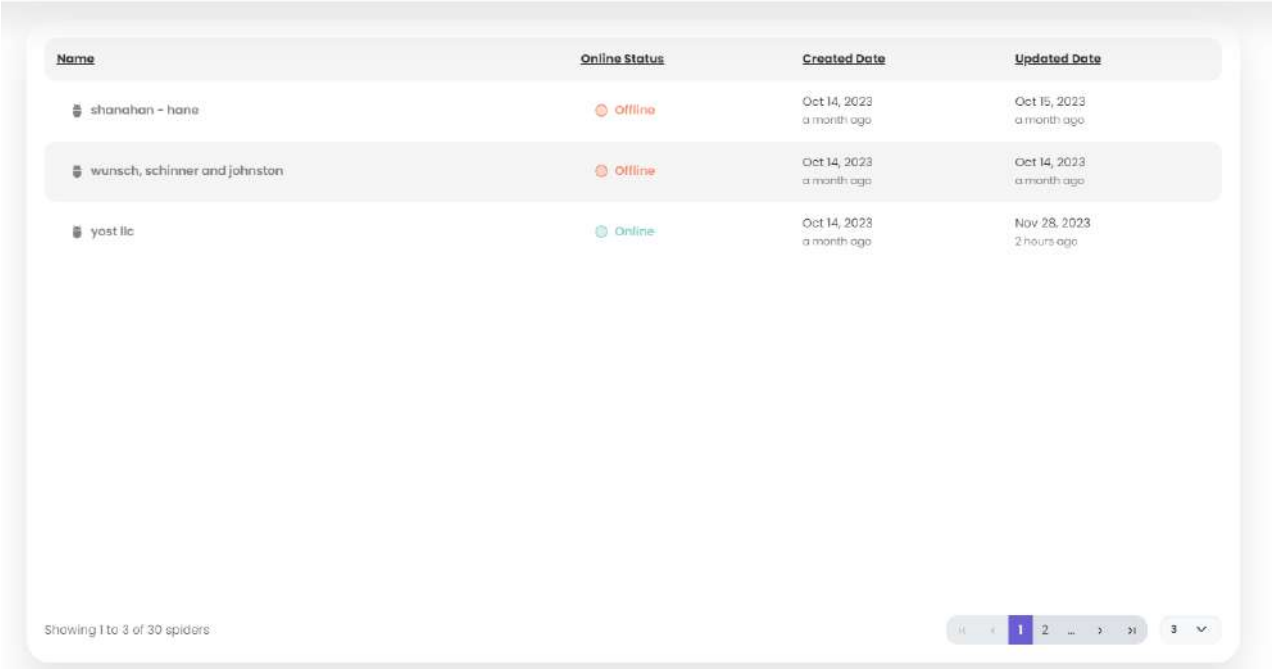


Рисунок 8.10 – Сторінка «Список сніферів»

8.2.4 Інформаційна сторінка «Список торрентів»

Важливим аспектом функціонування системи моніторингу є візуалізація даних про всі торренти, які були зафіксовані мережевими датчиками (сніферами). Адміністратор системи, обладнаний необхідними рівнями доступу, має змогу переходити до інтерфейсу, де презентовано перелік всіх торрентів. Цей перелік слугує вихідним пунктом для подальшого редиректу на сторінку, де аналізуються завантаження по кожному окремому торрент-файлу.

Процес аналізу ініціюється вибором конкретного торренту, внаслідок чого адміністратор перенаправляється на сторінку "Адреса". На цій сторінці адміністратору надається інформація про останню IP-адресу, яка була зареєстрована за завантаженням вказаного торренту. Така інформація є критичною для визначення поведінкових патернів користувачів та може бути використана для виявлення потенційно незаконної активності в мережі. Відповідну сторінку «Список торрентів» можна побачити на рисунку 8.11:

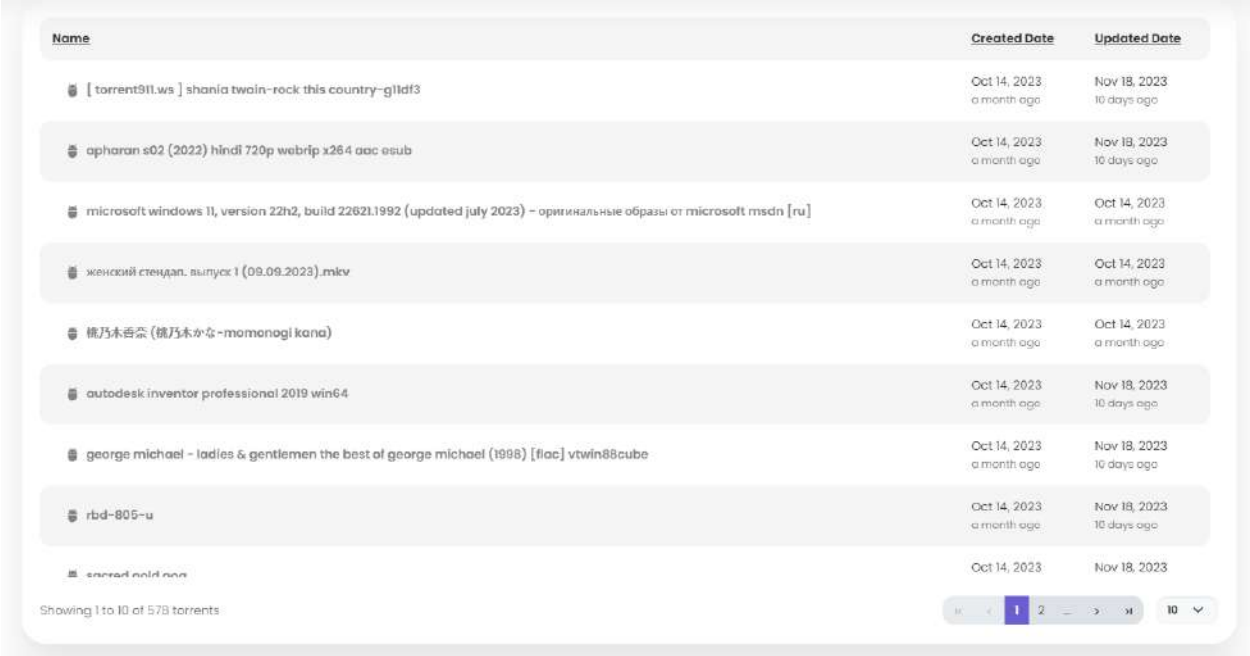


Рисунок 8.11 – Сторінка «Список торрентів»

8.3 Висновок за результатами розробки візуального інтерфейсу

Після розробки спеціалізованої аналітичної системи для правоохоронних органів, було створено веб-додаток із зручним користувацьким інтерфейсом, що дає змогу здійснювати моніторинг та аналіз IP-адрес, асоційованих із зафіксованими завантаженнями торрент-файлів. Система забезпечує класифікацію торрент-завантажень за допомогою штучного інтелекту та можливість генерації юридично оформлених запитів до інтернет-провайдерів прямо з інтерфейсу. Адміністратор системи має доступ до списку всіх торрентів, зафіксованих сніферами, та інформацію про стан цих сніферів (онлайн/офлайн).

Для розробки даного веб-додатку було обрано стек технологій, що включає NextJS, Typescript, RTKQuery на основі фреймворку React. Цей набір технологій забезпечує високу продуктивність, модульність та легкість підтримки коду, а також оптимізоване завантаження сторінок. Приклади програмного коду отриманих нами модулів наведено у лістингу Г.1. та Г.2.

Основні функції системи, включно з процедурами авторизації та реєстрації користувачів, представлені на окремих сторінках, кожна з яких має інтуїтивно зрозумілі команди та візуальні елементи для введення та обробки даних. Система надає зручні інструменти для пошуку та аналізу даних, що дозволяє ефективно здійснювати оперативний пошук та ведення досудових розслідувань.

ВИСНОВКИ

Під час виконання роботи було охоплено ряд ключових аспектів, пов’язаних із проєктуванням та розробкою програмного застосунку для деанонімізації користувачів в однорангових мережах, зокрема використовуючи протокол BitTorrent.

Основний прицнип роботи мережі BitTorrent полягає у необхідності використання DHT (розподілених геш-таблиць). У принципі працювання DHT-таблиць є загроза анонімності користувача.

За результатами дослідження було виявлено, що деанонімізація може здійснюватись через такі атаки як «Man in the Middle» , атака на ідентичність, атака Сивіли та специфічні методи деанонімізації в мережах BitTorrent.

Також у роботі був проведений юридичний аналіз легальності збору інформації для правоохоронних органів з метою пошуку правопорушників.

Отримавши відповідну вичерпну інформацію було визначено можливість створення програмного застосунку для подібної деанонімізації. Для цього був проведений аналіз кінцевих завдань застосунку, вибір відповідної мови програмування та розроблена діаграма працювання цього ПЗ.

Для функціонування цього ПЗ було розроблено повноцінний BitTorrent-клієнт який самостійно відстежує інформацію щодо завантажень у мережі BitTorrent.

З метою проведення аналізу та агрегації усієї наявної інформації було створено відповідний аналітичний сервер за допомогою програмної платформи NodeJS.

Для того щоб класифіковувати інформацію отриману від сніфера за можливим контентом, що завантажують користувачі, було створено відповідний штучний інтелект на базі бібліотеки Tensorflow.

Наприкінці, було розроблено лаконічний візуальний інтерфейс, був проведений аналіз можливих користувацьких сценаріїв його використання.

Таким чином, автором продемонстровані вразливості анонімності користувача у мережі BitTorrent та розроблене ПЗ яке може бути використане правоохоронними органами з метою деанонімізації користувачів.

Подальшим розвитком цієї роботи вбачаю розробку більш якісної системи штучного інтелекту та класифікації тексту за більшими параметрами з метою додавання ще більш точної відповіді.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Збірка «Тиждень науки-2023. Факультет радіоелектроніки та телекомунікацій». // Міністерство освіти і науки України. – 2023.– С. 115–117

2. «Розробка та програмна реалізація методики цифрової розвідки на основі відкритих джерел»// Всеукраїнський конкурс на кращу студентську наукову роботу – Режим доступу до ресурсу: https://lpnu.ua/sites/default/files/2021/pages/12564/rozvidka.pdf - Назва з екрана. (дата звернення 14.11.2023)

3. Ghareeb, M., Rouibia, S., Parrein, B., Raad, M., & Thareau, C. (2013). P2PWeb: A Client/Server and P2P hybrid architecture for content delivery over internet. 2013 Third International Conference on Communications and Information Technology (ICCIT). – Режим доступу до ресурсу: https://doi.org/10.1109/iccitechnology.2013.6579542. - Назва з екрана. (дата звернення 14.11.2023)

4. PURE P2P. – Режим доступу до ресурсу: https://www.ida.liu.se/conferences/p2p/p2p2001/pure.html. - Назва з екрана. (дата звернення 14.11.2023)

5. HYBRID P2P. – Режим доступу до ресурсу: https://www.ida.liu.se/conferences/p2p/p2p2001/hybrid.html. Назва з екрана. (дата звернення 14.11.2023)

6. The Global Internet Phenomena Report 2022 [Електронний ресурс]. – Режим доступу до ресурсу: https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2022/Phenomena%20Reports/GIPR%202022/Sandvine%20GIPR%20January%202022.pdf?hsCtaTracking=18fff708-438e-4e16-809d-34c3c89f4957%7C067d9d28-ef90-4645-9d46-c70d10279247– Назва з екрана. (дата звернення 14.11.2023)

7. Glossary of BitTorrent terms [Електронний ресурс]. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Glossary_of_BitTorrent_terms – Назва з екрана. (дата звернення 14.11.2023)

8. Tit-for-tat – Режим доступу до ресурсу https://en.wikipedia.org/wiki/Tit_for_tat#:~:text=Peer%2Dto%2Dpeer%20file%20sharing,-

See%20also%3A%20BitTorrent&text=BitTorrent%20peers%20use%20tit%2Dfor,to%20allocate%20to%20other%20peers. - Назва з екрана. (дата звернення 14.11.2023)

9. BitTorrent strategies The End Game [Електронний ресурс]. – Режим доступу до ресурсу: https://www.niallohiggins.com/2007/12/26/bittorrent-strategies-the-end-game/– Назва з екрана. (дата звернення 14.11.2023)

10. Superseeding [Електронний ресурс]. – Режим доступу до ресурсу: http://www.bittornado.com/docs/superseed.txt – Назва з екрана. (дата звернення 14.11.2023)

11. Credits in BitTorrent [Електронний ресурс]. – Режим доступу до ресурсу: https://repository.tudelft.nl/islandora/object/uuid%3A809eaec7-883c-47b0-9d57-8e605eaaaed1 – Назва з екрана. (дата звернення 14.11.2023)

12. BitTorrent/DHT [Електронний ресурс]. – Режим доступу до ресурсу: https://ru.wikibooks.org/wiki/BitTorrent/DHT – Назва з екрана. (дата звернення 14.11.2023)

13. Routing performance of structured overlay in Distributed Hash Tables (DHT) for P2P [Електронний ресурс]. – Режим доступу до ресурсу: https://www.researchgate.net/publication/333860645_Routing_performance_of_structured_overlay_in_Distributed_Hash_Tables_DHT_for_P2P – Назва з екрана. (дата звернення 14.11.2023)

14. A Taxonomy of Attack Methods on Peer-to-Peer Network [Електронний ресурс]. – Режим доступу до ресурсу: https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=B29E3C53DDF1D7BCE76692E8C315DEC4?doi=10.1.1.727.6966&rep=rep1&type=pdf – Назва з екрана. (дата звернення 14.11.2023)

15. Real-World Сивіли Attacks in BitTorrent Mainline DHT [Електронний ресурс]. – Режим доступу до ресурсу: https://www.cl.cam.ac.uk/~lw525/publications/security.pdf – Назва з екрана. (дата звернення 14.11.2023)

16. Судове рішення щодо кримінального провадження 12018040150000125 [Електронний ресурс]. – Режим доступу до ресурсу: https://reyestr.court.gov.ua/Review/76446942

17. Судове рішення щодо кримінального провадження 12022244000000060 [Електронний ресурс]. – Режим доступу до ресурсу: https://reyestr.court.gov.ua/Review/102980707

18. Рейтинг мов програмування DOU [Електронний ресурс]. – Режим доступу до ресурсу: https://dou.ua/lenta/articles/language-rating-2023/

19. bep_0005.rst_post. *BitTorrent.org*. [Електронний ресурс]. – Режим доступу до ресурсу: https://www.bittorrent.org/beps/bep_0005.html (дата звернення: 24.11.2023).

20. k-bucket [Електронний ресурс]. – Режим доступу до ресурсу: https://www.npmjs.com/package/k-bucket (дата звернення: 25.11.2023).

21. bittorrent-dht [Електронний ресурс]. – Режим доступу до ресурсу: https://www.npmjs.com/package/bittorrent-dht (дата звернення: 25.11.2023).

22. Documentation | NestJS - A progressive Node.js framework [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.nestjs.com/fundamentals (дата звернення: 25.11.2023).

23. What is BullMQ [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.bullmq.io/ (дата звернення: 25.11.2023).

24. Puppetter [Електронний ресурс]. – Режим доступу до ресурсу: https://pptr.dev/ (дата звернення: 25.11.2023).

25. Опендатабот [Електронний ресурс]. – Режим доступу до ресурсу: https://opendatabot.ua/ (дата звернення: 25.11.2023).

26. «Most popular torrent categories (from the OpenBay database dump) [OC]» [Електронний ресурс]. – Режим доступу до ресурсу: https://www.reddit.com/r/dataisbeautiful/comments/2raw0m/most_popular_torrent_categories_from_the_openbay (дата звернення: 25.11.2023).

27. Fenton N., Dementiev E. Bayesian Torrent Classification by File Name and Size Only. Workshop and Conference Proceedings. 2016. T. 52. URL: https://www.researchgate.net/publication/322635993_Bayesian_Torrent_Classification_by_File_Name_and_Size_Only (дата звернення: 28.11.2023)

28. TensorFlow 2 quickstart for beginners | TensorFlow Core [Електронний ресурс]. – Режим доступу до ресурсу: https://www.tensorflow.org/tutorials/quickstart/beginner?hl=en (дата звернення: 25.11.2023).

ДОДАТОК А

Код реалізації підробленого BitTorrent-клієнту

Лістинг А.1 - index.ts

```
import { DHTSniffer, parseMetaData } from "./sniffer";
import { SpiderEvents } from "./events";
import { runInCluster } from "./runCluster";
import { getIncidentSocket } from "./utils/getSocket";
import { Peer } from "./sniffer/type.t";

async function main() {
    const socket = getIncidentSocket();

    const sniffer = new DHTSniffer({
        port: 6881,
        maximumParallelFetchingTorrent: 40,
        maximumWaitingQueueSize: -1,
        concurrency: 32,
        refreshTime: 30000,
        downloadMaxTime: 20000,
        fetchedTupleSize: 100000,
        expandNodes: true,
        ignoreFetched: true,
        fetchedInfoHashSize: 100000,
        findNodeCacheSize: 100000,
        aggressiveLevel: 1,
    });

    socket.on("connect", () => {
        sniffer.start();
        process.on("SIGINT", () => {
            const arr = sniffer.exportWaitingQueue();
            const json = JSON.stringify(arr);
            if (arr.length > 0) {
                socket.emit(SpiderEvents.waitingQueueStoreDbSetMsg, json);
            }
            process.exit();
        });
    });

    function importPeersFromStorage(storage: any) {
        Object.values(storage).forEach((peer: Peer) =>
            sniffer.importPeer(peer)
        );
    }
    function updatePeerStorage(sniffer: any, storage: any): any {
        const usefulPeers = sniffer.exportUsefulPeers();
        const newStorage = structuredClone(storage);
        const now = Date.now();
        usefulPeers.forEach((peer) => {
            const peerKey = `${peer.host}:${peer.port}`;
            if (!newStorage[peerKey]) {
                newStorage[peerKey] = {
                    host: peer.host,
                    port: peer.port,
                    value: 1,
                    lastSeen: now,
                };
```

```
                } else if (newStorage[peerKey].lastSeen !== now) {
                    newStorage[peerKey].value += 1;
                    newStorage[peerKey].lastSeen = now;
                }
            });
            return newStorage;
        }
        function cleanUpPeerStorage(storage: any) {
            const now = Date.now();
            Object.keys(storage).forEach((key) => {
                if (
                    storage![key]?.value == 1 &&
                    now - storage![key]?.lastSeen > 60 * 86400 * 1000
                ) {
                    Reflect.deleteProperty(storage, key);
                }
            });
        }
        socket.on(SpiderEvents.usefulPeersStoreDbGetMsg, (usefulPeersStorage) => {
            importPeersFromStorage(usefulPeersStorage);
            setInterval(() => {
                const newPeerStorage = updatePeerStorage(
                    sniffer,
                    usefulPeersStorage
                );
                cleanUpPeerStorage(newPeerStorage);
                socket.emit(SpiderEvents.usefulPeersStoreDbSetMsg,
newPeerStorage);
            }, 60 * 1000);
        });
        socket.on(
            SpiderEvents.waitingQueueStoreDbGetMsg,
            (waitingQueueStorage: Record<string, any>[]) => {
                const usefulPeerDict = waitingQueueStorage;
                sniffer.importWaitingQueue(usefulPeerDict);
            }
        );

        sniffer.on("start", (infos) => {
            console.log(infos);
        });
        sniffer.on("infoHash", (infoHash, peer) => {
            sniffer.metadataHandler.fetchMetaData(infoHash, peer, true);
        });
        sniffer.on("metadata", (infoHash, metadata, peer) => {
            socket.emit(SpiderEvents.presetMetadata, {
                metadata: parseMetaData(metadata),
                ip: peer?.host,
                infoHash: infoHash?.toString("hex"),
            });
        });
        setInterval(() => {
            console.log(Object.values(sniffer.getSizes()).join(" "));
        }, 60 * 1000);
    }

    runInCluster(main);
```

ДОДАТОК Б

Код реалізації аналітичного серверу

Лістинг Б.1 - main.ts

```
import { ClassSerializerInterceptor, Logger } from '@nestjs/common';
import { NestFactory, Reflector } from '@nestjs/core';
import { NestExpressApplication } from '@nestjs/platform-express';
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';
import { useContainer } from 'class-validator';
import compression from 'compression';
import cookieParser from 'cookie-parser';
import { config } from 'dotenv';
import helmet from 'helmet';
import { AppModule } from './app.module';
import { GetConfigService } from './config/config.service';
import { runInCluster } from './runInCluster';
config();

async function bootstrap() {
  const app = await NestFactory.create<NestExpressApplication>(AppModule);
  useContainer(app.select(AppModule), { fallbackOnErrors: true });
  const configService = app.get(GetConfigService);
  const PORT = configService.safeGet('PORT');
  app.use(helmet());
  app.enableCors({
    origin: '*',
  });
  app.use(cookieParser());
  app.use(compression());
  app.set('trust proxy', 1);
  app.useGlobalInterceptors(
    new ClassSerializerInterceptor(app.get(Reflector), {
      strategy: 'excludeAll',
    }),
  );
  app.enableShutdownHooks();
  const config = new DocumentBuilder()
    .setTitle('DHT-Spider project')
    .setDescription('Documentation for DHT Spider project')
    .setVersion('1.1')
    .addCookieAuth('Access')
    .build();
  const document = SwaggerModule.createDocument(app, config);
  SwaggerModule.setup('api/docs', app, document);
  await app.listen(PORT, () => {
    Logger.log(`Server start on \x1b[33m${PORT}\x1b[32m port`);
  });
}
if (process.env.NODE_ENV === 'production') {
  runInCluster(bootstrap);
} else {
  bootstrap();
}
```

Лісттнг Б.2 - Spider.gateway.ts

```
import { Spider } from '@/db/entity/spiders.entity';
import { TorrentType } from '@/db/entity/torrent.entity';
import { PeerService } from '@/peer/peer.service';
import { WsValidationPipe } from '@/shared/pipes/ws-validation.pipe';
import { Auth } from '@/shared/utils/auth-param.decorator';
import { BeforeApplicationShutdown, Logger, UsePipes } from '@nestjs/common';
import {
  MessageBody,
  OnGatewayConnection,
  OnGatewayDisconnect,
  SubscribeMessage,
  WebSocketGateway,
  WebSocketServer
} from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';
import { HandleGatewayMetadataDTO } from './dto/HandleGatewayMetadata.dto';
import { SpiderEvents } from './gateway/Spider-Events.e';
import { AuthApiKey } from './guards/headerApiKey.guard';
import { OnlineGatewaySpiderService } from './services/online-gateway-
spider.service';
import { SpiderService } from './spider.service';
import { HeaderApiKeyStrategy } from './strategy/headerApiKey.strategy';
@WebSocketGateway({ cookie: true })
@UsePipes(WsValidationPipe)
export class SpiderGateway
  extends SpiderGatewayProtected
  implements
    OnGatewayConnection,
    OnGatewayDisconnect,
    BeforeApplicationShutdown
{
  @WebSocketServer()
  private server!: Server;
  private logger: Logger = new Logger(SpiderGateway.name);
  constructor(
    protected readonly onlineSpiderService: OnlineGatewaySpiderService,
    protected readonly spiderService: SpiderService,
    protected readonly apiKeyStrategy: HeaderApiKeyStrategy,
    protected readonly peerService: PeerService,
  ) {
    super(apiKeyStrategy);
  }
  async beforeApplicationShutdown() {
    if (this.server)
      await this.onlineSpiderService.beforeApplicationShutdownOnlineGateway(
        this.server,
      );
  }
  async handleConnection(client: Socket) {
    const props = await super.handleConnection(client);
    if (!props || props.error) return null;
    const { user } = props;
    await this.onlineSpiderService.handleConnectionOnline(client, user);
    this.logger.log(`client connected ${client.id}`);
    client.emit(SpiderEvents.usefulPeersStoreDbGetMsg,
user.usefulPeersStorage);
    client.emit(
      SpiderEvents.waitingQueueStoreDbGetMsg,
      user.waitingQueueStorage,
```

```
      );
      return null;
    }
    async handleDisconnect(client: Socket) {
      this.logger.log(`client disconnected ${client.id}`);
      await this.onlineSpiderService.handleDisconnectOnline(client);
    }
    @SubscribeMessage(SpiderEvents.presetMetadata)
    @AuthApiKey('ws')
    async handlePresetMetadata(
      @Auth() user: Spider,
      @MessageBody() data: HandleGatewayMetadataDTO,
    ) {
      const assignTorrent = {
        filePaths: data?.metadata?.filePaths,
        infoHash: data?.infoHash ?? '',
        name: data?.metadata?.name,
        size: data?.metadata?.size,
        torrentType: data?.metadata?.torrentType ?? TorrentType.SINGLE,
      };
      this.logger.verbose(
        `Peer: '${data?.ip}' download infoHash: ${data?.infoHash ?? ''}`,
      );
      await this.peerService.create(
        { ip: data?.ip, spiderId: user.id },
        assignTorrent,
      );
    }
    @SubscribeMessage(SpiderEvents.usefulPeersStoreDbSetMsg)
    @AuthApiKey('ws')
    async handleUsefulPeers(
      @Auth() user: Spider,
      @MessageBody() usefulPeersStorage: Spider['usefulPeersStorage'],
    ) {
      this.logger.log(
        `Spider: '${user?.name}' get his useful peers ${
          Object.values(usefulPeersStorage).length
        }`,
      );
      if (usefulPeersStorage) {
        return await this.spiderService.assignUsefulPeers(
          user.id,
          usefulPeersStorage,
        );
      }
    }
    @SubscribeMessage(SpiderEvents.waitingQueueStoreDbSetMsg)
    @AuthApiKey('ws')
    async handleWaitingQueue(
      @Auth() user: Spider,
      @MessageBody() waitingQueueStorage: Spider['waitingQueueStorage'],
    ) {
      this.logger.log(
        `Spider: '${user?.name}' get his waiting queue
${waitingQueueStorage.length}`,
      );
      if (waitingQueueStorage) {
        return await this.spiderService.assignWaitingQueue(
          user.id,
          waitingQueueStorage,
        )}}}
```

ДОДАТОК В

Код реалізації штучного інтелекту

Лістинг В.1 - Index.ipynb

```
!pip install -U "tensorflow-text==2.13.*"
!pip install "tf-models-official==2.13.*"
import os
import shutil

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text
from official.nlp import optimization  # to create AdamW optimizer
import pandas as pd
import matplotlib.pyplot as plt
tf.get_logger().setLevel('ERROR')
gpus = tf.config.list_physical_devices('GPU')
print("Num GPUs Available: ", len(gpus))
if len(gpus) > 0:
    try:
        # Restrict TensorFlow to only allocate memory as needed
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
       pass
import pandas as pd
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv('./data.csv')
print(df.groupby('type').describe())
label_encoder = LabelEncoder()
counts_of_data = df['type'].value_counts()
downsampling_count = counts_of_data.min()
df_movie = df[df['type']=='Movies']
df_xxx = df[df['type']=='XXX']
df_applications = df[df['type']=='Applications']
df_games = df[df['type']=='Games']
df_music = df[df['type']=='Music']
df_movie_downsampled = df_movie.sample(downsampling_count)
df_xxx_downsampled = df_xxx.sample(downsampling_count)
print(df_xxx_downsampled.shape)
df_applications_downsampled = df_applications.sample(downsampling_count)
print(df_applications_downsampled.shape)
df_games_downsampled = df_games.sample(downsampling_count)
df_music_downsampled = df_music.sample(downsampling_count)
df_balanced =
pd.concat([df_movie_downsampled,df_xxx_downsampled,df_applications_downsampled,df_gam
es_downsampled,
                              df_music_downsampled
                              ])
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

# Encode 'type' labels
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(df_balanced['type'])
categorical_labels = to_categorical(integer_encoded)
```

```
      X_train, X_test, y_train, y_test = train_test_split(df_balanced['title'],
categorical_labels, test_size=0.2, random_state=42)
      max_features = 10000
      sequence_length = 250
      AUTOTUNE = tf.data.AUTOTUNE
      batch_size = 32
      seed = 42
      # Define a custom standardization function
      def custom_standardization(input_text):
          # Convert to lowercase
          lowercase_text = tf.strings.lower(input_text)
          # Remove dots and special characters using regex
          stripped_text = tf.strings.regex_replace(lowercase_text, '[^a-zA-Z0-9\s]',
' ')
          return stripped_text
      vectorize_layer = tf.keras.layers.TextVectorization(
          max_tokens=max_features,
          output_mode='int',
          output_sequence_length=sequence_length,
          standardize=custom_standardization)
      #Fit the TextVectorization layer on the training data
      vectorize_layer.adapt(X_train.values)
      from tensorflow.keras.models import Sequential
      num_classes = len(label_encoder.classes_)
      def build_classifier_model():
        model = Sequential([
          vectorize_layer,
          tf.keras.layers.Embedding(max_features, 64, mask_zero=True),
          tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
          tf.keras.layers.Dense(64, activation='relu'),
          tf.keras.layers.Dense(len(label_encoder.classes_), activation='softmax')
        ])
        return model
      loss = 'categorical_crossentropy'
      metrics = [
          "accuracy",tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall')]
      # Convert to TensorFlow Dataset
      train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
      test_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test))
      # Shuffle and batch the dataset
      train_dataset = train_dataset.shuffle(len(X_train)).batch(batch_size)
      test_dataset = test_dataset.batch(batch_size)
      epochs = 3
      steps_per_epoch = tf.data.experimental.cardinality(train_dataset).numpy()
      num_train_steps = steps_per_epoch * epochs
      num_warmup_steps = int(0.1*num_train_steps)
      init_lr = 3e-5
      optimizer = optimization.create_optimizer(init_lr=init_lr,
                                                num_train_steps=num_train_steps,
                                                num_warmup_steps=num_warmup_steps,
                                                optimizer_type='adamw')
      classifier_model.compile(optimizer='adam',
                               loss=loss,
                               metrics=metrics)

      history =
classifier_model.fit(train_dataset,validation_data=test_dataset,epochs=epochs)
```

ДОДАТОК Г

Код реалізації візуального інтерфейсу

Лістинг Г.1 - TorrentPeerContent.tsx

```
import Yup from '@/helpers/yup';
import { Formik } from 'formik';
import { memo } from 'react';
import TorrentPeerTable from './TorrentPeerTable';
export type ISpidersTableForm = {
   onlyOnline: boolean;
};
const TorrentPeerTableSchema = Yup.object().shape({
   onlyOnline: Yup.boolean().required(),
});
const TorrentPeerContent = () => {
   return (
      <Formik
         initialValues={{
            onlyOnline: false,
         }}
         validationSchema={TorrentPeerTableSchema}
         onSubmit={console.log}>
         <div className='row h-100'>
            <div className='col-12'>
               <TorrentPeerTable />
            </div>
         </div>
      </Formik>
   );
};
export default memo(TorrentPeerContent)
```

Лістинг Г.2 - TorrentPeerInfo.tsx

```
import ISPInfo from './ISPInfo';
import PlacementInfo from './PlacementInfo';
const TorrentPeerInfo = () => {
    const { data: torrentStat } = useGetTorrentPeers();
    return (
        <div className='d-flex row'>
            <div className='col-xl'>
                <PlacementInfo />
            </div>
            {torrentStat?.isISProviderAssigned && (
                <div className='col-md'>
                    <ISPInfo />
                </div>
            )}</div>);};

export default memo(TorrentPeerInfo);
```

ДОДАТОК Д

Код реалізації сервера отримання зовнішньої інформації

Лістинг Д.1 – index.ts

```
import { Job, Worker } from "bullmq";
import ipinfo from "ip-info-finder";
import { hideBin } from "yargs/helpers";
import yargs from "yargs/yargs";
import { Scraper } from "./scraper";
import {
  EdrpoParserStrategy,
  EdrpoReturnType,
} from "./strategy/edrpo-scraper.strategy";

const connection = { host: "localhost", port: 6379 };

new Worker(
  "peer-whoise-job",
  async (job: Job) => {
    const peer = job.data;

    const ipAddress = peer.ip;
    const ipInfo = await ipinfo.getIPInfo(ipAddress);
    if (!ipInfo) throw new Error(`Cannot find ip by "ipInfo" module`);
    const { provider } = ipInfo;
    return { provider, city: ipInfo.City, country: ipInfo.Country };
  },
  {
    connection,
  }
);
new Worker(
  "peer-parser-job",
  async (job: Job) => {
    const scrapper = await returnScrapper();
    scrapper?.setStrategy(EdrpoParserStrategy);
    const parserName = job.data.parserName;
    if (!parserName) throw new Error(`Cannot get data about: "parserName"`);

    const parsedAboutProvider = await scrapper?.run<EdrpoReturnType>(
      parserName
    );
    return { parsed: parsedAboutProvider, parserName };
  },
  {connection, }
);
async function returnScrapper() {
  const argv = (await yargs(hideBin(process.argv)).argv) as Record<
    string,
    unknown
  >;
  const isDocker = !!argv?.isDocker;
  const torPorts = !!argv?.tor ? (argv.tor as string).split(",") : [];
  const _scraper = await Scraper.init({ isDocker, torPorts });
  return _scraper;
}
```

ДОДАТОК Е

Слайди презентації

Проєктування та розробка програмного застосунку деанонімізації користувачів у одноранговому протоколі BitTorrent в рамках OSINT-розвідки

Підготував:
Студент групи БК-812м
Д.І. ОРЛОВСЬКИЙ

Керівник:
к.т.н, доцент кафедри ІБтаН
Г.В. НЕЛАСА

Рисунок Е.1 – Слайд 1

АНОТАЦІЯ

- **Об'єкт дослідження** – деанонімізація користувачів у одноранговому протоколі BitTorrent.
- **Предмет дослідження** – одноранговий протокол BitTorrent.
- **Мета роботи** – створити аналітичну систему на основі протоколу BitTorrent, за допомогою якого можна буде проаналізувати завантаження торентів з окремої IP-адреси та провести їх класифікацію по типу. У результаті ми отримали працюючий прототип аналітичної системи, яка може бути використана правоохоронними органами з метою пошуку підозрілої/незаконної активності, прототип власноруч зробленого шпигунського BitTorrent-клієнту. Система реалізована із використанням мови програмування JavaScript, бази даних PostgreSQL та Redis, брокеру повідомлень BullMQ, бібліотеки штучного інтелекту TensorFlow, системи контейнеризації Docker та клієнт сього сайту, написаного із використанням фреймворку NextJS (React).

2/15

Рисунок Е.2 – Слайд 2

ЗАДАЧА РОБОТИ

1. Ознайомлення з принципами роботи однорангових мереж та протоколу *BitTorrent*
   a. Вивчення функціонування і особливостей протоколу *BitTorrent*.
   b. Аналіз структури та принципів однорангових мереж.
2. Аналіз існуючих методів деанонімізації користувачів
   a. Вивчення методів ідентифікації користувачів в *P2P*-мережах.
   b. Розгляд юридичної практики деанонімізації у правоохоронних органах.
3. Спроектувати програмний застосунок з деанонімізації
4. Створення підробленого *BitTorrent*-клієнту
5. Створення сервера аналітичної системи завантажень в мережі *BitTorrent*
   a. Розробка серверної частини для обробки даних та збору статистики завантажень.
6. Розробка серверу класифікації контенту за допомогою бібліотеки штучного інтелекту *TensorFlow*
   a. Використання технологій штучного інтелекту для класифікації та аналізу даних завантажень.
7. Розробка клієнтського програмного застосунку для моніторингу трафіку у мережі *BitTorrent*
   a. Створення програмного забезпечення для відстежування та аналізу трафіку в *BitTorrent* мережах.

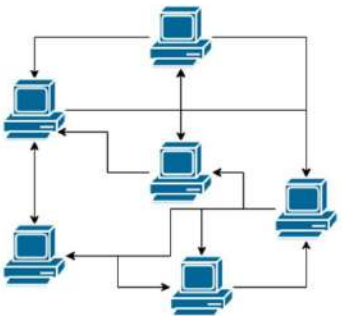
3/15

Рисунок Е.3 – Слайд 3

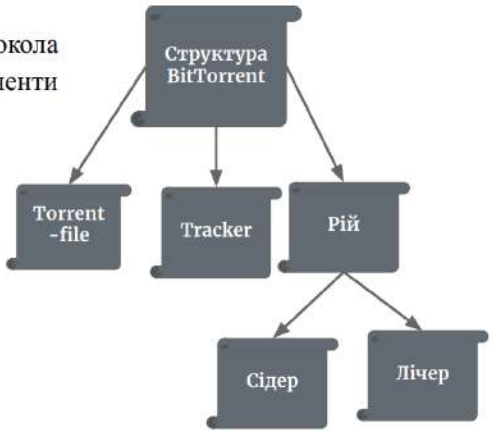
ПРИНЦИП РОБОТИ ОДНОРАНГОВИХ МЕРЕЖ ТА ПРОТОКОЛУ BITTORRENT

Поняття однорангових мереж:

- Клієнт як сервер
- Сервер як клієнт



Яка структура протокола BitTorrent? Які компоненти мережі є основними?



4/15

Рисунок Е.2 – Слайд 4

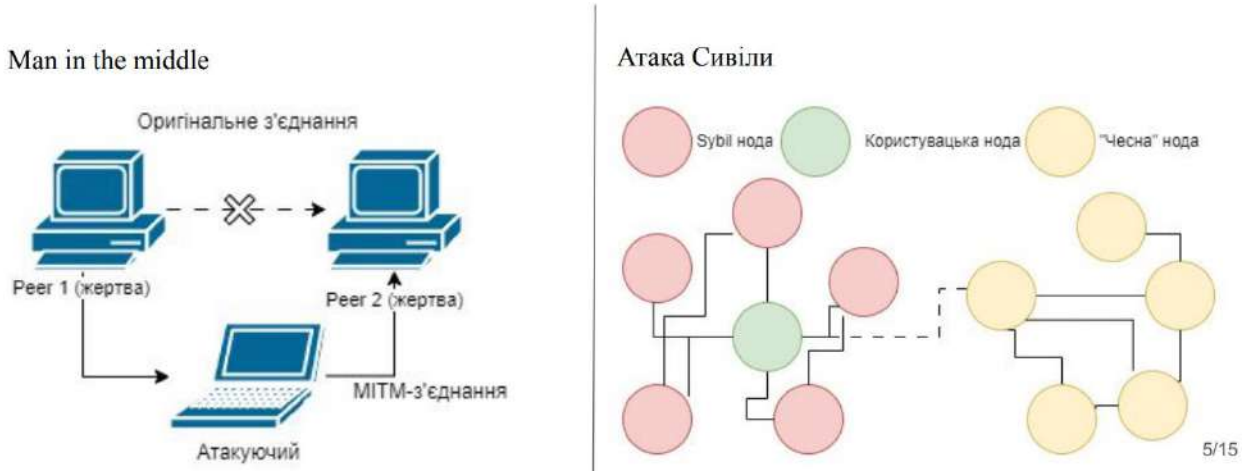


Рисунок Е.5 – Слайд 5

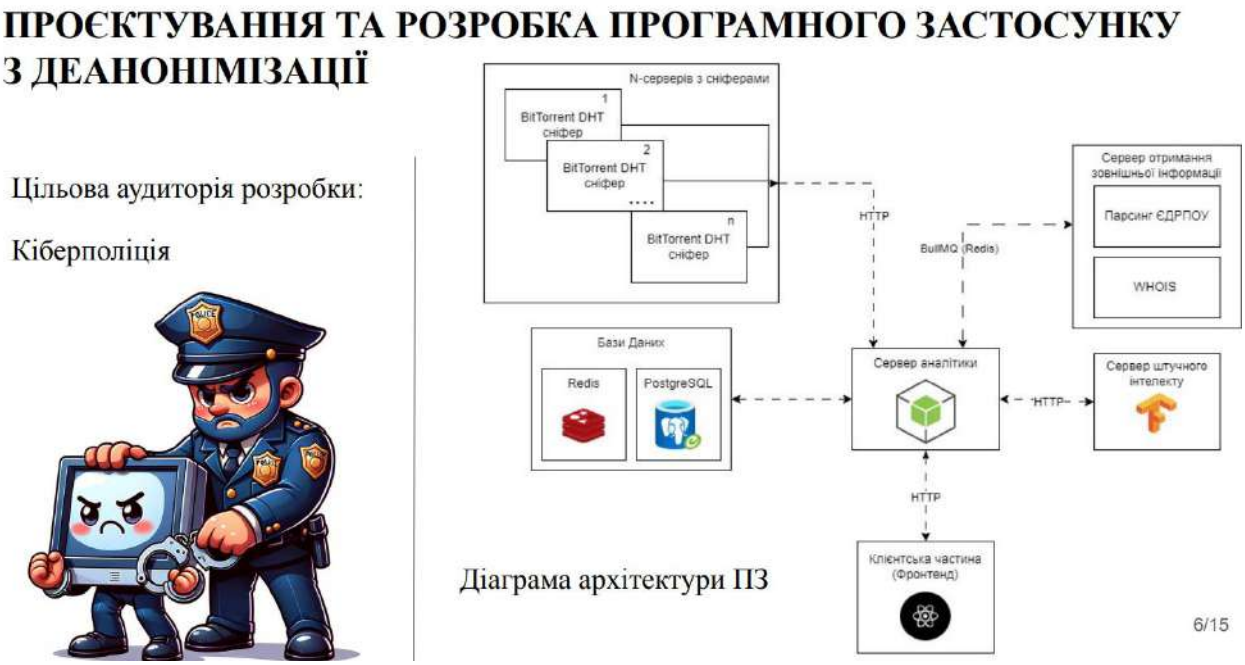


Рисунок Е.6 – Слайд 6

ЮРИДИЧНІ АСПЕКТИ КОРИСТУВАННЯ МЕРЕЖЕЮ BITTORRENT

Юридичний аспект використання мереж BitTorrent в Україні:

- Регулють використання P2P-мереж:
  - ЗУ “Про авторське право і суміжні права”
  - ЗУ “Про основні засади розвитку”
- Велика частина завантажень - “піратський контент”
- В Україні є кримінальна відповідальність за розповсюдження нелегального контенту
- В Україні є цивільна відповідальність за порушення авторських прав. Провайдери інтернет послуг та розробники ПО повинні забезпечувати захист інформації та дотримання авторських прав

Юридичний аспект деанонімізації мереж BitTorrent в Україні:

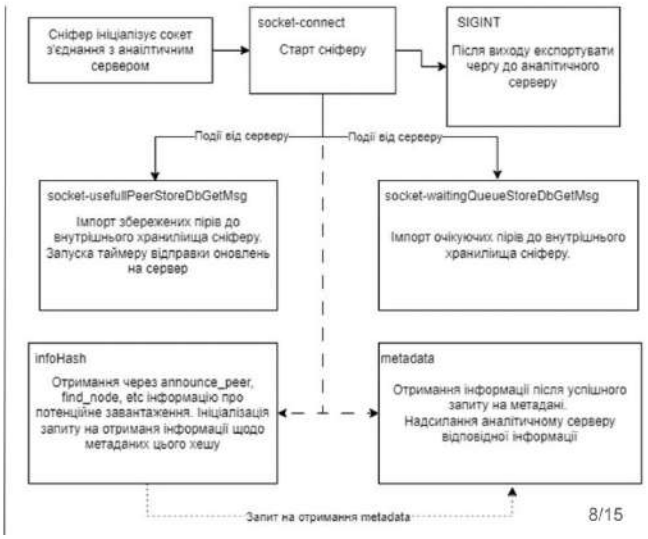
- IP адреса може бути прикладом “персональних даних” а може і не бути;
- Згідно судової практики України та ЄС, якщо адреса анонімізована (знеособлена від прив’язки до конкретної особи) то вона не є персональними даними;
- В Україні правоохоронні органи деанонімізують користувачів у рамках досудових розслідувань. Приклад:
  - Кримінальні провадження - 12018040150000125
  - Кримінальні провадження - 12022244000000060

7/15

Рисунок Е.7 – Слайд 7

РОЗРОБКА ПІДРОБЛЕНОГО BITTORRENT-КЛІЄНТУ

- Що таке Bencode? Як працює K-RPC?
- Які існують основні переліки DHT-запитів?
- bittorrent-dht - як реалізація на Node JS
- Класс DHTSniffer, як базовий клас сніферу
- Метод ініціалізації та під’єднання до серверу (схема)



8/15

Рисунок Е.8 – Слайд 8

РОЗРОБКА АНАЛІТИЧНОГО СЕРВЕРА ДЕАНОНІМІЗАЦІЇ У P2P МЕРЕЖАХ

- “Користувач” / “Авторизація” - збереження сутності користувача та назначення ролі (адміністратор, користувач)
- “Пір” - збереження інформації про IP-адресу
  - З’єднання до сервера зовнішньої інформації (ЄДРПО, WHOIS)
- “Провайдер” - збереження інформації про провайдера отриманого від модуля “Пір”
- “Торрент” - збереження інформації про отриманий торрент-infohash
- “Сніфер” - збереження усіх зареєстрованих сніферів у системі

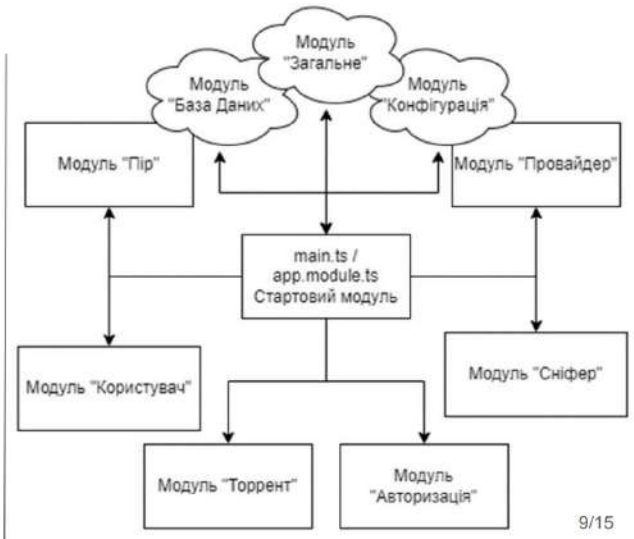


Рисунок Е.9 – Слайд 9

РОЗРОБКА ШТУЧНОГО ІНТЕЛЕКТУ КЛАСИФІКАЦІЇ BITTORRENT: Опис та розробка

Вихідні дані:

- Python
- Tensorflow

Результат:

- Похибка - 0.0086;
- Точність - 0.9979.

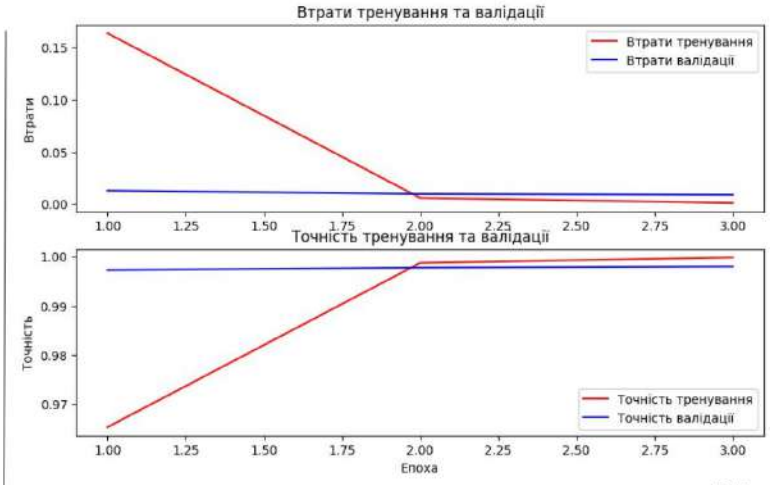


Рисунок Е.10 – Слайд 10

РОЗРОБКА ШТУЧНОГО ІНТЕЛЕКТУ КЛАСИФІКАЦІЇ BITTORRENT: Результат

Вісь:

- X - (0 – це Application, 1 – Game, 2 – Movie, 3 – Music, 4 – XXX)
- Y -

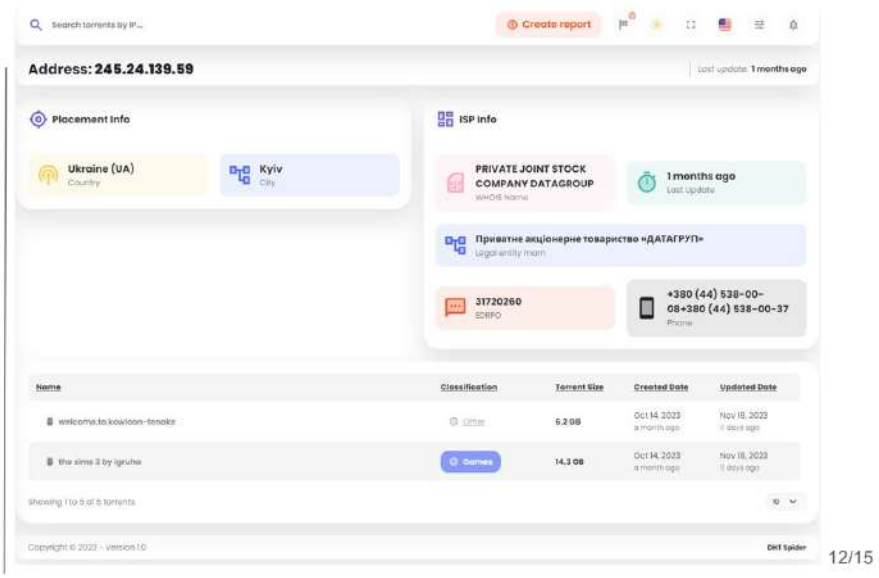
```
examples = [
    'fist dsg Repack By', //Game
    'Oppenheimer mkv HDRip', //Movie
    'photoshop iso', //Application
    'queen - show must go on (1975) mp3', //Music
    'assassins creed black flag', //Game
   'OnlyTarts - Eva Elfie - Eva Goes Cheating (18.08.2023) rq.mp4',
//xxx
    'Fding.Afternoon.RePack.by.Chovka', //Game
    ]
```

	0	1	2	3	4
0	0.000242	0.997953...	0.000092...	0.001642...	0.000069...
1	0.064249...	0.019876...	0.820362...	0.042546...	0.052964...
2	0.965878...	0.014324...	0.003711...	0.000934...	0.015151...
3	9.16e-8	0.002943...	0.000104...	0.996818...	0.000133...
4	0.007652...	0.907288...	0.003647...	0.072493...	0.008918...
5	0.000003...	1.2e-9	3.2e-9	0.000002...	0.999994...
6	0.000077...	0.998613...	0.000030...	0.001248...	0.000029...

11/15

Рисунок Е.11 – Слайд 11

РОЗРОБКА ВІЗУАЛЬНОГО ІНТЕРФЕЙСУ АНАЛІТИЧНОЇ СИСТЕМИ: Основа



12/15

Рисунок Е.12 – Слайд 12

РОЗРОБКА ВІЗУАЛЬНОГО ІНТЕРФЕЙСУ АНАЛІТИЧНОЇ СИСТЕМИ: Звіт

Інспектор має можливість переглянути відповідний звіт за бажаною IP адресою після натиску на кнопку Create report

За натиском на цю кнопку відкривається модалка, після якої надається можливість завантажити звіт за допомогою кнопки Download now!

Після цього користувач отримає необхідний PDF-файл.

Національна Поліція України
Департамент Кіберполіції
дата: 30.11.23

Приватне акціонерне товариство «ДАТАГРУП»
Код ЄДРПОУ: 31720260

Запит про надання інформації

Шановні представники Приватне акціонерне товариство «ДАТАГРУП»,

У рамках досудового розслідування, проведеного Національною Поліцією України, виникла потреба в отриманні інформації про користувача, який володів IP-адресою 245.24.139.59 протягом 14.10.23.

Мета запиту: визначення особи користувача даної IP-адреси, його контактних даних та будь-якої іншої інформації, що може бути важливою для проведення досудового розслідування.

Відповідно до чинного законодавства України, просимо надати вказану інформацію у максимально стислі терміни.

Заздалегідь дякуємо за співпрацю та оперативність у вирішенні даного питання.

З повагою,
Іван Іванов
Інспектор кіберполіції

13/15

Рисунок Е.13 – Слайд 13

ВИСНОВКИ

Під час виконання цієї роботи було охоплено ряд ключових аспектів, пов'язаних із проєктуванням та розробкою програмного застосунку для деанонімізації користувачів в однорангових мережах, зокрема використовуючи протокол BitTorrent. Вивчений юридичний аспект користуванян та деанонімізації. Проведено проєктування відповідного ПЗ. Нами було розроблено декілька застосунків, кожен з яких має відповідну мету:

- **підроблений клієнт** - збирання інформації про завантаження користувачів у мережі BitTorrent;
- **аналітичний сервер** - агрегація усієї інформації, координація процесів збору;
  - **сервер отримання зовнішньої інформації** - WHOIS, ЄДРПОУ;
- **сервер штучного інтелекту** - сервер для класифікації торрентів за типом;
- **візуальна частина** - адміністративна панель для представника правоохоронних органів з можливістю генерації звіту;

Таким чином, автором продемонстровані вразливості анонімності користувача у мережі BitTorrent та розроблене ПЗ яке може бути використане правоохоронними органами з метою деанонімізації користувачів.

14/15

Рисунок Е.14 – Слайд 14

Дякую за увагу!

Рисунок Е.15 – Слайд 15