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

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-2
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Гонем А.В.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Марченко К.М.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Гонему Аміру Ваєльовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення*

2. Керівник роботи *Марченко Костянтин Миколайович, канд. техн. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 47-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення*

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

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Марченко К.М.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Гонем А.В.
(прізвище та ініціали)

АНОТАЦІЯ

Гонем А.В. Програмне забезпечення структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

Метою розробки є програмне забезпечення структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

Результат роботи – програмна реалізація структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

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

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

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

Програму розроблено в середовищі Python.

Ключові слова: комп'ютерна інженерія, GERT-модель

ABSTRACT

Gonem A.V. Software for the structural-logical GERT model of the technology of spreading malicious software. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for the structural-logical GERT model of the technology of spreading malicious software.

The purpose of the development is the software for the structural-logical GERT model of the technology of spreading malicious software.

The result of the work is the software implementation of the structural-logical GERT model of the technology of spreading malicious software.

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

A convenient user interface has been developed. Instructions for working with software are provided.

The program can be used on PCs with Windows 10/11.

The program is developed in Python.

Keywords: computer engineering, GERT model

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 2
ВСТУП..... 3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 5
1.1 Призначення системи..... 5
1.2 Область застосування..... 6
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 8
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти..... 8
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування..... 19
2.3 Розгорнута постановка завдання 24
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 26
3.1 Опис функціонування системи 26
3.2 Розробка структурної схеми..... 35
3.3 Розробка функціональної схеми 41
3.4 Розробка діаграми процесів..... 55
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ..... 57
4.1 Розробка блок-схем та опис алгоритмів функціонування системи..... 57
4.2 Захист розробленого програмного забезпечення..... 65
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ..... 67
6 ОСНОВНІ ВИСНОВКИ..... 73
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 75

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

LLQ – Low Latency Queuing

PPS – Priority Processor Sharing

PSIDDRM– Progressive Suspected-Infected-Detected-Death-Recovered Markov

PSIDDR – Progressive Suspected-Infected-Detected-Death-Recovered

PSIDR – Progressive Suspected-Infected-Detected-Recovered

PSIDRM – Progressive Suspected-Infected-Detected-Recovered Markov

PSIDRT – Progressive Suspected-Infected-Detected-Recovered with Topology

QoSS – Quality of Service

RRP – Role Resolution Protocol

RSVP – Resource Reservation Protocol

SEIQR– Suspected-Exposed-Infected-Quarantined-Recovered

SI – Suspected-Infected

SIM – Suspected-Infected Markov

SIR – Suspected-Infected-Recovered

SIRT – Suspected-Infected-Recovered with Topology

SIRM – Suspected-Infected-Recovered Markov

SIT – Suspected-Infected with Topology

SI-WF^{2}Q – Stratified/Interleaved Worst-case Fair Weighted Fair Queueing

SCFQ – Self-Clocked Fair Queuing

VC – Virtual Clock

VFT – Virtual Finish Time

VST – Virtual Start Time

WFQ – Worst-Case Queuing

WFQI – Worst-Case Queuing Improved

WF^{2}Q – Worst-Case Fair Weighed Fair Queuing

ВСТУП

Актуальність теми. Хмарний захист від вірусів – це технічна функція, яка визначає роботу антивірусу . Відмінності між хмарним і традиційним антивірусом найбільш помітні, коли йдеться про зберігання визначень зловмисного програмного забезпечення та сканування на віруси.

Традиційні антивірусні рішення зберігають відомі визначення шкідливих програм і виконують сканування безпосередньо на пристрої. З іншого боку, хмарний антивірус робить все в хмарі. У результаті визначення зловмисного програмного забезпечення в хмарі оновлюються в режимі реального часу , а це означає, що антивірус може миттєво виявити найновіші загрози.

З традиційними рішеннями захисту оновлення потрібно виконувати вручну. Протягом цього часу, між випуском оновлення та встановленням, нове та невідоме зловмисне програмне забезпечення має більший шанс заразити ваш пристрій.

Ще одна перевага хмарного антивіруса – краща продуктивність комп'ютера. Це тому, що хмарне сканування потребує менше обчислювальної потужності вашого пристрою і, як наслідок, не погіршує його продуктивність.

Мета й завдання дослідження. Метою роботи є програмне забезпечення структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

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

1 Огляд існуючих систем структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

2 Дослідження структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

3 Програмна реалізація структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Хмарні антивірусні рішення – це ті, які зберігають інформацію про варіанти шкідливих програм у хмарі, а не на пристрої користувача. Традиційні, так звані «спискові» антивірусні рішення зберігають списки завідомо несправних бітів коду на самому пристрої, що може негативно вплинути на продуктивність машини.

Щоб краще зрозуміти хмарне антивірусне програмне забезпечення, допоможе зрозуміти «хмару». Хмара – це просто децентралізований простір для зберігання даних, до яких ваш комп'ютер має доступ через Інтернет.

Зберігаючи визначення загроз (файли, класифіковані як зловмисне програмне забезпечення або небезпечні IP-адреси та URL-адреси) у хмарі, а не на самому пристрої, хмарному антивірусному програмному забезпеченню не потрібно зберігати всі ці мільйони визначень на власному жорсткому диску, звільняючи місце.

А оскільки оновлення можна надсилати у ваше антивірусне програмне забезпечення віддалено через хмару, ви не застрягнете зі статичним списком загроз, від яких програмне забезпечення знає, що захищає. Як тільки будуть виявлені нові загрози, скажімо, командою дослідників загроз, які підтримують ваше антивірусне програмне забезпечення, оновлення можна розмістити на всіх пристроях, які його використовують. Завдяки цьому забезпечується захист майже в режимі реального часу від постійної зміни загроз, які існують в Інтернеті.

1.2 Область застосування

Переваги хмарних антивірусних рішень:

4 Доступ до більшої бази даних загроз без необхідності зберігати її на жорсткому диску.

5 Менший інсталяційний агент для вашого антивірусного програмного забезпечення, тому він займає менше місця.

6 Оновлення визначень майже в реальному часі на основі даних, зібраних від усієї мережі користувачів.

Існуючий каталог поліморфного зловмисного програмного забезпечення, фішингу нульового дня та аналогічних загроз є величезним. Зберігання такої величезної кількості даних на одному комп'ютері призвело б до зниження продуктивності до такої міри, що комп'ютер став би майже непрацездатним під час сканування.

На щастя, значний прогрес у хмарних обчисленнях був досягнутий на рубежі нового тисячоліття. Для тих компаній, які можуть використовувати потужність хмари для антивірусного програмного забезпечення, чекають серйозні покращення швидкості та продуктивності. Менші агенти інсталяції означали менше часу та місця під час інсталяції. Менша кількість визначень, що зберігаються на пристрої, стала означати менше або повну відсутність перерв для рутинного сканування.

Окрім зменшення навантаження на пам'ять для окремих пристроїв, оновлення визначень – ці виправлення курсу, додані до програмного забезпечення для захисту від нещодавно виявлених загроз – займають хвилини, щоб досягти програмного забезпечення, а не дні чи тижні, які можуть знадобитися, щоб зробити ті самі оновлення без допомоги хмари. Якщо вам коли-небудь доводилося чекати, поки комп'ютер оновиться, антивірусне програмне забезпечення оновлюється, ви знаєте, яке розчарування. Це не кажучи вже про загрози, яким ви могли наражатися в проміжку між оновленнями версій.

Хмарні антивірусні рішення не менш важливі для бізнесу, ніж домашніх користувачів. Навпаки, якщо інсталювати систему захисту кінцевих точок на великій кількості пристроїв, повільна інсталяція, спричинена великими агентами, характерними для традиційних антивірусних рішень, може справді збільшитися.

Крім того, хмарний антивірус для бізнесу гарантує, що кожен пристрій буде захищено від нових загроз лише за кілька хвилин після того, як їх виявить будь-яка захищена кінцева точка в усьому світі, без оновлень вручну чи інших перерв у роботі.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Щоб допомогти вам вибрати найкращий хмарний антивірус, ми перевірили для вас десяток різних постачальників. Нижче ви знайдете докладні описи чудових продуктів, які ми вибрали.

TotalAV

TotalAV – найкращий хмарний антивірус, оскільки він забезпечує розширене хмарне сканування невідомих або підозрілих файлів у режимі реального часу. Крім того, ви отримуєте повний пакет безпеки з багатьма додатковими функціями для захисту вашого пристрою, а також конфіденційної інформації.

Захист: цей хмарний антивірус показав чудові результати в незалежних тестах – 99% захисту від zero-day і 99,96% від широко поширеного шкідливого програмного забезпечення. Тож ви можете бути впевнені, що ваша система захищена від вірусів, хробаків, троянських програм чи навіть програм-вимагачів.

Продуктивність: не знадобиться багато часу, щоб переконатися, що на вашому пристрої немає шкідливих програм. Під час наших тестів швидке сканування за допомогою TotalAV займало всього 14 секунд, а повне сканування – 6 хвилин 36 секунд. Це надзвичайно висока продуктивність, оскільки більшості антивірусів потрібно більше 20 хвилин для повної перевірки системи.

Особливості: окрім хмарного захисту в режимі реального часу, цей антивірус також пропонує додаткові функції для онлайн-безпеки, наприклад VPN. Крім того, ви отримуєте менеджер паролів, щоб мінімізувати ризики злому

облікового запису, і AdBlock, щоб видалити нав’язливу та потенційно шкідливу рекламу під час перегляду.

Сумісність: неважливо, який тип пристрою ви використовуєте, оскільки цей хмарний антивірус сумісний з усіма основними операційними системами. Це включає Windows, MacOS, Android та iOS. І настільні, і мобільні програми добре організовані та прості в налаштуванні.

Ціна: цей постачальник пропонує кілька варіантів підписки, а ціни починаються від 19,00 доларів США на рік за 3 пристрої з 30-денною гарантією повернення грошей. Крім того, є безкоштовна версія, за допомогою якої ви можете спробувати продукт перед покупкою.

Переваги:

– Високий рівень виявлення шкідливих програм.
– Дуже швидке сканування.
– Багатофункціональний продукт.
– Підтримує всі основні пристрої.
– 30-денна гарантія повернення грошей.

Недоліки:

– Безкоштовна версія з обмеженнями.

Surfshark AV

Surfshark AV також є одним із найкращих хмарних антивірусних рішень. Окрім сканування в режимі реального часу та за розкладом, він має систему Surfshark Cloud Protect, яка перевіряє нерозпізнані файли в хмарі, щоб краще виявляти рідкісні типи зловмисного програмного забезпечення.

Захист: ви можете очікувати високих показників виявлення зловмисного програмного забезпечення, оскільки під час наших тестів він успішно виявив 7/10 зловмисних файлів серед 100 000 звичайних. Це дійсно чудовий результат, враховуючи, що ми використовували різні типи вірусів, включаючи трояни, шпигунське програмне забезпечення, рекламне програмне забезпечення та інші. Високі показники Surfshark у виявленні зловмисного програмного забезпечення

не залишилися непоміченими – його послідовний захист приніс йому престижну нагороду AV-TEST «Кращий продукт», що відображає його надійність у реальних сценаріях.

Продуктивність: ще одна чудова перевага Surfshark AV полягає в тому, що він легкий – середнє використання ЦП під час нашого повного тесту сканування становило близько 22%, і для його завершення знадобилося 25 хвилин. Найвищий пік становив 53%, але він швидко впав. Тож вплив на продуктивність ПК не помітний.

Особливості: цей провайдер пропонує пакет безпеки «все в одному». Для цього ви отримуєте першокласний Surfshark VPN – один із найкращих доступних на даний момент сервісів VPN. Крім того, є можливість використовувати приватну пошукову систему, щоб приховати свою діяльність в Інтернеті та навіть отримувати сповіщення, якщо ваші дані витікають.

Сумісність: цей хмарний антивірус має спеціальні програми для Windows, Android і Mac. Ми перевірили кожен із них – дизайн не захаращений, і вони надзвичайно прості у використанні. Однак, оскільки це відносно новий продукт, програми для iOS ще немає.

Ціна: ціни на підписку починаються від 36,32 доларів США на рік за 5 пристроїв і включають 30-денну гарантію повернення грошей. Ви також можете вибрати тривалість підписки залежно від ваших потреб.

Переваги:

– Чудовий захист від шкідливих програм.
– Включає першокласний VPN для конфіденційності в Інтернеті.
– Захист у режимі реального часу.
– Зручні програми.
– 30-денна гарантія повернення грошей.

Недоліки:

– Не працює з iOS.

NordVPN Threat Protection

NordVPN Threat Protection – це чудове рішення, яке використовує хмарні технології для захисту пристроїв від шкідливих сайтів, програм, трекерів і реклами. Крім того, оскільки NordVPN спочатку є постачальником послуг VPN, він також дозволяє підключатися до захищеного сервера VPN і шифрувати ваш трафік для додаткового рівня безпеки.

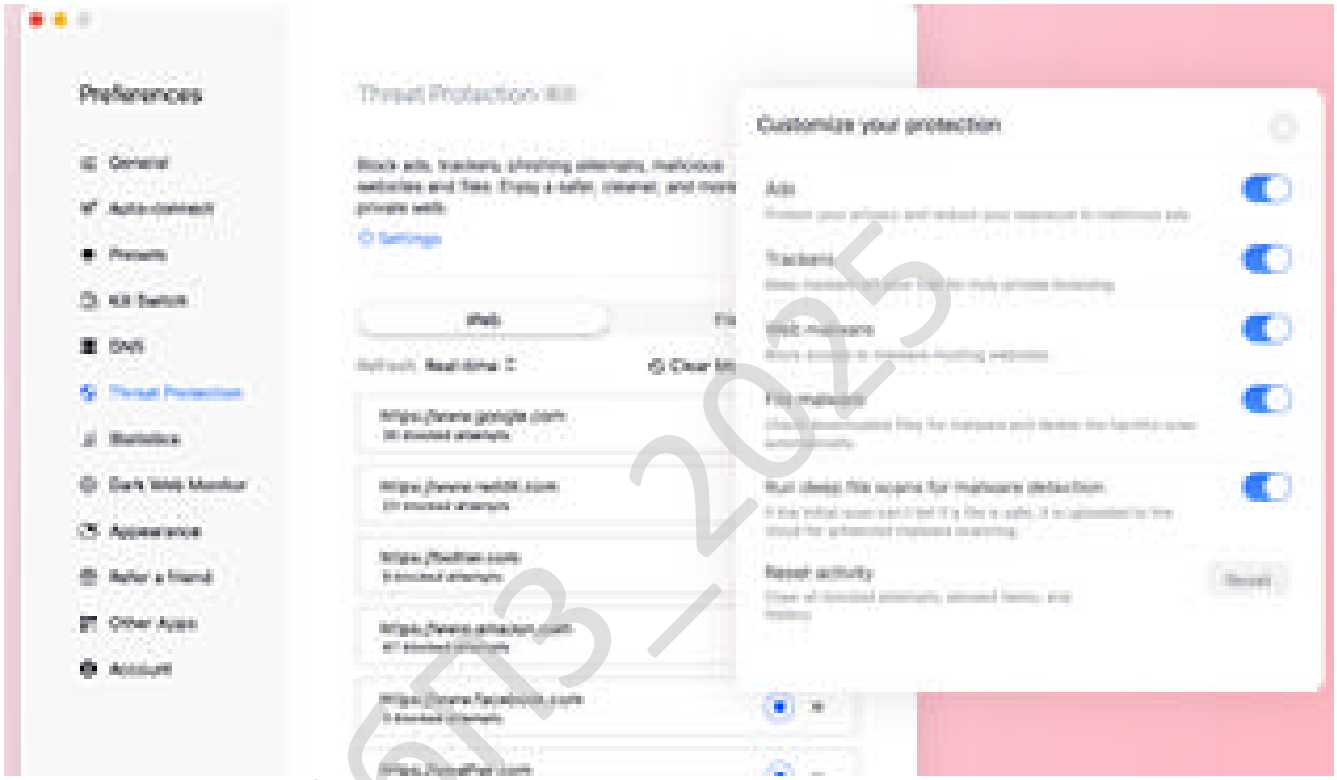

Рисунок 2.1 – Інтерфес користувача NordVPN Threat Protection

Захист: функція захисту від загроз включає моніторинг у реальному часі для ефективного блокування потенційно небезпечних веб-сайтів, завантажень, трекерів і реклами. Після первинної перевірки він додатково завантажує підозрілі файли в хмару для вторинної перевірки. Таким чином ви отримуєте ще сильніший захист від онлайн-загроз.

Продуктивність: це чудовий вибір для користувачів, яким потрібна дуже легка безпека на основі хмари. Він не займає багато місця, і після встановлення

програми NordVPN він безперервно працює у фоновому режимі, не перериваючи вашу звичайну діяльність.

Особливості: окрім запобігання зловмисному програмному забезпеченню, це рішення безпеки також дозволяє захистити вашу конфіденційну інформацію, розблокувати потокові платформи та насолоджуватися безпечним торрент-завантаженням. Для цього вам просто потрібно ввімкнути VPN-з'єднання та вибрати сервер у бажаному місці.

Сумісність: ви можете використовувати функцію захисту від загроз і служби VPN на всіх типах пристроїв, включаючи Windows, macOS, Android та iOS. Програми дуже зручні, сучасні та прості в налаштуванні.

Ціна: Threat Protection постачається разом із усіма планами NordVPN без додаткових витрат. Ціни на підписку починаються від 3,99 доларів США на місяць за 10 пристроїв і включають 30-денну гарантію повернення грошей. Крім того, для Android доступна безкоштовна 7-денна пробна версія.

Переваги:

– Повноцінний захист від різних загроз.
– Сканування завантажень у реальному часі.
– Один із найкращих VPN включено.
– Легке рішення.
– 30-денна гарантія повернення грошей.

Недоліки:

– Немає сканування системи в реальному часі.

McAfee

Антивірус McAfee також є хмарним, що робить його хорошим вибором для безпеки. Крім того, оскільки він виконує сканування в хмарі, а не на самому пристрої, це не сильно впливає на продуктивність вашого пристрою. Тож ви можете легко продовжувати виконувати звичайні завдання, поки ваша система перевіряється.

Захист: незалежні тестові лабораторії демонструють чудові показники

виявлення зловмисного програмного забезпечення McAfee. Він ефективний на 99,4% у виявленні зловмисного програмного забезпечення нульового дня та на 100% ефективний, коли йдеться про поширені загрози.

Продуктивність: під час наших власних тестів ми також отримали хороші результати продуктивності. Швидке сканування зайняло 6 хвилин 20 секунд, а повне сканування – 22 хвилини. Крім того, він не використовував значну кількість процесора, тому це також не сильно впливає на продуктивність комп'ютера.

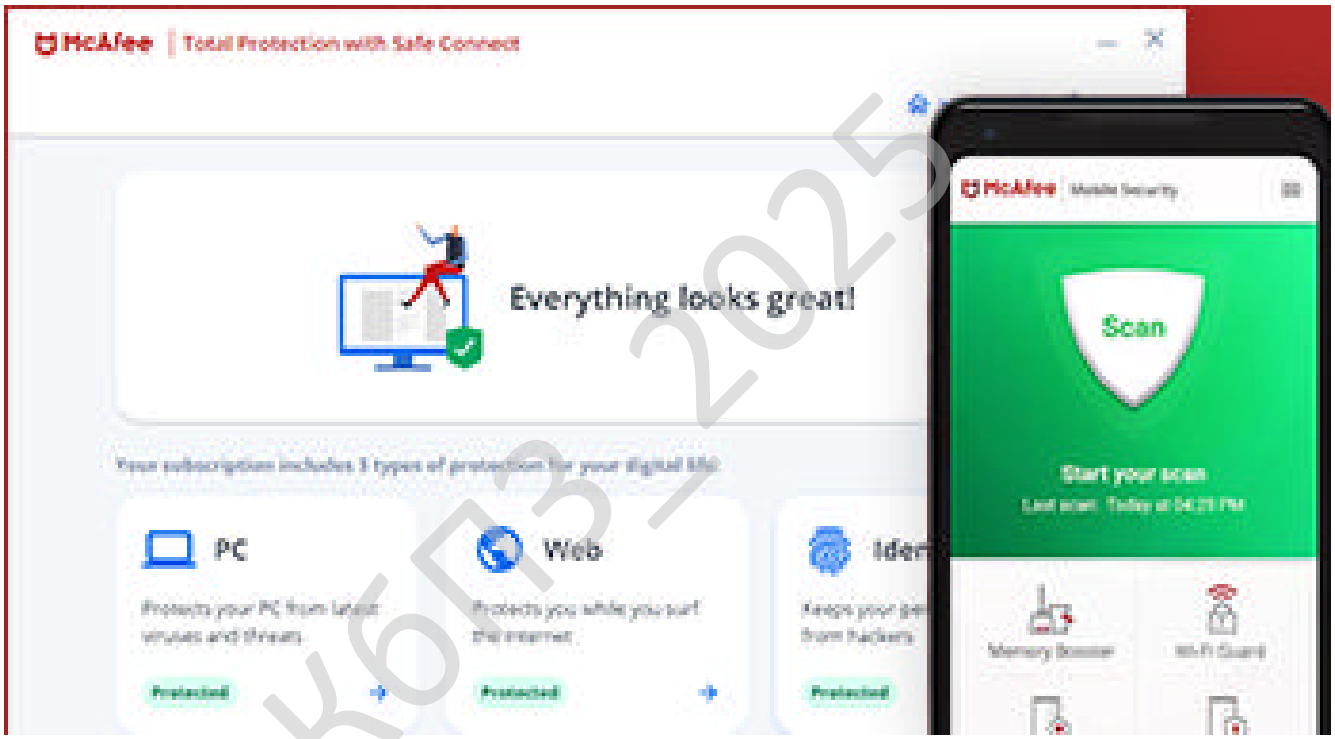

Рисунок 2.2 – Інтерфейс користувача McAfee

Особливості: важливо зазначити, що McAfee містить брандмауер, який запобігає проникненню зловмисного програмного забезпечення в реальному часі. Крім того, коли мова йде про додаткові засоби, цей антивірус також досить щедрий. Ви отримуєте McAfee Safe Connect VPN , шредер файлів, менеджер паролів та інструменти для прискорення.

Сумісність: цей хмарний антивірус підтримує всі популярні настільні та

мобільні пристрої – Windows, macOS, iOS та Android. Хоча завантажувати та встановлювати програми легко, спочатку інтерфейс може заплутати. Тому вам, можливо, доведеться витратити деякий час на ознайомлення.

Ціна: McAfee коштує від 39,99 доларів на рік із 30-денною гарантією повернення грошей. Існує також безкоштовна версія, яку можна спробувати на своєму Android або iPhone.

Переваги:

– Хороші показники виявлення шкідливих програм.
– Не сильно впливає на продуктивність ПК.
– Багато додаткових функцій.
– Включає брандмауер.
– 30-денна гарантія повернення грошей.

Недоліки:

– Програми трохи захаращені.
– Плутані тарифні плани.

Avira

Антивірус Avira – ще один надійний варіант для пошуку найкращих хмарних служб захисту. Він має преміальну та безкоштовну версії, обидві з яких використовують хмарні технології під час сканування пристроїв на наявність шкідливих програм.

Захист: згідно з результатами незалежної лабораторії тестування, Avira забезпечує 100% захист від зловмисного програмного забезпечення нульового дня та 99,98% від поширених загроз. Таким чином, ви точно отримаєте надійний захист як від поширених, так і від невідомих вірусів.

Продуктивність: ми протестували опцію повного сканування, і для завершення всієї перевірки системи знадобилося лише 3 хвилини 30 секунд. Хоча це виглядає вражаюче, варто зазначити, що воно потребує досить багато потужності ЦП і може помітно вплинути на продуктивність вашого ПК під час процесу сканування.

Особливості: окрім хмарного сканування, Avira також пропонує менеджер паролів, щоб запобігти злому облікового запису. Крім того, є необмежений VPN, засіб для видалення трекерів і функція захисту від фішингу, щоб захистити вас в Інтернеті.

Сумісність: цей антивірус працює з усіма основними операційними системами, такими як Windows, macOS, Android та iOS. Під час наших тестів на встановлення знадобилося близько 4-5 хвилин, а програми були інтуїтивно зрозумілими та зручними.

Ціна: плани передплати Avira починаються від 26,99 доларів США на рік і пропонують 30-денну гарантію повернення грошей. Крім того, ви завжди можете скористатися безкоштовною версією, щоб перевірити, як працює хмарна технологія, перш ніж купувати преміум-план.

Переваги:

– Багаторівневий захист із хмарною технологією.
– Швидке сканування.
– Включає VPN і менеджер паролів.
– Щедра безкоштовна версія.
– 30-денна гарантія повернення грошей.

Недоліки:

– Брандмауер відсутній у безкоштовній версії.
– Деякі функції доступні лише в Windows.
– Досить великий вплив на продуктивність ПК.

Як тестував ці хмарні антивіруси

Вибір найкращих хмарних антивірусних рішень вимагає оцінки кожного з них за певними критеріями. Щоб надати вам продукцію найвищого класу, ми використали наступні:

– Оцінки виявлення шкідливих програм. Ми перевірили, наскільки добре хмарний антивірус може захищати від вірусів та інших кіберзагроз. Для цього ми використали звіти надійних і незалежних лабораторій тестування, таких як AV-

Test і AV-Comparatives.

– Вплив на продуктивність ПК. Перевірка всіх файлів на комп'ютері на наявність зловмисного програмного забезпечення може зайняти багато потужності ЦП і мати значний вплив на його продуктивність. З цієї причини ми також оцінили показник впливу та перевірили, які постачальники хмарних антивірусних програм пропонують легкі рішення.

– Сумісність. Усі типи операційних систем вразливі до атак. Так само ми вибрали продукти, сумісні з найпопулярнішими пристроями, включаючи Windows, MacOS і Linux. Таким чином ви можете отримати висококласний захист від зловмисного програмного забезпечення, не замислюючись, чи працює воно з вашою ОС.

– Доступність безкоштовної пробної версії. Крім того, чудово, якщо ви можете випробувати програмне забезпечення, перш ніж укладати довгостроковий план. Таким чином, ми перевірили, які хмарні антивіруси пропонують безкоштовні пробні версії або включають гарантії повернення грошей. Обидва варіанти дозволяють перевірити продукт без ризику.

– Ціна. Хоча багато хто може подумати, що вища ціна означає кращий захист, це не завжди так. Ми порівняли співвідношення ціни та якості доступних хмарних антивірусних рішень і вибраних продуктів, які пропонують кілька планів передплати, включно з доступними.

Чому варто використовувати хмарний антивірус?

Хмарні антивірусні рішення мають численні переваги. Ось причини, чому ви повинні використовувати один:

– Отримайте кращий захист від нових шкідливих програм. Оскільки хмарний антивірус не зберігає визначення шкідливих програм на пристрої, вони постійно оновлюються в реальному часі в хмарі. Це означає, що антивірус швидше отримує інформацію про останні кіберзагрози та може виявити їх майже миттєво.

– Насолоджуйтеся продуктивністю пристрою без змін під час

сканування. Процес перевірки комп'ютерних файлів на віруси також завершується в хмарі. Як наслідок, він потребує менше обчислювальної потужності вашого пристрою – отже, більше його доступно для щоденних завдань.

– Майте більше пам'яті пристрою. Незважаючи на те, що хмарну антивірусну програму потрібно інсталювати на пристрої, як і традиційну, вона займає менше місця. Отже, ви можете мати більше пам'яті пристрою для інших програм.

Як працює хмарне антивірусне програмне забезпечення?

Операції антивірусного програмного забезпечення в хмарі базуються на перенесенні процесу сканування з пристрою в хмару. Таким чином, він використовує менше ресурсів комп'ютера для процесу та може краще виявляти зловмисне програмне забезпечення в реальному часі.

Ось коротке пояснення того, як працює хмарний антивірус:

1. Виявлення. Антивірус виявляє підозрілий файл і надсилає його цифровий відбиток у хмару.

2. Експертиза. Дані файлу порівнюються з базою даних відомих шкідливих програм у режимі реального часу.

3. Оцінка. Якщо цифровий відбиток збігається з інформацією в базі даних, файл визначається як шкідливий; якщо відповідності не знайдено, то файл позначається як безпечний.

4. Повернення інформації. Інформація надсилається назад на пристрій із подальшими інструкціями щодо збереження файлу або негайного його видалення.

Традиційний чи хмарний антивірус – що краще?

Хмарний антивірус є кращим за традиційний, оскільки він краще виявляє найновіші шкідливі програми та менше впливає на продуктивність комп'ютера. Простіше кажучи, ви можете отримати висококласний захист від найновіших вірусів і водночас безперервно виконувати свою звичайну діяльність.

Крім того, антивіруси з хмарним захистом використовують штучний інтелект (AI) і машинне навчання (ML), щоб покращити виявлення шкідливих програм. Крім того, встановлення одного на вашому пристрої потребує менше місця для зберігання, тому у вас залишиться багато для інших програм.

Однак хмарний антивірус також має недолік. Наприклад, оскільки всі перевірені файли порівнюються з інформацією в хмарі, процес сканування може тривати довше, ніж звичайним антивірусом.

Чи варто використовувати безкоштовний хмарний антивірус?

Ні , ви не повинні використовувати безкоштовний хмарний антивірус, якщо це безкоштовна версія або безкоштовна пробна версія надійного преміум-постачальника. Це тому, що інструменти freemium ненадійні та можуть становити загрозу безпеці та конфіденційності.

Тим більше, що хмарні антивірусні рішення завантажують усі комп'ютерні файли для порівняння в хмару – неперевірені продукти можуть викрасти вашу конфіденційну інформацію та продати її третім особам .

Якщо ви хочете уникнути ризиків, але все одно користуватися безкоштовними послугами, ми рекомендуємо використовувати безкоштовні версії або пробні версії, які пропонують преміальні постачальники. Наприклад, TotalAV дозволяє насолоджуватися хмарним захистом без ризику з 30-денною гарантією повернення грошей.

Висновок

Найкращий хмарний антивірус, як-от TotalAV , є чудовим рішенням для надійного захисту від поширених і невідомих шкідливих програм . Це тому, що база даних вірусів, яка використовується для виявлення, постійно оновлюється в режимі реального часу, що полегшує ідентифікацію всіх типів загроз.

Крім того, використання антивіруса з хмарною технологією замість традиційного може допомогти уникнути негативного впливу на продуктивність комп'ютера . Це означає, що ви можете продовжувати свою звичайну діяльність без будь-яких перерв, поки триває сканування.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Python – динамічна інтерпретована об'єктно-орієнтована скриптова мова програмування із строгою динамічною типізацією. Офіційний сайт мови програмування Python https://www.python.org/. Python – багатоцільова мова програмування, яка дозволяє писати код, що добре читається. Відносний лаконізм мови Python дозволяє створити програму, яка буде набагато коротше свого аналога, написаного на іншій мові. Python – багатоплатформова мова програмування. Це означає, що програми на Python можна запускати в різних операційних системах без будь-яких змін.

Ще однією перевагою Python є його стандартна бібліотека, яка встановлюється разом з Python і містить готові інструменти для роботи з операційною системою, веб-сторінками, базами даних, різними форматами даних, для побудови графічного інтерфейсу програм тощо. Програми, написані на мові програмування Python, можуть бути як невеликими скриптами, так і складними системами. Python абсолютно безкоштовний.

Швидкість виконання коду Python

Один з можливих недоліків Python – швидкість виконання коду. Python не є компільованою мовою. Код на Python спочатку компілюється у внутрішній байт-код, який потім виконується інтерпретатором Python. У більшості випадків при використанні Python виходять програми повільніші в порівнянні з такими мовами, як C.

Втім, сучасні комп'ютери мають таку обчислювальну потужність, що для більшості застосунків швидкість розробки важливіша швидкості виконання, а програми на Python зазвичай пишуться набагато швидше.

Окрім того, Python легко розширюється модулями, написаними на C або C++. Такі модулі можуть використовуватися для виконання частин програми, що створюють інтенсивне навантаження на процесор.

Використання Python

Python використовується для різних цілей: для створення ігор і веб-застосунків, розробки внутрішніх інструментів для різноманітних проектів. Мова також широко застосовується в науковій області для досліджень і розв'язування прикладних завдань.

Застосування мови програмування Python:

1. BitTorrent – протокол для обміну даними.

2. Ubuntu Software Center – вільне програмне забезпечення для пошуку, установки і видалення пакунків в системі Ubuntu Linux.

3. Blender – програма для створення тривимірної комп'ютерної графіки, що включає засоби моделювання, анімації, вимальовування, пост-обробки відео, а також створення відеоігор.

4. GIMP – растровий графічний редактор, із підтримкою векторної графіки.

5. World of Tanks.

6. Вільна енциклопедія Вікіпедія.

7. Пошукова система Google.

8. DropBox – файловий хостинг, що включає персональне хмарне сховище, синхронізацію файлів і програму-клієнт.

9. YouTube – популярне відеосховище.

Версії Python

Мови програмування з часом змінюються – розробники додають в них нові можливості, а також виправляють помилки. Так з'являються різні версії мови. Наприклад, код написаний на Python 2 у більшості випадків не буде працювати у версії Python 3 без внесення додаткових змін.

Процесор є найважливішим компонентом в комп'ютері. Одна з основних функцій процесора – це обробка даних згідно комп'ютерної програми, яка є списком інструкцій, шляхом виконання арифметичних і логічних операцій над фрагментами даних.

Кожна інструкція в програмі – це команда, яка «повідомляє» процесору, яку операцію він повинен виконати. Процесор комп'ютера може розуміти лише ті інструкції, які написані на машинній мові. Машинна мова – це штучна мова, створена для передачі команд комп'ютеру. За допомогою машинної мови створюються ефективні програми, оскільки розробник отримує доступ до всіх можливостей процесора. Машинна мова – мова низького рівня.

Інструкція машинної мови існує для кожної операції, яку процесор здатний виконати – є інструкція для додавання чисел, є інструкція для віднімання чисел і т.д. Увесь набір інструкцій, який центральний процесор може виконати, відомий як набір інструкцій процесора.

Наприклад, у вас є певна програма, яка зберігається на диску вашого комп'ютера. Для виконання програми, ви здійснюєте подвійний клік на значку програми. Це змушує програму копіюватися з диска в оперативну пам'ять, після чого процесор комп'ютера виконує копію програми, яка знаходиться в оперативній пам'яті.

Коли процесор виконує інструкції програми, він бере участь у процесі, який є відомим як цикл fetch – decode – execute (отримати – декодувати – виконати). Цей цикл виконується для кожної інструкції у програмі і складається з трьох кроків:

Отримати

Програма – це послідовність інструкцій на машинній мові. Першим кроком циклу є завантаження (отримання) наступної інструкції з пам'яті в процесор.

Декодувати

Інструкція машинної мови – це двійкове число, яке представляє команду, що повідомляє процесору виконати певну операцію. На цьому кроці процесор декодує інструкцію, яку було «витягнуто» з пам'яті, для визначення того, яка операція повинна виконуватись.

Виконати

Останній крок циклу – виконати операцію.

Хоча процесор комп'ютера розуміє тільки машинну мову, людині непрактично писати програми на машинній мові. Така програма може мати тисячі або навіть мільйони бінарних інструкцій, і написання такої програми буде дуже обтяжливим процесом.

З цієї причини була створена мова асемблера як альтернатива машинній мові. Замість використання двійкових чисел для написання інструкцій, мова асемблера використовує короткі слова, відомі як мнемокоди.

Незважаючи на те, що мова асемблера не вимагає двійкових інструкцій, як у випадку машинної мови, проте вона вимагає високих знань про процесор. Використовуючи мову асемблера, навіть для найпростішої програми, необхідно написати велику кількість інструкцій.

Мова програмування високого рівня дозволяє створювати складні програми, не знаючи, як працює процесор, і не записуючи великої кількості інструкцій низького рівня. Крім того, більшість мов програмування високого рівня використовують слова, які легко зрозуміти.

Python – одна із популярних сучасних мов програмування високого рівня. Python – інтерпретована мова програмування. Python – це високорівнева інтерпретована мова програмування, на відміну від C++, яка є прикладом компільованої мови програмування. Назва Python відноситься як до мови програмування, так і до інтерпретатора – комп'ютерної програми, яка зчитує початковий код (написаний на Python) і виконує інструкції (команди).

Для перекладу мови високого рівня на машинну мову доступні два типи програм:

1. Компілятор.

2. Інтерпретатор.

Завантаження Python

Версії інтерпретатора Python для різних операційних систем доступні для безкоштовного завантаження за адресою https://www.python.org/downloads.

Середовище програмування для Python

Для написання програм використовують текстові редактори або інтегровані середовища розробки, які включають в себе різні інструменти для роботи з кодом: засіб для написання коду (текстовий редактор), інтерактивний інтерпретатор, відлагоджувач тощо.

Текстові редактори та інтегровані середовища програмування для Python:

– IDLE – стандартний редактор Python. Встановлюється разом з Python для користувачів Windows, окремим пакунком для користувачів Linux.

– Notepad++ – безкоштовний текстовий редактор початкового коду, який підтримує велику кількість мов, в тому числі і Python. Лише для користувачів Windows.

– Visual Studio Code – це легкий, але потужний редактор початкового коду, який розповсюджується безкоштовно і доступний у версіях для платформ Linux, Windows і macOS.

– PyScripter – інтегроване середовище розробки для мови програмування Python. Для користувачів Windows. Поширюється безкоштовно.

– Wing IDE 101 – вільне інтегроване середовище для Python, розроблене для навчання програмістів-початківців. Для користувачів Linux, Windows і macOS. Поширюється безкоштовно.

– Geany – вільний текстовий редактор з базовими елементами інтегрованого середовища розробки, доступний для операційних систем Linux, Windows і macOS.

– PyCharm – інтегроване середовище розробки для мови програмування Python. PyCharm є власницьким програмним забезпеченням. Наявна безкоштовна версія Community з усіченим набором можливостей. Для користувачів Linux, Windows і macOS.

– Thonny – IDE для вивчення програмування мовою Python. Для користувачів Linux, Windows і macOS.

– Mu – редактор коду Python для програмістів-початківців. Для користувачів Linux, Windows і macOS.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

У сучасних умовах динамічного розвитку інформаційно-телекомунікаційних технологій все більшу важливість здобувають програмні засоби контролю й керування інформаційними ресурсами. При цьому на практиці дуже часто виникає проблема несанкціонованого проникнення в програмне забезпечення ТКС шляхом впровадження ЗПО.

Проведені дослідження [2, 3] показали, що в цей час існує й постійно обновляється велика кількість подібного роду шкідливих програм, що роблять значний вплив на процес нормального функціонування ТКС. Аналіз злочинного програмного забезпечення дозволив представити загальну класифікацію програмних погроз і шкідливого програмного забезпечення.

Аналіз літератури [1-3], а так само проведені дослідження широкого спектра існуючих комп'ютерних вірусів показали, що найнебезпечнішими в цей час представляються віруси типу Flame. Їхньою відмінною рисою є наявність безлічі компонентів коду, функцією яких є виконання різноманітних шкідливих дій: розмноження в мережах різних типів з використанням різних протоколів, викрадення й знищення конфіденційної інформації, вивід з ладу комп'ютерних систем, шпигунство, взаємодія зі злочинним програмним забезпеченням іншого типу й т.д.

Дослідження комп'ютерних вірусів типу Flame дозволили виділити основні етапи технології їхнього поширення в ТКС і проілюструвати її структурно-логічну модель у вигляді діаграми переходів, представленої на рисунку 3.1.

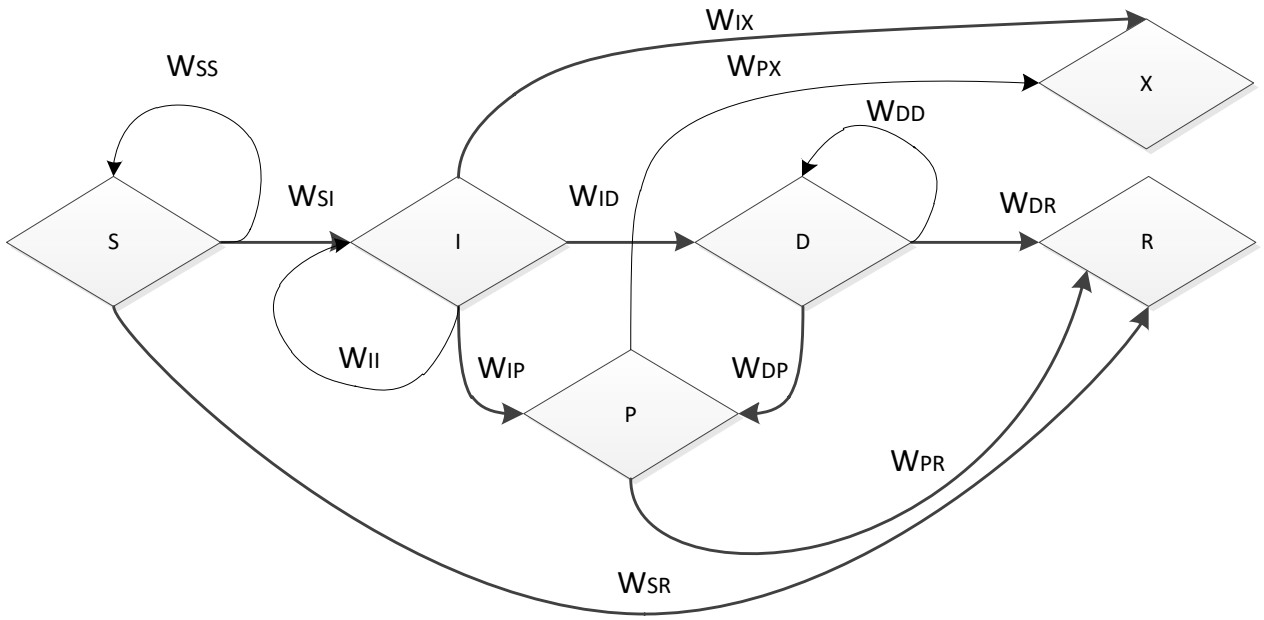

Рисунок 3.1 – Структурно-логічна GERT-модель технології поширення комп'ютерних вірусів типу Flame

Як видно із цього рисунка процес функціонування оконечного встаткування (вузлів) ТКС в умовах поширення досліджуваного типу комп'ютерних вірусів можна представити у вигляді сукупності станів:

– телекомунікаційні вузли заражені (I);

– телекомунікаційні вузли не заражені (S);

– телекомунікаційні вузли вилікувані й мають імунітет (R);

– заражені телекомунікаційні вузли с виявленим комп'ютерним вірусом (D);

– у результаті дій злочинного програмного забезпечення телекомунікаційні вузли повністю вийшли з ладу (X);

– у результаті дій злочинного програмного забезпечення телекомунікаційні вузли частково вийшли з ладу (P).

Представлена на рисунку 3.1. модель може бути описана в такий спосіб. Гілка S→I (W_{SI}) інтерпретує процес проникнення комп'ютерного вірусу в незаражений k-й комп'ютер з імовірністю $P_{зар}^{(k,s^{(i)})}$ в деякому вихідному стані

ТКС $s^{(i)}$. Гілка S→R (W_{SR}) описує процес імунізації незараженого k-го комп'ютера у вихідному стані з імовірністю імунізації $P_{им}^{(k,s^{(i)})}$. Гілка S→S (W_{SS}) інтерпретує процес нормального функціонування системи (без зараження) з імовірністю $\left(1-P_{зар}^{(k,s^{(i)})}-P_{им}^{(k,s^{(i)})}\right)$. Гілка I→D (W_{ID}) характеризує процес детектування зараженого k-го комп'ютера в деякому вихідному стані ТКС $s^{(i)}$ з імовірністю $P_{дет}^{(k,s^{(i)})}$. Гілка I→X (W_{IX}) описує процес повного виведення з ладу k-ого вузла з імовірністю $P_{вис}^{(k,s^{(i)})}$. Гілка I→P (W_{IP}) інтерпретує процес часткового виведення k-ого вузла з ладу з імовірністю $P_{чвис}^{(k,s^{(i)})}$. Гілка I→I (W_{II}) передбачає можливість ситуації, коли вузол телекомунікації залишиться зараженим з імовірністю $\left(1-P_{дет}^{(k,s^{(i)})}-P_{вис}^{(k,s^{(i)})}-P_{чвис}^{(k,s^{(i)})}\right)$. Гілка D→R (W_{DR}) описує процес лікування зараженого k-го комп'ютера в деякому стані ТКС $s^{(i)}$ з імовірністю $P_{леч}^{(k,s^{(i)})}$. Гілка D→P характеризує процес часткового виходу з ладу встаткування після того як злочинне програмне забезпечення типу Flame виявлено. При цьому ймовірність такого переходу дорівнює $P_{чв}^{(k,s^{(i)})}$. Гілка D→D (W_{DD}) передбачає можливість ситуації, коли телекомунікаційний вузол виявить злочинне програмне забезпечення типу Flame, але при цьому процесу імунізації не відбудеться (виявлення за допомогою тільки евристичного аналізатора). При цьому ймовірність такого переходу стану дорівнює $\left(1-P_{леч}^{(k,s^{(i)})}-P_{чв}^{(k,s^{(i)})}\right)$. Гілка P→R (W_{PR}) інтерпретує процес відновлення нормального працездатного стану й імунізації встаткування після ситуації часткового виходу з ладу. Імовірність такої події в деякому стані ТКС $s^{(i)}$ дорівнює $P_{лечв}^{(k,s^{(i)})}$. Гілка P→X

(W_{PX}) описує процес повного виходу з ладу зараженого телекомунікаційного встаткування з імовірністю $\left(1 - P_{лечв}^{(k,s^{(i)})}\right)$.

$$
\begin{array}{c|cccccc}
 & S & I & D & P & R & X \\
S & \left(\begin{array}{l}1 - \\ - P_{зар}^{(k,s^{(i)})} - \\ - P_{им}^{(k,s^{(i)})}\end{array}\right) & P_{зар}^{(k,s^{(i)})} & 0 & 0 & P_{им}^{(k,s^{(i)})} & 0 \\
I & 0 & \left(\begin{array}{l}1 - \\ - P_{дет}^{(k,s^{(i)})} - \\ - P_{вис}^{(k,s^{(i)})} - \\ - P_{чвис}^{(k,s^{(i)})}\end{array}\right) & P_{дет}^{(k,s^{(i)})} & P_{чвис}^{(k,s^{(i)})} & 0 & P_{вис}^{(k,s^{(i)})} \\
D & 0 & 0 & \left(\begin{array}{l}1 - \\ - P_{леч}^{(k,s^{(i)})} - \\ - P_{чв}^{(k,s^{(i)})}\end{array}\right) & P_{чв}^{(k,s^{(i)})} & P_{леч}^{(k,s^{(i)})} & 0 \\
P & 0 & 0 & 0 & 0 & P_{лечв}^{(k,s^{(i)})} & \left(\begin{array}{l}1 - \\ - P_{лечв}^{(k,s^{(i)})}\end{array}\right) \\
R & 0 & 0 & 0 & 0 & 1 & 0 \\
X & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
$$

Рисунок 3.2 – Матриця ймовірностей переходів між станами відповідно до GERT-моделі технології поширення комп'ютерних вірусів типу Flame

Варто помітити, що для спрощення розроблювальної GERT-моделі ухвалено рішення не розглядати інші переходи у зв'язку з поставленими при моделюванні обмеженнями («стан «виведений з ладу» – кінцевий стан об'єкта», «імунізований вузол кінцева крапка моделі», «незаражений вузол не може бути виведений з ладу», «процес лікування й імунізації без попереднього детектування не передбачений»), при цьому передбачається, що ймовірність переходів R→R і

X→X дорівнює одиниці, а ймовірності переходів, що залишилися, дорівнюють нулю.

Зазначені вище допущення й обмеження дозволили сформувати матрицю ймовірностей переходів з одного стану в інше й представити її у вигляді рисунок 3.2.

Надалі даний підхід дозволить сформулювати й представити ряд допущень, обмежень і вхідних даних для розробки імітаційної моделі технології поширення комп'ютерних вірусів у ТКС.

Аналіз ряду робіт [1, 2, 4, 6], а також проведені дослідження процесу передачі даних у ТКС дозволили сформувати характеристики розглянутих в GERT-моделі гілок і параметри розподілу й представити їх у табл. 3.1.

Відповідно до характеристик гілок GERT-мережі еквівалентну W-функцію часу поширення в ТКС комп'ютерних вірусів типу Flame з кінцевим результатом лікування й імунізації вузлів ТКС можна представити як:

$$W_E(s) = \frac{W_{SI}W_{ID}W_{DR} + W_{SI}W_{IP}W_{PR} + W_{SI}W_{ID}W_{DP}W_{PR} + W_{SR}}{1 - W_{SS} - W_{SI}W_{II} - W_{SI}W_{ID}W_{DD}} =$$

$$= \frac{\left(p_1(\lambda_2/\lambda_2 - s)(\lambda_8/\lambda_8 - s)\begin{pmatrix} p_4 p_8(\lambda_4/\lambda_4 - s) + p_5 p_9(\lambda_5/\lambda_5 - s) + \\ + p_4 p_7 p_9(\lambda_4/\lambda_4 - s)(\lambda_5/\lambda_5 - s) \end{pmatrix} \right)}{\left(\begin{pmatrix} 1 - (1 - p_1 - p_2)(\lambda_1/\lambda_1 - s) - p_1(\lambda_2/\lambda_2 - s) \times \\ \times \begin{pmatrix} (1 - p_4 - p_5 - p_6)(\lambda_7/\lambda_7 - s) + \\ + (p_4(1 - p_7 - p_8)(\lambda_4/\lambda_4 - s)(\lambda_9/\lambda_9 - s)) \end{pmatrix} \end{pmatrix} \right)}. \quad (3.1)$$

Еквівалентну W-функцію часу поширення в ТКС комп'ютерних вірусів типу Flame з кінцевим результатом виходу вузлів ТКС із ладу можна представити як:

$$W_E(s) = \frac{W_{SI}W_{IX} + W_{SI}W_{IP}W_{PX} + W_{SI}W_{ID}W_{DP}W_{PX}}{1 - W_{SS} - W_{SI}W_{II} - W_{SI}W_{ID}W_{DD}} =$$

$$= \frac{p_1(\lambda_2/\lambda_2 - s)\begin{pmatrix} p_6(\lambda_6/\lambda_6 - s) + \\ +(1-p_9)(\lambda_5/\lambda_5 - s)(\lambda_6/\lambda_6 - s)(p_5 + p_4(\lambda_4/\lambda_4 - s)) \end{pmatrix}}{1 - (1-p_1-p_2)(\lambda_1/\lambda_1 - s) - p_1(\lambda_2/\lambda_2 - s)\times} \quad (3.2$$

$$\times\begin{pmatrix} (1-p_4-p_5-p_6)(\lambda_7/\lambda_7 - s) + \\ +(p_4(1-p_7-p_8)(\lambda_4/\lambda_4 - s)(\lambda_9/\lambda_9 - s)) \end{pmatrix}$$

Таблиця 3.1 – Характеристики гілок моделі

№ п/п	Гілка	W-функція	Імовірність	Виробляюча функція моментів
1.	(S,S)	W_{SS}	1-1- P_1- P_2	$\lambda_1/(\lambda_1 - s)$
3.	(S,I)	W_{SI}	P_1	$\lambda_2/(\lambda_2 - s)$
3.	(S,R)	W_{SR}	P_2	$\lambda_3/(\lambda_3 - s)$
4.	(I,D)	W_{ID}	P_4	$\lambda_4/(\lambda_4 - s)$
5.	(I,P)	W_{IP}	P_5	$\lambda_5/(\lambda_5 - s)$
6.	(I,X)	W_{IX}	P_6	$\lambda_6/(\lambda_6 - s)$
7.	(I,I)	W_{II}	1-1- P_4 - P_5 - P_6	$\lambda_7/(\lambda_7 - s)$
8.	(D,P)	W_{DP}	P_7	$\lambda_5/(\lambda_5 - s)$
9.	(D,R)	W_{DR}	P_8	$\lambda_8/(\lambda_8 - s)$
10.	(D,D)	W_{DD}	1-1- P_7 - P_8	$\lambda_9/(\lambda_9 - s)$
11.	(P,R)	W_{PR}	P_9	$\lambda_8/(\lambda_8 - s)$
13.	(P,X)	W_{PX}	1-1- P_9	$\lambda_6/(\lambda_6 - s)$

Варто помітити, що розглянута на рисунку 3.1 GERT-мережа, описує технологію поширення злочинного програмного забезпечення на високому рівні стратифікації [8]. У цьому зв'язку еквівалентна W-функція часу поширення в ТКС

комп'ютерних вірусів досліджуваного типу представляється з однієї сторони досить складної для аналізу, а з іншої сторони у зв'язку з уведеними обмеженнями моделювання (обрана однотипна виробляюча функція моментів, гілки W_{IX}, W_{PX}, так само як і гілки W_{DR}, W_{PR}, а також W_{DP}, W_{IP} характеризуються тим самим параметром розподілу й ін.) точність результатів не відповідає висунутим вимогам.

Аналіз ряду робіт [3, 6] показав, що в цій ситуації для усунення зазначених недоліків доцільно звузити область визначення розроблювальної математичної моделі шляхом декомпозиції й еквівалентних перетворень, що спрощують.

Еквівалентні перетворення, що спрощують, GERT-моделі технології поширення комп'ютерних вірусів

Проведені дослідження показали, що використання найбільш відомого підходу декомпозиції загального завдання математичного моделювання на ряд приватних завдань у більшості випадків дозволяє вирішити поставлені завдання із заданою точністю. Однак у деяких випадках просте використання методу декомпозиції не дозволяє досягти необхідного рівня обчислювальної складності. Тому в дисертаційній роботі для рішення завдання математичного моделювання технології поширення комп'ютерних вірусів типу Flame за допомогою GERT-мереж пропонується комплексне використання підходів декомпозиції й еквівалентних перетворень, що спрощують.

Для цього можна скористатися методикою, представленої в роботах [3, 5, 6]. Відповідно до цієї методики першим кроком математичного GERT-моделювання є декомпозиція розглянутого об'єкта на страти й вибір з них найбільш проблемної області дослідження.

Проведені дослідження показали, що процес стратифікації GERT-моделі обраної проблемної області є багато в чому суб'єктивним, і тільки розроблювач, виходячи зі свого розуміння мети моделювання, може визначити рівні деталізації, число й взаємозв'язки елементів. Для процесу функціонування ТКС у ході

виявлення, лікування й імунізації комп'ютерних вірусів типу Flame рекомендується визначити наступна розбивка на страти.

1) K-страта. Низькорівневі алгоритми обробки даних: алгоритми формування сигнатур (хешування й ін.); алгоритми сигнатурного й евристичного аналізу (за допомогою мереж Маркова, нечіткої логіки й ін.); алгоритми скремблювання й завадостійкого кодування (додання потоку бітової інформації властивостей псевдовипадкової послідовності); алгоритми перевірки цілісності інформації й ін.

2) (K+1)-страта. Механізми забезпечення необхідного рівня якості обслуговування (QoS).

3) (K+2)-страта. Низькорівневе моделювання каналів зв'язку ТКС шляхом завдання їхніх типів, і параметрів необхідних для виконання поставленого завдання (наприклад, пропускної здатності, часу затримки поширення й ін.) [2, 6].

4) (K+3)-страта. Моделювання каналів зв'язку середнього рівня (з урахуванням специфіки розбивки даних на інформаційні пакети й кадри). При цьому параметрами, необхідними для виконання поставленого завдання можуть бути мінімальний і максимальний розмір інформаційного пакета (кадру), імовірності помилки в інформаційному пакеті (кадрі), час доставки інформаційного пакета й ін. [6].

5) (K+4)-страта. Моделювання протоколів і засобів передачі даних виконуючі транспортні функції й функції доставки повідомлень між кінцевими вузлами (наприклад, TCP або UDP [5]), комунікаційних протоколів транспортного рівня, протоколів маршрутизації рівня доступу (наприклад, OSPF, RIP, BGP і ін. [5]).

6) (K+5)-страта. Моделювання додатків за допомогою команд декількох типів, у тому числі команд обробки даних, відправлення й читання повідомлень, читання й записи даних у файл, установлення сесій і припинення програми до одержання повідомлень. Для кожного додатка задається так званий репертуар команд [5].

7) (K +6)-страта. Моделювання інформаційного трафіку різних мережних служб із урахуванням характерних особливостей, основних показників і законів розподілу шуканих випадкових величин.

8) (K +7)-страта. Моделювання мереж на самому верхньому рівні абстракції. Укрупнене моделювання мереж з комутацією пакетів (наприклад, TCP/IP [5]), мереж з комутацією осередків (наприклад, ATM [5]), мереж з комутацією міток (наприклад, MPLS [1]) і ін. При цьому кожний тип глобального сервісу характеризується рядом специфічних параметрів, таких як мінімальний і максимальний розмір кадру, і необхідні ресурси для передачі службової інформації й ін.

Аналіз процесів протікають у ТКС показав, що в умовах зовнішніх деструктивних впливів комп'ютерних вірусів типу Flame одними йз основних етапів, що дозволяють досягти стану R (див. рисунок 3.2) (телекомунікаційні вузли вилікувані й мають імунітет) є етапи виявлення, лікування й імунізації злочинного програмного забезпечення. Проведені дослідження показали, що математичну формалізацію зазначеного процесу доцільно представити на (K +4) рівні стратифікації.

Особливо важливим даний рівень стратифікації представляється при математичному описі процесів виявлення, лікування й імунізації за допомогою хмарних антивірусних систем, оскільки саме на цьому рівні можливий облік основних протоколів передачі даних транспортного (TCP, UDP) і мережного (OSPF, RIP, BGP і ін.) рівнів.

Процедура еквівалентного перетворення, що спрощує, GERT-мережі до еквівалентної дуги представляється набором елементарних операцій перетворення, у результаті яких можна одержати еквівалентні характеристичні функції. Загальна методика еквівалентного перетворення, що спрощує, GERT-мережі а також основні математичні викладення, що характеризують дану методику представлені в роботах [3, 6].

3.2 Розробка структурної схеми

Процес інформаційного обміну робочих станцій з вузлами, що надають послуги «хмарної» антивірусного захисту, являє собою чітко організовану функціональну структуру, що є сукупністю алгоритмів формування сигнатур, транспортування, комутації, маршрутизації й обробки спеціалізованими аналізаторами. Структурна схема системи представлена на рисунку 3.3.

Проведені дослідження основних підходів математичного моделювання показали, що найбільш зручною, наочною й багатобічною формою опису технології передачі метаданих в «хмарні» антивірусні системи є граф алгоритмів на основі GERT-мережі.

Для розглянутого в статті приклада під графом алгоритмів розуміється орграф $G=(X,U)$ вершини x_i якого відображають приватні реалізації i-х алгоритмів системи. Вершинам графа приписується вага, що відповідає часу реалізації алгоритму. (В окремих випадках це може бути ймовірність показання на той або інший вихід вузлів графа, що вимагається для виконання пам'ять, помилки визначення тих або інших величин, пов'язаних з реалізацією алгоритму й т.д.). Приватні реалізації алгоритмів у розглянутому графі GERT-мережі ототожнюється дугами графа з певними умовними ймовірностями й виробляючими функціями моментів гілки.

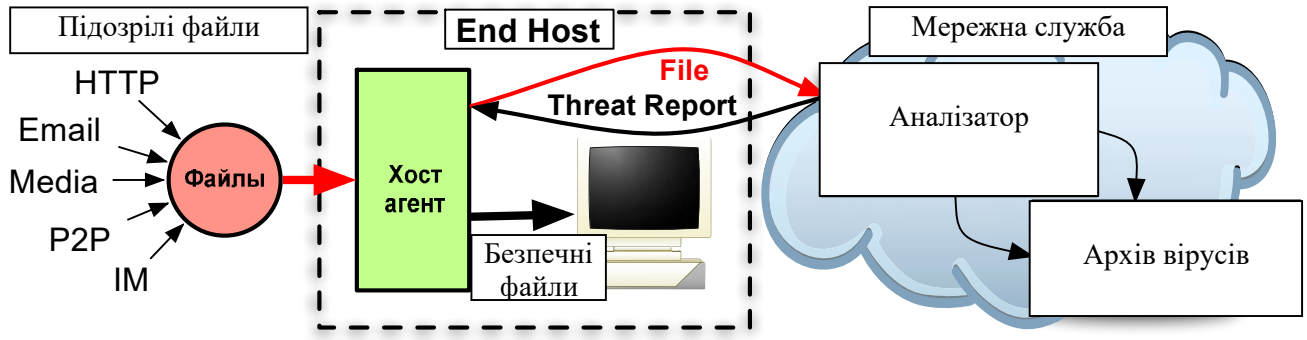

Рисунок 3.3 – Структурна схема системи

Скористаємося представленими на рисунку 3.3 даними для розробки GERT-моделі ТКС у процесі передачі метаданих в «хмарні» антивірусні системи.

Типова модель алгоритмів формування й передачі метаданих в «хмарні» антивірусні системи відповідно до (K +4) рівнем стратифікації представлена на рисунку 3.6.

Ця модель може бути описана в такий спосіб.

Гілка (1,2) інтерпретує час формування метаданих (сигнатур). Гілка (2,3) задає час передачі кадру (пакета) метаданих від передавача до трансляторів (маршрутизаторам).

Гілка (3,3) відображає можливий несправний стан комутаційного встаткування, що транслює (маршрутизаторів) на обраних маршрутах.

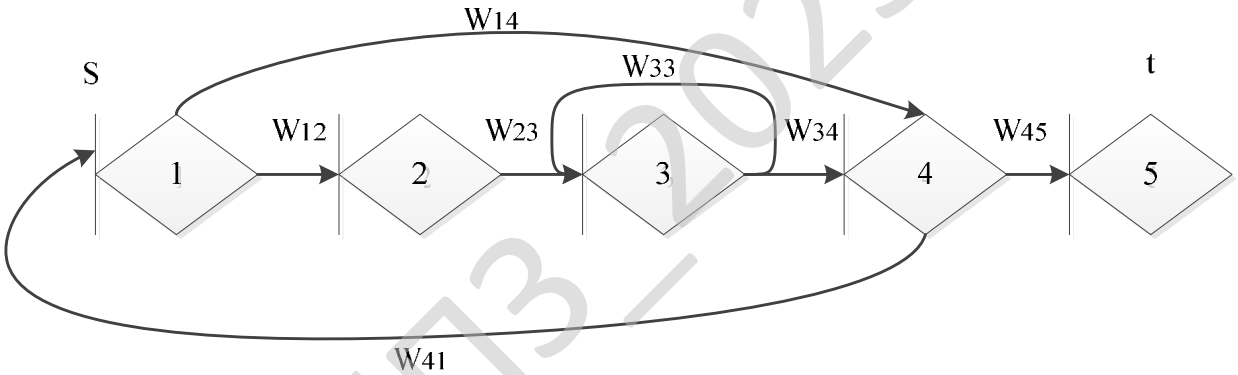

Рисунок 3.4 – Модель алгоритмів формування й передачі метаданих в «хмарні» антивірусні системи

Гілка (3,4) описує час комутації кадру (пакета) у телекомунікаційному встаткуванні.

Гілки (4,1) і (4,5) задають випадковий час передачі квитанції про правильність (помилці) доставки кадрів (пакетів) відповідно до протоколу транспортного рівня (TCP).

Вузол 5 відбиває стан системи в момент аналізу метаданих на предмет наявності комп'ютерних вірусів.

У ряді практичних випадків з метою підвищення ймовірності виявлення комп'ютерних вірусів існує необхідність антивірусного аналізу не сформованих на прикінцевому устаткуванні сигнатур, а даних, збережених на цьому встаткуванні в повному обсязі. Цієї ситуації відповідає гілка (1,4).

Гілки (3,4), (4,1) і (4,5) доцільно описувати ідентичними параметрами розподілу, тому що вони задають схожі операції передачі даних невеликого обсягу.

Аналіз ряду робіт [1, 3, 6], а також проведені дослідження процесу передачі даних у мультисервісних телекомунікаційних мережах дозволили сформувати характеристики гілок і параметри розподілу у вигляді, представленому в табл. 3.2.

Аналіз даних, представлених у табл. 3.3. показав високу структурну складність розроблювальної GERT-мережі. Особливо гостро дана проблема фіксується на ділянці, сформованій з вузлів 2-3-4 (гілки (2,3), (3,3)).

Таблиця 3.2 – Характеристики гілок моделі

№ п/п	Гілка	W-функція	Імовірність	Виробляюча функція моментів
1	(1,2)	W_{12}	p_1	$\lambda_1 /(\lambda_1 - s)$
2	(1,4)	W_{14}	$1\text{-}p_1$	$\lambda_2 /(\lambda_2 - s)$
3	(2,3)	W_{23}	p_2	$\lambda_3 /(\lambda_3 - s)$
4	(3,3)	W_{33}	p_3	$\lambda_4 /(\lambda_4 - s)$
5	(3,4)	W_{34}	1-1- p3	$\lambda_5 /(\lambda_5 - s)$
6	(4,5)	W_{45}	p_4	$\lambda_5 /(\lambda_5 - s)$
7	(4,1)	W_{41}	1-1- p_4	$\lambda_5 /(\lambda_5 - s)$

З метою спрощення розглянутої на рисунку 3.6 моделі скористаємося методикою еквівалентних перетворень, що спрощують(K +4) рівні стратифікації.

У результаті перетворень, що спрощують, сформуємо GERT-мережу, представлену на рисунку 3.7. Як видно із цього рисунка в результаті перетворень, що спрощують, гілки (2,3) і (3,3) минулого замінені на еквівалентну гілка.

Оновлені дані характеристик гілок мережі представлені в табл. 3.3.

Відповідно до характеристик гілок GERT-мережі визначимо еквівалентну W-функцію часу передачі файлу як:

$$W_E(s) = \frac{W_{13}W_{34} + W_{12}W_{23}W_{34}}{1 - W_{13}W_{31} - W_{12}W_{23}W_{31}} =$$

$$= \frac{\left(\frac{p_4\lambda_5 q_1\lambda_1(\lambda_1 - s)(\lambda_3 - s)((\lambda_4 - s) - p_3\lambda_4) + p_1\lambda_1 p_4\lambda_5 p_2\lambda_3(\lambda_2 - s)}{(\lambda_2 - s)(\lambda_5 - s)(\lambda_3 - s)((\lambda_4 - s) - p_3\lambda_4)} \right)}{\left(\begin{array}{l} \left(\begin{array}{l} (\lambda_1 - s)(\lambda_2 - s)(\lambda_5 - s)(\lambda_3 - s)((\lambda_4 - s) - p_3\lambda_4) - \\ - q_1\lambda_2 q_4\lambda_5(\lambda_1 - s)(\lambda_3 - s)((\lambda_4 - s) - p_3\lambda_4) - p_1\lambda_1 q_4\lambda_5 p_2\lambda_3(\lambda_2 - s) \end{array} \right) \times \\ \times (1/(\lambda_2 - s)(\lambda_1 - s)(\lambda_5 - s)(\lambda_3 - s)((\lambda_4 - s) - p_3\lambda_4)) \end{array} \right)},$$

де $1 - p_1 = q_1$, $1 - p_2 = q_2$, $1 - p_3 = q_3$, $1 - p_4 = q_4$.

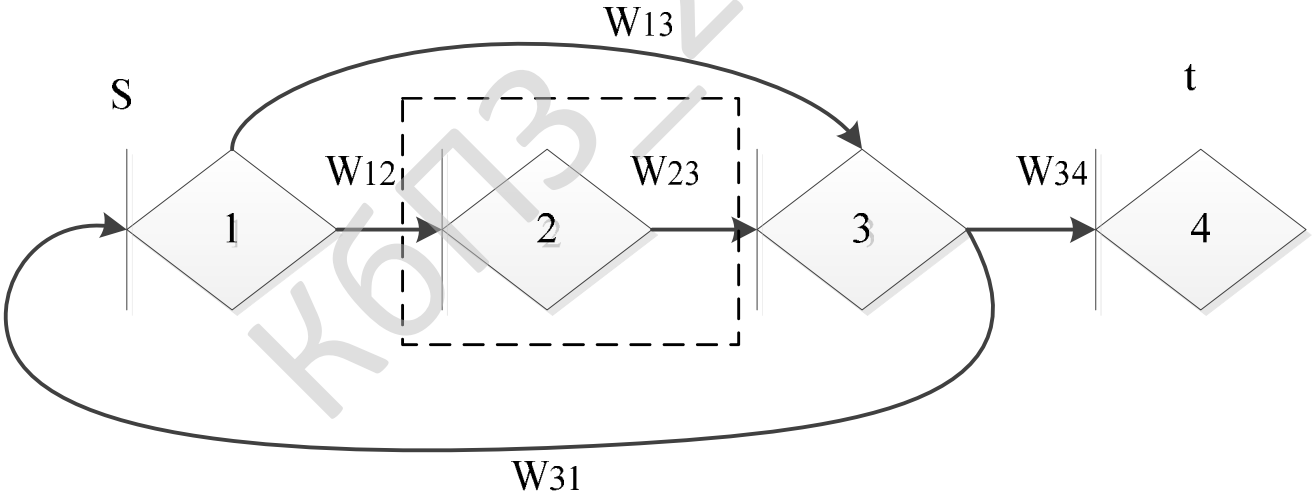

Рисунок 3.5 – Спрощена модель алгоритмів формування й передачі метаданих в «хмарні» антивірусні системи

Таблиця 3.3 – Характеристики гілок моделі

№ п/п	Гілка	W-функція	Параметр розподілу
1	(1,2)	W_{12}	$p_1\lambda_1 /(\lambda_1 - s)$
2	(1,3)	W_{13}	$(1- p_1)\lambda_2 /(\lambda_2 - s)$
3	(2,3)	W_{23}	$p_2\lambda_3/(\lambda_3 - s)((\lambda_4 - s) - p_3\lambda_4)$
4	(3,4)	W_{34}	$p_4\lambda_5 /(\lambda_5 - s)$
5	(3,1)	W_{31}	$(1 - p_4)\lambda_5 /(\lambda_5 - s)$

Проведені дослідження показали, що в складних GERT-мережах з можливими циклами відсутні прості методи знаходження особливих крапок функції $\Phi_E(z)$ заміни дійсних змінних $(z = -i\varsigma)$, де ς – дійсна змінна. Зв'язано це з тим, що для знаходження особливих крапок необхідно вирішувати нелінійні рівняння, і чим складніше структура GERT-мережі, тим складніше й вихідне рівняння [6]. Тому в ході моделювання виконуючи комплексне перетворення одержимо:

$$\Phi(z) = \frac{uz^3 - kz^2 + wz + h}{\left(z^3 + vz^2 + rz + c\right)}, \tag{3.3}$$

де:

$$u = p_4\lambda_5 q_1\lambda_2,$$

$$k = p_4\lambda_4 q_1\lambda_2(p_3\lambda_4 - \lambda_3 - \lambda_1 - \lambda_4),$$

$$w = p_4\lambda_5 q_1\lambda_2 \cdot (p_3\lambda_3\lambda_4 - \lambda_1\lambda_3 - \lambda_3\lambda_4 - \lambda_1\lambda_4 + p_3\lambda_1\lambda_4),$$

$$h = p_4\lambda_4 q_1\lambda_1\lambda_2\lambda_3\lambda_4 q_3,$$

$$v = \lambda_3 - \lambda_4 - \lambda_2 + q_1q_4\lambda_2\lambda_5 + p_3\lambda_4,$$

$$r = \lambda_3\lambda_4 + \lambda_3\lambda_2 - q_1\lambda_2 q_4\lambda_5\lambda_3 - p_3\lambda_3\lambda_4 + \lambda_2\lambda_4 - q_1\lambda_2 q_4\lambda_5\lambda_4 - p_3\lambda_2\lambda_4 + q_1\lambda_2 q_4\lambda_5 p_3\lambda_4,$$

$$c = \lambda_3\lambda_4\lambda_2 - q_1\lambda_2 q_4\lambda_3\lambda_4\lambda_5 - p_3\lambda_3\lambda_4\lambda_2 + q_1\lambda_2 q_4\lambda_3\lambda_4 p_3.$$

З вираження (3.3) видно, що функція $\Phi(z)$ має тільки прості полюси обумовлені коріннями рівняння $z^3 + vz^2 + rz + c = 0$. У цьому випадку щільність розподілу ймовірностей часу передачі повідомлення дорівнює:

$$\varphi(x) = \frac{1}{2\pi i} \int_{-i\infty}^{i\infty} e^{zx} \frac{uz^3 - kz^2 + wz + h}{\left(z^3 + vz^2 + rz + c\right)} dz. \qquad (3.4)$$

Використовуючи спеціалізований математичний пакет Mathcad, визначимо прості полюси z функції $\Phi(z)$ й знайдемо щільність розподілу ймовірностей $\varphi(x)$ часу передачі метаданих в «хмарні» антивірусні системи. При цьому в якості початкових даних визначимо наступні параметри гілок GERT-мережі:

$$p_1 = 0{,}9,\ p_2 = 0{,}99999,\ p_3 = 0{,}99999,\ p_4 = 0{,}99999,$$

$$\lambda_1 = 1,\ \lambda_2 = 0{,}099,\ \lambda_3 = 0{,}9,\ \lambda_4 = 0{,}5,\ \lambda_5 = 0{,}4.$$

Для зазначеного приклада функція $\Phi(z)$ має прості полюси:

$$z := \begin{pmatrix} -0.67 \\ 0 \\ -0.117 \end{pmatrix}.$$

Відповідно до формули (3.4) $\varphi(x)$ дорівнює:

$$\frac{1}{z} 0{,}159155 \times 2{,}71828^{(-0{,}0835025 - 0{,}364433i)z} \times$$
$$\times (z 2{,}71828^{(0/920508 + 0{,}364433i)z}\, Ei((x - 0{,}837005)z) \times$$
$$\times (1/02026h - 0{,}714769k + 0{,}85396v + 0{,}598265) +$$
$$+ 2{,}71828^{(0{,}728866i)z}\, zEi((x + (0{,}0835025 - 0{,}364433i))z) \times$$
$$\times (-(0{,}142616 - 0{,}131097i)k - (0{,}42698 + 0{,}293502i)v +$$
$$+ (0{,}0358675 + 0{,}0629208i) + (-0{,}510128 + 1{,}28851i)h) -$$
$$- (0{,}510128 + 1{,}28851i)hzEi((x + (0{,}0835025 + 0{,}364433i))z) -$$
$$- (0{,}142616 + 0{,}131097i)kzEi((x + (0{,}0835025 + 0{,}364433i))z) -$$
$$- (0{,}42698 - 0{,}293502i)vzEi((x + (0{,}0835025 + 0{,}364433i))z) +$$
$$+ (0{,}0358675 + 0{,}0629208i)zEi((x + (0{,}0835025 + 0{,}364433i))z) +$$
$$+ 2{,}71828^{z(x + (0{,}0835025 + 0{,}364433i))}) + const$$

На рисунку 3.6. представлений графік щільності розподілу часу передачі метаданих.

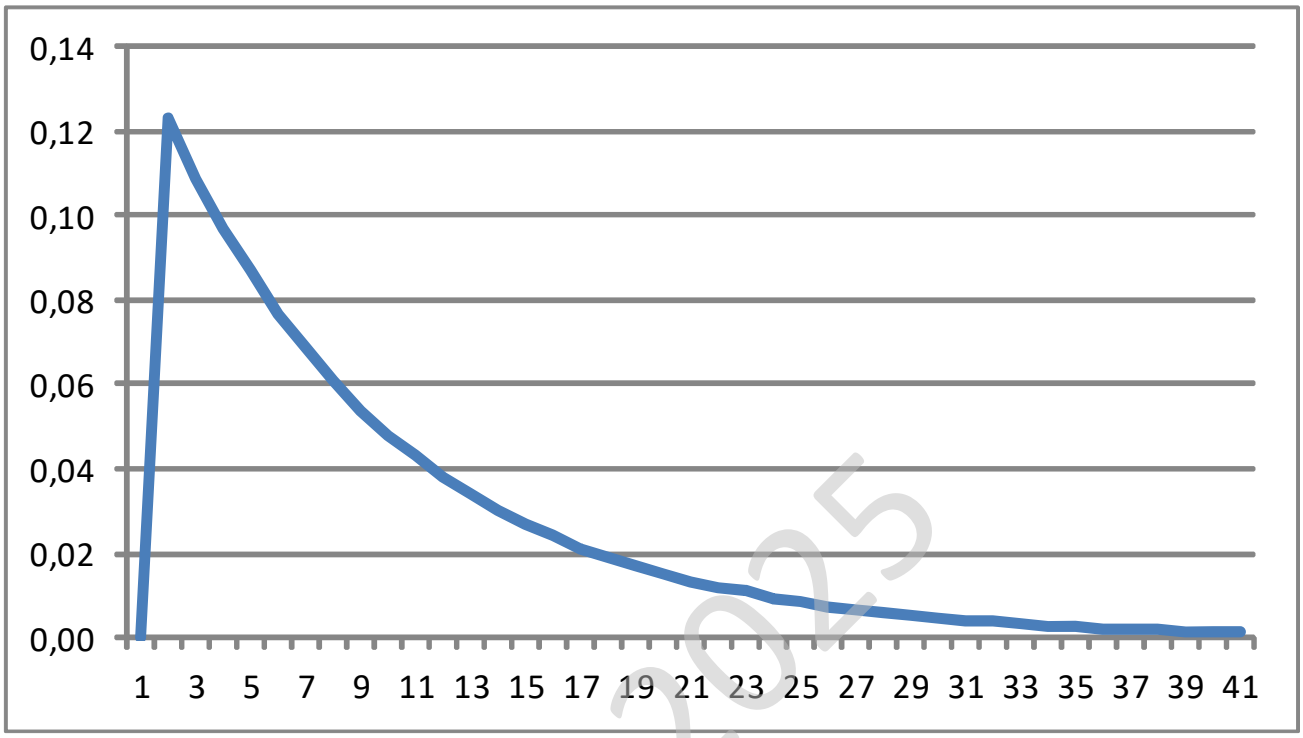

Рисунок 3.6 – Щільність розподілу часу передачі метаданих в «хмарні» антивірусні системи

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.7. Так, як функціональна схема є більш подобним описом функціональних можливостей структурної схеми, то вона буде представляти собою, більш детальний варіант структурної схеми.

З рисунку видно, що розроблена система складається з наступних частин:

– Блок виявлення розповсюдження шкідливого програмного забезпечення.

– Блок визначення деструктивних дій та виду атаки.

– Блок зберігання результатів.

– Блок моніторингу мережі.

– Блок аналізу мережної статистики.

– Блок визначення топології мережі.

Блок виявлення розповсюдження шкідливого програмного забезпечення

Блок виявлення розповсюдження шкідливого програмного забезпечення:

– Блок визначення профілю поводження нормального трафіку.

– Блок заміни направлення трафіку.

– Блок усунення розповсюдження шкідливого програмного забезпечення.

При первісному розгортанні рішення по DDoS адміністратор створює профіль поводження нормального трафіку. Цей процес іменується навчанням. Компанія використовує додатки звичайним образом протягом 24 годин протягом одного тижня, і трафік додатка проходить через Детектор аномалій трафіку. У період навчання Детектор аномалій трафіку збирає базову інформацію для розуміння нормальної роботи мережі, куди входять:

– Інтенсивність пакетів для кожного типу пакетів, обмірювана як кількість пакетів у секунду (pps).

– Співвідношення пакетів, наприклад, співвідношення пакетів SYN і пакетів FIN.

– Кількість одночасних TCP-з'єднань, відкритих одним джерелом.

Базова інформація збирається по кожній цільовій адресі хост-ПК, цільовій підмережі, вихідній адресі хост-ПК і вихідній підмережі.

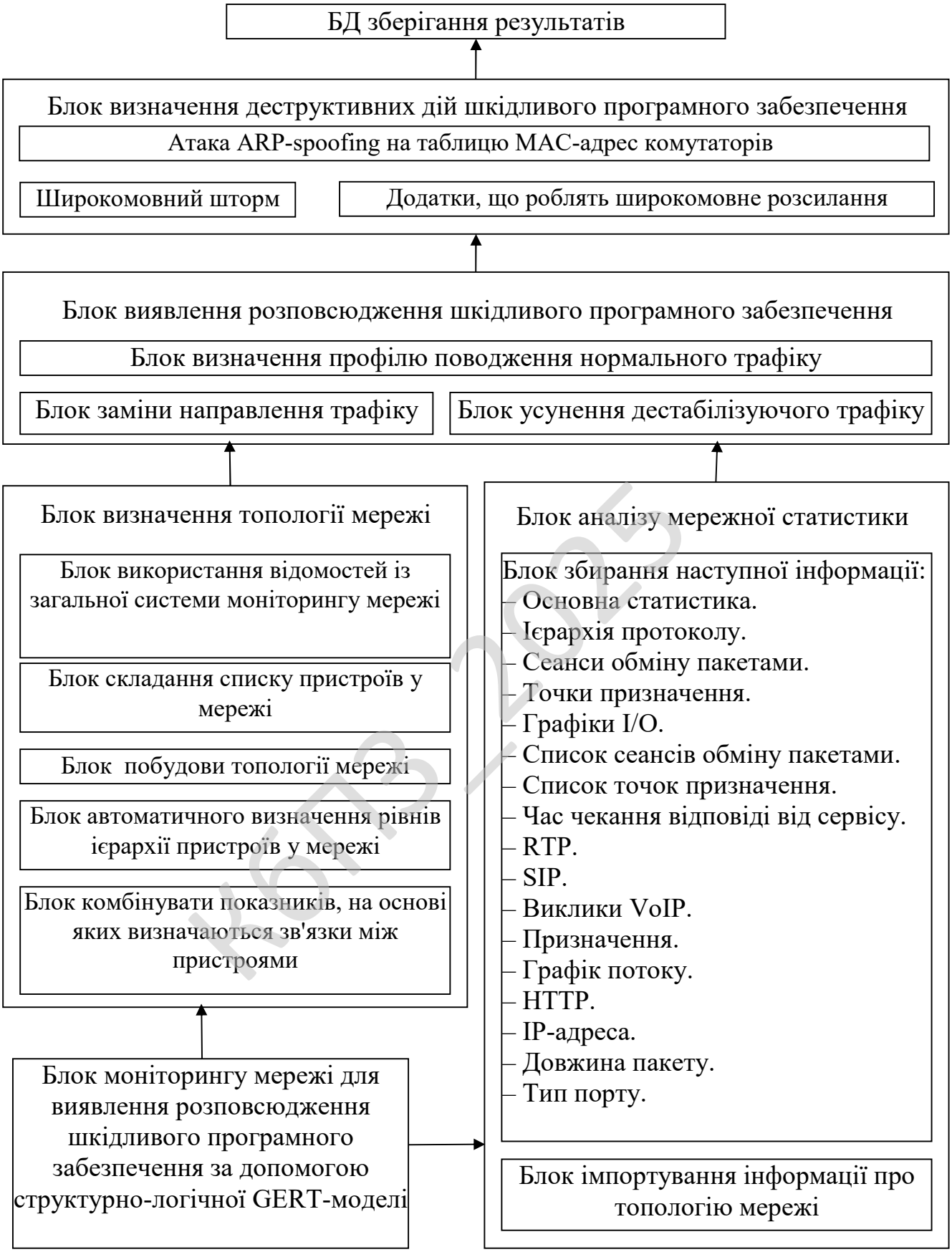

Рисунок 3.7 – Функціональна схема системи

Після закінчення періоду навчання Детектор аномалій трафіку переводиться в режим моніторингу, а Блок усунення розповсюдження шкідливого програмного забезпечення – у резервний режим готовності. Доти, поки немає атаки, що активно розвивається, вхідний трафік з мережі Інтернет проходить через комутатор без якого-небудь втручання з боку Блоку усунення розповсюдження шкідливого програмного забезпечення. Копія вхідного трафіку посилає для аналізу на Детектор аномалій трафіку через зовнішній аналізатор протоколів (SPAN) або віртуальні списки ACL. Якщо Детектор аномалій трафіку виявляє дестабілізуюче в порівнянні з базовою інформацією поводження трафіку, починається процес усунення:

– Детектор аномалій трафіку направляє в Блок усунення розповсюдження шкідливого програмного забезпечення команду почати процес зміни напрямку.

– Блок усунення розповсюдження шкідливого програмного забезпечення відхиляє (“захоплює”) трафік, адресований на атакуєму IP-адресу, переадресуючи його на самого себе.

– Блок усунення розповсюдження шкідливого програмного забезпечення піддає трафік багатоступінчастому аналізу й застосовує контрзаходи для відділення благонадійних джерел від джерел атаки. Цей процес іменується очищенням або вичищенням.

– Блок усунення розповсюдження шкідливого програмного забезпечення скидає трафік атаки й пересилає благонадійний трафік назад на нормальний маршрут проходження трафіку до мети. Цей процес іменується ін'єкцією.

Атаки DDoS – Виявлення й усунення

У таблиці 3.4 перераховані типи атак DDoS, які може виявляти й усувати Блок усунення розповсюдження шкідливого програмного забезпечення.

Таблиця 3.4 – Категорії й особливі типи атак DDoS

Категорія атаки	Особливі типи атак
Атаки з дефіцитом ресурсів	Атаки пакетного розміру, характерна риса яких – фрагментированні або великі пакети. Приклади: teardrop і ping-of-death.
	Атаки зомбі-комп'ютерів/мереж зомбі-комп'ютерів з низькою інтенсивністю схожі на атаки із заповненням смуги пропущення за тим виключенням, що кожне джерело атаки посилає множинні запити з невеликим обсягом в одиницю часу.
	Атаки DNS з рекурсивним переглядом DNS.
Атаки із заповненням смуги пропущення	Лавинні атаки зі спуфінгом або без спуфінга: – Прапор TCP (SYN, SYN-ACK, ACK, FIN). – Протокол керування повідомленнями в Інтернет (ICMP). – Протокол користувальницьких датаграмм (UDP). Приклади: лавинна атака SYN, smurf, LAND і UDP – лавинні атаки.
	Атаки зомбі-комп'ютерів/мереж зомбі-комп'ютерів, у яких кожний вихідний зомбі-пк або мережа відкриває множинні TCP-з'єднання й, у деяких випадках, видає багаторазові запити HTTP.
	Атаки DNS, наприклад, лавинна атака із запитами DNS.

Детектор аномалій трафіку

Детектор аномалій трафіку – це пасивний пристрій моніторингу, що постійно виявляє ознаки, що вказують на присутність атаки DDoS, спрямованої проти захищеного місця призначення, також іменованого зоною. Це може бути сервер, інтерфейс міжмережного екрана або інтерфейс маршрутизатора. Детектор аномалій трафіку аналізує копії всього вхідного трафіку, адресуємого в захищені

зони, через SPAN або відгалуження пасивної мережі. Цей аналіз включає зіставлення поточного поводження трафіку з базовими граничними параметрами, які також іменуються зональною політикою, для виявлення розповсюдження шкідливого програмного забезпечення. Якщо дестабілізуюче поводження виявлене й виглядає як можлива атака, Детектор аномалій трафіку через позаполосну управлінську мережу Ethernet посилає в Блок усунення розповсюдження шкідливого програмного забезпечення сигнал про початок аналізу й усунення атаки.

Блок усунення розповсюдження шкідливого програмного забезпечення

Блок усунення розповсюдження шкідливого програмного забезпечення – це автономний пристрій аналізу й фільтрації трафіку. Починаючи прийом трафіку, адресованого в конкретну зону, що, очевидно, піддається атаці, Блок усунення розповсюдження шкідливого програмного забезпечення проводить точний аналіз цього трафіку. Якщо результати аналізу підтверджують, що трафік злочинний, Блок усунення розповсюдження шкідливого програмного забезпечення застосовує контрзаходи, наприклад, механізми анти-спуфінга й фільтрацію різного рівня (таблиця 3.1). Кінцевий результат полягає в тому, що трафік зі злочинних джерел скидається, а трафік із благонадійних джерел пересилається в передбачений пункт призначення.

Блок визначення деструктивних дій та виду атаки

Блок визначення деструктивних дій та виду атаки:

– Широкомовний шторм.

– Додатки, що роблять інтенсивне широкомовне розсилання, наприклад: широкомовні чати й мережні ігри.

– Атака ARP-spoofing на таблицю mac-адрес комутаторів.

Широкомовний шторм

Широкомовний шторм – лавина (сплеск) широкомовних акетів (на другому рівні моделі OSI – кадрів). Розмноження некоректно сформованих

широкомовних повідомлень у кожному вузлі приводить до експонентного росту їхнього числа й паралізує роботу мережі. Звичайно такі пакети використовуються мережними сервісами для оповіщення станцій про свою присутність. Вважається нормальним, якщо широкомовні пакети становлять не більше 10% від загального числа пакетів у мережі.

Також досить часто до шторму приводять кільця в мережі при некоректному настроюванні протоколу Spanning Tree, оскільки в заголовку пакетів Ethernet немає інформації про час життя кадру, як, наприклад, у пакетів IP. Крім цього широкомовний шторм застосовується (навмисно) зломщиками.

Відповідно до галузевого стандарту де-факто число широкомовних і багатоадресних кадрів у мережі не повинне перевищувати 8-10% від загального числа кадрів.

Широкомовний кадр – це кадр, адресований всім станціям у домені мережі. Багатоадресний кадр – це кадр, адресований групі станцій у домені мережі. Оскільки широкомовний кадр адресований всім станціям, то, одержавши його, станції повинні перервати свою роботу й обробити такий кадр. Це сповільнює роботу всієї мережі.

Якщо відношення числа широкомовних кадрів до загального числа кадрів більше 10%, то такий ефект називається "широкомовним штормом".

Широкомовний шторм може бути наслідком дефектів устаткування або неправильного настроювання параметрів активного встаткування. Найчастіше це явище спостерігається в розподілених мережах NetWare, побудованих на основі комутаторів, або коли дані між сегментами або доменами мережі можуть передаватися більш ніж по одному потенційному шляху. Якщо один з комутаторів такої мережі не підтримує протокол Spanning Tree (звичайно IEEE 802.1d) або останній неправильно настроєний або збоїть, то в мережі починається некерована циркуляція широкомовних кадрів.

Виявлення "широкомовного шторму" є не настільки тривіальним завданням, як це може здатися на перший погляд. Для його виявлення недостатньо взяти загальне число широкомовних кадрів і поділити його на загальне число кадрів, що пройшли по мережі.

Для цього ви повинні визначити: яку частку становлять широкомовні кадри в кожний інтервал часу (наприклад, за одну мінуту) і яка при цьому утилізація каналу зв'язку. Якщо, наприклад, за одну мінуту по мережі пройшло 4 кадри, а 2 з них були широкомовними, то це ще не виходить, що ви спостерігаєте "широкомовний шторм".

Захист від широкомовних штормів (broadcast storm)

Одна з характерних несправностей мережного програмного забезпечення – мимовільна генерація з високою інтенсивністю широкомовних пакетів. Широкомовним штормом вважається ситуація, у якій відсоток широкомовних пакетів перевищує 20% від загальної кількості пакетів у мережі. Звичайний комутатор або міст сліпо передає такі пакети на всі свої порти, як того вимагає його логіка роботи, засмічуючи, таким чином, мережу. Боротьба із широкомовним штормом у мережі, з'єднаної комутаторами, жадає від адміністратора відключення портів, що генерують широкомовні пакети. Маршрутизатор не поширює такі ушкоджені пакети, оскільки в коло його завдань не входить копіювання широкомовних пакетів в усі поєднувані їм мережі. Тому маршрутизатор є прекрасним засобом боротьби із широкомовним штормом, щоправда, якщо мережа розділена на достатню кількість підмереж.

ARP-spoofing

ARP-spoofing – техніка атаки в Ethernet мережах, що дозволяє перехоплювати трафік між хостами. Заснована на використанні протоколу ARP.

При використанні в розподіленій обчислювальній системи (РВМ) алгоритмів віддаленого пошуку існує можливість здійснення в такій мережі типової віддаленої атаки «помилковий об'єкт РВМ». Аналіз безпеки протоколу ARP показує, що, перехопивши на атакуючому хості усередині даного сегмента мережі широкомовний ARP-запит, можна послати помилкову ARP-відповідь, у якій оголосити себе шуканим хостом (наприклад, маршрутизатором), і надалі активно контролювати мережний трафік дезінформованного хосту, впливаючи на нього за схемою «помилковий об'єкт РВМ».

Протокол ARP призначений для перетворення IP-адрес в MAC-адреси. Найчастіше мова йде перетворенні в адреси Ethernet, але ARP використовується й у мережах інших технологій: Token Ring, FDDI і інших.

Алгоритм роботи ARP

Протокол може використовуватися в наступних випадках:

1. Хост A хоче передати IP-пакет вузлу B, що перебуває з ним в одній мережі.

2. Хост A хоче передати IP-пакет вузлу B, що перебуває з ним у різних мережах, і користується для цього послугами маршрутизатора R.

У кожному із цих випадку вузлом A буде використовуватися протокол ARP, тільки в першому випадку для визначення MAC-адреси вузла B, а в другому – для визначення MAC-адреси маршрутизатора R. В останньому випадку пакет буде переданий маршрутизатору для подальшої ретрансляції.

Далі для простоти розглядається перший випадок, коли інформацією обмінюються вузли, що перебувають безпосередньо в одній мережі. (Випадок коли пакет адресований вузлу, який знаходиться за маршрутизатором, відрізняється тільки тим, що в пакетах переданих після того як ARP-перетворення завершене, використовується IP-адреса одержувача, але MAC-адреса маршрутизатора, а не одержувача.)

Проблеми ARP

Протокол ARP є абсолютно незахищеним. Він не має ніякого способу перевірки дійсності пакетів: як запитів, так і відповідей. Ситуація стає ще більш складною, коли може використовуватися мимовільний ARP (gratuitous ARP).

Мимовільний ARP – таке поводження ARP, коли ARP-відповідь надсилається, коли в цьому (з погляду одержувача) немає особою необхідності. Мимовільна ARP-відповідь це пакет-відповідь ARP, присланий без запиту. Він застосовується для визначення конфліктів IP-адрес у мережі: як тільки станція одержує адресу по DHCP або адреса привласнюється вручну, розсилається ARP-відповідь gratuitous ARP.

Мимовільний ARP може бути корисний у наступних випадках:

– Відновлення ARP-таблиць, зокрема, у кластерних системах.

– Інформування комутаторів.

– Повідомлення про включення мережного інтерфейсу.

Незважаючи на ефективність мимовільного ARP, він є особливо небезпечним, оскільки з його допомогою можна запевнити віддалений вузол у тому, що MAC-адреса якої-небудь системи, що перебуває з нею в одній мережі, змінилася й указати, яка адреса використовується тепер.

До виконання ARP-spoofing'а в ARP-таблиці вузлів A і B існують записи з IP- і MAC-адресами один одного. Обмін інформацією виробляється безпосередньо між вузлами A і B.

У ході виконання ARP-spoofing'а комп'ютер C, що виконує атаку, відправляє ARP-відповіді (без одержання запитів):

– вузлу A: з IP-адресою вузла B і MAC-адресою вузла C;

– вузлу B: з IP-адресою вузла A і MAC-адресою вузла C.

У силу того що комп'ютери підтримують мимовільний ARP (gratuitous ARP), вони модифікують власні ARP-таблиці й поміщають туди записи, де замість справжніх MAC-адрес комп'ютерів A і B коштує MAC-адреса комп'ютера C.

Після того як атака виконана, коли комп'ютер A хоче передати пакет комп'ютеру B, він знаходить в ARP-таблиці запис (він відповідає комп'ютеру C) і визначає з її MAC-адресу одержувача. Відправлений по цьому MAC-адресу пакет приходить комп'ютеру C замість одержувача. Комп'ютер C потім ретранслює пакет тому, кому він дійсно адресований – тобто комп'ютеру B.

Блок аналізу мережної статистики

Блок збирання наступної інформації:

– Основна статистика (Summary).

– Ієрархія протоколу (Protocol Hierachy).

– Сеанси обміну пакетами (Conversations).

– Точки призначення (Endpoints).

– Графіки I/O (IO Graphs).

– Список сеансів обміну пакетами (Conversation List).

– Список точок призначення (Endpoint List).

– Час чекання відповіді від сервісу (Service Response Time).

– RTP.

– SIP.

– Виклики VoIP (VoIP Calls).

– Призначення (Destination).

– Графік потоку (Flow Graph).

– HTTP.

– IP-адреса (IP address).

– Довжина пакету (Packet Length).

– Тип порту (Port Type).

Розпишемо їх більш детально.

1. Основна статистика. Доступні такі елементи основної статистики, як:

– Властивості захоплених файлів.

– Час захвату.

– Інформація про фільтр захвату.

– Інформація про фільтр відображення.

2. Ієрархія протоколу. Статистика ієрархії протоколу допомагає аналізувати пакети, розбиваючи відображені дані, які належать чинному рівню OSI.

3. Сеанси обміну пакетами. Якщо ви використовуєте протокол TCP/IP або програму, яка працює із цим протоколом, ви маєте побачити чотири активних вкладок для обміну пакетами за допомогою Ethernet, IP, TCP та UDP. «Діалог» між комп’ютерами відображає трафік між двома активними хостами. Номер, зазначений на вкладці після назви протоколу, означає кількість «діалогів» між хостами. Номер, зазначений на вкладці після назви протоколу, означає кількість «діалогів» між хостами, наприклад, «Ethernet:6».

4. Точки призначення. Точки призначення забезпечують статистику даними про відправку та прийом пакетів. Номер, зазначений на вкладці після назви протоколу, вказує на кількість точок призначення. Наприклад, «Ethernet:6».

5. Графіки I/O. Основний графік може бути отриманий за допомогою команди «IO graphs» (Графіки I/O). Ще декілька графіків можуть бути додані у тому ж вікні на основі фільтрів відображення.

6. Час чекання відповіді від сервісу. 13 протоколів доступні для глибокого аналізу.

7. RTP. RTP (Real-time Transport Protocol, протокол передачі у реальному часі, RFC 3550) – це протокол для передачі звука та відео через IP-мережу. Він працює у початку протоколу дейтаграм користувача (User Datagram Protocol, UDP). Він часто використовується у сукупності з протоколами SIP або H.233, забезпечуючи виконання сигнальних завдань.

8. SIP. SIP (Session Initiation Protocol, протокол встановлення сесії, RFC 3261) – це сигнальний протокол, який оголошує відео– або VoIP-сесії. Він працює разом із протоколом RTP, який використовується для передачі мультимедійних даних.

9. Виклики VoIP.

VoIP (Voice over IP, голосовий зв’язок за допомогою Інтернету) взагалі використовує два типи протоколів:

– сигнальні протоколи, такі, як SIP або H.323

– переносні протоколи, наприклад, RTP

10 Призначення. Відображення усіх IP-адрес призначення мережевих пакетів.

11. Графік потоків. Графіки потоків забезпечує послідовний аналіз TCP-з’єднань. Перші три строки містять оголошення TCP– з’єднання з послідовностями «SYN», «SYN ACK» та «ACK».

12 HTTP. HTTP (Hypertext Transfer Protocol, протокол передачі гіпертексту) – це протокол типу «клієнт-сервер», який використовується для передачі HTML-файлів. HTTP-клієнт (у більшості випадків це web-браузер) відсилає HTTP-запит до web-серверу із полем «URL», який допомагає знайти потрібний файл. Web-сервер відповідає HTTP-пакетом та забезпечує клієнт необхідною web-сторінкою.

Меню «HTTP» містить три підменю:

– «Load Distribution» (Розподіл пакетів).

– «Packet Counter» (Лічильник пакетів).

– «Requests» (Запити).

14 IP-адреса. Відображення IP-адреси джерела або призначення мережевих пакетів.

15. Довжина пакету.

16. Тип порту. Відображення статистики портів TCP або UDP.

Блок визначення топології мережі

Блок визначення топології мережі:

– Блок використання відомостей із загальної системи моніторингу мережі, а не опитування пристрою додатково.

– Блок складання списку пристроїв у мережі, автоматично, ґрунтуючись на дані системи моніторингу.

– Блок побудови топології мережі, за станом на задану дату й відстеження змін у топології протягом часу.

– Блок автоматичного визначення рівнів ієрархії пристроїв у мережі, з виділенням периферійних, проміжних і центральних вузлів;

– Блок побудови топології мережі, незалежно від використовуваної системи моніторингу й програмно-апаратних платформ;

– Блок комбінувати показників, на основі яких визначаються зв'язки між пристроями, і при їхньому обчисленні виконувати перевірку на значимість із використанням статистичних критеріїв.

Можливі варіанти зміни напрямку трафіку

Фахівці з IT можуть використовувати описані нижче варіанти зміни напрямку трафіку з його пересиланням з мережі, розташованого вище лежачого оператора зв'язку, на Блок усунення розповсюдження шкідливого програмного забезпечення. Цей процес також іменується “захватом” трафіку:

– Повідомлення прикордонного шлюзового протоколу (Border Gateway Protocol, BGP) із Блок усунення розповсюдження шкідливого програмного забезпечення на маршрутизатори, розташовані у вище лежачого оператора зв'язку, з інформацією про те, що трафік, адресований на захищену адресу призначення, буде переспрямований на Блок усунення розповсюдження шкідливого програмного забезпечення.

– Використання зовнішніх механізмів зміни напрямку трафіку, наприклад, маршрутизаторів віддаленого відновлення BGP.

– Повідомлення про ін'єкцію очищеного трафіку на маршруті (Route Health Injection, RHI) від Блок усунення розповсюдження шкідливого програмного забезпечення для процесу маршрутизації в Catalyst серії 6500 або в систему нагляду серії 7600. Ці повідомлення поміщають статичний маршрут у глобальну таблицю маршрутизації, у якій модуль Блок усунення розповсюдження шкідливого програмного забезпечення позначений як наступний вузол.

Можливі варіанти ін'єкції трафіку

Ін'єкція трафіку – це процес, застосовуваний у Блок усунення розповсюдження шкідливого програмного забезпечення для пересилання очищеного благонадійного трафіку в точку призначення, що піддається атаці. Рішення підтримує різні варіанти ін'єкції трафіку. У варіанті 2-ого рівня топології, очищений трафік пересилається із Блок усунення розповсюдження шкідливого програмного забезпечення на статично-конфігуруєму наступну адресу заходу. Ця адреса перебуває на маршрутизаторі, розташованому нижче й з'єднаним з тої ж VLAN або підмережею, що й інтерфейс/VLAN ін'єкції трафіку. Ін'єкцію трафіку на 2-му рівні найпростіше конфігурувати, оскільки тут не

потрібно вносити які-небудь істотні зміни в конфігурацію маршрутизатора, розташованого нижче.

Варіанти ін'єкції трафіку 3-го рівня:

– Маршрутизація й пересилання по VPN (VPN Routing and Forwarding, VRF).

– Маршрутизація на основі політики (Policy-Based Routing, PBR).

– Транкинг VLAN (VLAN Trunking).

– Інкапсуляція по загальній маршрутизації (GRE) або інкапсуляція IP у тунелі IP (IPIP).

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування).

Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

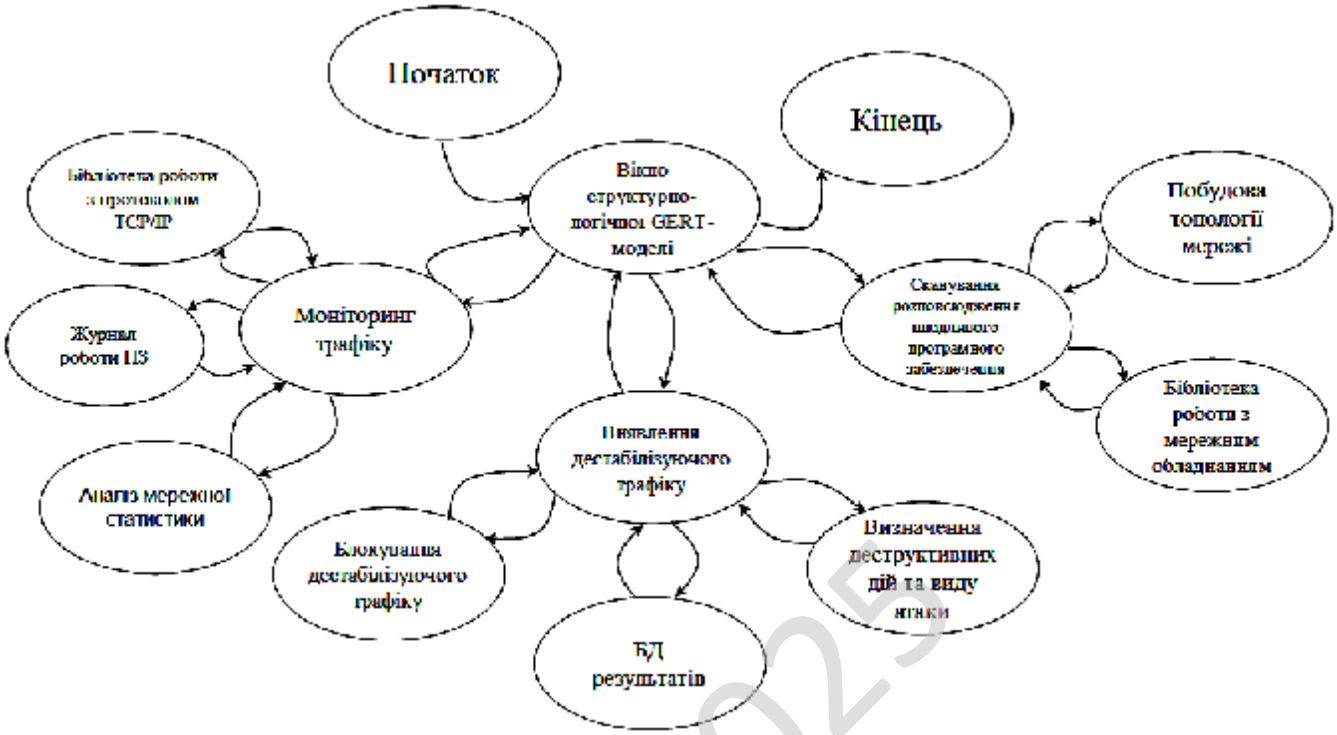

Рисунок 3.8 – Діаграма взаємодії процесів

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Первинною стадією без якої не відбувається розробка програмного забезпечення це звичайно розробка блок-схем. На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 та 4.3 зображено роботу підпрограм. З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограм та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограм виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Опис алгоритмів функціонування системи. Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми. При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення. При складанні блок-схем програмного забезпечення і напрацювання алгоритмів я зіткнувся з масою проблем, які вимагали напрацювання процедур і функцій над основною проблематикою. Для чого були створені додаткові класи, типи даних і константи, що забезпечило вирішення проблем.

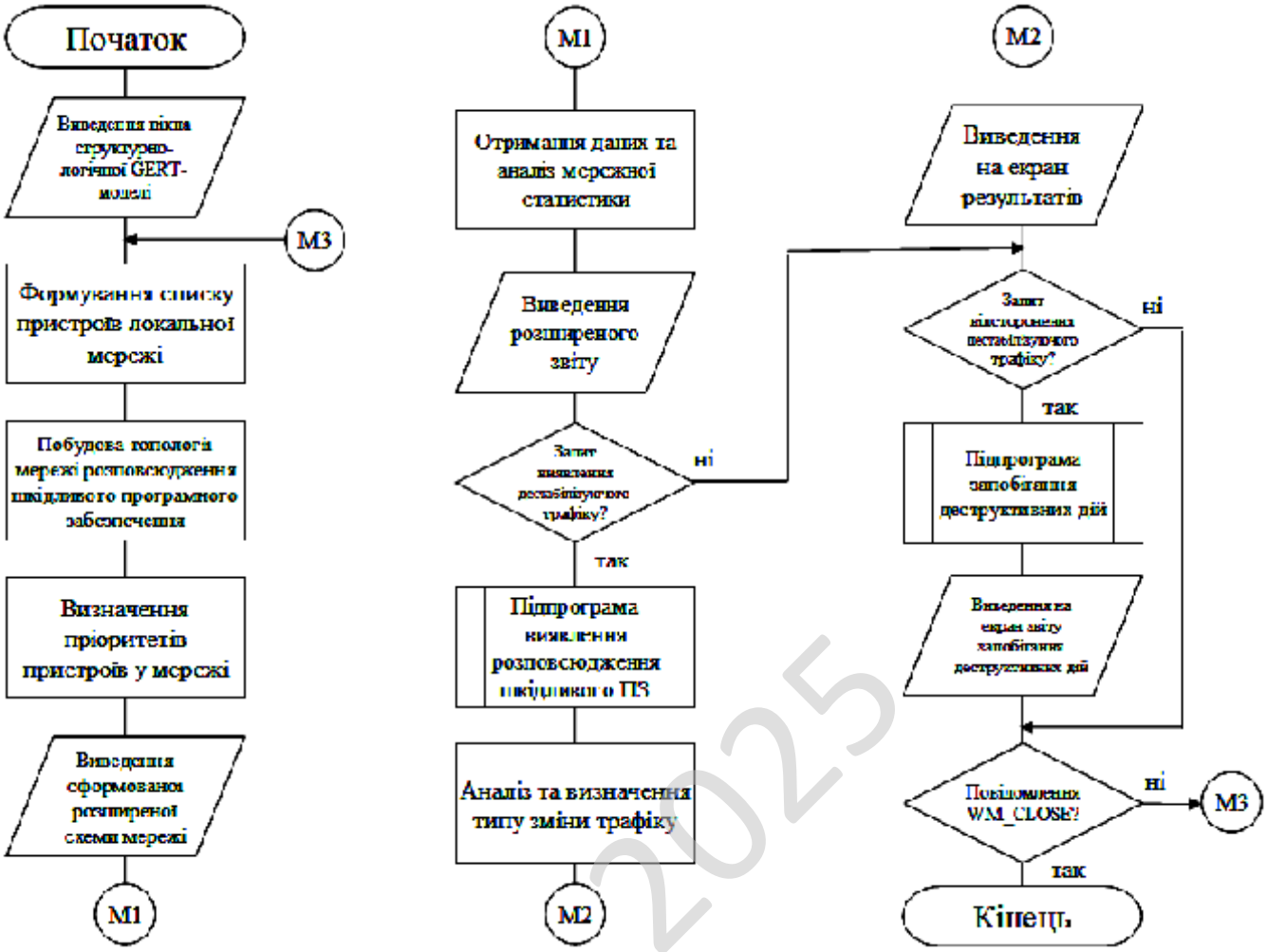

Рисунок 4.1 – Блок-схема основної програми

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування.

Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

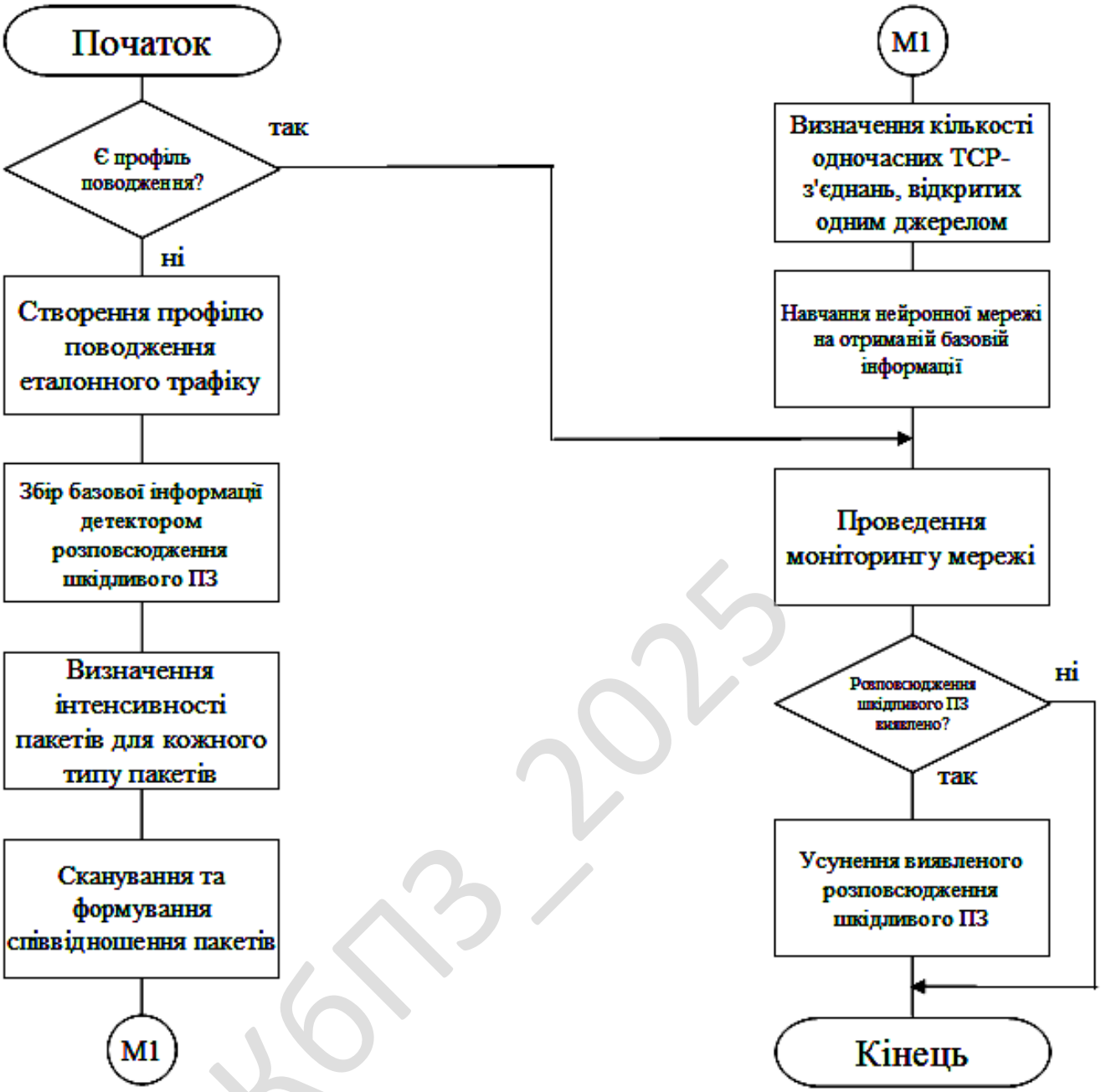

Рисунок 4.2 – Блок-схема роботи підпрограми

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код. Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки. Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

– Керівникам проектів, які керують розподілом завдань і контролем за проектом.

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

– Бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії.

– Програмістам які реалізовують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

Блок-схема алгоритму еквівалентного перетворення GERT-мережі представлена на рисунку 4.3.

Рисунок 4.3 – Блок-схема алгоритму еквівалентного перетворення GERT-мережі

Відмінною рисою даного алгоритму є введення процедур визначення внутрішніх однорівневих підмереж і виключення петель першого роду.

Проведені дослідження показали сумірність результатів моделювання еквівалентної GERT-мережі з первісною структурою. Саме це дозволило провести математичне моделювання й аналіз комп'ютерних вірусів типу Flame, і зробити вивід про те, що підхід, заснований на GERT-моделюванні, дозволяє врахувати ряд деструктивних факторів, властивому даному типу вірусів, і тим самим розширити спектр можливих сценаріїв деструктивних впливів до 30%.

Скористаємося описаною в даному підрозділі методикою еквівалентних перетворень, що спрощують, для математичної формалізації технології передачі даних у процесі інформаційного обміну спеціалізованими сигнатурами з «хмарними» антивірусними системами.

Проведемо порівняльні дослідження розробленої математичної моделі технології поширення комп'ютерних вірусів у ТКС. Для такого дослідження й відповідно оцінки в якості еталонної виберемо математичну модель PSIDDR, представлену в роботах [3, 4, 8] на основі біологічного підходу моделювання. За даними джерел [3, 4] зазначена математична модель найбільше адекватно описує процес поширення комп'ютерних вірусів і враховує п'ять можливих станів вузлів ТКС у процесі їхнього функціонування в умовах зовнішніх деструктивних впливів.

Виходячи з виділених в [3, 4] даних, у математичній моделі PSIDDR, узагальнена структура ТКС може бути представлена за допомогою вираження:

$$N = S(t) + I(t) + D(t) + R(t) + X(t),$$

де:

S(t) – кількість уразливих об'єктів; I(t) – кількість заражених об'єктів; R(t) – кількість вилікуваних об'єктів, що володіють імунітетом; D(t) – кількість об'єктів, у яких виявлений вірус; X(t) – кількість виведених з ладу вузлів; N – загальна кількість об'єктів у системі.

З урахуванням зазначених особливостей функціонування ТКС модель PSIDDR математично можна представити у вигляді системи:

$$\begin{cases} \dfrac{dS(t)}{dt} = -\beta \dfrac{S(t)I(t)}{N} - \alpha S(t) \\ \dfrac{dI(t)}{dt} = \beta \dfrac{S(t)I(t)}{N} - (\gamma + \chi)I(t) \\ \dfrac{dR(t)}{dt} = \omega D(t) + \alpha S(t) \\ \dfrac{dD(t)}{dt} = \gamma I(t) - (\omega + \chi)D(t) \\ \dfrac{dX(t)}{dt} = \chi I(t) + \chi D(t) \\ \dfrac{dS(t)}{dt} + \dfrac{dI(t)}{dt} + \dfrac{dR(t)}{dt} + \dfrac{dD(t)}{dt} + \dfrac{dX(t)}{dt} = 0, \end{cases} \quad (4.1)$$

де:

β – частота зараження;

α – імовірність імунізації до стадії зараження;

χ – імовірність того, що вірус атакує вузол з фатальними наслідками;

γ – імовірність того, що вірус на даному вузлі буде виявлений;

ω – імовірність лікування;

S(t) – кількість уразливих об'єктів;

I(t) – кількість заражених об'єктів;

R(t) – кількість вилікуваних (з імунітетом) об'єктів;

X(t) – кількість виведених з ладу об'єктів;

D(t) – кількість виявлених заражених об'єктів (на першій стадії дорівнює 0);

N – загальна кількість об'єктів у системі.

Як вихідні параметри моделювання й порівняльної оцінки були обрані числові значення характеристик процесу поширення комп'ютерних вірусів, характерні реальному функціонуванню ТКС локального рівня:

– α= 0,08;

– γ= 0,3;

– ω= 0,3;

– β= 0,3.

На рисунку 4.4 представлені графіки залежності кількості заражених (I), виведених з ладу (X), і вилікуваних (R) об'єктів від часу функціонування комп'ютерної системи, у різних початкових умовах зараженості мережі.

Так, на рисунку 4.4.а приводиться сімейство кривих, що характеризує перераховані процеси в умовах, коли ймовірність $\chi = 0{,}01$, а рівень зараження ТКС на момент початку другої стадії $U = \frac{I}{N} = 0{,}9$. Аналогічно на рисунку 4.4.б визначені наступні початкові умови: $\chi = 0{,}1$, $U=0{,}9$; на рисунку 4.4.в: $\chi = 0{,}1$, $U=1$, на рисунку 4.4.д: $\chi = 0{,}01$, $U = 1$.

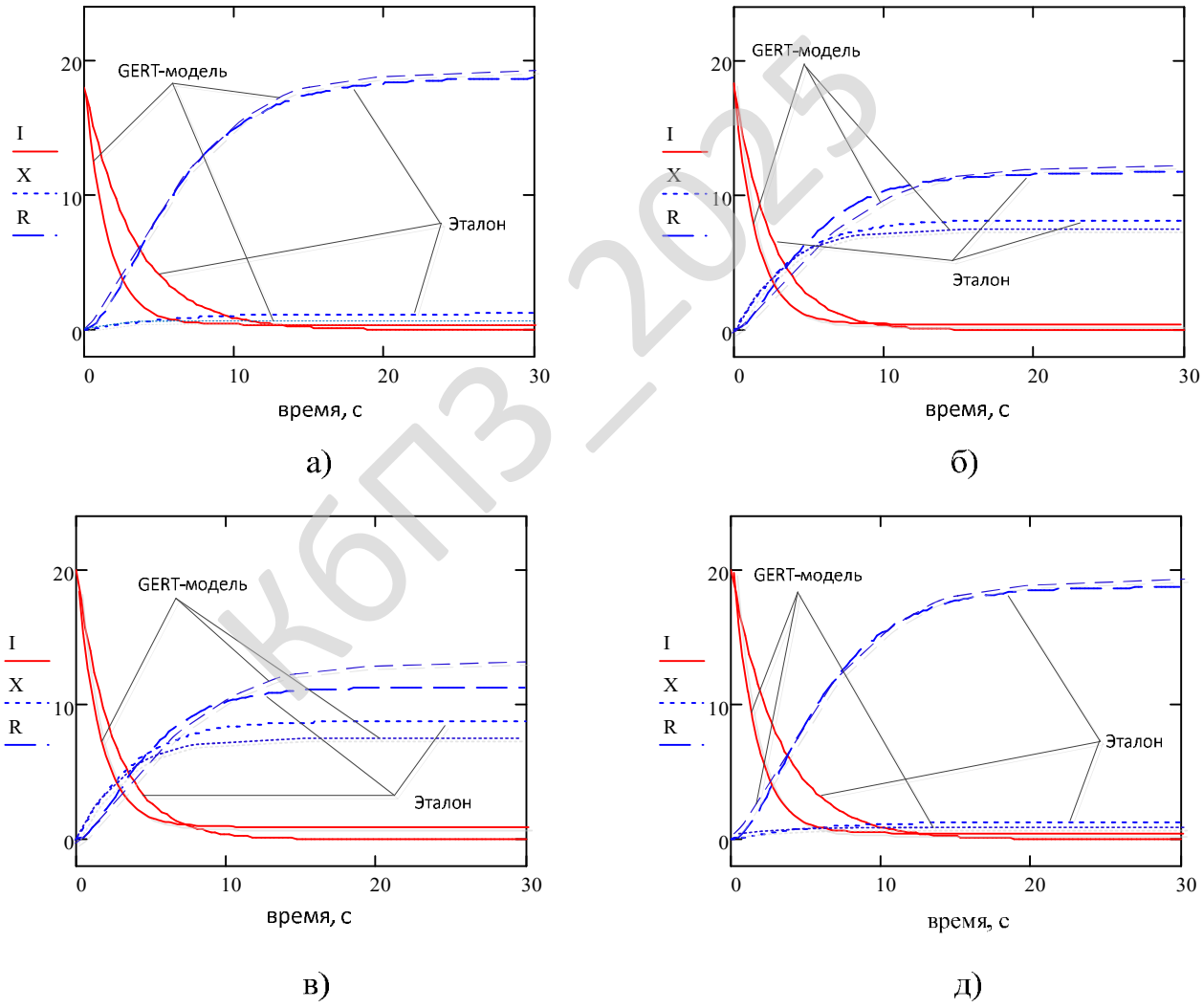

Рисунок 4.4 – Графіки залежності кількості заражених (I), виведених з ладу (X), і вилікуваних (які володіють імунітетом (R)) об'єктів від часу функціонування інформаційно-телекомунікаційної мережі

Як видно з рисунка, у першому досліджуваному випадку (рисунок 4.3.а), кінцева кількість виведених з ладу об'єктів $X \approx \{1,2\}$, у другому випадку (рисунок 4.3.б) ця кількість $X \approx \{7,8\}$, у третьому (рисунок 3.8.в) – $X \approx \{8,9\}$, у четвертому (рисунок 4.4.д) – $X \approx \{1,2\}$.

Крім цього з рисунка 4.4. видно, що облік в GERT-моделі ключової інформації про стани телекомунікаційних вузлів (інтелектуальних вузлів комутації) у процесі деструктивних впливів комп'ютерних вірусів дозволив підвищити точність отриманих результатів у порівнянні з еталоном до 1,4 рази.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм SEED – у криптографії симетричний блоковий криптоалгоритм на основі Мережі Фейстеля, розроблений Корейським агентством інформаційної безпеки (Korean Information Security Agency, KISA) в 1998 році. В алгоритмі використовується 128-бітний блок і ключ довжиною 128 біт. Алгоритм одержав широке поширення й використовується фінансовими й банківськими структурами, виробничими підприємствами й бюджетними установами Південної Кореї, оскільки 40-бітний SSL не забезпечує на даний момент мінімально необхідного рівня безпеки. Агентством по захисту інформації специфіковане використання шифру SEED у протоколах TLS і S/MIME. У той же час, алгоритм SEED не реалізований у більшості сучасних браузерів і інтернет-додатків, що утрудняє його використання в даній сфері поза межами Південної Кореї.

SEED являє собою мережу Фейстеля з 16 раундами, 128-бітовими блоками й 128-бітовим ключем. Алгоритм використовує дві 8×8 таблиці підстановки, які, як такі з Safer, виведені з дискретного зведення в ступінь (у цьому випадку, x^{247} і x^{251} – плюс деякі «несумісні операції»). Це є деякою подібністю с MISTY1 у рекурсивності його структури: 128-бітовий повний шифр – мережа Фейстеля з F-

функцією, що впливає на 64-бітові половини, у той час як сама F-функція – Мережа Фейстеля, складена з G-функції, що впливає на 32-розрядні половини. Однак рекурсія не простягнеться далі, тому що G-функція – не Мережа Фейстеля. В G-функції 32-розрядне слово розглядають як чотири 8-бітових байта, кожний з яких проходить через одну або іншу таблицю підстановки, потім поєднується в помірковано комплексному наборі булевих функцій таким чином, що кожний біт виводу залежить від 3 з 4 вхідних байтів.

SEED має складний ключовий розклад, генеруючи тридцять два 32-розрядних додаткових символу, використовуючи G-функції на серіях обертань вихідного неопрацьованого ключа, комбінованого зі спеціальними раундовими константами (як в TEA) від «Золотого співвідношення» (англ. Golden ratio).

Згідно з дослідженнями KISA, алгоритм SEED «надійно протистоїть відомим атакам».

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс програмного забезпечення, розробленого у результаті виконання бакалаврської дипломної роботи.

Розроблене програмне забезпечення структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення складається з наступних функціональних блоків:

– Навігаційне меню: Файл; Дані; Система; Звіти; Налаштування; Довідка.

– Вікно виведення результату роботи структурно-логічної GERT-моделі.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

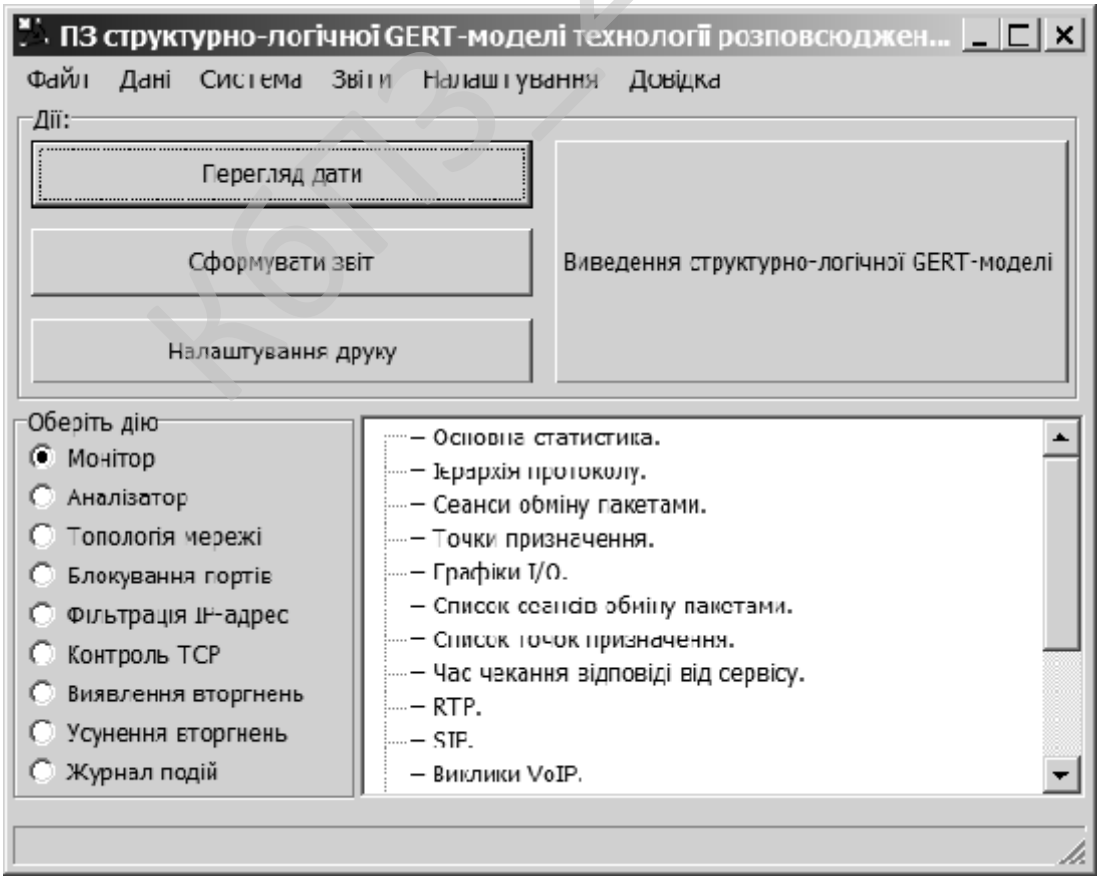

Рисунок 5.1 – Головне вікно розробленого ПЗ

Для перегляду короткої довідки про програму слід натиснути на основному вікні кнопку авторського права, після чого на екрані з’явиться вікно показане на рисунку 5.2.

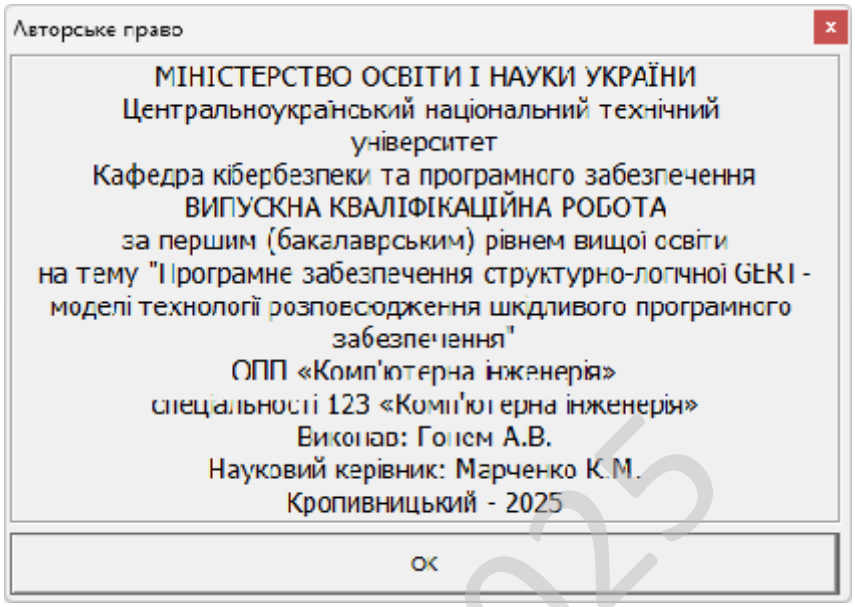

Рисунок 5.2 – Вікно розробника ПЗ

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування форматом білої скриньки та чорної скриньки. Тестування форматом білої скриньки засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

– Виконуються всі цикли (у межах їхніх кордонів та діапазонів).

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

– Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

– У програмі можуть бути пропущені деякі маршрути.

– Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

– Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

– Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

– При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

– Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Проводилось тестування чорної скриньки. Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ITc).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його OT).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITc;

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору OT.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій;

– Помилок інтерфейсу;

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних;

– Помилок характеристик (необхідна ємність пам'яті і т.д.);

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – Shareware.

Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання. Основний принцип умовно-безплатного ПЗ - «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості. Якщо після

закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму.

В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

– Досліджена система структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

– На основі отриманих результатів досліджень створена програмна реалізація структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

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

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

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для

структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм SEED.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Massimo Bertaccini. Cryptography Algorithms. Packt Publishing. 2022. 358 p.

2. Alyssa Miller. Cybersecurity Career Guide. Manning Publications. 2022. 368 p.

3. Awais Rashid, Howard Chivers, George Danezis, Emil Lupu, Andrew Martin. CyBOK The Cyber Security Body of Knowledge. The National Cyber Security Centre. 2019. 854 p.

4. Loren Kohnfelder. Designing Secure Software. No Starch Press. 2022. 332 p.

5. Samir Kumar Rakshit. Ethical Hacker's Penetration Testing Guide. BPB Online. 2022. 509 p.

6. Corey J. Ball. Hacking APIs. No Starch Press. 2022. 353 p.

7. Kevin Beaver. Hacking for Dummies. John Wiley & Sons. 2022. 419 p.

8. Mark S. Merkow. Practical Security for Agile and DevOps. CRC Press. 2022. 236 p

9. Derek Fisher. Application Security Program Handbook. Manning Publications. 2021. 155 p.

10. Cameron Wyatt PH.D. Kali Linux Tutorial. Independently published. 2021. 60 p.

11. Alex Matrosov, Eugene Rodionov, Sergey Bratus. Rootkits and Bootkits. No Starch Press. 2019. 450 p.

12. Kuznetsov O., Frontoni E., Kuznetsova Y., Smirnov O., Moskovchenko I. «Trust-Based Security Architecture for Edge Computing: A Simulation Study of Dynamic Trust Evolution and Attack Detection». *CEUR Workshop Proceedings,* 2024, 3909, pp. 227–241.

13. Lakhno, V., Malyukov, V., Smirnov, O., Bebeshko, B., Chubaievskiy, V., Zhumadilova, M., Malyukova, I., Smirnov, S. «Multifactorial Model for Targeted

Attacks Counteracting Within the Framework of a Multi-Step Quality Game with Fuzzy Information». *8th International Symposium on Intelligent Informatics, ISI 2023*, 2025. vol 389. pp 377-389. Springer, Singapore.

14. Kuznetsov, O., Frontoni, E., Kryvinska, N., Chevardin, V., Smirnov, O. «Wireless Network Encryption Stream Ciphers, Computational Modeling, and Security Analysis». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 379–402.

15. Kuznetsov, O., Frontoni, E., Kryvinska, N., Smirnov, O., Imoize, G.L. «Computational Modeling of Enhanced Spread Spectrum Codes for Asynchronous Wireless Communication». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 403–447.

16. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

17. Akhalaia, G., Iavich, M., Iashvili, G., Prysiazhnyy, D., Smirnova, T. «Secure Encrypted Connection on Georgian Website». *CEUR Workshop Proceedings,* 2023, 3550, pp. 313-320.

18. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56

19. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

20. Kuznetsov, O., Kandiy, S., Frontoni, E., Smirnov, O. «Trade-offs in Post-Quantum Cryptography: A Comparative Assessment of BIKE, HQC, and Classic McEliece». *CEUR Workshop Proceedings,* Volume 3504, 2023, pp. 1-11.

21. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022,

22. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». *In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies*, vol 131. 2023. Springer, Singapore. pp. 21-34.

23. Smirnov O.A., Al-Oraiqat A.M., Ulichev O.S., Meleshko Ye.V., Al-Rawashdeh H.S., Polishchuk L.I. «Modeling strategies for information influence dissemination in social networks». *Journal of Ambient Intelligence and Humanized Computing* Volume 13, Issue 5. Springer, Cham. 2022, pp. 2463-2477.

24. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021,* Cracow, Poland, 22-25 September 2021. P. 414-418

25. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». *4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) - 2021*, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

26. Smirnov O., Kuznetsov A., Girzheva O., Kiian A., Nakisko O., Kuznetsova T. «Advanced Code-Based Electronic Digital Signature Scheme». *2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020,* Kharkiv, 6 October 2020-9 October 2020, P. 358-362.

27. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova K. «Data hiding scheme based on spread sequence addressing». *CEUR Workshop Proceedings* Volume 2805, 2020, Pages *44-58.*

28. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». *International Journal of Computing*; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

29. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

30. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

31. Smirnov O., Kuznetsov A., Arischenko A., Chepurko I., Onikiychuk A., Kuznetsova T. «Pseudorandom sequences for spread spectrum image steganography». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 122-131.

32. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 1-14.

33. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». *Lecture Notes in Networks and Systems*, vol 152. Springer, Cham. 2021, pp 66-84.

34. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

35. Smirnov O., Kuznetsov A., Kiian A., Babenko V., Perevozova I.,Chepurko I. «New Approach to the Implementation of Post-Quantum Digital Signature Scheme». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 166-171.

36. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

37. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

38. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

39. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». *International Journal of Computer Network and Information Security (IJCNIS)*. Vol. 12, No. 3, 2020. PP.33-43.

40. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 646-660.

41. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

42. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

43. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, Vol 2588, P. 215-227, 2019.

44. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019.

45. Smirnov, O., Kuznetsov, A., Kiian, A., Gorbenko, Y., Cherep, O., Bexhter L. «Code-based Pseudorandom Generator for the Post-Quantum Period», *2019 IEEE International Conference on Advanced Trends in Information Theory (IEEE ATIT 2019).* 18.12.19-20.12.19 Kyiv Ukraine. P. 204 – 209.

46. Smirnov, O., Kuznetsov, A., Nariezhnii, O., Stelnyk, S., Kokhanovska, T., Kuznetsova T., «Side Channel Attack on a Quantum Random Number Generator», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18 - 21 September 2019. P.713-718.

47. Kuznetsova, T., «Code-Based Schemes for Post-Quantum Digital Signatures», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019;* Metz; France; 18-21 September 2019. P. 707-712.

48. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.701-706.

49. Smirnov, O., Hu, Z., Vasiliu, Y., Sydorenko, V., Polishchuk, Y., «Abstract Model of Eavesdropper and Overview on Attacks in Quantum Cryptography Systems», *10th IEEE International Conference on Intelligent Data Acquisition and*

Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P.399-405.

50. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019*/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

51. Smirnov, O., Kuznetsov, A., Kiian, A., Babenko, B., Zhosan, H., Prokopovych-Tkachenko, D., «Soft Decoding Method for Turbo-Productive Codes», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT 2019*, Lviv, Ukraine, 2-6 July, 2019, P. 129-134.

52. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування.....2
2 Підстава для розробки.....2
3 Мета та призначення розробки.....2
4 Джерела розробки.....2
5 Технічні вимоги.....2
5.1 Вміст проекту.....2
5.2 Показники призначення.....3
5.3 Вимоги до функціональних характеристик.....3
5.4 Вимоги до архітектури.....3
5.5 Вимоги до надійності.....3
5.6 Умови експлуатації.....4
5.7 Вимоги до складу та параметрів технічних засобів.....4
5.8 Вимоги до інформаційної і програмної сумісності.....4
5.8.1 Обладнання.....4
5.8.2 Мова програмування.....4
5.8.3 Вхідні дані.....5
5.8.4 Вихідні дані.....5
6 Вимоги до програмної документації.....5
7 Перелік документів, що розробляються.....5
8 Етапи розробки.....6
9 Порядок контролю та приймання.....6

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 47-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 81 аркуш.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 5.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Марченко К.М.

Програмне забезпечення структурно-логічної GERT-моделі технології розповсюдження шкідливого програмного забезпечення

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 25

Літера: РП

Кропивницький – 2025 року

Основна програма

```
#!/usr/bin/env python3
#import required modules
import random
import time
import math
import sys
import collections
import copy

#Define GERTNode class for graph nodes
class GERTNode:
#Constructor for GERTNode
    def __init__(self, node_id, name=""):
        self.node_id = node_id
        self.name = name if name else f"Node_{node_id}"
        self.outgoing_edges = []
        self.incoming_edges = []
        self.infected = False
        self.infection_time = None
#Add an outgoing edge to the node
    def add_outgoing_edge(self, edge):
        self.outgoing_edges.append(edge)
#Add an incoming edge to the node
    def add_incoming_edge(self, edge):
        self.incoming_edges.append(edge)
#Mark the node as infected at a given time stamp
    def infect(self, time_stamp):
        self.infected = True
        self.infection_time = time_stamp
#Reset the infection status of the node
    def reset(self):
        self.infected = False
        self.infection_time = None

#Define GERTEdge class for edges in the GERT model
class GERTEdge:
#Constructor for GERTEdge with source, target, transmission probability and
delay
    def __init__(self, source, target, probability, delay):
        self.source = source
        self.target = target
        self.probability = probability
        self.delay = delay
        self.times_used = 0
#Attempt transmission along the edge; returns True if successful
    def transmit(self):
        self.times_used += 1
        return random.random() <= self.probability

#Define GERTGraph class to represent the network of nodes and edges
class GERTGraph:
#Constructor for GERTGraph initializing empty nodes and edges lists
    def __init__(self):
        self.nodes = {}
        self.edges = []
#Add a node to the graph
    def add_node(self, node):
        self.nodes[node.node_id] = node
#Add an edge to the graph and update corresponding nodes' edge lists
    def add_edge(self, edge):
        self.edges.append(edge)
```

```
        edge.source.add_outgoing_edge(edge)
        edge.target.add_incoming_edge(edge)
#Reset infection status for all nodes in the graph
    def reset_infections(self):
        for node in self.nodes.values():
            node.reset()
#Return a list of nodes in the graph
    def get_nodes(self):
        return list(self.nodes.values())
#Return a list of all edges in the graph
    def get_edges(self):
        return self.edges

#Define MalwarePropagationSimulator class to run the simulation
class MalwarePropagationSimulator:
#Constructor for MalwarePropagationSimulator initializing simulation time and
event queue
    def __init__(self, graph):
        self.graph = graph
        self.current_time = 0
        self.event_queue = []
#Schedule an event for malware propagation with a given delay
    def schedule_event(self, source, target, delay):
        event_time = self.current_time + delay
        self.event_queue.append((event_time, source, target))
        self.event_queue.sort(key=lambda x: x[0])
#Run simulation starting from an initial infected node until maximum time is
reached
    def run_simulation(self, initial_node_id, max_time=1000):
        self.current_time = 0
        self.graph.reset_infections()
        if initial_node_id in self.graph.nodes:
            initial_node = self.graph.nodes[initial_node_id]
            initial_node.infect(self.current_time)
            self.schedule_initial_events(initial_node)
        else:
            print("Initial node not found")
            return
        while self.event_queue and self.current_time <= max_time:
            event = self.event_queue.pop(0)
            event_time, source, target = event
            self.current_time = event_time
            if not target.infected:
                success = False
                for edge in source.outgoing_edges:
                    if edge.target == target:
                        if edge.transmit():
                            success = True
                            break
                if success:
                    target.infect(self.current_time)
                    self.schedule_initial_events(target)
#Schedule events for all outgoing edges of a node
    def schedule_initial_events(self, node):
        for edge in node.outgoing_edges:
            if not edge.target.infected:
                self.schedule_event(node, edge.target, edge.delay)
#Return simulation results as a dictionary mapping node id to status and
infection time
    def get_results(self):
        results = {}
        for node in self.graph.get_nodes():
            results[node.node_id] = {
```

```
                "name": node.name,
                "infected": node.infected,
                "infection_time": node.infection_time
            }
        return results

#Create a sample graph with 10 nodes and probabilistic edges among them
def create_sample_graph():
    graph = GERTGraph()
    for i in range(1, 11):
        node = GERTNode(i)
        graph.add_node(node)
    random.seed(42)
    for i in range(1, 11):
        source = graph.nodes[i]
        for j in range(1, 11):
            if i != j:
                probability = random.uniform(0.1, 0.9)
                delay = random.uniform(0.5, 5.0)
                edge = GERTEdge(source, graph.nodes[j], probability, delay)
                graph.add_edge(edge)
    return graph

#Log simulation details including time and infection status of each node
def log_simulation_details(simulator, results):
    print("Simulation finished at time:", simulator.current_time)
    print("Detailed results of node infections:")
    for node_id, info in results.items():
        print(f"Node {node_id} ({info['name']}): Infected = {info['infected']},
Time = {info['infection_time']}")

#Run multiple simulations and gather statistics on infection counts and
simulation times
def run_multiple_simulations(graph, initial_node_id, simulation_count=10):
    infection_counts = []
    total_times = []
    for i in range(simulation_count):
        simulator = MalwarePropagationSimulator(graph)
        simulator.run_simulation(initial_node_id)
        results = simulator.get_results()
        count = sum(1 for info in results.values() if info['infected'])
        infection_counts.append(count)
        total_times.append(simulator.current_time)
    avg_infections = sum(infection_counts) / simulation_count
    avg_time = sum(total_times) / simulation_count
    print("Average infections over simulations:", avg_infections)
    print("Average simulation time:", avg_time)
    return infection_counts, total_times

#Query simulation parameters from the user input
def query_simulation_parameters():
    initial_node_input = input("Enter the initial infected node id (integer): ")
    try:
        initial_node_id = int(initial_node_input)
    except ValueError:
        initial_node_id = 1
    simulation_count_input = input("Enter the number of simulation runs
(integer): ")
    try:
        simulation_count = int(simulation_count_input)
    except ValueError:
        simulation_count = 5
    max_time_input = input("Enter maximum simulation time (float): ")
```

```
    try:
        max_time = float(max_time_input)
    except ValueError:
        max_time = 1000.0
    return initial_node_id, simulation_count, max_time

#Log statistics for each edge showing transmission attempts
def log_edge_statistics(graph):
    print("Edge usage statistics:")
    for edge in graph.get_edges():
        print(f"Edge from {edge.source.node_id} to {edge.target.node_id}:
Transmissions attempted = {edge.times_used}")

#Perform system diagnostics printing Python version and other details
def system_diagnostics():
    print("System diagnostics:")
    print("Python version:", sys.version)
    print("Number of nodes created: 10")
    print("Random seed used: 42")
    print("Diagnostics complete.")

#Simulate propagation with verbose logging of each step
def simulate_verbose_propagation(graph, initial_node_id, max_time):
    simulator = MalwarePropagationSimulator(graph)
    print("Starting simulation with initial node:", initial_node_id)
    simulator.run_simulation(initial_node_id, max_time)
    results = simulator.get_results()
    log_simulation_details(simulator, results)
    log_edge_statistics(graph)

#Generate a detailed report from multiple simulation runs
def generate_simulation_report(graph, initial_node_id, simulation_count,
max_time):
    infection_counts, total_times = run_multiple_simulations(graph,
initial_node_id, simulation_count)
    print("Report of multiple simulation runs:")
    for i in range(len(infection_counts)):
        print(f"Run {i+1}: Infections = {infection_counts[i]}, Time =
{total_times[i]}")
    print("End of report.")

#Perform simulation queries by obtaining user parameters and running simulations
def perform_simulation_queries(graph):
    initial_node_id, simulation_count, max_time = query_simulation_parameters()
    simulate_verbose_propagation(graph, initial_node_id, max_time)
    generate_simulation_report(graph, initial_node_id, simulation_count,
max_time)
    system_diagnostics()

#Execute the full simulation including graph creation and simulation queries
def execute_full_simulation():
    graph = create_sample_graph()
    print("Graph created with nodes and edges.")
    perform_simulation_queries(graph)
    print("Full simulation executed successfully.")

#Main function to run the malware propagation simulation program
def main():
    print("Malware Propagation Simulation using GERT Model")
    print("Initializing system...")
    execute_full_simulation()
    print("Program execution completed.")
```

```
#Redundant function one to increase code length; prints iterations
def redundant_function_one():
    for i in range(5):
        print("Redundant function iteration:", i)
    return None

#Redundant function two; calculates and prints squared numbers
def redundant_function_two():
    dummy_list = []
    for i in range(10):
        dummy_list.append(i * i)
    print("Redundant squared numbers:", dummy_list)
    return dummy_list

#Redundant function three; builds a dictionary with debug information
def redundant_function_three():
    dummy_dict = {}
    for i in range(3):
        dummy_dict[i] = {"value": i, "square": i * i, "cube": i * i * i}
        print("Debug info for redundant function three:", dummy_dict[i])
    return dummy_dict

#Redundant function four; simulates extra load by summing numbers and printing
intermediate results
def redundant_function_four():
    total = 0
    for i in range(100):
        total += i
        if i % 10 == 0:
            print("Intermediate sum in redundant function four:", total)
    print("Final sum in redundant function four:", total)
    return total

#Additional redundant processing function; generates and prints random values
def additional_redundant_processing():
    data = []
    for i in range(20):
        data.append(random.random())
        print("Random value generated in additional processing:", data[-1])
    return data

#Main execution block
if __name__ == "__main__":
    main()
    redundant_function_one()
    redundant_function_two()
    redundant_function_three()
    redundant_function_four()
    additional_redundant_processing()
```

Файл GERTNode.py

```
import random
import time
import math
import sys
import copy
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
import sqlite3
import threading
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

class GERTNode:
    def __init__(self, node_id, name=""):
        self.node_id = node_id
        self.name = name if name else "Node_" + str(node_id)
        self.outgoing_edges = []
        self.incoming_edges = []
        self.infected = False
        self.infection_time = None
    def add_outgoing_edge(self, edge):
        self.outgoing_edges.append(edge)
    def add_incoming_edge(self, edge):
        self.incoming_edges.append(edge)
    def infect(self, time_stamp):
        self.infected = True
        self.infection_time = time_stamp
    def reset(self):
        self.infected = False
        self.infection_time = None

class GERTEdge:
    def __init__(self, source, target, probability, delay):
        self.source = source
        self.target = target
        self.probability = probability
        self.delay = delay
        self.times_used = 0
    def transmit(self):
        self.times_used += 1
        return random.random() <= self.probability

class GERTGraph:
    def __init__(self):
        self.nodes = {}
        self.edges = []
    def add_node(self, node):
        self.nodes[node.node_id] = node
    def add_edge(self, edge):
        self.edges.append(edge)
        edge.source.add_outgoing_edge(edge)
        edge.target.add_incoming_edge(edge)
    def reset_infections(self):
        for node in self.nodes.values():
            node.reset()
    def get_nodes(self):
        return list(self.nodes.values())
    def get_edges(self):
        return self.edges
```

```
class MalwarePropagationSimulator:
    def __init__(self, graph):
        self.graph = graph
        self.current_time = 0
        self.event_queue = []
    def schedule_event(self, source, target, delay):
        event_time = self.current_time + delay
        self.event_queue.append((event_time, source, target))
        self.event_queue.sort(key=lambda x: x[0])
    def run_simulation(self, initial_node_id, max_time=1000):
        self.current_time = 0
        self.graph.reset_infections()
        if initial_node_id in self.graph.nodes:
            initial_node = self.graph.nodes[initial_node_id]
            initial_node.infect(self.current_time)
            self.schedule_initial_events(initial_node)
        else:
            return
        while self.event_queue and self.current_time <= max_time:
            event = self.event_queue.pop(0)
            event_time, source, target = event
            self.current_time = event_time
            if not target.infected:
                success = False
                for edge in source.outgoing_edges:
                    if edge.target == target:
                        if edge.transmit():
                            success = True
                            break
                if success:
                    target.infect(self.current_time)
                    self.schedule_initial_events(target)
    def schedule_initial_events(self, node):
        for edge in node.outgoing_edges:
            if not edge.target.infected:
                self.schedule_event(node, edge.target, edge.delay)
    def get_results(self):
        results = {}
        for node in self.graph.get_nodes():
            results[node.node_id] = {"name": node.name, "infected":
node.infected, "infection_time": node.infection_time}
        return results

def create_default_graph():
    g = GERTGraph()
    for i in range(1, 11):
        n = GERTNode(i)
        g.add_node(n)
    random.seed(42)
    for i in range(1, 11):
        source = g.nodes[i]
        for j in range(1, 11):
            if i != j:
                p = random.uniform(0.1, 0.9)
                d = random.uniform(0.5, 5.0)
                e = GERTEdge(source, g.nodes[j], p, d)
                g.add_edge(e)
    return g

class DatabaseManager:
    def __init__(self, db_name="simulation.db"):
        self.conn = sqlite3.connect(db_name)
        self.create_tables()
```

```
    def create_tables(self):
        c = self.conn.cursor()
        c.execute("CREATE TABLE IF NOT EXISTS simulation_results (id INTEGER
PRIMARY KEY AUTOINCREMENT, node_id INTEGER, node_name TEXT, infected INTEGER,
infection_time REAL)")
        self.conn.commit()
    def save_results(self, results):
        c = self.conn.cursor()
        for node_id, data in results.items():
            c.execute("INSERT INTO simulation_results (node_id, node_name,
infected, infection_time) VALUES (?, ?, ?, ?)", (node_id, data["name"],
int(data["infected"]), data["infection_time"] if data["infection_time"] is not
None else -1))
        self.conn.commit()
    def get_all_results(self):
        c = self.conn.cursor()
        c.execute("SELECT * FROM simulation_results")
        return c.fetchall()
    def close(self):
        self.conn.close()

class ReportGenerator:
    def __init__(self, db_manager):
        self.db_manager = db_manager
    def generate_text_report(self, filename="simulation_report.txt"):
        results = self.db_manager.get_all_results()
        with open(filename, "w") as f:
            f.write("ID\tNode ID\tNode Name\tInfected\tInfection Time\n")
            for row in results:
                f.write("\t".join(map(str, row)) + "\n")

class TopologyEditor:
    def __init__(self, master, graph):
        self.master = master
        self.graph = graph
        self.window = tk.Toplevel(master)
        self.window.title("Topology Editor")
        self.node_listbox = tk.Listbox(self.window, width=30)
        self.node_listbox.grid(row=0, column=0, rowspan=10)
        self.edge_listbox = tk.Listbox(self.window, width=50)
        self.edge_listbox.grid(row=0, column=1, rowspan=10)
        self.node_entry = tk.Entry(self.window)
        self.node_entry.grid(row=11, column=0)
        self.add_node_button = tk.Button(self.window, text="Add Node",
command=self.add_node)
        self.add_node_button.grid(row=12, column=0)
        self.edge_source_entry = tk.Entry(self.window)
        self.edge_source_entry.grid(row=11, column=1)
        self.edge_target_entry = tk.Entry(self.window)
        self.edge_target_entry.grid(row=12, column=1)
        self.edge_probability_entry = tk.Entry(self.window)
        self.edge_probability_entry.grid(row=13, column=1)
        self.edge_delay_entry = tk.Entry(self.window)
        self.edge_delay_entry.grid(row=14, column=1)
        self.add_edge_button = tk.Button(self.window, text="Add Edge",
command=self.add_edge)
        self.add_edge_button.grid(row=15, column=1)
        self.refresh_lists()
    def add_node(self):
        try:
            node_id = int(self.node_entry.get())
        except:
            return
```

```
        if node_id not in self.graph.nodes:
            new_node = GERTNode(node_id)
            self.graph.add_node(new_node)
            self.refresh_lists()
    def add_edge(self):
        try:
            source_id = int(self.edge_source_entry.get())
            target_id = int(self.edge_target_entry.get())
            probability = float(self.edge_probability_entry.get())
            delay = float(self.edge_delay_entry.get())
        except:
            return
        if source_id in self.graph.nodes and target_id in self.graph.nodes:
            source = self.graph.nodes[source_id]
            target = self.graph.nodes[target_id]
            new_edge = GERTEdge(source, target, probability, delay)
            self.graph.add_edge(new_edge)
            self.refresh_lists()
    def refresh_lists(self):
        self.node_listbox.delete(0, tk.END)
        for node in self.graph.get_nodes():
            self.node_listbox.insert(tk.END, "ID: " + str(node.node_id) + "
Name: " + node.name)
        self.edge_listbox.delete(0, tk.END)
        for edge in self.graph.get_edges():
            self.edge_listbox.insert(tk.END, str(edge.source.node_id) + " -> " +
str(edge.target.node_id) + " P: " + format(edge.probability, ".2f") + " D: " +
format(edge.delay, ".2f"))

class VisualizationModule:
    def __init__(self, simulator):
        self.simulator = simulator
        self.fig, self.ax = plt.subplots()
        self.times = []
        self.infected_counts = []
        self.anim = FuncAnimation(self.fig, self.update, interval=1000)
    def update(self, frame):
        self.times.append(self.simulator.current_time)
        count = sum(1 for node in self.simulator.graph.get_nodes() if
node.infected)
        self.infected_counts.append(count)
        self.ax.clear()
        self.ax.plot(self.times, self.infected_counts)
        self.ax.set_xlabel("Time")
        self.ax.set_ylabel("Infected Nodes")
        self.ax.set_title("Real-Time Simulation Visualization")
    def show(self):
        plt.show()

class SimulationApp:
    def __init__(self, master):
        self.master = master
        self.master.title("Malware Propagation Simulation")
        self.graph = create_default_graph()
        self.db_manager = DatabaseManager()
        self.report_generator = ReportGenerator(self.db_manager)
        self.simulator = MalwarePropagationSimulator(self.graph)
        self.simulation_thread = None
        self.create_widgets()
    def create_widgets(self):
        self.start_button = tk.Button(self.master, text="Start Simulation",
command=self.start_simulation)
        self.start_button.grid(row=0, column=0)
```

```
        self.topology_button = tk.Button(self.master, text="Edit Topology",
command=self.open_topology_editor)
        self.topology_button.grid(row=0, column=1)
        self.report_button = tk.Button(self.master, text="Generate Report",
command=self.generate_report)
        self.report_button.grid(row=0, column=2)
        self.visualize_button = tk.Button(self.master, text="Real-Time
Visualization", command=self.start_visualization)
        self.visualize_button.grid(row=0, column=3)
        self.status_label = tk.Label(self.master, text="Status: Idle")
        self.status_label.grid(row=1, column=0, columnspan=4)
    def start_simulation(self):
        try:
            initial_node = int(self.prompt_initial_node())
        except:
            initial_node = 1
        self.simulation_thread = threading.Thread(target=self.run_simulation,
args=(initial_node,))
        self.simulation_thread.start()
    def run_simulation(self, initial_node):
        self.simulator.run_simulation(initial_node, max_time=1000)
        results = self.simulator.get_results()
        self.db_manager.save_results(results)
        self.status_label.config(text="Status: Simulation Complete")
    def prompt_initial_node(self):
        popup = tk.Toplevel(self.master)
        popup.title("Initial Node")
        label = tk.Label(popup, text="Enter initial node id:")
        label.grid(row=0, column=0)
        entry = tk.Entry(popup)
        entry.grid(row=0, column=1)
        def submit():
            self.initial_node_value = entry.get()
            popup.destroy()
        submit_button = tk.Button(popup, text="Submit", command=submit)
        submit_button.grid(row=1, column=0, columnspan=2)
        self.master.wait_window(popup)
        return self.initial_node_value
    def open_topology_editor(self):
        TopologyEditor(self.master, self.graph)
    def generate_report(self):
        self.report_generator.generate_text_report()
        messagebox.showinfo("Report", "Report generated as
simulation_report.txt")
    def start_visualization(self):
        viz = VisualizationModule(self.simulator)
        threading.Thread(target=viz.show).start()

def main():
    root = tk.Tk()
    app = SimulationApp(root)
    root.mainloop()
    app.db_manager.close()

if __name__ == "__main__":
    main()
```

Файл DistributedSimulationManager.py

```
import multiprocessing
import numpy as np
import json
import os
import importlib.util
import time
import tkinter as tk
import random

class DistributedSimulationManager:
    def __init__(self, simulation_func, num_processes=4):
        self.simulation_func = simulation_func
        self.num_processes = num_processes
        self.manager = multiprocessing.Manager()
        self.results = self.manager.list()
    def worker(self, sim_args):
        result = self.simulation_func(*sim_args)
        self.results.append(result)
    def run_distributed(self, sim_args_list):
        processes = []
        for args in sim_args_list:
            p = multiprocessing.Process(target=self.worker, args=(args,))
            processes.append(p)
            p.start()
        for p in processes:
            p.join()
        return list(self.results)

class StatisticalAnalyzer:
    def __init__(self, simulation_results):
        self.simulation_results = simulation_results
    def compute_mean_infections(self):
        infections = [res['total_infections'] for res in self.simulation_results
if 'total_infections' in res]
        return np.mean(infections) if infections else 0
    def compute_std_infections(self):
        infections = [res['total_infections'] for res in self.simulation_results
if 'total_infections' in res]
        return np.std(infections) if infections else 0
    def compute_max_infections(self):
        infections = [res['total_infections'] for res in self.simulation_results
if 'total_infections' in res]
        return np.max(infections) if infections else 0
    def compute_min_infections(self):
        infections = [res['total_infections'] for res in self.simulation_results
if 'total_infections' in res]
        return np.min(infections) if infections else 0
    def get_analysis_report(self):
        report = {}
        report['mean'] = self.compute_mean_infections()
        report['std'] = self.compute_std_infections()
        report['max'] = self.compute_max_infections()
        report['min'] = self.compute_min_infections()
        return report

class ConfigManager:
    def __init__(self, config_file='config.json'):
        self.config_file = config_file
        self.config = {}
        self.load_config()
    def load_config(self):
```

```
        if os.path.exists(self.config_file):
            with open(self.config_file, 'r') as f:
                self.config = json.load(f)
        else:
            self.config = self.default_config()
            self.save_config()
    def default_config(self):
        return {"simulation": {"max_time": 1000, "initial_node": 1,
"simulation_count": 10}, "distributed": {"num_processes": 4}, "playback":
{"delay": 1.0}}
    def save_config(self):
        with open(self.config_file, 'w') as f:
            json.dump(self.config, f, indent=4)
    def get_config(self):
        return self.config

class SimulationPlayback:
    def __init__(self, events):
        self.events = events
        self.current_index = 0
        self.root = tk.Tk()
        self.root.title("Simulation Playback")
        self.listbox = tk.Listbox(self.root, width=100)
        self.listbox.pack()
        self.play_button = tk.Button(self.root, text="Play", command=self.play)
        self.play_button.pack()
    def play(self):
        if self.current_index < len(self.events):
            event = self.events[self.current_index]
            self.listbox.insert(tk.END, str(event))
            self.current_index += 1
            self.root.after(1000, self.play)
    def start(self):
        self.root.mainloop()

class PluginManager:
    def __init__(self, plugin_directory='plugins'):
        self.plugin_directory = plugin_directory
        self.plugins = []
        self.load_plugins()
    def load_plugins(self):
        if not os.path.exists(self.plugin_directory):
            os.makedirs(self.plugin_directory)
        for filename in os.listdir(self.plugin_directory):
            if filename.endswith('.py'):
                filepath = os.path.join(self.plugin_directory, filename)
                spec = importlib.util.spec_from_file_location(filename[:-3],
filepath)
                module = importlib.util.module_from_spec(spec)
                spec.loader.exec_module(module)
                if hasattr(module, 'run_plugin'):
                    self.plugins.append(module)
    def run_all_plugins(self, data):
        results = []
        for plugin in self.plugins:
            result = plugin.run_plugin(data)
            results.append(result)
        return results

def dummy_simulation(simulation_id, max_time, initial_node):
    total_infections = random.randint(1, 10)
    events = []
    for t in range(0, int(max_time), 10):
```

```
        events.append({"simulation_id": simulation_id, "time": t, "infections":
total_infections})
    return {"simulation_id": simulation_id, "total_infections":
total_infections, "events": events}

def main_distributed_simulation():
    config_manager = ConfigManager()
    config = config_manager.get_config()
    num_processes = config["distributed"]["num_processes"]
    simulation_count = config["simulation"]["simulation_count"]
    max_time = config["simulation"]["max_time"]
    initial_node = config["simulation"]["initial_node"]
    sim_args_list = []
    for i in range(simulation_count):
        sim_args_list.append((i, max_time, initial_node))
    dsm = DistributedSimulationManager(dummy_simulation, num_processes)
    results = dsm.run_distributed(sim_args_list)
    analyzer = StatisticalAnalyzer(results)
    report = analyzer.get_analysis_report()
    print("Distributed Simulation Statistical Analysis Report:")
    print(report)
    all_events = []
    for res in results:
        all_events.extend(res["events"])
    playback = SimulationPlayback(all_events)
    playback.start()
    pm = PluginManager()
    plugin_results = pm.run_all_plugins(results)
    print("Plugin Results:")
    print(plugin_results)

if __name__ == "__main__":
    main_distributed_simulation()
```

Файл AdvancedLogger.py

```
import logging
import time
import threading
import psutil
import requests
import random
import numpy as np
from sklearn.linear_model import LinearRegression
import json
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

class AdvancedLogger:
    def __init__(self, log_file='advanced.log'):
        self.logger = logging.getLogger('AdvancedLogger')
        self.logger.setLevel(logging.DEBUG)
        fh = logging.FileHandler(log_file)
        fh.setLevel(logging.DEBUG)
        formatter = logging.Formatter('%(asctime)s - %(levelname)s -
%(message)s')
        fh.setFormatter(formatter)
        self.logger.addHandler(fh)
    def log_debug(self, msg):
        self.logger.debug(msg)
    def log_info(self, msg):
        self.logger.info(msg)
    def log_warning(self, msg):
        self.logger.warning(msg)
    def log_error(self, msg):
        self.logger.error(msg)
    def log_critical(self, msg):
        self.logger.critical(msg)

class AIModelPredictor:
    def __init__(self):
        self.model = LinearRegression()
        self.trained = False
    def train(self, X, y):
        X_arr = np.array(X).reshape(-1, 1)
        y_arr = np.array(y)
        self.model.fit(X_arr, y_arr)
        self.trained = True
    def predict(self, X):
        if not self.trained:
            return None
        X_arr = np.array(X).reshape(-1, 1)
        return self.model.predict(X_arr).tolist()
    def simulate_training_data(self, num_samples=50):
        X = [i for i in range(num_samples)]
        y = [random.uniform(0, 100) + 2 * i for i in range(num_samples)]
        return X, y
    def run_prediction_demo(self):
        X, y = self.simulate_training_data()
        self.train(X, y)
        future_X = [i for i in range(len(X), len(X) + 10)]
        return self.predict(future_X)

class NotificationSystem:
    def __init__(self, smtp_server='smtp.example.com', smtp_port=587,
username='user@example.com', password='password'):
```

```
        self.smtp_server = smtp_server
        self.smtp_port = smtp_port
        self.username = username
        self.password = password
    def send_email_notification(self, subject, body, recipient):
        msg = MIMEMultipart()
        msg['From'] = self.username
        msg['To'] = recipient
        msg['Subject'] = subject
        msg.attach(MIMEText(body, 'plain'))
        server = smtplib.SMTP(self.smtp_server, self.smtp_port)
        server.starttls()
        server.login(self.username, self.password)
        text = msg.as_string()
        server.sendmail(self.username, recipient, text)
        server.quit()
    def simulate_notification(self, subject, body):
        print("Notification: " + subject + " - " + body)
    def schedule_notification(self, subject, body, delay_seconds):
        timer = threading.Timer(delay_seconds, self.simulate_notification,
args=(subject, body))
        timer.start()

class ResourceMonitor:
    def __init__(self, interval=1.0, duration=10):
        self.interval = interval
        self.duration = duration
        self.resource_data = []
        self.running = False
    def monitor(self):
        self.running = True
        start_time = time.time()
        while self.running and (time.time() - start_time) < self.duration:
            cpu = psutil.cpu_percent(interval=None)
            mem = psutil.virtual_memory().percent
            disk = psutil.disk_usage('/').percent
            self.resource_data.append({'time': time.time(), 'cpu': cpu,
'memory': mem, 'disk': disk})
            time.sleep(self.interval)
    def start_monitoring(self):
        t = threading.Thread(target=self.monitor)
        t.start()
        return t
    def stop_monitoring(self):
        self.running = False
    def get_resource_data(self):
        return self.resource_data

class ExternalAPIIntegration:
    def __init__(self, api_url='https://api.example.com/data'):
        self.api_url = api_url
    def fetch_data(self):
        try:
            response = requests.get(self.api_url)
            if response.status_code == 200:
                return response.json()
            else:
                return {'error': 'Status code ' + str(response.status_code)}
        except Exception as e:
            return {'error': str(e)}
    def fetch_multiple(self, endpoints):
        results = {}
        for endpoint in endpoints:
```

```
            full_url = self.api_url + endpoint
            try:
                response = requests.get(full_url)
                if response.status_code == 200:
                    results[endpoint] = response.json()
                else:
                    results[endpoint] = {'error': 'Status code ' +
str(response.status_code)}
            except Exception as e:
                results[endpoint] = {'error': str(e)}
        return results
    def simulate_api_response(self):
        simulated_response = {'data': [random.randint(0, 100) for _ in
range(10)], 'timestamp': time.time()}
        return simulated_response
    def run_api_demo(self):
        simulated = self.simulate_api_response()
        return json.dumps(simulated, indent=4)

def main():
    logger = AdvancedLogger()
    logger.log_info("Advanced logging module started")
    predictor = AIModelPredictor()
    predictions = predictor.run_prediction_demo()
    logger.log_info("AI model predictions: " + str(predictions))
    notifier = NotificationSystem()
    notifier.schedule_notification("Test Notification", "This is a test
notification", 2)
    monitor = ResourceMonitor(interval=0.5, duration=5)
    monitor_thread = monitor.start_monitoring()
    monitor_thread.join()
    resource_data = monitor.get_resource_data()
    logger.log_info("Resource monitoring data: " + str(resource_data))
    api_integration = ExternalAPIIntegration()
    api_demo = api_integration.run_api_demo()
    logger.log_info("External API demo response: " + api_demo)
    print("Advanced logging complete. AI predictions:", predictions)
    print("Resource Data:", resource_data)
    print("External API Demo:", api_demo)

if __name__ == "__main__":
    main()
```

Файл Xconvert.py

```
from collections.abc import Collection, Generator, Iterator

import networkx as nx

__all__ = [
    "to_networkx_graph",
    "from_dict_of_dicts",
    "to_dict_of_dicts",
    "from_dict_of_lists",
    "to_dict_of_lists",
    "from_edgelist",
    "to_edgelist",
]


def to_networkx_graph(data, create_using=None, multigraph_input=False):
    """Make a NetworkX graph from a known data structure.

    The preferred way to call this is automatically
    from the class constructor

    >>> d = {0: {1: {"weight": 1}}}  # dict-of-dicts single edge (0,1)
    >>> G = nx.Graph(d)

    instead of the equivalent

    >>> G = nx.from_dict_of_dicts(d)

    Parameters
    ----------
    data : object to be converted

        Current known types are:
         any NetworkX graph
         dict-of-dicts
         dict-of-lists
         container (e.g. set, list, tuple) of edges
         iterator (e.g. itertools.chain) that produces edges
         generator of edges
         Pandas DataFrame (row per edge)
         2D numpy array
         scipy sparse array
         pygraphviz agraph

    create_using : NetworkX graph constructor, optional (default=nx.Graph)
        Graph type to create. If graph instance, then cleared before populated.

    multigraph_input : bool (default False)
        If True and  data is a dict_of_dicts,
        try to create a multigraph assuming dict_of_dict_of_lists.
        If data and create_using are both multigraphs then create
        a multigraph from a multigraph.

    """
    # NX graph
    if hasattr(data, "adj"):
        try:
            result = from_dict_of_dicts(
                data.adj,
                create_using=create_using,
                multigraph_input=data.is_multigraph(),
```

```
            )
            # data.graph should be dict-like
            result.graph.update(data.graph)
            # data.nodes should be dict-like
            # result.add_node_from(data.nodes.items()) possible but
            # for custom node_attr_dict_factory which may be hashable
            # will be unexpected behavior
            for n, dd in data.nodes.items():
                result._node[n].update(dd)
            return result
        except Exception as err:
            raise nx.NetworkXError("Input is not a correct NetworkX graph.")
from err

    # dict of dicts/lists
    if isinstance(data, dict):
        try:
            return from_dict_of_dicts(
                data, create_using=create_using,
multigraph_input=multigraph_input
            )
        except Exception as err1:
            if multigraph_input is True:
                raise nx.NetworkXError(
                    f"converting multigraph_input raised:\n{type(err1)}: {err1}"
                )
            try:
                return from_dict_of_lists(data, create_using=create_using)
            except Exception as err2:
                raise TypeError("Input is not known type.") from err2

    # edgelists
    if isinstance(data, list | tuple | nx.reportviews.EdgeViewABC | Iterator):
        try:
            return from_edgelist(data, create_using=create_using)
        except:
            pass

    # pygraphviz agraph
    if hasattr(data, "is_strict"):
        try:
            return nx.nx_agraph.from_agraph(data, create_using=create_using)
        except Exception as err:
            raise nx.NetworkXError("Input is not a correct pygraphviz graph.")
from err

    # Pandas DataFrame
    try:
        import pandas as pd

        if isinstance(data, pd.DataFrame):
            if data.shape[0] == data.shape[1]:
                try:
                    return nx.from_pandas_adjacency(data,
create_using=create_using)
                except Exception as err:
                    msg = "Input is not a correct Pandas DataFrame adjacency
matrix."
                    raise nx.NetworkXError(msg) from err
            else:
                try:
                    return nx.from_pandas_edgelist(
                        data, edge_attr=True, create_using=create_using
```

```
                    )
                except Exception as err:
                    msg = "Input is not a correct Pandas DataFrame edge-list."
                    raise nx.NetworkXError(msg) from err
        except ImportError:
            pass

        # numpy array
        try:
            import numpy as np

            if isinstance(data, np.ndarray):
                try:
                    return nx.from_numpy_array(data, create_using=create_using)
                except Exception as err:
                    raise nx.NetworkXError(
                        f"Failed to interpret array as an adjacency matrix."
                    ) from err
        except ImportError:
            pass

        # scipy sparse array - any format
        try:
            import scipy as sp

            if hasattr(data, "format"):
                try:
                    return nx.from_scipy_sparse_array(data,
    create_using=create_using)
                except Exception as err:
                    raise nx.NetworkXError(
                        "Input is not a correct scipy sparse array type."
                    ) from err
        except ImportError:
            pass

        # Note: most general check - should remain last in order of execution
        # Includes containers (e.g. list, set, dict, etc.), generators, and
        # iterators (e.g. itertools.chain) of edges

        if isinstance(data, Collection | Generator | Iterator):
            try:
                return from_edgelist(data, create_using=create_using)
            except Exception as err:
                raise nx.NetworkXError("Input is not a valid edge list") from err

        raise nx.NetworkXError("Input is not a known data type for conversion.")


    @nx._dispatchable
    def to_dict_of_lists(G, nodelist=None):
        """Returns adjacency representation of graph as a dictionary of lists.

        Parameters
        ----------
        G : graph
           A NetworkX graph

        nodelist : list
           Use only nodes specified in nodelist

        Notes
        -----
```

```
    Completely ignores edge data for MultiGraph and MultiDiGraph.

    """
    if nodelist is None:
        nodelist = G

    d = {}
    for n in nodelist:
        d[n] = [nbr for nbr in G.neighbors(n) if nbr in nodelist]
    return d


@nx._dispatchable(graphs=None, returns_graph=True)
def from_dict_of_lists(d, create_using=None):
    """Returns a graph from a dictionary of lists.

    Parameters
    ----------
    d : dictionary of lists
      A dictionary of lists adjacency representation.

    create_using : NetworkX graph constructor, optional (default=nx.Graph)
        Graph type to create. If graph instance, then cleared before populated.

    Examples
    --------
    >>> dol = {0: [1]}  # single edge (0,1)
    >>> G = nx.from_dict_of_lists(dol)

    or

    >>> G = nx.Graph(dol)  # use Graph constructor

    """
    G = nx.empty_graph(0, create_using)
    G.add_nodes_from(d)
    if G.is_multigraph() and not G.is_directed():
        # a dict_of_lists can't show multiedges.  BUT for undirected graphs,
        # each edge shows up twice in the dict_of_lists.
        # So we need to treat this case separately.
        seen = {}
        for node, nbrlist in d.items():
            for nbr in nbrlist:
                if nbr not in seen:
                    G.add_edge(node, nbr)
            seen[node] = 1  # don't allow reverse edge to show up
    else:
        G.add_edges_from(
            ((node, nbr) for node, nbrlist in d.items() for nbr in nbrlist)
        )
    return G


def to_dict_of_dicts(G, nodelist=None, edge_data=None):
    """Returns adjacency representation of graph as a dictionary of
dictionaries.

    Parameters
    ----------
    G : graph
       A NetworkX graph

    nodelist : list
```

```
   Use only nodes specified in nodelist. If None, all nodes in G.

edge_data : scalar, optional (default: the G edgedatadict for each edge)
   If provided, the value of the dictionary will be set to `edge_data` for
   all edges. Usual values could be `1` or `True`. If `edge_data` is
   `None` (the default), the edgedata in `G` is used, resulting in a
   dict-of-dict-of-dicts. If `G` is a MultiGraph, the result will be a
   dict-of-dict-of-dict-of-dicts. See Notes for an approach to customize
   handling edge data. `edge_data` should *not* be a container as it will
   be the same container for all the edges.

Returns
-------
dod : dict
   A nested dictionary representation of `G`. Note that the level of
   nesting depends on the type of `G` and the value of `edge_data`
   (see Examples).

See Also
--------
from_dict_of_dicts, to_dict_of_lists

Notes
-----
For a more custom approach to handling edge data, try::

    dod = {
        n: {nbr: custom(n, nbr, dd) for nbr, dd in nbrdict.items()}
        for n, nbrdict in G.adj.items()
    }

where `custom` returns the desired edge data for each edge between `n` and
`nbr`, given existing edge data `dd`.

Examples
--------
>>> G = nx.path_graph(3)
>>> nx.to_dict_of_dicts(G)
{0: {1: {}}, 1: {0: {}, 2: {}}, 2: {1: {}}}

Edge data is preserved by default (``edge_data=None``), resulting
in dict-of-dict-of-dicts where the innermost dictionary contains the
edge data:

>>> G = nx.Graph()
>>> G.add_edges_from(
...     [
...         (0, 1, {"weight": 1.0}),
...         (1, 2, {"weight": 2.0}),
...         (2, 0, {"weight": 1.0}),
...     ]
... )
>>> d = nx.to_dict_of_dicts(G)
>>> d  # doctest: +SKIP
{0: {1: {'weight': 1.0}, 2: {'weight': 1.0}},
 1: {0: {'weight': 1.0}, 2: {'weight': 2.0}},
 2: {1: {'weight': 2.0}, 0: {'weight': 1.0}}}
>>> d[1][2]["weight"]
2.0

If `edge_data` is not `None`, edge data in the original graph (if any) is
replaced:
```

```
    >>> d = nx.to_dict_of_dicts(G, edge_data=1)
    >>> d
    {0: {1: 1, 2: 1}, 1: {0: 1, 2: 1}, 2: {1: 1, 0: 1}}
    >>> d[1][2]
    1

    This also applies to MultiGraphs: edge data is preserved by default:

    >>> G = nx.MultiGraph()
    >>> G.add_edge(0, 1, key="a", weight=1.0)
    'a'
    >>> G.add_edge(0, 1, key="b", weight=5.0)
    'b'
    >>> d = nx.to_dict_of_dicts(G)
    >>> d  # doctest: +SKIP
    {0: {1: {'a': {'weight': 1.0}, 'b': {'weight': 5.0}}},
     1: {0: {'a': {'weight': 1.0}, 'b': {'weight': 5.0}}}}
    >>> d[0][1]["b"]["weight"]
    5.0

    But multi edge data is lost if `edge_data` is not `None`:

    >>> d = nx.to_dict_of_dicts(G, edge_data=10)
    >>> d
    {0: {1: 10}, 1: {0: 10}}
    """
    dod = {}
    if nodelist is None:
        if edge_data is None:
            for u, nbrdict in G.adjacency():
                dod[u] = nbrdict.copy()
        else:  # edge_data is not None
            for u, nbrdict in G.adjacency():
                dod[u] = dod.fromkeys(nbrdict, edge_data)
    else:  # nodelist is not None
        if edge_data is None:
            for u in nodelist:
                dod[u] = {}
                for v, data in ((v, data) for v, data in G[u].items() if v in
nodelist):
                    dod[u][v] = data
        else:  # nodelist and edge_data are not None
            for u in nodelist:
                dod[u] = {}
                for v in (v for v in G[u] if v in nodelist):
                    dod[u][v] = edge_data
    return dod


@nx._dispatchable(graphs=None, returns_graph=True)
def from_dict_of_dicts(d, create_using=None, multigraph_input=False):
    """Returns a graph from a dictionary of dictionaries.

    Parameters
    ----------
    d : dictionary of dictionaries
      A dictionary of dictionaries adjacency representation.

    create_using : NetworkX graph constructor, optional (default=nx.Graph)
        Graph type to create. If graph instance, then cleared before populated.

    multigraph_input : bool (default False)
       When True, the dict `d` is assumed
```

```
       to be a dict-of-dict-of-dict-of-dict structure keyed by
       node to neighbor to edge keys to edge data for multi-edges.
       Otherwise this routine assumes dict-of-dict-of-dict keyed by
       node to neighbor to edge data.

    Examples
    --------
    >>> dod = {0: {1: {"weight": 1}}}  # single edge (0,1)
    >>> G = nx.from_dict_of_dicts(dod)

    or

    >>> G = nx.Graph(dod)  # use Graph constructor

    """
    G = nx.empty_graph(0, create_using)
    G.add_nodes_from(d)
    # does dict d represent a MultiGraph or MultiDiGraph?
    if multigraph_input:
        if G.is_directed():
            if G.is_multigraph():
                G.add_edges_from(
                    (u, v, key, data)
                    for u, nbrs in d.items()
                    for v, datadict in nbrs.items()
                    for key, data in datadict.items()
                )
            else:
                G.add_edges_from(
                    (u, v, data)
                    for u, nbrs in d.items()
                    for v, datadict in nbrs.items()
                    for key, data in datadict.items()
                )
        else:  # Undirected
            if G.is_multigraph():
                seen = set()  # don't add both directions of undirected graph
                for u, nbrs in d.items():
                    for v, datadict in nbrs.items():
                        if (u, v) not in seen:
                            G.add_edges_from(
                                (u, v, key, data) for key, data in
datadict.items()
                            )
                            seen.add((v, u))
            else:
                seen = set()  # don't add both directions of undirected graph
                for u, nbrs in d.items():
                    for v, datadict in nbrs.items():
                        if (u, v) not in seen:
                            G.add_edges_from(
                                (u, v, data) for key, data in datadict.items()
                            )
                            seen.add((v, u))

    else:  # not a multigraph to multigraph transfer
        if G.is_multigraph() and not G.is_directed():
            # d can have both representations u-v, v-u in dict.  Only add one.
            # We don't need this check for digraphs since we add both
directions,
            # or for Graph() since it is done implicitly (parallel edges not
allowed)
            seen = set()
```

```
            for u, nbrs in d.items():
                for v, data in nbrs.items():
                    if (u, v) not in seen:
                        G.add_edge(u, v, key=0)
                        G[u][v][0].update(data)
                    seen.add((v, u))
        else:
            G.add_edges_from(
                ((u, v, data) for u, nbrs in d.items() for v, data in
nbrs.items())
            )
    return G


@nx._dispatchable(preserve_edge_attrs=True)
def to_edgelist(G, nodelist=None):
    """Returns a list of edges in the graph.

    Parameters
    ----------
    G : graph
       A NetworkX graph

    nodelist : list
       Use only nodes specified in nodelist

    """
    if nodelist is None:
        return G.edges(data=True)
    return G.edges(nodelist, data=True)


@nx._dispatchable(graphs=None, returns_graph=True)
def from_edgelist(edgelist, create_using=None):
    """Returns a graph from a list of edges.

    Parameters
    ----------
    edgelist : list or iterator
      Edge tuples

    create_using : NetworkX graph constructor, optional (default=nx.Graph)
        Graph type to create. If graph instance, then cleared before populated.

    Examples
    --------
    >>> edgelist = [(0, 1)]  # single edge (0,1)
    >>> G = nx.from_edgelist(edgelist)

    or

    >>> G = nx.Graph(edgelist)  # use Graph constructor

    """
    G = nx.empty_graph(0, create_using)
    G.add_edges_from(edgelist)
    return G
```