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

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“ Програмне забезпечення для моделювання роботи комп’ютерних мереж”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-2
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Наконечний Б. М.
«____» _____ 20 р.

Керівник проекту
доктор технічних наук, професор
_____ Мелешко Є. В.
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти

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.
_____О.А.Смірнов
« __ » _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Наконечному Богдану Максимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення для моделювання роботи комп'ютерних мереж*

керівник роботи *Мелешко Єлизавета Владиславівна, д-р техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

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

2. Строк подання студентом роботи до захисту *24.05.2025 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення для моделювання роботи комп'ютерних мереж*

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

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

5. 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*

6. Дата видачі завдання 17.01.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	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	22.05.2025 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Наконечний Б. М. Програмне забезпечення для моделювання роботи комп'ютерних мереж. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення, яке призначено для моделювання роботи комп'ютерних мереж.

Метою роботи є створення системи для моделювання роботи комп'ютерних мереж.

Результат роботи – програмна реалізація системи моделювання роботи комп'ютерних мереж.

В процесі роботи над реалізацією системи виконано дослідження існуючих методів, алгоритмів та програмних засобів. Розроблено та реалізовано власне програмне забезпечення, здійснено опис всіх його компонентів.

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

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

Програму розроблено на мові програмування Python.

Ключові слова: комп'ютерна інженерія, комп'ютерні мережі, імітаційне моделювання

ABSTRACT

Nakonechnyi B. M. Software for simulating computer networks. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi 2025.

In this qualification bachelor's thesis, software has been developed that is intended for modeling the operation of computer networks.

The purpose of the work is to create a system for modeling the operation of computer networks.

The result of the work is a software implementation of a system for modeling the operation of computer networks.

In the process of working on the implementation of the system, a study of existing methods, algorithms and software tools was carried out. The software itself was developed and implemented, and all its components were described.

A convenient user interface was developed. Instructions for working with the software tools are provided.

The program can be used on IBM PC architecture computers with Windows 10/11.

The program was developed in the Python programming language.

Keywords: computer engineering, computer networks, simulation modeling

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

– **IP (Internet Protocol)** – мережевий протокол, що використовується для адресації та маршрутизації пакетів даних у комп’ютерних мережах.

– **JSON (JavaScript Object Notation)** – формат обміну даними, що представляє інформацію у вигляді пар "ключ-значення" і є зручним для зберігання та передачі даних.

– **MST (Мінімальне кістякове дерево, Minimum Spanning Tree)** – підграф зв’язного зваженого графа, що містить усі вершини та мінімізує загальну вагу ребер.

– **OSPF (Open Shortest Path First)** – динамічний протокол маршрутизації, що використовує алгоритм Дейкстри для визначення найкоротших шляхів у мережі.

– **SIR (Susceptible-Infected-Recovered)** – епідеміологічна модель розповсюдження інфекцій у популяції, що описує три стани індивідів: сприйнятливий (S), інфікований (I) та одужалий (R).

– **SPF (Shortest Path First)** – алгоритм знаходження найкоротшого шляху у графі, який використовується в мережевих протоколах маршрутизації, таких як OSPF (Open Shortest Path First).

– **STP (Spanning Tree Protocol)** – мережевий протокол, який запобігає появі петель у комутаційних мережах шляхом блокування зайвих зв’язків.

– **Безмасштабна мережа (модель Barabási-Albert)** – мережа, у якій степеневий розподіл вузлів вказує на наявність невеликої кількості "супервузлів" із великою кількістю зв’язків.

– **Випадкова мережа (модель Erdős-Rényi)** – граф, у якому ребра розподіляються випадково між вершинами з рівноймовірним з’єднанням.

– **Граф (Graph)** – математична структура, що складається з множини

вершин (вузлів) та множини ребер (зв’язків між вершинами).

– **Модель Barabási-Albert** – безмасштабна модель мережі, що ґрунтується на механізмі зростання та преференційного приєднання, в результаті чого формується степеневий розподіл вузлів за ступенем.

– **Модель Bursty Traffic Model (нерівномірний трафік)** – модель, що характеризується періодами високої та низької активності трафіку, відображаючи реальні патерни навантаження.

– **Модель Circular Layout (розміщення по колу)** – метод розташування вузлів графа у вигляді кола, що підходить для представлення циклічних структур.

– **Модель Erdős-Rényi** – класична модель випадкових графів, у якій кожне ребро додається між двома вузлами з імовірністю p незалежно від інших зв’язків.

– **Модель Kamada-Kawai Layout (алгоритм Камади-Каваї)** – метод візуалізації графів, що мінімізує відхилення між фактичними та бажаними відстанями між вузлами.

– **Модель Poisson Traffic Model (потік, що відповідає розподілу Пуассона)** – математична модель трафіку, де події (наприклад, прихід пакетів) розподілені згідно з розподілом Пуассона.

– **Модель Random Traffic Model (випадкові затримки)** – модель трафіку, де часи між подіями змінюються випадковим чином.

– **Модель Spectral Layout (спектральне розташування)** – метод розташування вузлів графа на основі спектрального аналізу його матриці суміжності.

– **Модель Spring Layout (пружинна модель)** – метод візуалізації графів, у якому вузли розташовуються так, ніби вони зв’язані пружинами, що прагнуть до рівноваги.

– **Модель Watts-Strogatz** – модель малого світу, яка генерує мережі з короткими шляхами між вузлами та високим коефіцієнтом кластеризації.

– **Модель малого світу (модель Watts-Strogatz)** – мережева модель, що характеризується малою середньою довжиною шляху між вузлами та високою кластеризацією.

– **ПЗ (Програмне забезпечення, Software)** – сукупність програм, процедур та правил, необхідних для виконання певних завдань на комп'ютері або іншому пристрої.

– **Складні мережі (Complex Networks)** – мережі з нерегулярною структурою та складними взаємозв'язками між вузлами, що можуть мати властивості, такі як мала середня довжина шляху або висока кластеризація.

ВСТУП

Актуальність теми. Розвиток інформаційних технологій та глобальна інтеграція мережевих систем створюють нові можливості для оптимізації різноманітних процесів у сфері управління, бізнесу та наукових досліджень. Одним із основних елементів сучасних інформаційних технологій є комп'ютерні мережі, які забезпечують взаємодію між різними пристроями, системами та користувачами. В умовах швидкого розвитку нових технологій, зокрема віртуалізації, хмарних обчислень та Інтернету речей, важливою задачею є правильне проектування, тестування та оптимізація комп'ютерних мереж.

Програмне забезпечення для моделювання роботи комп'ютерних мереж є важливим інструментом для розробників, інженерів і дослідників. Воно дозволяє створювати віртуальні моделі мереж, тестувати їх на різні сценарії навантаження та оцінювати ефективність роботи перед реальним впровадженням. Це дозволяє знизити ризики, пов'язані з помилками у проектуванні, а також оптимізувати використання ресурсів мережі.

Програмне забезпечення для моделювання комп'ютерних мереж дозволяє ефективно аналізувати роботу мережевих компонентів і виявляти потенційні проблеми на етапі проектування.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи для моделювання роботи комп'ютерних мереж.

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

– Огляд існуючих систем для моделювання роботи комп'ютерних мереж.

– Проектування системи для моделювання роботи комп'ютерних мереж на основі теорії складних мереж.

– Програмна реалізація системи для моделювання роботи комп'ютерних мереж.

Проблеми, пов’язані з проектуванням, аналізом та оптимізацією комп’ютерних мереж, вимагають нових підходів до моделювання, що дозволяють врахувати складну природу цих систем.

Основними завданнями роботи є дослідження основ теорії складних мереж, розробка моделі комп’ютерної мережі на основі цієї теорії, реалізація програмного забезпечення для імітації роботи мережі, а також тестування розробленого продукту за різними сценаріями. Враховуючи наявність складних взаємозв’язків і багаточисельних факторів, цей підхід дозволить більш точно і ефективно аналізувати різні параметри роботи комп’ютерних мереж, що є важливим для їх оптимізації в умовах реальних умов експлуатації.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно здійснювати програмне імітаційне моделювання роботи комп’ютерних мереж на основі теорії складних мереж.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Програмне імітаційне моделювання роботи комп'ютерних мереж має на меті створення інструменту для детального аналізу, проектування та оптимізації мережевих систем з урахуванням їх складних структурних та функціональних властивостей. Основне призначення розроблюваної системи полягає в наступному:

– *Аналіз роботи комп'ютерних мереж*. Система дозволяє моделювати і вивчати поведінку комп'ютерних мереж при різних умовах, що включають зміну топології мережі, навантаження, збої або аномальні ситуації. Це дозволяє оцінити ефективність роботи мережі в різних сценаріях без необхідності проводити дорогі та складні експерименти в реальних умовах.

– *Оцінка стійкості мережі до збоїв*. Використовуючи теорію складних мереж, система дозволяє моделювати вплив збоїв на роботу мережі, що дозволяє оцінити її стійкість до відмов окремих вузлів чи зв'язків. Це особливо важливо для мереж, де високі вимоги до безпеки та надійності.

– *Оптимізація топології мережі*. Система допомагає досліджувати оптимальні шляхи для підключення вузлів, налаштування маршрутизації та балансування навантаження з урахуванням специфічних вимог до швидкості передачі даних, затримок і пропускної здатності.

– *Прогнозування ефективності мережі в умовах зростання навантаження*. Система дозволяє моделювати майбутнє навантаження на мережу та її здатність ефективно обробляти великий обсяг даних. Це дає змогу здійснити своєчасне коригування конфігурації мережі та підвищити її ефективність.

– *Тестування нових алгоритмів і методів*. Система також дозволяє тестувати нові алгоритми для управління мережею, маршрутизації, оптимізації потоків даних та інших аспектів, що стосуються ефективності роботи мережі. Це дозволяє визначити найбільш ефективні методи для конкретних сценаріїв без ризику для реальної інфраструктури.

Таким чином, основне призначення даної системи полягає в забезпеченні ефективного, науково обґрунтованого та безпечного підходу до проектування, аналізу та оптимізації комп'ютерних мереж, що дозволяє знизити ризики, пов'язані з їх використанням та управлінням.

1.2 Область застосування

Програмне імітаційне моделювання роботи комп'ютерних мереж може бути застосоване в різних сферах, де використання ефективних, надійних і масштабованих мереж є критично важливим. Основні області застосування цієї системи включають:

– *Проектування та оптимізація комп'ютерних мереж*. Система може бути використана для розробки та тестування нових комп'ютерних мереж у різних сферах, таких як корпоративні, навчальні, індустріальні та урядові мережі. Вона дозволяє оптимізувати структуру мережі для досягнення максимальної ефективності та стійкості до збоїв.

– *ІТ-компанії та постачальники телекомунікаційних послуг*. Компанії, що займаються проектуванням і обслуговуванням великих мереж, можуть використовувати таку систему для моделювання роботи своїх мереж, оцінки їхньої продуктивності, навантаження та визначення слабких місць, що потребують покращення.

– *Дослідження та розробки в галузі телекомунікацій*. Науково-дослідні організації та університети можуть використовувати систему для досліджень в області телекомунікацій, розвитку нових алгоритмів маршрутизації, управління

потоками даних, а також для вивчення властивостей складних мереж та їхнього впливу на ефективність передачі даних.

– *Освіта та тренування фахівців*. Система може використовуватися в навчальних закладах для підготовки студентів з напрямку комп'ютерних мереж, теорії складних систем і інформаційних технологій. Вона надає студентам можливість на практиці ознайомитися з процесами проектування та управління мережами, що є корисним як в теоретичних курсах, так і в лабораторних роботах.

– *Інженерія інформаційних систем*. Великі корпоративні чи урядові інформаційні системи вимагають високої надійності та ефективності. Імітаційне моделювання дозволяє тестувати різні стратегії управління такими мережами, виявляти вразливі місця і запобігати можливим поломкам в майбутньому.

– *Розробка і тестування нових технологій в області IT*. Технології, що з'являються, як-от 5G, Інтернет речей (IoT) чи хмарні технології, потребують моделювання їх роботи до впровадження в реальні умови. Система дозволяє моделювати нові інфраструктури мереж, оцінюючи їх ефективність в умовах високих навантажень і непередбачуваних змін.

– *Аналіз великих даних*. Система може бути застосована для моделювання роботи великих розподілених систем, таких як хмарні обчислення, де ефективність роботи мережі має безпосередній вплив на швидкість обробки даних. Моделювання дозволяє аналізувати такі системи на різних етапах їх розвитку.

Таким чином, область застосування системи є надзвичайно широкою і охоплює як технічні, так і наукові, освітні та практичні сфери, де важливо мати можливість ефективно моделювати, аналізувати і оптимізувати складні комп'ютерні мережі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Програмне забезпечення для моделювання комп'ютерних мереж можна розділити на такі категорії:

1. Симулятори комп'ютерних мереж:

– Дозволяють імітувати роботу комп'ютерної мережі без потреби в реальному обладнанні.

– Використовуються для досліджень, навчання, тестування.

– Приклади: NS-3, OMNeT++, GNS3, Packet Tracer, Mininet.

2. Емулятори мереж:

– Дають змогу створювати віртуальні мережі, які працюють аналогічно реальним.

– Використовуються для тестування протоколів, безпеки, продуктивності.

– Приклади: GNS3, EVE-NG, CORE (Common Open Research Emulator), Cloonix.

3. Інструменти аналізу трафіку:

– Використовуються для моніторингу та аналізу продуктивності мереж.

– Приклади: Wireshark, NetFlow, PRTG Network Monitor.

4. Системи автоматизації та оркестрації:

– Дозволяють моделювати та керувати мережами у великих інфраструктурах.

– Приклади: Ansible, OpenStack, Kubernetes (для SDN/NFV).

Основні технології, що використовуються у симуляторах та емуляторах комп’ютерних мереж:

1. Віртуалізація та контейнеризація:

– Docker + Kubernetes – для створення віртуальних середовищ із мережевими компонентами.

– KVM, VirtualBox, VMware – для запуску віртуальних машин із власними мережевими конфігураціями.

2. Програмно-Конфігуровані Мережі (SDN)

– Використання контролерів, таких як OpenDaylight, ONOS, Ryu, дозволяє динамічно змінювати топологію мережі та керувати маршрутизацією.

– Інструмент Mininet дозволяє створювати віртуальні SDN-мережі для тестування.

3. Віртуалізація функцій мережі (NFV)

– Використовується у великих провайдерів для заміни апаратних маршрутизаторів та файрволів на віртуальні екземпляри.

– OpenStack + Neutron – популярна платформа для мережевої віртуалізації.

Розглянемо детальніше конкретне програмне забезпечення.

NS-3 (Network Simulator 3) – це **дискретно-ситуативний симулятор мереж**, який використовується для досліджень у галузі комп'ютерних мереж. Він дозволяє моделювати протоколи, мережеві топології та аналізувати ефективність роботи мереж.

Переваги:

- Реалістичне моделювання мережевих протоколів (TCP, UDP, Wi-Fi, LTE).
- Гнучкість – можна програмувати власні модулі на C++ або Python.
- Підтримка інтеграції з реальними мережами через DCE (Direct Code Execution).

Недоліки:

- Високий поріг входу – потребує знань C++ та структури моделювання.

- Відсутність графічного інтерфейсу – управління через термінал.

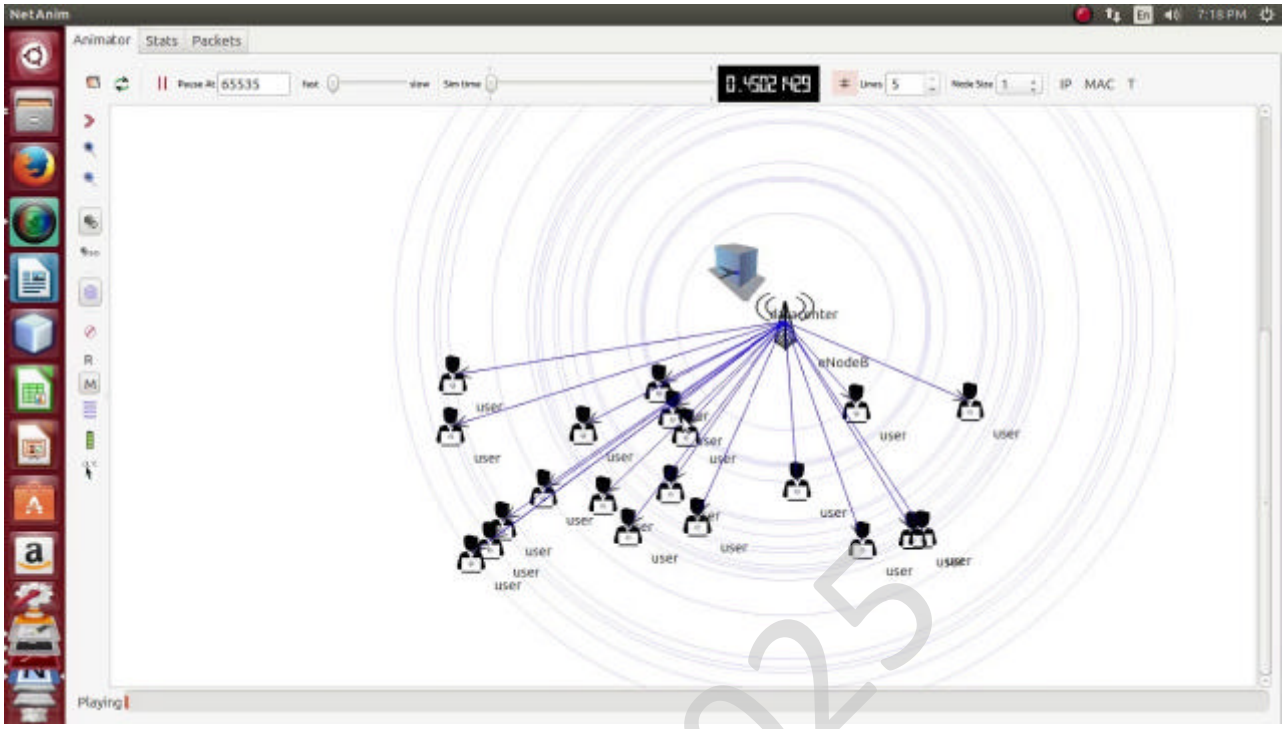

Рисунок 2.1 – Приклад роботи NS-3 (Network Simulator 3)

OMNeT++ (Objective Modular Network Testbed in C++) – це **модульний симулятор мереж**, який широко використовується в дослідженнях та навчанні. Він надає інструменти для моделювання великих мережевих систем, бездротових мереж, мобільних мереж та IoT.

Переваги:

- Модульна архітектура – дозволяє легко змінювати компоненти.
- Вбудований **GUI** – зручна візуалізація мережевих процесів.
- Велика бібліотека протоколів та можливість розширення.

Недоліки:

- Високі вимоги до ресурсів при складному моделюванні.
- Менше реалістичних сценаріїв порівняно з емуляторами.

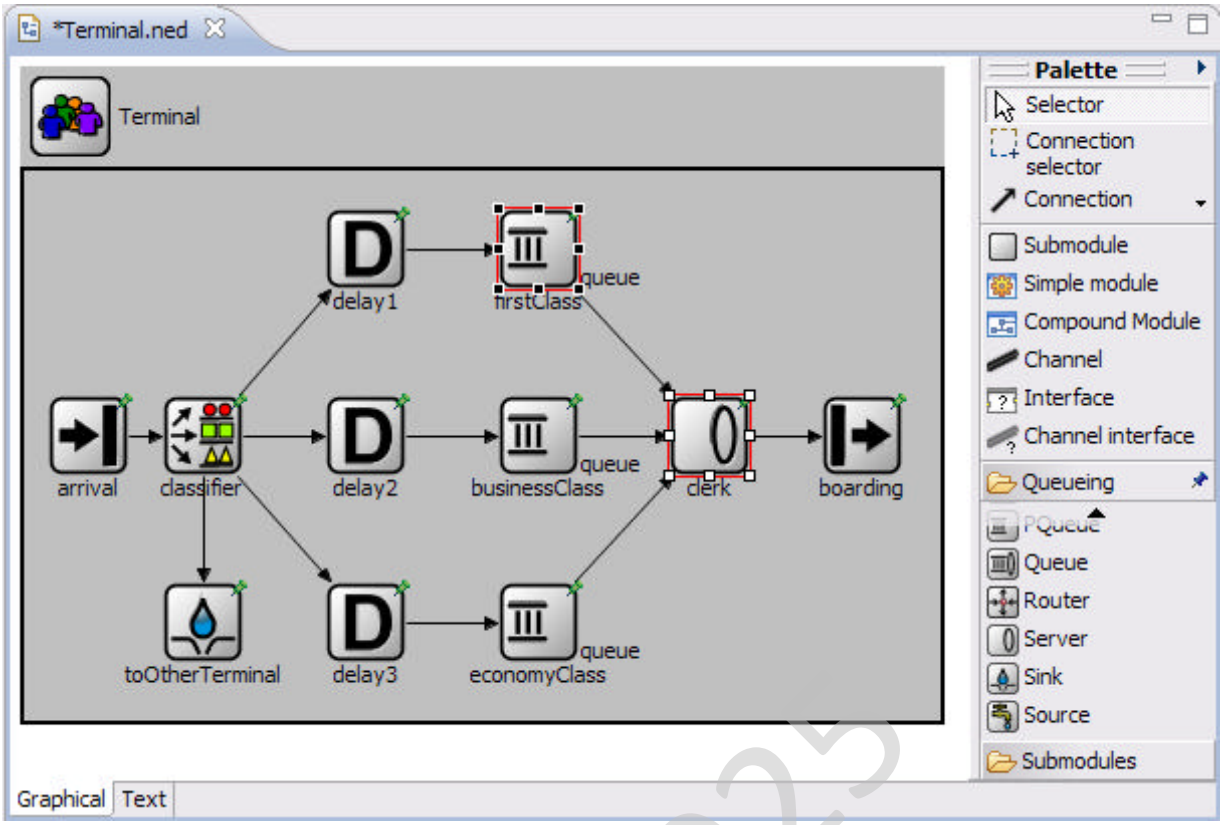

Рисунок 2.2 – Приклад роботи OMNeT++ (Objective Modular Network Testbed in C++)

GNS3 (Graphical Network Simulator 3) – це **емулятор мережевого обладнання**, який дозволяє створювати віртуальні мережі з реальними образами маршрутизаторів та комутаторів (наприклад, Cisco, Juniper).

Переваги:

- Підтримка реального мережевого ПЗ (Cisco IOS, Juniper, MikroTik).
- Можливість інтеграції з фізичними мережами.
- Графічний інтерфейс для побудови топологій.

Недоліки:

- Високі вимоги до ресурсів (процесор та оперативна пам’ять).
- Ліцензійні обмеження для використання реальних образів.

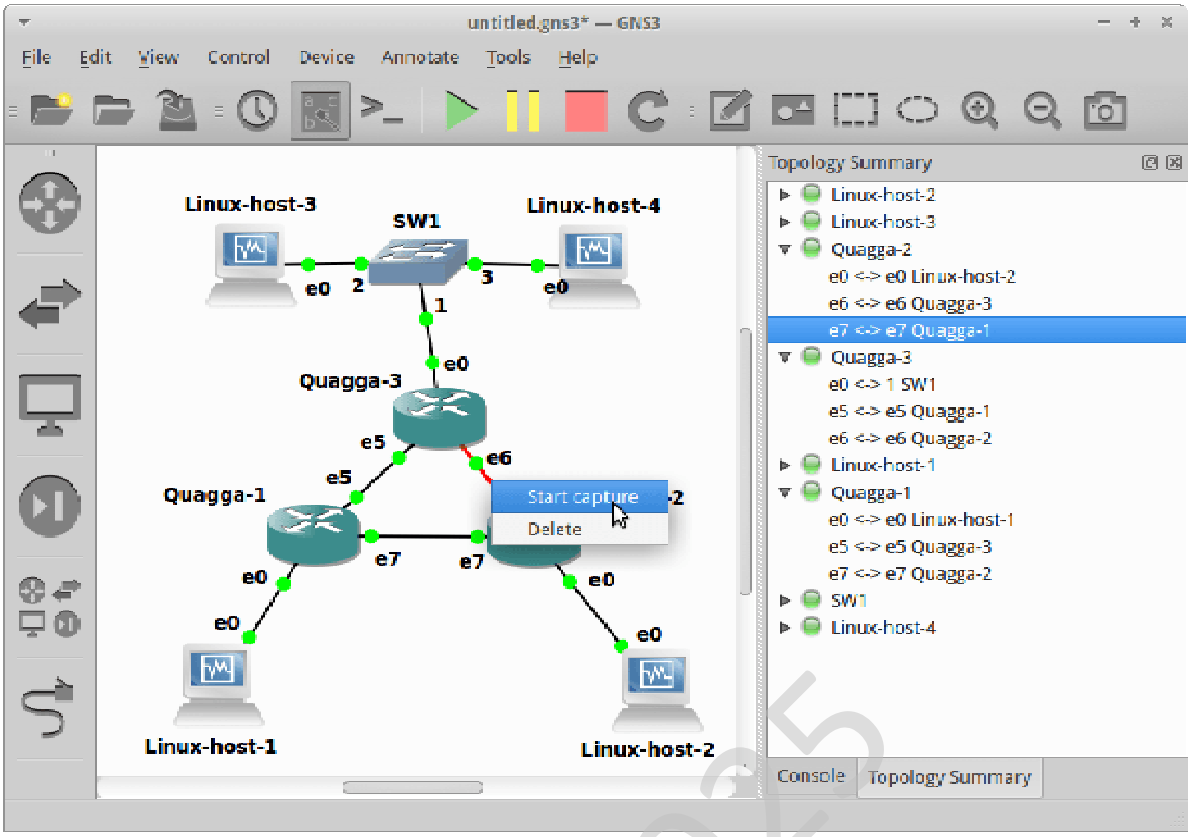

Рисунок 2.3 – Приклад роботи GNS3 (Graphical Network Simulator 3)

Packet Tracer – це **мережевий емулятор від Cisco**, який використовується для навчання. Він дозволяє створювати та тестувати мережі, налаштовувати комутатори та маршрутизатори.

Переваги:

- Легкість у використанні – не потребує знань програмування.
- Вбудований інтерактивний режим навчання.
- Можливість моделювання основних протоколів (IPv4, IPv6, OSPF, STP тощо).

Недоліки:

- Обмежений функціонал – не можна працювати з реальними образами ОС мережевого обладнання.
- Призначений лише для навчання.

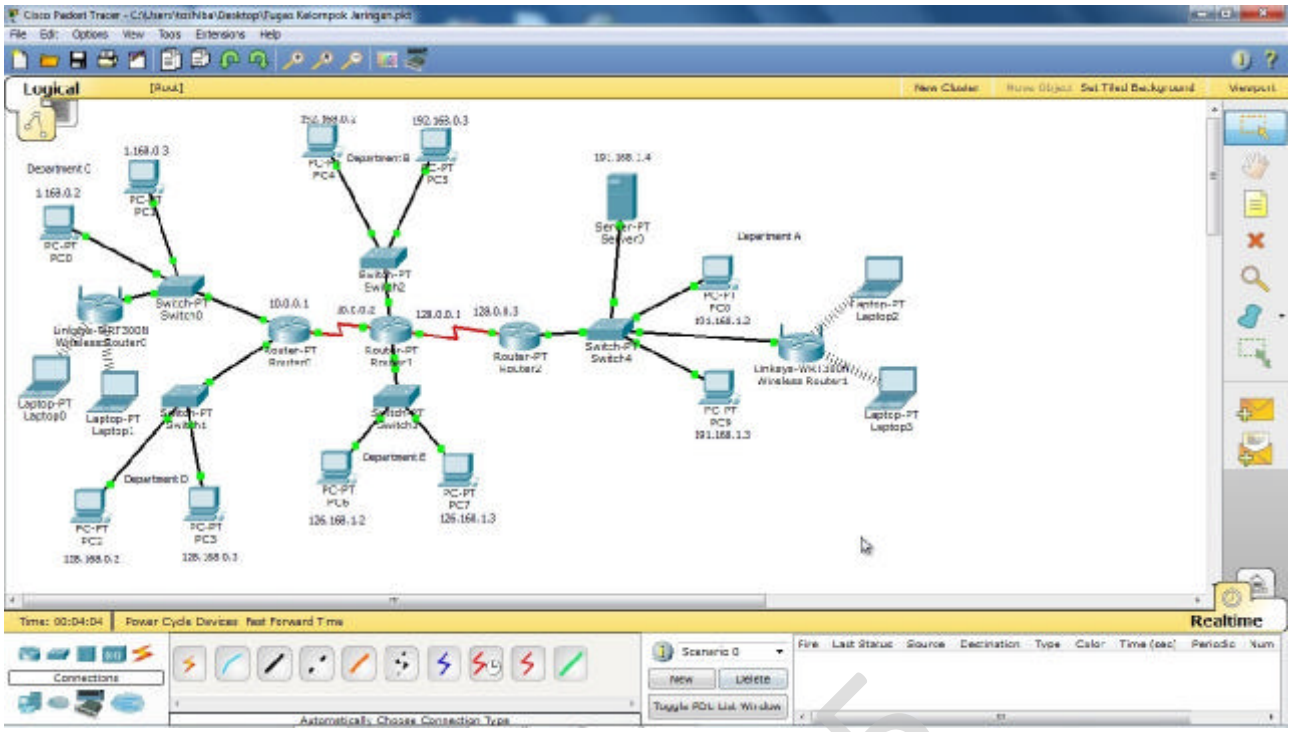

Рисунок 2.4 – Приклад роботи Packet Tracer

Mininet – це **емулятор мереж SDN (Software-Defined Networking)**, що дозволяє створювати віртуальні топології та тестувати OpenFlow-контролери.

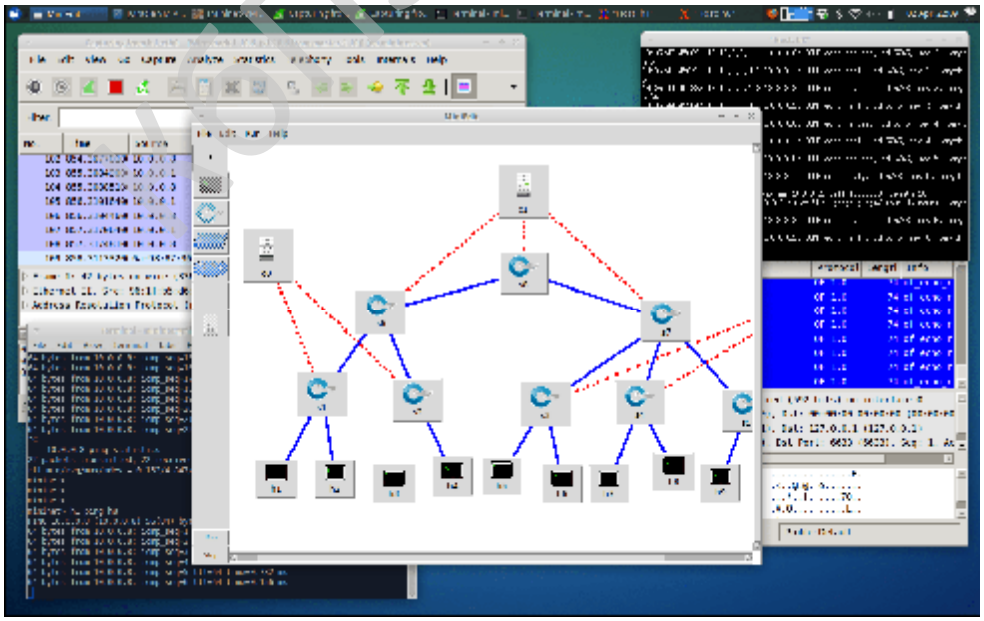

Рисунок 2.5 – Приклад роботи Mininet

Переваги:

- Легкий та швидкий запуск віртуальних мереж.
- Використовується в дослідженнях SDN і підтримує OpenFlow.
- Працює у середовищі Linux.

Недоліки:

- Не підходить для традиційних мереж без SDN.
- Вимагає знань Linux та мережевого програмування.

Wireshark – це **аналізатор трафіку**, що дозволяє переглядати, фільтрувати та аналізувати мережеві пакети у реальному часі.

Переваги:

- Підтримка більшості мережевих протоколів.
- Використовується для аналізу мережевої безпеки та усунення збоїв.
- Безкоштовний та відкритий код.

Недоліки:

- Не призначений для моделювання – лише аналізує трафік.
- Потребує знань мережевих технологій для правильного аналізу даних.

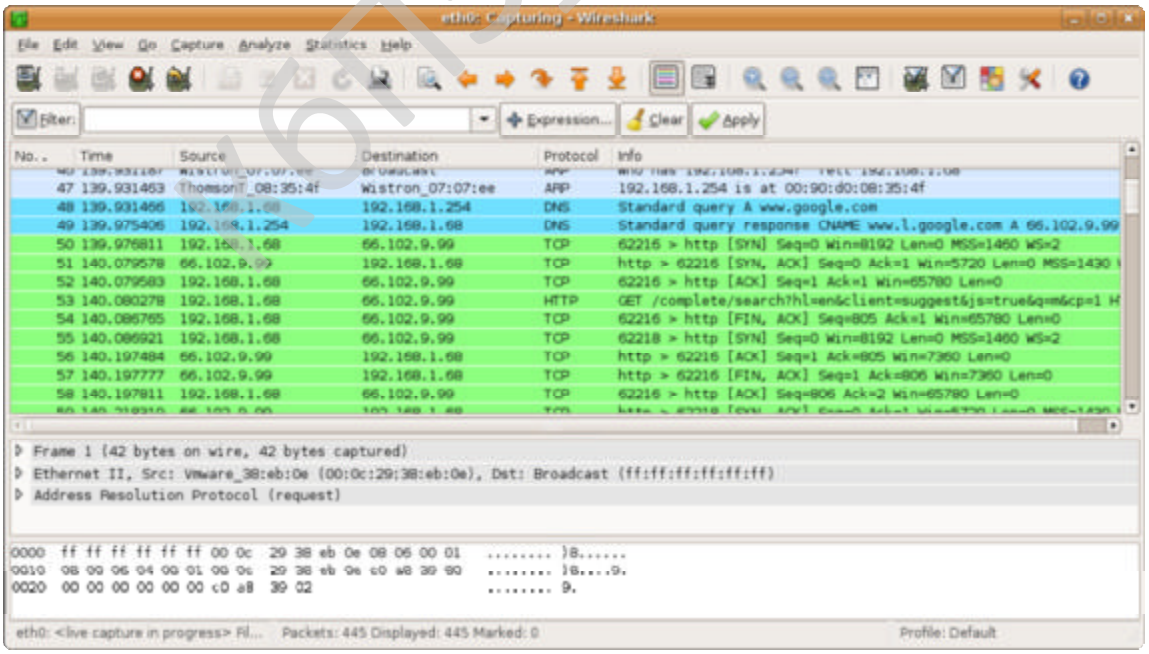

Рисунок 2.6 – Приклад роботи Wireshark

Таблиця 2.1 – Порівняльний аналіз популярних програмних рішень для моделювання комп'ютерних мереж

Назва	Тип	Особливості	Переваги	Обмеження
NS-3	Симулятор	Складне моделювання протоколів та топологій	Висока точність, велика підтримка спільноти	Високий поріг входу
OMNeT++	Симулятор	Гнучка модульна архітектура	Підтримка розширень, візуалізація	Потребує значних ресурсів
GNS3	Емулятор	Підтримка реальних образів Cisco	Реалістичність, інтеграція з реальними мережами	Високі вимоги до обладнання
Packet Tracer	Емулятор	Від Cisco, орієнтований на навчання	Легкий у використанні, безкоштовний	Обмежена реалістичність
Mininet	Емулятор SDN	Підтримка OpenFlow	Використовується для SDN-тестів	Вимагає Linux
Wireshark	Аналізатор трафіку	Аналіз пакетів	Глибокий аналіз мережевих процесів	Не підходить для моделювання

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для розробки програмного забезпечення для моделювання роботи комп'ютерних мереж було обрано наступні засоби:

– Мова програмування: Python.

– Середовище розробки: Anaconda та Spyder.

– Методологія аналізу: Теорія складних мереж.

Обґрунтуємо їх вибір, враховуючи специфіку проекту.

Python є оптимальним вибором для моделювання та аналізу комп'ютерних мереж через такі переваги:

1. Легкість у використанні та розширюваність:

- Python має зрозумілий синтаксис, що спрощує розробку та тестування коду.
- Велика кількість бібліотек дозволяє легко додавати новий функціонал.

2. Підтримка бібліотек для мережевого моделювання. Python має багато бібліотек, що дозволяють працювати з мережами:

- NetworkX – для аналізу графів та моделювання топологій мереж.
- Scapy – для роботи з пакетами мережевого трафіку.
- NS-3 (через pybind) – для симуляції мережевих протоколів.

3. Гнучкість та інтеграція:

- Python легко інтегрується з базами даних, веб-сервісами та хмарними обчисленнями.
- Підтримує багатопоточність та асинхронне програмування для моделювання реальних мережевих процесів.

Python підходить для виконання мети даної роботи з наступних причин:

- Дозволяє легко будувати та аналізувати топології мереж.
- Забезпечує можливість працювати з графами, що важливо для теорії складних мереж.

– Має великий набір бібліотек для обробки даних та візуалізації результатів.

Anaconda – потужне середовище для обчислень та аналізу даних, комплексне середовище для роботи з Python, що має ряд переваг:

– Включає основні бібліотеки для наукових обчислень (NumPy, SciPy, Pandas, Matplotlib).

– Має власний менеджер пакетів conda, що спрощує встановлення необхідного програмного забезпечення.

– Підтримує віртуальні середовища, що дозволяє ізолювати залежності проєкту.

– Spyder – спеціалізоване IDE для наукових досліджень та зручне інтегроване середовище, що поєднує:

– Інтерактивний Python-консоль для швидкого тестування коду.

– Редактор з підсвіткою синтаксису для написання складних скриптів.

– Інструменти для налагодження та профілювання коду.

Anaconda та Spyder підходить для виконання мети даної роботи з декількох причин:

– Дозволяють ефективно працювати з мережевими симуляціями та аналізом графів.

– Мають всі необхідні бібліотеки "з коробки".

– Підходять для візуалізації топологій мереж.

Теорія складних мереж – це математичний підхід, що використовує графи для моделювання реальних мережевих систем.

Чому теорія складних мереж була використана у цій роботі для моделювання комп’ютерних мереж:

– Комп’ютерні мережі – це графи (вузли – це маршрутизатори/сервери, ребра – це з'єднання між ними).

– Аналіз топології мереж дозволяє досліджувати їхню надійність, масштабованість та вразливість.

- Дослідження алгоритмів маршрутизації та виявлення "вузьких місць".

Інструменти для роботи з графами у Python:

- NetworkX – створення, моделювання та аналіз складних мереж.
- Graph-tool – прискорена обробка великих графів.
- Matplotlib / Gephi – візуалізація мережевих топологій.

Обраний підхід має наступні переваги:

- Дає змогу будувати реалістичні моделі мереж.
- Дозволяє імітувати поведінку трафіку та мережевих атак.
- Полегшує порівняння різних мережевих архітектур.

Обрані інструменти забезпечують гнучкість, потужність та надійність у роботі з мережевими моделями. Вони дозволяють ефективно будувати, тестувати та аналізувати комп'ютерні мережі у симуляційному середовищі. Цей вибір є оптимальним для даної кваліфікаційної бакалаврської роботи.

2.3 Розгорнута постановка завдання

Метою роботи є розробка програмного забезпечення для моделювання роботи комп’ютерних мереж з використанням Python, Anaconda, Spyder та теорії складних мереж.

Реалізована система повинна дозволяти:

- Створювати та аналізувати різні топології мереж.
- Симулювати передачу даних та маршрутизацію.
- Оцінювати ефективність роботи мережі за різних навантажень.
- Візуалізувати результати у вигляді графів та діаграм.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. Аналіз предметної області та існуючих рішень.

- Огляд сучасного програмного забезпечення для моделювання комп’ютерних мереж (NS-3, OMNeT++, GNS3, Packet Tracer, Mininet).
- Визначення переваг та обмежень існуючих систем.

– Обґрунтування вибору мови програмування, середовища розробки та методу досліджень для даної роботи.

2. Розробка методики моделювання:

– Визначення типів мережевих топологій (дерево, кільце, зірка, масштабовані графи тощо).

– Використання графової моделі для представлення мережі.

– Формування метрик ефективності мережі (пропускна здатність, затримка, навантаження на вузли).

– Вибір підходу для симуляції роботи мережевих протоколів.

3. Реалізація програмного забезпечення:

– Розробка модулів для створення та аналізу топології.

– Імітація трафіку та аналіз роботи протоколів.

– Візуалізація результатів – створення графіків, таблиць, схем взаємодії вузлів.

– Оптимізація продуктивності коду.

4. Тестування та верифікація моделі

– Проведення серії експериментів для оцінки роботи системи.

– Порівняння результатів симуляції з реальними мережевими даними.

– Аналіз можливих помилок та їх усунення.

5. Підготовка документації

– Опис реалізованих алгоритмів та структур даних.

– Інструкція для користувачів та розробників.

– Аналіз отриманих результатів та висновки.

Очікувані результати

В результаті виконання роботи буде:

– Розроблено програмне забезпечення для моделювання роботи комп’ютерних мереж.

– Оцінено ефективність різних топологій та алгоритмів маршрутизації.

– Проаналізовано основні фактори, що впливають на продуктивність

мережі.

– Підготовлено рекомендації щодо використання отриманих результатів у реальних мережах.

Ця робота дозволить створити інструмент для аналізу роботи комп’ютерних мереж, що базується на сучасних підходах до моделювання. Вона може бути корисною для адміністраторів мереж, дослідників та студентів, які вивчають мережеві технології.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

В розроблюваному програмному забезпеченні було реалізовано:

– Генерацію структури комп’ютерної мережі.

– Візуалізацію графу комп’ютерної мережі.

– Симуляцію руху мережевого трафіку по комп’ютерній мережі.

– Симуляцію поширення комп’ютерних вірусів по комп’ютерній мережі.

– Аналіз структури комп’ютерної мережі та рекомендації з її оптимізації й усунення слабких місць.

Для генерації структури комп’ютерної мережі було використано теорію складних мереж.

Генерація структури складних мереж є важливим завданням в аналізі мереж та графів. Існує кілька основних алгоритмів, які використовуються для генерації таких мереж:

1. Стохастичні (випадкові) моделі. Ці моделі генерують мережі за певними ймовірнісними правилами. Приклади таких моделей:

– **Модель Ердеша-Реньї (Erdős–Rényi, ER).** Визначається фіксованою кількістю вузлів N і ймовірністю зв’язку між ними p. Модель не враховує неоднорідність вузлів і не пояснює появу "хабів" (вузлів з високим ступенем зв’язності).

– **Модель Ваттса-Строгаца (Watts-Strogatz, WS).** Починається з регулярного графа (наприклад, циклу) і поступово перезапускає ребра з ймовірністю p. Генерує "малі світи" (small-world networks), що характеризуються високим коефіцієнтом кластеризації і малими середніми відстанями.

– **Модель Барбаші-Альберт (Barabási–Albert, BA).** Ґрунтується на механізмі "прикріплення, пропорційного ступеню" (preferential attachment).

Генерує безмасштабні (scale-free) мережі, які мають степеневий закон розподілу ступенів вузлів.

2. Детерміновані (фрактальні) моделі:

– **Генерація мереж за рекурсивними правилами.** Наприклад, фрактальні дерева або самоорганізовані критичні системи. Використовуються для моделювання природних мереж, таких як кровоносні або нейронні мережі.

– **Модель Куна та Бассетта (Kuhn & Bassett).** Включає топологічні правила зростання мереж, що нагадують структури біологічних систем.

3. Еволюційні моделі:

– **Мережі з адаптивною динамікою (Adaptive Networks).** Структура мережі змінюється залежно від динаміки системи. Використовуються в соціальних мережах, фінансових ринках, біологічних системах.

– **Мережі на основі агентних моделей (Agent-based models).** Вузли є агентами, які взаємодіють за певними правилами, що змінюють структуру мережі. Приклад: модель Сігеля (Segal model) для соціальних взаємодій.

4. Гібридні моделі. Комбінують елементи випадкових і детермінованих алгоритмів. Приклади таких мереж:

– **Модель Дорога-Мендеса (Dorogovtsev-Mendes).** Включає ріст і преференційне прикріплення, але також допускає випадкові з'єднання.

– **Модель Кронекера (Kronecker Graphs).** Створює великі мережі шляхом рекурсивного збільшення меншого базового графа.

5. Алгоритми на основі реальних даних

– **Моделі на основі статистичних закономірностей (Stochastic Block Models, SBM).** Групують вузли у "спільноти" з певними внутрішніми та міжгруповими зв'язками.

– **Алгоритми реконструкції мереж із часових рядів.** Використовуються в нейронауках і фінансах для аналізу взаємозв'язків між змінними.

Вибір алгоритму залежить від типу мережі та цілей дослідження. Для

моделювання комп'ютерних мереж було вирішено використати моделі Ердеша-Реньї, Ваттса-Строгаца та Барбаші-Альберт, так як за допомогою них зручно моделювати технічні мережі.

Візуалізація графів – це процес розташування вершин і ребер у просторі для наочного представлення їхньої структури. Існує кілька основних підходів та алгоритмів для цього:

1. Алгоритми на основі фізичних моделей. Ці алгоритми моделюють сили між вершинами та ребрами, щоб знайти «енергетично оптимальне» розташування графа. Приклади:

– **Spring Layout (Fruchterman-Reingold).** Базується на пружинній моделі – вершини відштовхуються одна від одної, а ребра діють як пружини. Підходить для середніх за розміром графів. Реалізовано в NetworkX як spring_layout.

– **Kamada-Kawai Layout.** Оптимізує розташування графа, мінімізуючи різницю між геометричними та топологічними відстанями. Дає гарні результати для малих і середніх графів.

– **ForceAtlas2 (Gephi).** Покращена версія Fruchterman-Reingold з урахуванням ваг ребер. Добре працює для візуалізації соціальних мереж.

– **sfdp (Scalable Force-Directed Placement).** Оптимізована версія force-directed layout для **великих графів**.

2. Геометричні та топологічні алгоритми. Ці методи базуються на визначених геометричних правилах. Приклади:

– **Circular Layout.** Всі вершини розташовуються по колу. Корисно для кільцевих структур (наприклад, телефонні мережі).

– **Spectral Layout.** Використовує власні вектори матриці Лапласа для розташування вершин. Ефективний для розріджених мереж.

– **Grid Layout.** Розташовує вершини у вигляді решітки. Використовується, коли потрібна впорядкована візуалізація.

– **Radial Layout.** Вершини розташовуються концентричними колами навколо центрального вузла. Добре підходить для ієрархічних структур.

3. Ієрархічні алгоритми. Застосовуються для деревоподібних структур. Приклади:

– **Sugiyama (Layered Layout).** Використовується для орієнтованих графів (наприклад, діаграми процесів). Реалізований у Graphviz (DOT layout).

– **Tree Layout.** Оптимізоване представлення для дерев (ієрархій, організаційних структур).

4. Рандомізовані та спеціалізовані методи:

– **Random Layout.** Вершини розміщуються випадковим чином. Використовується як тестовий варіант.

– **Geographical Layout.** Використовується для графів, прив'язаних до географічних координат (наприклад, транспортні мережі).

5. 3D-алгоритми:

– **Graph-embedding in 3D space.** Використовуються тривимірні варіанти force-directed або spectral layout. Можливість обертати граф у 3D-просторі.

– **VR-орієнтовані підходи.** Використання віртуальної реальності для інтерактивної навігації по графу.

Для реалізації візуалізації графа комп'ютерної мережі було вирішено обрати наступні алгоритми: Пружинна модель Spring Layout, по колу Circular Layout, спектральний Spectral Layout та Kamada-Kawai Layout.

Симуляція обміну трафіком між комп'ютерами мережі використовується для дослідження продуктивності, оптимізації маршрутів, виявлення вузьких місць і забезпечення безпеки. Для цього існують різні алгоритми та моделі.

У розроблюваному програмному забезпеченні були використані наступні алгоритми симуляції руху трафіку:

– **Poisson Traffic Model (випадковий потік).** Принцип роботи: модель заснована на розподілі Пуассона, який описує випадкові незалежні події, що відбуваються через певні проміжки часу. Використовується для імітації випадкових запитів, наприклад, коли користувачі надсилають пакети в мережу в різний час. Інтервали між пакетами моделюються як експоненційно розподілені

значення. Застосування: аналіз мереж загального користування (інтернет-запити, мережеві сервери), моделювання потоків у хмарних сервісах, випробування QoS (Quality of Service) у телекомунікаціях. Особливості: потік має випадковий розподіл у часі, не враховує великі сплески навантаження, підходить для середньоінтенсивного навантаження.

– **Bursty Traffic Model (нерівномірний потік)**. Принцип роботи: трафік надходить не рівномірно, а періодичними "сплесками" (bursts); вузли надсилають велику кількість пакетів за короткий період, потім перерва, потім знову активність; інтервали між сплесками описуються експоненційним або нормальним розподілом. Застосування: відеострімінг та VoIP (мережі з піковими навантаженнями), файлові сервери та CDN (Content Delivery Network), мережі з великими пакетами даних (наприклад, IoT). Особливості: створює пікове навантаження, яке може викликати перевантаження мережі, дозволяє аналізувати буферизацію та відкладену передачу, використовується у динамічних середовищах.

– **Random Traffic Model (випадкові затримки)**. Принцип роботи: ця модель не має визначеного розподілу для надходження трафіку, використовується, коли немає статистичної інформації про поведінку користувачів,затр имки між пакетами генеруються випадковим чином (наприклад, рівномірний розподіл). Застосування: тестування мережевого обладнання у невизначених умовах, імітація нестабільного трафіку (наприклад, публічні wi-fi), генерація тестових сценаріїв для експериментів. Особливості: не враховує закономірності трафіку, використовується для рандомізованих тестів, допомагає оцінити погіршення продуктивності при хаотичному трафіку.

Розглянемо алгоритми для управління рухом трафіку (маршрутизації, балансування та розподілу трафіку). Основні групи таких алгоритмів:

1. Маршрутизаційні алгоритми (Routing Algorithms). Визначають, яким шляхом дані передаватимуться між вузлами. Приклади:

– **Dijkstra’s Algorithm (Shortest Path First, SPF).** Знаходить

найкоротший шлях між двома вузлами. Використовується в протоколах OSPF (Open Shortest Path First).

– **Bellman-Ford Algorithm.** Визначає найкоротший шлях, але підтримує негативні ваги ребер. Використовується в протоколах RIP (Routing Information Protocol).

– ***A (A-star) Algorithm*.*** Покращена версія алгоритму Дейкстри, використовує евристику для швидшого пошуку.

– **Flooding Algorithm.** Пакети надсилаються всім можливим напрямкам, що гарантує доставку, але створює високий трафік.

– **Random Walk Routing.** Пакет випадково передається сусідам, поки не досягне цілі.

– **Ant Colony Optimization (ACO).** Імітує поведінку мурах для знаходження оптимального маршруту шляхом залишення "феромонів".

2. Алгоритми управління потоками трафіку (Traffic Flow Control). Забезпечують ефективне використання пропускної здатності мережі.

– **Leaky Bucket Algorithm.** Обмежує швидкість відправки пакетів, щоб запобігти перевантаженню.

– **Token Bucket Algorithm.** Дозволяє гнучке управління швидкістю, дозволяючи відправляти більше даних за короткі періоди.

– **RED (Random Early Detection).** Випадково відкидає пакети при високому завантаженні мережі, запобігаючи заторам.

3. Алгоритми балансування навантаження (Load Balancing). Розподіляють трафік між вузлами для уникнення перевантаження. Приклади:

– **Round Robin Load Balancing.** Трафік розподіляється по черзі між серверами.

– **Least Connections Algorithm.** Трафік надсилається серверу, у якого найменше активних з'єднань.

– **Weighted Load Balancing.** Сервери отримують навантаження залежно від їхніх потужностей.

4. Алгоритми запобігання перевантаженню мережі. Допомагають уникати заторів і збоїв у мережах. Приклади:

– **TCP Congestion Control (Slow Start, AIMD).** TCP поступово збільшує швидкість передачі даних, щоб не перевантажити мережу.

– **Explicit Congestion Notification (ECN).** Дозволяє маршрутизаторам сигналізувати про **перевантаження**, не відкидаючи пакети.

– **Fair Queueing (FQ).** Забезпечує справедливий розподіл трафіку між різними потоками.

У розроблюваному програмному забезпеченні було реалізовано алгоритм SPF.

Оцінка ефективності роботи мережі залежить від таких параметрів, як пропускна здатність, затримки, втрати пакетів і балансування трафіку. Для різних рівнів навантаження (низьке, середнє, високе) можна аналізувати:

– **Середній час затримки (Latency).** Час, який витрачається на передачу пакетів між вузлами. **Метрика**: середнє значення затримок усіх переданих пакетів.

– **Пропускна здатність (Throughput).** Обсяг даних, який передається за одиницю часу. **Метрика**: кількість пакетів або байтів, переданих за секунду.

– **Завантаженість вузлів (Node Load).** Відображає, які вузли обробляють найбільшу кількість трафіку. **Метрика**: кількість отриманих та переданих пакетів кожним вузлом.

– **Втрати пакетів (Packet Loss).** Вказує на перевантаження або несправності в мережі. **Метрика**: відсоток пакетів, які не дійшли до місця призначення.

– **Середня довжина маршруту (Path Length).** Число проміжних вузлів, через які проходять пакети. **Метрика**: середня довжина всіх маршрутів.

Щоб ефективно аналізувати роботу мережі при різних навантаженнях, можна використовувати такі графіки:

– **Середній час затримки** – лінійний графік, що показує зміну затримки

при збільшенні трафіку.

Пропускна здатність – гістограма або лінійний графік, що відображає кількість переданих пакетів за час.

Завантаженість вузлів – гістограма або лінійний графік, що показує, які вузли найчастіше використовуються.

Втрати пакетів – лінійний графік, що відображає відсоток втрачених пакетів.

Довжина маршруту – коробкова діаграма, що показує статистику довжин маршрутів.

У розроблюваному програмному забезпеченні реалізовано наступні алгоритми маршрутизації:

Було вирішено додати наступний аналіз мережі у розроблене програмне забезпечення:

- Розрахунок метрик мережі для заданої кількості ітерацій.
- Створення графіків для кожного параметра.
- Порівняння роботи мережі під різними навантаженнями.

Програма оцінює роботу мережі за допомогою таких метрик та діаграм:

1. Середній час затримки – будується гістограма затримок.
2. Довжина маршрутів – аналізується через гістограму довжин шляхів.
3. Завантаженість вузлів – показується на стовпчастій діаграмі.

Під час запуску користувач обирає:

- Модель трафіку (Poisson, Bursty, Random).
- Кількість ітерацій (відправок пакетів).

Після кожної ітерації:

- Відображається візуалізація поточного пакету.
- На екрані виводяться деталі маршруту та затримки.

Аналіз структури графа комп’ютерної мережі допомагає визначити вразливі місця, такі як вузли або ребра, чия відмова може серйозно вплинути на роботу системи.

Нижче наведені основні алгоритми та метрики, які можна використовувати для аналізу:

1. Аналіз центральності вузлів (Node Centrality). Ці метрики допомагають знайти ключові вузли, які є критичними для зв’язності мережі. Приклади:

- Degree Centrality (Ступенева центральність). Вимірює кількість з’єднань кожного вузла. Вразливість: вузли з високою degree centrality є ключовими для комунікації; їх видалення може викликати розрив у мережі.
- Betweenness Centrality (Посередницька центральність). Показує, через які вузли проходить найбільше найкоротших шляхів. Вразливість: якщо такий вузол відмовить, багато маршрутів стануть довшими або неможливими.
- Closeness Centrality (Близькість). Вимірює середню відстань вузла до всіх інших. Вразливість: вузли з низькою близькістю можуть бути ізольовані.
- Eigenvector Centrality (Центральність власного вектора). Показує, наскільки вузол пов'язаний із впливовими вузлами. Вразливість: вузли з високим значенням eigenvector centrality можуть бути точками обвалу системи.

2. Визначення вузьких місць у мережі (Bottlenecks). Алгоритми для пошуку критичних місць:

- Articulation Points (Точки артикуляції). Вузли, видалення яких розриває граф на частини. Вразливість: втрата такого вузла може розірвати мережу.
- Bridges (Мости). Ребра, видалення яких роз’єднує граф. Вразливість: такі з’єднання можуть стати єдиною точкою відмови.
- Edge Betweenness Centrality. Оцінює важливість ребра в мережі, як часто воно використовується у найкоротших шляхах. Вразливість: видалення ребра з високим edge betweenness може вплинути на передачу трафіку.

3. Аналіз зв’язності та стійкості (Resilience). Допомагає визначити, наскільки легко мережу можна вивести з ладу. Приклади:

- Connected Components (Компоненти зв’язності). Показує, скільки окремих частин має граф. Вразливість: велика кількість компонент означає

низьку стійкість мережі.

– K-Core Analysis (K-ядерний аналіз). Визначає вузли, які є частиною стабільного центрального кластера. Вразливість: вузли поза основним ядром є слабкими ланками.

– Network Robustness Index. Оцінює, наскільки добре мережа може витримати атаки. Вразливість: низьке значення індексу означає вразливу структуру.

4. Аналіз мережевої відмовостійкості (Failure & Attack Simulation). Оцінка наслідків атак або випадкових відмов. Приклади:

– Random Node Failure. Видаляються випадкові вузли, і аналізується вплив на зв'язність. Вразливість: якщо граф швидко розпадається, мережа нестабільна.

– Targeted Node Attack (Атака на найбільш важливі вузли). Видаляються вузли з високою centrality, щоб оцінити вплив. Вразливість: якщо мережа втрачає зв'язність при видаленні кількох вузлів, її варто покращити.

У розробленому програмному забезпеченні було реалізовано:

– Аналіз точок артикуляції та мостів – виявити критичні вузли та ребра.

– Підрахунок центральностей – знайти найбільш впливові вузли.

– Перевірку компонент зв'язності – визначити, які вузли можуть стати точками розриву.

– Симуляцію атак та відмов – перевірити, як зміниться мережа при видаленні ключових вузлів.

Також було реалізовано наступний аналіз та формування рекомендацій:

1. Покращення топології мережі:

– Аналіз мінімального кістякового дерева (Minimum Spanning Tree, MST). Визначається оптимальна структуру мережі для мінімізації витрат. Алгоритм: Крускала-Прима. Застосування: Якщо потрібно зменшити кількість зв'язків, зберігаючи зв'язність.

– Знаходження оптимального центру мережі. Визначає вузол із

найменшою середньою відстанню до всіх інших вузлів. Метрика: Центр графа (Graph Center). Застосування: корисно для визначення оптимального розташування сервера або комутатора, моделювання альтернативних маршрутів, визначення альтернативних шляхів між вузлами для підвищення надійності, Алгоритм: K-shortest paths. Застосування: Поліпшення маршрутизації трафіку.

2. Аналіз продуктивності мережі:

– Обчислення середнього діаметра графа. Визначає найгірший сценарій для передачі даних. Метрика: Діаметр графа (Graph Diameter). Застосування: Оптимізація затримок у мережі.

– Моделювання навантаження на канали. Аналіз трафіку на основі топології. Метрика: Максимальна пропускна здатність шляхів. Застосування: Виявлення перевантажених сегментів.

– Балансування навантаження у мережі. Визначення точок, які обробляють надмірну кількість трафіку. Метрика: Посередницька центральність вузлів (Betweenness Centrality). Застосування: Поліпшення балансування для рівномірного розподілу трафіку.

3. Аналіз безпеки мережі:

– Визначення критичних вузлів та ребер. Виявлення точок одиничної відмови (Single Point of Failure). Алгоритм: Пошук точок артикуляції та мостів. Застосування: Посилення безпеки уразливих ділянок мережі.

– Виявлення підозрілих шаблонів зв’язності. Ідентифікація аномалій у структурі мережі. Метрика: Коефіцієнт кластеризації (Clustering Coefficient). Застосування: Виявлення небезпечних структур, що можуть бути ознаками DDoS-атак.

– Аналіз проникнення (Attack Simulation). Моделювання атаки на мережу шляхом видалення ключових вузлів. Метрика: Число компонент зв’язності після видалення. Застосування: Перевірка надійності мережі до кіберзагроз.

Аналіз поширення комп'ютерних вірусів у мережі допомагає оцінити ризики, виявити критичні вузли та розробити стратегії захисту. Для цього використовуються епідеміологічні моделі, адаптовані до кіберпростору.

Основні моделі поширення вірусів:

– SIS (Susceptible-Infected-Susceptible). Вузол може заразитися, потім вилікуватися, але залишається вразливим. Підходить для мереж із повторюваними загрозами (наприклад, ботнети, руткіти).

– SIR (Susceptible-Infected-Removed). Вузол може заразитися, потім виліковується та більше не заражається. Використовується для разових атак (наприклад, віруси-вимагачі).

– SEIR (Susceptible-Exposed-Infected-Removed). Включає стадію латентного періоду, коли вірус поширюється, але ще не активний. Підходить для мережевих атак із затримкою активації.

– SIQR (Susceptible-Infected-Quarantined-Recovered). Включає механізм ізоляції заражених вузлів. Використовується у захищених мережах із антивірусами та IDS.

Фактори, які впливають на поширення вірусу:

– Ступінь вузлів (Degree Centrality) – вузли з великою кількістю зв'язків заражають більше інших.

– Посередницька центральність (Betweenness Centrality) – вузли, через які проходить багато трафіку, стають ключовими точками поширення.

– Коефіцієнт кластеризації (Clustering Coefficient) – високе значення означає, що вірус довше поширюється локально.

– Щільність мережі (Density) – у щільних графах вірус поширюється швидше.

– Коефіцієнт відмови (Failure Rate) – ймовірність видалення зараженого вузла через безпеку.

Було реалізовано модель SIR за наступним алгоритмом:

– Вибір початкових заражених вузлів (рандомно або через центральність).

– Розповсюдження вірусу за правилами обраної моделі.

– Візуалізація процесу (кольорове позначення заражених, здорових і видалених вузлів).

– Звіт по фінальному стану мережі.

3.2 Розробка структурної схеми

Структурна схема програмного забезпечення складається з наступних блоків, рис. 3.1:

- Генерація графу комп'ютерної мережі.
- Візуалізацію структури комп'ютерної мережі.
- Симуляцію руху мережевого трафіку.
- Симуляція поширення комп'ютерних вірусів.
- Аналіз структури комп'ютерної мережі.
- Рекомендації з оптимізації комп'ютерної мережі.
- Рекомендації з усунення слабких місць комп'ютерної мережі.

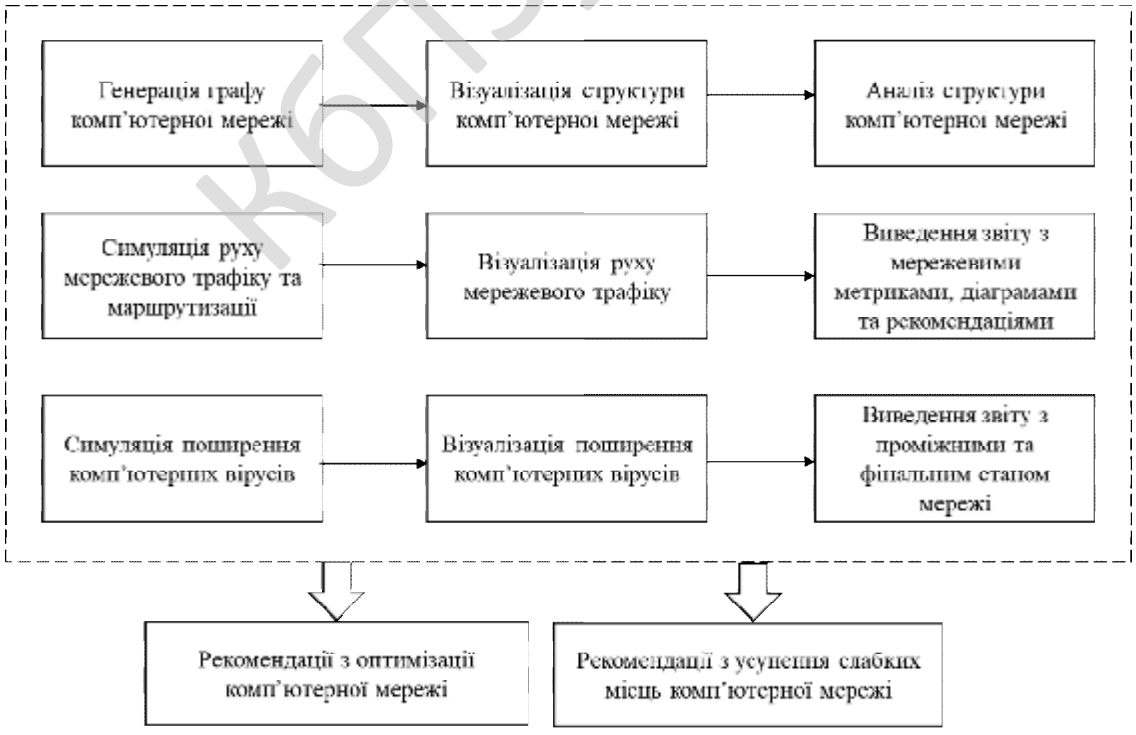

Рисунок 3.1 – Структурна схема системи

3.3 Розробка функціональної схеми

На рисунку 3.2 наведена функціональна схема системи.

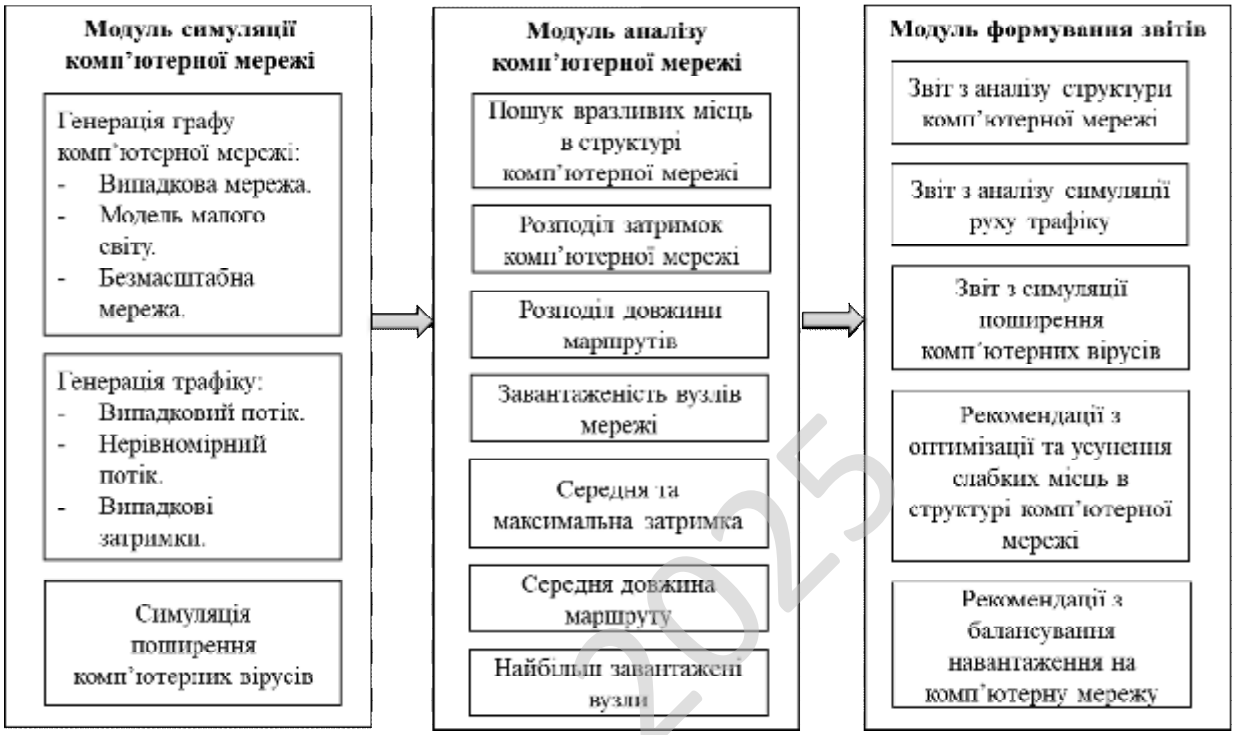

Рисунок 3.2 – Функціональна схема системи

Розроблювана система складається з наступних модулів:

– Модуль симуляції комп'ютерної мережі.

– Модуль аналізу комп'ютерної мережі.

– Модуль формування звітів.

Модуль симуляції комп'ютерної мережі містить наступні функції:

– Генерація графу комп'ютерної мережі методами випадкова мережа (модель Erdős-Rényi), модель малого світу (модель Watts-Strogatz) та безмасштабна мережа (модель Barabási-Albert).

– Генерація трафіку методами випадковий потік, нерівномірний потік, випадкові затримки.

– Симуляція поширення комп'ютерних вірусів.

Модуль аналізу комп’ютерної мережі містить наступні функції:

– Пошук вразливих місць в структурі комп’ютерної мережі.

– Розподіл затримок комп’ютерної мережі.

– Розподіл довжини маршрутів.

– Завантаженість вузлів мережі.

– Середня та максимальна затримка.

– Середня довжина маршруту.

– Найбільш завантажені вузли.

Модуль формування звітів містить наступні функції:

– Звіт з аналізу структури комп’ютерної мережі.

– Звіт з аналізу симуляції руху трафіку.

– Звіт з симуляції поширення комп’ютерних вірусів.

– Рекомендації з оптимізації та усунення слабких місць в структурі комп’ютерної мережі.

– Рекомендації з балансування навантаження на комп’ютерну мережу.

3.4 Розробка діаграми процесів

Діаграма процесів системи зображена на рис. 3.3. Як видно з рисунку, у системі є наступні процеси:

– Головне вікно програми.

– Генерація структури комп’ютерної мережі.

– Аналіз структури комп’ютерної мережі.

– Звіт та рекомендації після аналізу структури.

– Симуляція руху трафіку.

– Звіт та рекомендації після аналізу руху трафіку.

– Візуалізація роботи комп’ютерної мережі.

– Візуалізація графу мережі.

– Симуляція поширення комп’ютерних вірусів.

– Звіт після симуляції поширення вірусів.

Рисунок 3.3 – Діаграма процесів системи

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рис. 4.1 наведена блок-схема основної програми.

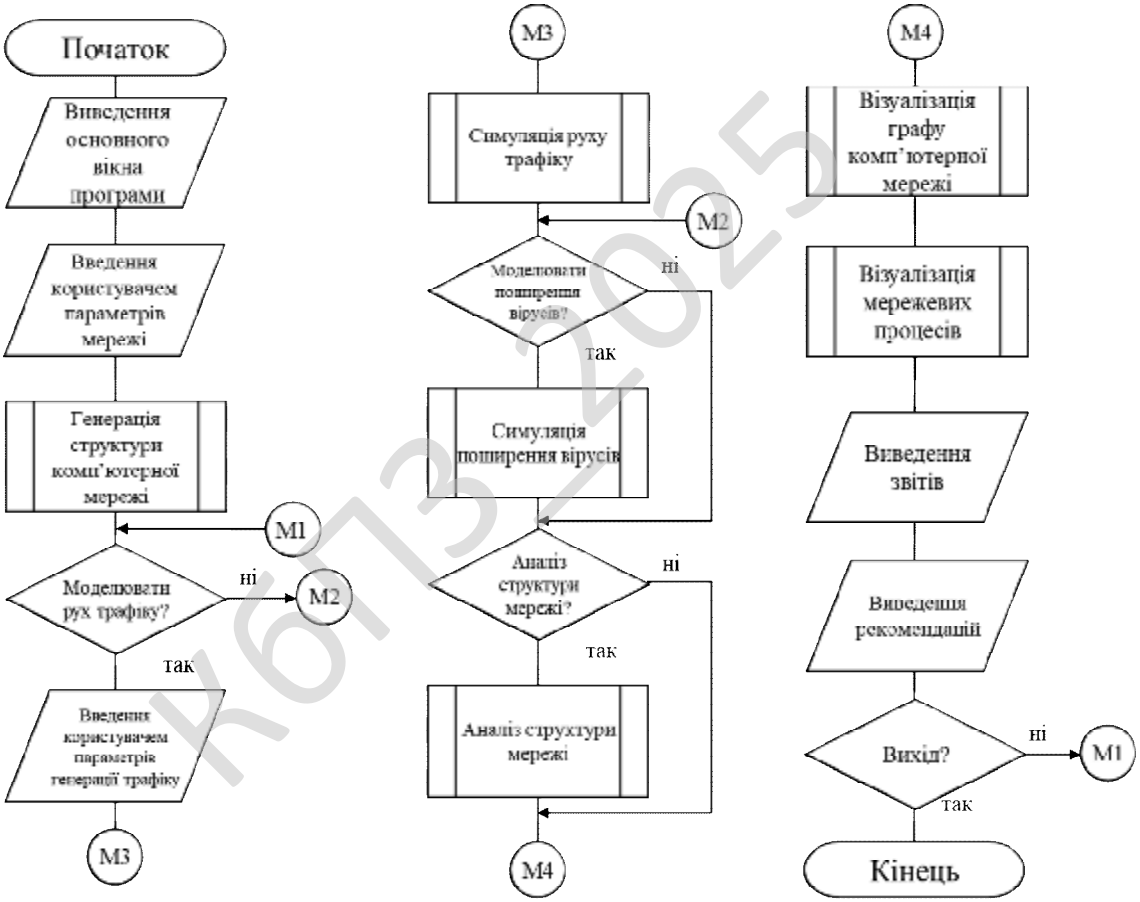

Рисунок 4.1 – Блок-схема роботи основної програми

Спочатку користувачу дається вибір між трьома популярними алгоритмами генерації структури мереж:

– Erdős–Rényi (випадкова мережа).

– Watts–Strogatz (модель малого світу).

– Barabási–Albert (безмасштабна мережа).

Потім генерується граф мережі відповідно до вибраного алгоритму й відбувається його візуалізація із допомогою бібліотеки matplotlib.

Далі матриця суміжності графу згенерованої мережі зберігається у файл у форматі JSON.

Якщо необхідно здійснити наступні дії з графом мережі, то зчитується матриця суміжності з файлу. Користувач обирає один із алгоритмів візуалізації:

– Spring Layout (пружинна модель).

– Circular Layout (розміщення по колу).

– Spectral Layout (спектральне розташування).

– Kamada-Kawai Layout (алгоритм Камади-Каваї).

Граф візуалізується відповідно до вибраного алгоритму.

Для симуляції руху трафіку використовується алгоритм SPF (Shortest Path First) для маршрутизації пакетів. Сам обмін трафіком між вузлами симулюється за допомогою однієї з моделей на вибір користувача:

– Poisson Traffic Model (потік, що відповідає розподілу Пуассона).

– Bursty Traffic Model (нерівномірний трафік).

– Random Traffic Model (випадкові затримки).

При цьому відбувається візуалізація графу мережі і маршрутів передачі трафіку, відображаються значення затримок на маршрутах.

Вузли-відправники позначаються червоним кольором, вузли-отримувачі – зеленим, інші вузли – блакитним. Користувач може обрати кількість ітерацій (відправок пакетів). На екран виводиться детальна інформація про передачу трафіку (відправник, отримувач, маршрут, затримка). Інформація про передачу трафіку виводиться на екран.

Після завершення симуляції формується звіт. Виводиться значення метрик мережі, зокрема:

– Середня та максимальна затримка.

– Середня довжина маршруту.

– Найбільш завантажені вузли.

Аналізуються проблеми мережі та виводяться рекомендації:

– Якщо затримки занадто високі → пропонується оптимізація маршрутів або збільшення пропускної здатності.

– Якщо вузли перевантажені → рекомендація балансувати навантаження.

– Якщо маршрути надто довгі → пропонується змінити топологію мережі.

Також будуються графіки розподілу затримок, довжин маршрутів та завантаженості вузлів.

Якщо необхідно проаналізувати структуру мережі, то здійснюється наступний аналіз:

1. Аналізується центральність вузлів (degree, betweenness, closeness, eigenvector centrality).

2. Визначаються вузькі місця – точки артикуляції та мости.

3. Перевіряється зв’язність мережі – кількість компонент і найбільший підграф.

4. Візуалізується граф, підсвічуючи критичні вузли (червоним) та критичні ребра (сірим).

5. Даються рекомендації щодо покращення структури мережі:

– Вказуються перевантажені вузли та пропонується балансування навантаження.

– Визначаються критичні маршрути та радиться дублювання зв’язків.

– Аналізується зв’язність мережі та рекомендується усунення ізольованих підграфів.

Можливий також поглиблений аналіз мережі, включаючи:

– Мінімальне кістякове дерево (MST) – визначається оптимальна топологія мережі.

– Центральні вузли мережі – визначаються найефективніші точки маршрутизації.

Симуляція поширення комп'ютерних вірусів відбувається наступним чином:

1. Завантажується граф мережі з файлу.
2. Візуалізується структура мережі (початковий стан).
3. Підсвічуються різними кольорами вузли:

– Червоний – заражені.

– Сірий – видалені (ізольовані).

– Блакитний – здорові.

4. Користувач обтирає кількість ітерацій симуляції.
5. Кожна ітерація візуалізується та виводиться у консоль, показуючи:

– Кількість заражених вузлів.

– Кількість видалених (відновлених або ізольованих) вузлів.

6. Механізм зараження та відновлення:

– Вузли заражають своїх сусідів з імовірністю infection_rate.

– Імовірність відновлення зараженого вузла визначається recovery_rate.

7. Відображається фінальний стан мережі та формується звіт.

На рис. 4.2 наведена блок-схема генерації структури комп'ютерної мережі на основі моделі Ердеша-Реньї (випадкова мережа). Кроки даного алгоритму наступні:

1. Створення початкового графу:

– Вибирається кількість вузлів N та ймовірність з'єднань P.

– Створюється порожній граф без зв'язків (вузли є, але ребер немає).

2. Перевірка кожної можливої пари вузлів:

– Перебираємо всі пари вузлів (u, v), де $u \neq v$ (тобто кожен вузол перевіряється з усіма іншими).

– Для кожної такої пари випадково визначаємо, чи буде між ними зв'язок.

3. Ймовірнісне додавання ребра:

– Для кожної пари (u, v) генерується випадкове число R від 0 до 1.

– Якщо $R < P$, то між u і v додається ребро.

– Таким чином, P визначає густину графа: якщо $P = 0$, граф буде повністю розрідженим (без зв’язків); якщо $P = 1$, граф буде повністю з'єднаним (кожен вузол з’єднаний з усіма іншими); якщо $P = 0.5$, граф буде приблизно наполовину заповненим.

4. Отримання випадкового графа:

– Після перевірки всіх пар отримуємо випадкову структуру мережі.

– Чим вищий параметр P, тим більше зв’язків у графі.

– Граф може бути зв'язним або розділеним на компоненти, залежно від P та N.

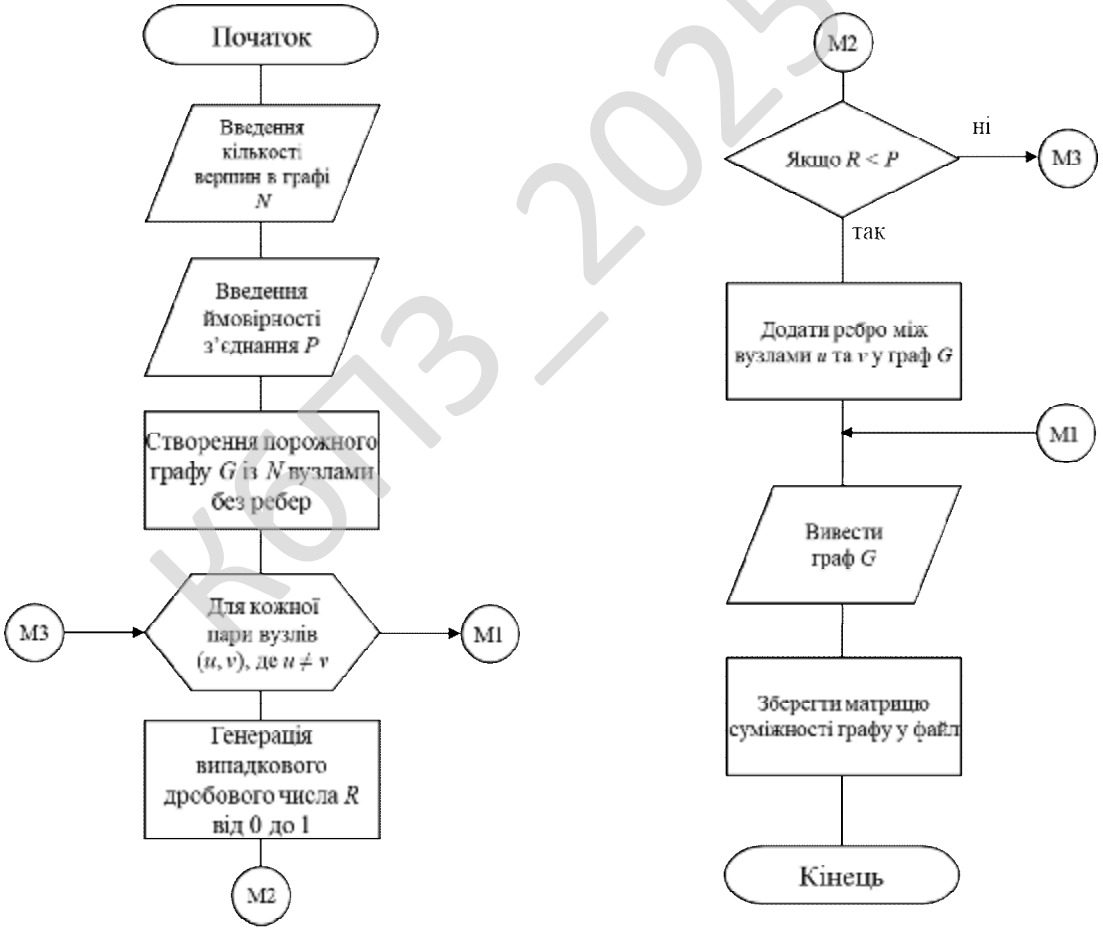

Рисунок 4.2 – Блок-схема генерації структури комп’ютерної мережі на основі моделі Ердеша-Реньї (випадкова мережа)

Розглянемо деякі реалізовані підпрограми.

Генерацію та візуалізацію графу комп'ютерної мережі було реалізовано за допомогою наступного коду:

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import json

def generate_network(algorithm, num_nodes, extra_param=None):
    if algorithm == "erdos-renyi":
        p = extra_param if extra_param else 0.1  # Ймовірність з'єднання
        G = nx.erdos_renyi_graph(num_nodes, p)
    elif algorithm == "watts-strogatz":
        k = extra_param if extra_param else 4  # Кількість сусідів
        p = 0.3  # Ймовірність перемикання ребер
        G = nx.watts_strogatz_graph(num_nodes, k, p)
    elif algorithm == "barabasi-albert":
        m = extra_param if extra_param else 2  # Кількість зв'язків для нового
вузла
        G = nx.barabasi_albert_graph(num_nodes, m)
    else:
        print("Введено некоректні дані. Використовується модель за
замовчуванням - модель Erdős–Rényi.")
        G = nx.erdos_renyi_graph(num_nodes, 0.1)
    return G

def visualize_graph(G):
    plt.figure(figsize=(8, 6))
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels=True, node_size=500, node_color='lightblue',
edge_color='gray')
    plt.show()

def save_adjacency_matrix(G, filename="adjacency_matrix.json"):
    adjacency_matrix = nx.to_numpy_array(G).tolist()
    with open(filename, "w") as f:
        json.dump(adjacency_matrix, f)
    print(f"Матрицю суміжності збережено у файл {filename}")

def load_and_visualize_graph(filename="adjacency_matrix.json"):
    with open(filename, "r") as f:
        adjacency_matrix = json.load(f)
    G = nx.from_numpy_array(np.array(adjacency_matrix))
    visualize_graph(G)
    print("Граф зчитано та візуалізовано з матриці суміжності.")

if __name__ == "__main__":
    print("Оберіть опцію:")
    print("1: Генерація нового графа")
    print("2: Завантаження та візуалізація графа")

    option = input("Введіть номер опції (1-2): ")

    if option == "1":
        print("Оберіть алгоритм генерації соціальної мережі:")
        print("1: Erdős–Rényi (Випадкова мережа)")
        print("2: Watts–Strogatz (Модель малого світу)")
        print("3: Barabási–Albert (Безмасштабна мережа)")
```

```
        choice = input("Введіть номер алгоритму (1-3): ")
        num_nodes = int(input("Введіть кількість вузлів у мережі: "))

        if choice == "1":
            p = float(input("Введіть ймовірність з'єднання (наприклад, 0.1):
"))
            G = generate_network("erdos-renyi", num_nodes, p)
        elif choice == "2":
            k = int(input("Введіть середню кількість сусідів (наприклад, 4):
"))
            G = generate_network("watts-strogatz", num_nodes, k)
        elif choice == "3":
            m = int(input("Введіть кількість зв'язків для нового вузла
(наприклад, 2): "))
            G = generate_network("barabasi-albert", num_nodes, m)
        else:
            print("Некоректний вибір, використовується Erdős–Rényi.")
            G = generate_network("erdos-renyi", num_nodes, 0.1)

        visualize_graph(G)
        save_adjacency_matrix(G)
    elif option == "2":
        load_and_visualize_graph()
    else:
        print("Некоректний вибір, завершення програми.")
```

Симуляцію руху трафіку у мережі реалізовано наступним чином:

```
import networkx as nx
import numpy as np
import json
import random
import matplotlib.pyplot as plt
from collections import Counter

def load_graph_from_file(filename="adjacency_matrix.json"):
    with open(filename, "r") as f:
        adjacency_matrix = json.load(f)
    G = nx.from_numpy_array(np.array(adjacency_matrix))
    return G

def shortest_path_routing(G, source, target):
    try:
        path = nx.shortest_path(G, source=source, target=target, weight=None)
        return path
    except nx.NetworkXNoPath:
        return None

def simulate_traffic(G, num_packets=10, traffic_model="poisson"):
    traffic_data = []
    nodes = list(G.nodes())

    for _ in range(num_packets):
        src, dst = random.sample(nodes, 2)
        path = shortest_path_routing(G, src, dst)

        if path is None:
            continue
```

```
            if traffic_model == "poisson":
                delay = np.random.poisson(5)
            elif traffic_model == "bursty":
                delay = np.random.exponential(2)
            else:
                delay = random.uniform(1, 5)

            traffic_data.append((src, dst, path, delay))

        return traffic_data

    def analyze_and_visualize(G, traffic_data):
        pos = nx.spring_layout(G)
        delays = []
        path_lengths = []
        node_load = Counter()

        for i, (src, dst, path, delay) in enumerate(traffic_data):
            plt.figure(figsize=(8, 6))

            senders = [src]
            receivers = [dst]

            node_colors = ["red" if node in senders else "green" if node in
    receivers else "lightblue" for node in G.nodes()]
            nx.draw(G, pos, with_labels=True, node_size=500,
    node_color=node_colors, edge_color='gray')

            path_edges = list(zip(path, path[1:]))
            nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color='blue',
    width=2)
            mid_node = path[len(path) // 2]
            plt.text(pos[mid_node][0], pos[mid_node][1], f"{delay:.1f}s",
    fontsize=9, color='blue')

            plt.title(f"Ітерація {i+1}: {src} -> {dst}, Затримка: {delay:.1f}s")
            plt.show()

            print(f"Ітерація {i+1}: {src} -> {dst} через {path}, затримка:
    {delay:.1f}s")

            delays.append(delay)
            path_lengths.append(len(path))
            for node in path:
                node_load[node] += 1

        avg_delay = np.mean(delays)
        max_delay = np.max(delays)
        avg_path_length = np.mean(path_lengths)
        max_node_load = max(node_load.values()) if node_load else 0
        busiest_nodes = [node for node, load in node_load.items() if load ==
    max_node_load]

        print("\n--- АНАЛІЗ МЕРЕЖІ ---")
        print(f"Середня затримка: {avg_delay:.2f} с")
        print(f"Максимальна затримка: {max_delay:.2f} с")
        print(f"Середня довжина маршруту: {avg_path_length:.2f} вузлів")
        print(f"Найбільш завантажені вузли: {busiest_nodes} (Обробили
    {max_node_load} запитів)")

        if avg_delay > 10:
```

```
        print("⚠ Високі затримки в мережі! Рекомендація: Оптимізуйте
маршрутизацію або додайте додаткові канали.")
    if max_node_load > len(G.nodes()) / 2:
        print("⚠ Деякі вузли перевантажені! Рекомендація: Додайте
балансування навантаження або резервні вузли.")
    if avg_path_length > len(G.nodes()) / 3:
        print("⚠ Довгі маршрути передачі! Рекомендація: Оптимізуйте топологі
мережі для зменшення відстані між вузлами.")

    # Побудова аналітичних графіків
    plt.figure(figsize=(10, 5))
    plt.hist(delays, bins=5, color='blue', alpha=0.7)
    plt.xlabel("Затримка (s)")
    plt.ylabel("Частота")
    plt.title("Розподіл затримок у мережі")
    plt.show()

    plt.figure(figsize=(10, 5))
    plt.hist(path_lengths, bins=5, color='green', alpha=0.7)
    plt.xlabel("Довжина маршруту")
    plt.ylabel("Частота")
    plt.title("Розподіл довжин маршрутів")
    plt.show()

    plt.figure(figsize=(10, 5))
    plt.bar(node_load.keys(), node_load.values(), color='purple')
    plt.xlabel("Вузли")
    plt.ylabel("Завантаження")
    plt.title("Завантаженість вузлів у мережі")
    plt.show()

if __name__ == "__main__":
    G = load_graph_from_file()

    print("Оберіть модель передачі трафіку:")
    print("1: Poisson Traffic Model (випадковий потік)")
    print("2: Bursty Traffic Model (нерівномірний потік)")
    print("3: Random Traffic Model (випадкові затримки)")

    choice = input("Введіть номер моделі (1-3): ")

    if choice == "1":
        traffic_model = "poisson"
    elif choice == "2":
        traffic_model = "bursty"
    elif choice == "3":
        traffic_model = "random"
    else:
        print("Некоректний вибір, використовується Poisson Model.")
        traffic_model = "poisson"

    num_packets = int(input("Введіть кількість ітерацій (відправок пакетів):
"))
    traffic_data = simulate_traffic(G, num_packets=num_packets,
traffic_model=traffic_model)
    analyze_and_visualize(G, traffic_data)
```

Візуалізація графу комп'ютерної мережі у розробленому програмному забезпеченні реалізована наступним чином:

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import json

def load_graph_from_file(filename="adjacency_matrix.json"):
    with open(filename, "r") as f:
        adjacency_matrix = json.load(f)
    G = nx.from_numpy_array(np.array(adjacency_matrix))
    return G

def visualize_graph(G, layout_type="spring"):
    plt.figure(figsize=(8, 6))

    if layout_type == "spring":
        pos = nx.spring_layout(G)
    elif layout_type == "circular":
        pos = nx.circular_layout(G)
    elif layout_type == "spectral":
        pos = nx.spectral_layout(G)
    elif layout_type == "kamada-kawai":
        pos = nx.kamada_kawai_layout(G)
    else:
        print("Невідомий тип розташування. Використовується spring layout.")
        pos = nx.spring_layout(G)

    nx.draw(G, pos, with_labels=True, node_size=500, node_color='lightblue',
edge_color='gray')
    plt.title(f"Граф із використанням {layout_type} layout")
    plt.show()

if __name__ == "__main__":
    print("Оберіть алгоритм візуалізації графа:")
    print("1: Spring Layout (пружинна модель)")
    print("2: Circular Layout (по колу)")
    print("3: Spectral Layout (спектральний)")
    print("4: Kamada-Kawai Layout")

    choice = input("Введіть номер алгоритму (1-4): ")
    G = load_graph_from_file()

    if choice == "1":
        visualize_graph(G, "spring")
    elif choice == "2":
        visualize_graph(G, "circular")
    elif choice == "3":
        visualize_graph(G, "spectral")
    elif choice == "4":
        visualize_graph(G, "kamada-kawai")
    else:
        print("Некоректний вибір, використовується spring layout.")
        visualize_graph(G, "spring")
```

4.2 Захист розробленого програмного забезпечення

При розповсюдженні програмного забезпечення важливо визначити, на яких умовах його можна використовувати, змінювати та поширювати. Одним із популярних варіантів ліцензування є Creative Commons (CC), яка дозволяє гнучко регулювати права на використання програмного коду.

Creative Commons (CC) – це набір відкритих ліцензій, що дозволяють авторам захистити свої права та водночас визначити умови, за яких інші можуть використовувати їхні розробки.

На відміну від традиційного авторського права, де "всі права захищені" (All rights reserved), Creative Commons дозволяє використовувати принцип "Деякі права захищені" (Some rights reserved).

Використання ліцензії Creative Commons у розробці програмного забезпечення має кілька переваг:

– Гнучкість у ліцензуванні – розробник сам обирає, що дозволено, а що заборонено.

– Прозорість – користувачі точно знають, що вони можуть робити з програмою.

– Захист авторських прав – програма залишається вільно доступною, але використання контролюється.

– Можливість комерційного використання (залежно від обраного типу ліцензії).

Creative Commons пропонує 6 основних варіантів ліцензування, які комбінують чотири ключові умови:

– BY – авторство (Attribution) – необхідно вказувати автора.

– NC – некомерційне використання (Non-Commercial) – не можна використовувати у комерційних цілях.

– ND – без похідних творів (No Derivatives) – не можна змінювати код.

– SA – ліцензія з розповсюдженням на тих самих умовах (ShareAlike) –

модифікації коду повинні поширюватися з тією ж ліцензією.

Найпопулярніші варіанти CC для програмного забезпечення:

1. **CC BY** – дозволяє використовувати програму в будь-яких цілях (у тому числі комерційних) за умови зазначення автора.

2. **CC BY-SA** – дозволяє змінювати та розповсюджувати програму, але модифіковані версії повинні мати ту саму ліцензію.

3. **CC BY-NC** – забороняє комерційне використання.

4. **CC BY-ND** – дозволяє розповсюджувати програму, але забороняє зміни.

Для захисту створеного ПЗ можна обрати ліцензію **Creative Commons CC BY-SA**, яка:

- **Гарантує авторські права** розробника (всі, хто використовують код, повинні вказувати автора).
- **Забезпечує відкритість проєкту** – користувачі можуть змінювати код, але поширювати його лише з тією ж ліцензією.
- **Дозволяє комерційне використання** – інші можуть вільно використовувати програму навіть у бізнесі, якщо дотримуються умов ліцензії.

Для додавання ліцензії до коду треба у файлі LICENSE.txt написати:

```
Copyright (c) [Ім'я розробника]
This software is licensed under Creative Commons Attribution-ShareAlike (CC BY-SA) 4.0.
You are free to use, modify, and distribute this software under the same license terms.
Full text: https://creativecommons.org/licenses/by-sa/4.0/
```

Також можна додати коментар у код:

```
"""
Програмне забезпечення ліцензоване за CC BY-SA 4.0
Автор: [Ім'я розробника]
"""
```

Creative Commons дозволяє гнучко ліцензувати програмне забезпечення, не забороняючи його використання, але забезпечуючи правову охорону

авторства.

Для відкритого ПЗ найкраще підходять CC BY та CC BY-SA.

Вибір ліцензії залежить від мети проєкту – чи можна змінювати код, використовувати його в комерційних цілях тощо.

Використання CC дає змогу розробнику контролювати поширення своєї програми і водночас сприяє її популяризації.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Для впровадження в експлуатацію розробленого програмного забезпечення треба встановити нижченаведені бібліотеки для мови програмування Python:

– **networkx** – Бібліотека для створення, аналізу та візуалізації графів. Використовується для генерації мереж за різними моделями (Erdős–Rényi, Watts–Strogatz, Barabási–Albert), аналізу центральності вузлів, пошуку критичних точок, мінімальних остовних дерев і виконання інших графових операцій.

– **matplotlib.pyplot** – Бібліотека для візуалізації даних. Використовується для відображення графів та побудови різних аналітичних графіків, таких як розподіл затримок, довжини маршрутів та завантаженості вузлів.

– **numpy** – Бібліотека для роботи з масивами та лінійною алгеброю. Використовується для перетворення матриці суміжності в графи, моделювання випадкових процесів та розрахунків у графових алгоритмах.

– **json** – Вбудована бібліотека Python для серіалізації та десеріалізації JSON-даних. Використовується для збереження та завантаження матриці суміжності графа, що дозволяє працювати з мережею між різними запусками програми.

Приклад роботи розробленого додатку наведений на рис. 5.1-5.10.

```
Оберіть опцію:
1: Генерація нового графа
2: Завантаження та візуалізація графа
Введіть номер опції (1-2): 1
Оберіть алгоритм генерації соціальної мережі:
1: Erdős–Rényi (Випадкова мережа)
2: Watts–Strogatz (Модель малого світу)
3: Barabási–Albert (Безмасштабна мережа)
Введіть номер алгоритму (1-3): 1
Введіть кількість вузлів у мережі: 10
Введіть ймовірність з'єднання (наприклад, 0.1): 0.5
Матрицю суміжності збережено у файл adjacency_matrix.json
```

Рисунок 5.1 – Приклад користувацького діалогу на початку роботи програми

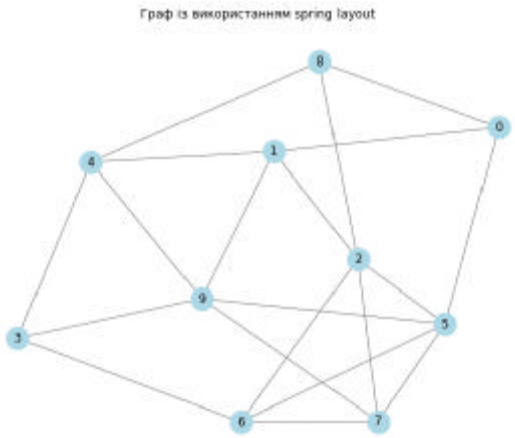

Рисунок 5.2 – Приклад графу комп’ютерної мережі, візуалізованого алгоритмом spring layout

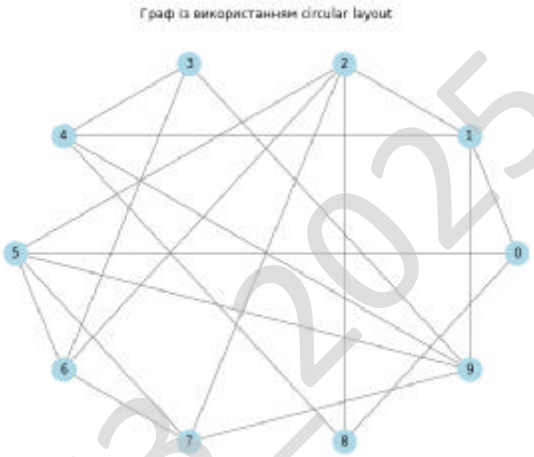

Рисунок 5.3 – Приклад графу комп’ютерної мережі, візуалізованого алгоритмом circular layout

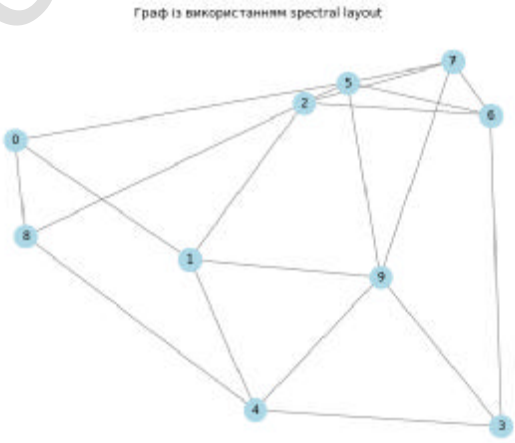

Рисунок 5.4 – Приклад графу комп’ютерної мережі, візуалізованого алгоритмом spectral layout

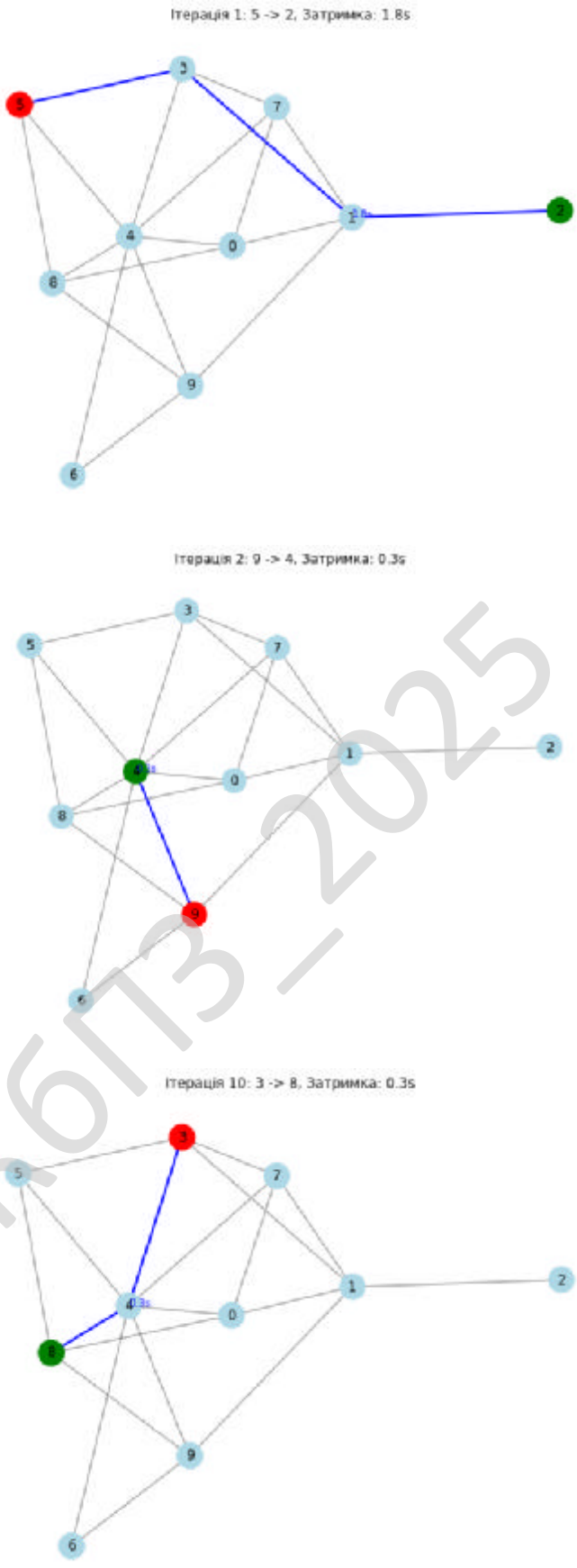

Рисунок 5.5 – Приклад симуляції трафіку, маршрутизації та візуалізації даних процесів

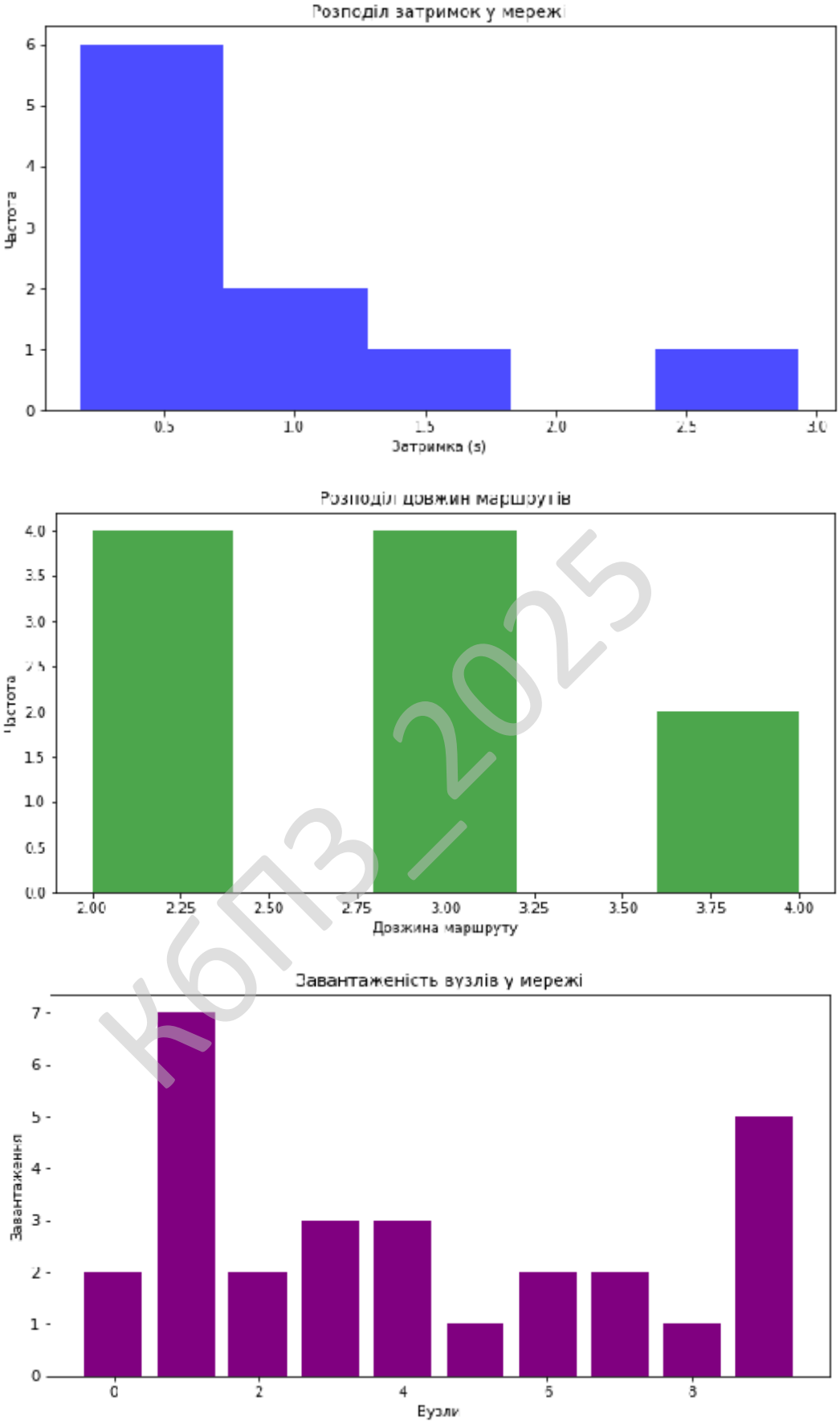

Рисунок 5.6 – Приклад діаграм для звіту після симуляції трафіку

```
Оберіть модель передачі трафіку:
1: Poisson Traffic Model (випадковий потік)
2: Bursty Traffic Model (нерівномірний потік)
3: Random Traffic Model (випадкові затримки)
Введіть номер моделі (1-3): 2
Введіть кількість ітерацій (відправок пакетів): 10
Ітерація 1: 5 -> 2 через [5, 3, 1, 2], затримка: 1.8s
Ітерація 2: 9 -> 4 через [9, 4], затримка: 0.3s
Ітерація 3: 1 -> 6 через [1, 9, 6], затримка: 1.2s
Ітерація 4: 2 -> 6 через [2, 1, 9, 6], затримка: 2.9s
Ітерація 5: 7 -> 4 через [7, 4], затримка: 0.2s
Ітерація 6: 3 -> 1 через [3, 1], затримка: 0.2s
Ітерація 7: 1 -> 0 через [1, 0], затримка: 0.2s
Ітерація 8: 0 -> 9 через [0, 1, 9], затримка: 0.4s
Ітерація 9: 7 -> 9 через [7, 1, 9], затримка: 1.3s
Ітерація 10: 3 -> 8 через [3, 4, 8], затримка: 0.3s

--- АНАЛІЗ МЕРЕЖІ ---
Середня затримка: 0.89 с
Максимальна затримка: 2.93 с
Середня довжина маршруту: 2.80 вузлів
Найбільш завантажені вузли: [1] (Обробили 7 запитів)
⚠ Деякі вузли перевантажені! Рекомендація: Додайте балансування
навантаження або резервні вузли.
```

Рисунок 5.7 – Приклад користувацького діалогу в підпрограмі симуляції трафіку та звіту після нього

Рисунок 5.8 – Приклад аналізу вразливостей комп’ютерної мережі

```
--- АНАЛІЗ ЦЕНТРАЛЬНОСТІ ВУЗЛІВ ---
Вузли з найвищою центральністю:
Ступенева центральність: 20
Посередницька центральність: 47
Близькість: 47
Eigenvector Centrality: 28

    РЕКОМЕНДАЦІЇ ЗА ЦЕНТРАЛЬНІСТЮ
1. Вузли з високою ступеневою центральністю (Degree Centrality) часто є точками високого навантаження.
    > Вузол 20 може стати перевантаженим. Розгляньте додавання резервних з'єднань.
2. Вузли з високою посередницькою центральністю (Betweenness Centrality) є ключовими маршрутами.
   -> Вузол 47 є критичним для багатьох маршрутів. Його відмова може сильно вплинути на мережу.
    > Додайте альтернативні шляхи, щоб знизити ризики.
3. Вузли з високою близькістю (Closeness Centrality) забезпечують швидкий доступ до всієї мережі.
    > Вузол 47 важливий для швидкої доставки пакетів. Його потрібно захистити від перевантаження.
4. Вузли з високою eigenvector centrality мають великий вплив у мережі.
   -> Вузол 20 пов'язаний із ключовими вузлами. Варто забезпечити його надійне з'єднання.

--- ВУЗЬКІ МІСЦЯ В МЕРЕЖІ ---
Точки артикуляції (видалення яких розриває мережу): [16, 17, 18, 19, 27, 28, 29, 20, 23, 39, 37, 38,
40, 41, 42, 43, 44, 45, 46, 47, 14, 49, 9, 12, 6, 5, 4, 3, 2, 1, 32, 34, 35, 0]
Мости (ключові з'єднання між частинами мережі): [(0, 1), (0, 32), (0, 35), (1, 2), (2, 3), (3, 4), (4,
5), (6, 12), (9, 49), (10, 12), (11, 12), (13, 14), (14, 21), (14, 49), (15, 16), (16, 17), (17, 18),
(18, 19), (19, 20), (19, 29), (20, 30), (22, 23), (24, 39), (25, 32), (26, 27), (27, 28), (28, 29),
(31, 32), (33, 34), (34, 35), (36, 37), (37, 38), (38, 39), (39, 40), (40, 41), (41, 42), (42, 43),
(43, 44), (44, 45), (45, 46), (46, 47), (48, 49)]

--- РЕКОМЕНДАЦІЇ ПО ВРАЗЛИВИХ МІСЦЯХ ---
1. Вузли-артикуляції є критичними точками.
    > Захистіть або дублюйте з'єднання для вузлів [16, 17, 18, 19, 27, 28, 20, 23, 39, 37, 38, 40,
41, 42, 43, 44, 45, 46, 47, 14, 49, 9, 12, 6, 5, 4, 3, 2, 1, 32, 34, 35, 0].
2. Мости — це єдині з'єднання між частинами мережі.
   -> Додайте резервні канали замість [(0, 1), (0, 32), (0, 35), (1, 2), (2, 3), (3, 4), (4, 5), (6,
12), (9, 49), (10, 12), (11, 12), (13, 14), (14, 21), (14, 49), (15, 16), (16, 17), (17, 18), (18,
19), (19, 20), (19, 29), (20, 30), (22, 23), (24, 39), (25, 32), (26, 27), (27, 28), (28, 29), (31,
32), (33, 34), (34, 35), (36, 37), (37, 38), (38, 39), (39, 40), (40, 41), (41, 42), (42, 43), (43,
44), (44, 45), (45, 46), (46, 47), (48, 49)].

--- АНАЛІЗ ЗВ'ЯЗНОСТІ ---
Кількість компонент зв'язності: 1
Розмір найбільшої компоненти: 50 вузлів

--- РЕКОМЕНДАЦІЇ ПО ЗВ'ЯЗНОСТІ ---
2. Забезпечте резервні шляхи для ключових з'єднань.
```

Рисунок 5.9 – Приклад звіту після аналізу вразливостей комп’ютерної мережі

```
Мережу успішно завантажено з файлу adjacency_matrix.json.
Введіть кількість ітерацій моделі: 10

--- ПОЧАТОК СИМУЛЯЦІЇ ---
Ітерація 1: 6 заражених, 0 видалених.
Ітерація 2: 14 заражених, 2 видалених.
Ітерація 3: 25 заражених, 2 видалених.
Ітерація 4: 21 заражених, 9 видалених.
Ітерація 5: 15 заражених, 15 видалених.
Ітерація 6: 10 заражених, 20 видалених.
Ітерація 7: 9 заражених, 21 видалених.
Ітерація 8: 5 заражених, 25 видалених.
Ітерація 9: 4 заражених, 26 видалених.
Ітерація 10: 3 заражених, 27 видалених.

--- СИМУЛЯЦІЯ ЗАВЕРШЕНА ---
Фінальний стан: 3 заражених, 27 видалених.
```

Рисунок 5.10 – Приклад звіту після симуляції поширення комп’ютерних вірусів по комп’ютерній мережі

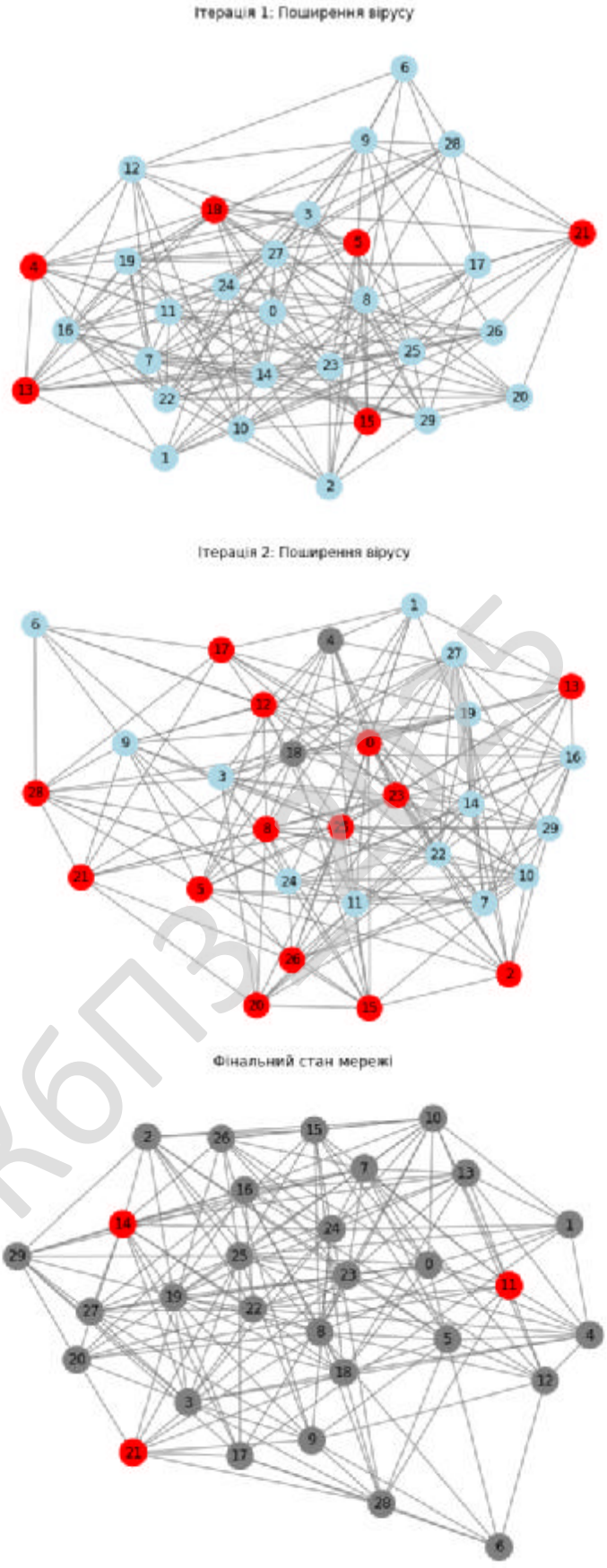

Рисунок 5.11 – Приклад симуляції поширення комп’ютерних вірусів по комп’ютерній мережі

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної роботи, призначено для моделювання роботи комп'ютерних мереж.

Для вирішення поставленої мети було проведено:

– Огляд існуючих систем для моделювання роботи комп'ютерних мереж.

– Проектування системи для моделювання роботи комп'ютерних мереж на основі теорії складних мереж.

– Програмна реалізація системи для моделювання роботи комп'ютерних мереж.

Реалізовані під час виконання кваліфікаційної роботи алгоритми дозволяють успішно вирішувати завдання моделювання роботи комп'ютерних мереж.

Розроблене програмне забезпечення представляє собою додаток, що можна використовувати практично на будь-якому сучасному комп'ютері.

Програмне забезпечення розроблялося у об'єктно-орієнтованій парадигмі, що робить його гнучким та зручним для підтримки та масштабування.

Для розробки програмного забезпечення системи моделювання роботи комп'ютерних мереж використовувалися мова програмування Python та середовище розробки Anaconda і Spyder. Дані мова програмування і середовище розробки дозволили ефективно вирішити поставлену мету та задачі кваліфікаційної роботи.

У п'ятому розділі пояснювальної записки надаються усі необхідні рекомендації з встановлення програмного забезпечення та інструкція для його використання.

Для захисту розробленого програмного забезпечення було обрано вільну ліцензію Creative Commons.

Програмне забезпечення системи моделювання роботи комп'ютерних

мереж є важливим інструментом при проектуванні комп’ютерних мереж. Враховуючи все більшу складність комп’ютерних мереж, використання такого роду програмного забезпечення стає не просто рекомендацією, а необхідністю. Тож, розроблене у цій кваліфікаційній роботі програмне забезпечення має важливе призначення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. – Pearson, 2001. – 620 p.13. Кормен Т., Лейзерсон Ч., Риверст Р., Штайн К. Алгоритмы: построение и анализ, 3-е издание – М.: Диалектика, 2019. – 1328 p.

2. Areström E., Carlsson N. Early online classification of encrypted traffic streams using multi-fractal features. IEEE INFOCOM 2019 - IEEE Conf. on Computer Communications Workshops (INFOCOM WKSHPS). 2019. P. 84–89. DOI: 10.1109/INFCOMW.2019.8845127.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

18. Fast low-rank matrix approximation with locality sensitive hashing for quick anomaly detection / G. Xie, K. Xie, J. Huang et al. IEEE INFOCOM 2017: IEEE

Conf. on Computer Communications. 2017. P. 1–9.

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

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

21. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology 1st Edition. – Cambridge University Press, 2008. – 556 p.

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

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

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

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

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

DOI: 10.3390/data4010005.

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

28. Lambert K. A. Fundamentals of Python: First Programs, 2nd Edition. – Cengage, 2019.

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

30. Lutz M. Learning Python, 5th Edition Fifth Edition. - O'Reilly Media, 2016. - 1643 p.

31. Lutz M. Python: Pocket Reference Fourth Edition. - O'Reilly Media, 2016. - 210 p.

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

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

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

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

36. Mirchandani P. B., Zou N. Queuing models for analysis of traffic adaptive signal control. IEEE Transactions on Intelligent Transportation Systems. 2007.

Vol. 8, no. 1. P. 50–59. DOI: 10.1109/TITS.2006.888619.

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

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

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

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

41. Random Geometric Graph // NetworkX documentation – URL: https://networkx.org/documentation/stable/auto_examples/drawing/plot_random_geometric_graph.html#sphx-glr-auto-examples-drawing-plot-random-geometric-graph-py

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

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

44. Sobh T., Elleithy K., Mahmood A. Novel Algorithms and Techniques in Telecommunications and Networking. Springer, 2010. P. 41–46.

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

46. The Python Tutorial. –https://docs.python.org/3/tutorial/index.html

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

48. Traag V. A. Algorithms and dynamical models for communities and reputation in social networks. Springer International Publishing. 2014. P. 229. URL:

https://doi.org/10.1007/978-3-319-06391-1.

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

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

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

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

53. Xu Y., Li Q., Meng S. (2019). Self-similarity Analysis and Application of Network Traffic. In: Yin, Y., Li, Y., Gao, H., Zhang, J. (eds) Mobile Computing, Applications, and Services. MobiCASE 2019. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 290. Springer, Cham. https://doi.org/10.1007/978-3-030-28468-8_9

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

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

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи для моделювання роботи комп’ютерних мереж.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу №47-02 від 17.01.2025 року, видане на кафедрі кібербезпеки та програмного забезпечення.

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення системи для моделювання роботи комп’ютерних мереж.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– систему для моделювання роботи комп'ютерних мереж;
– 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 архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

Комп’ютер Intel Core i7/8 ГБ /1 Tb/ GeForce GT 1030 2GB або сумісні з ним.

5.8.2 Мова програмування

Програму розроблено на мовах програмування 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.
– Функціональна схема системи.
– Діаграма процесів.
– Блок-схема алгоритму роботи програми.
– Пояснювальна записка.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

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.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 24.05.2025 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист __.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

_____ Є.В. Мелешко

Програмне забезпечення для моделювання роботи комп’ютерних мереж

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 15

Літера: РП

Кропивницький – 2025 року

Файл main.py основної програми

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import json

def generate_network(algorithm, num_nodes, extra_param=None):
    if algorithm == "erdos-renyi":
        p = extra_param if extra_param else 0.1  # Ймовірність з'єднання
        G = nx.erdos_renyi_graph(num_nodes, p)
    elif algorithm == "watts-strogatz":
        k = extra_param if extra_param else 4  # Кількість сусідів
        p = 0.3  # Ймовірність перемикання ребер
        G = nx.watts_strogatz_graph(num_nodes, k, p)
    elif algorithm == "barabasi-albert":
        m = extra_param if extra_param else 2  # Кількість зв'язків для нового
вузла
        G = nx.barabasi_albert_graph(num_nodes, m)
    else:
        print("Введено некоректні дані. Використовується модель за замовчуванням
- модель Erdős–Rényi.")
        G = nx.erdos_renyi_graph(num_nodes, 0.1)
    return G

def visualize_graph(G):
    plt.figure(figsize=(8, 6))
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels=True, node_size=500, node_color='lightblue',
edge_color='gray')
    plt.show()

def save_adjacency_matrix(G, filename="adjacency_matrix.json"):
    adjacency_matrix = nx.to_numpy_array(G).tolist()
    with open(filename, "w") as f:
        json.dump(adjacency_matrix, f)
    print(f"Матрицю суміжності збережено у файл {filename}")

def load_and_visualize_graph(filename="adjacency_matrix.json"):
    with open(filename, "r") as f:
        adjacency_matrix = json.load(f)
    G = nx.from_numpy_array(np.array(adjacency_matrix))
    visualize_graph(G)
    print("Граф зчитано та візуалізовано з матриці суміжності.")

if __name__ == "__main__":
    print("Оберіть опцію:")
    print("1: Генерація нового графа")
    print("2: Завантаження та візуалізація графа")

    option = input("Введіть номер опції (1-2): ")

    if option == "1":
        print("Оберіть алгоритм генерації комп’ютерної мережі:")
        print("1: Erdős–Rényi (Випадкова мережа)")
        print("2: Watts–Strogatz (Модель малого світу)")
        print("3: Barabási–Albert (Безмасштабна мережа)")

        choice = input("Введіть номер алгоритму (1-3): ")
        num_nodes = int(input("Введіть кількість вузлів у мережі: "))

        if choice == "1":
            p = float(input("Введіть ймовірність з'єднання (наприклад, 0.1): "))
            G = generate_network("erdos-renyi", num_nodes, p)
        elif choice == "2":
```

```
            k = int(input("Введіть середню кількість сусідів (наприклад, 4): "))
            G = generate_network("watts-strogatz", num_nodes, k)
        elif choice == "3":
            m = int(input("Введіть кількість зв'язків для нового вузла
(наприклад, 2): "))
            G = generate_network("barabasi-albert", num_nodes, m)
        else:
            print("Некоректний вибір, використовується Erdős–Rényi.")
            G = generate_network("erdos-renyi", num_nodes, 0.1)

        visualize_graph(G)
        save_adjacency_matrix(G)
    elif option == "2":
        load_and_visualize_graph()
    else:
        print("Некоректний вибір, завершення програми.")
```

Файл visual_graph.py основної програми

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import json

def load_graph_from_file(filename="adjacency_matrix.json"):
    with open(filename, "r") as f:
        adjacency_matrix = json.load(f)
    G = nx.from_numpy_array(np.array(adjacency_matrix))
    return G

def visualize_graph(G, layout_type="spring"):
    plt.figure(figsize=(8, 6))

    if layout_type == "spring":
        pos = nx.spring_layout(G)
    elif layout_type == "circular":
        pos = nx.circular_layout(G)
    elif layout_type == "spectral":
        pos = nx.spectral_layout(G)
    elif layout_type == "kamada-kawai":
        pos = nx.kamada_kawai_layout(G)
    else:
        print("Невідомий тип розташування. Використовується spring layout.")
        pos = nx.spring_layout(G)

    nx.draw(G, pos, with_labels=True, node_size=500, node_color='lightblue',
edge_color='gray')
    plt.title(f"Граф із використанням {layout_type} layout")
    plt.show()

if __name__ == "__main__":
    print("Оберіть алгоритм візуалізації графа:")
    print("1: Spring Layout (пружинна модель)")
    print("2: Circular Layout (по колу)")
    print("3: Spectral Layout (спектральний)")
    print("4: Kamada-Kawai Layout")

    choice = input("Введіть номер алгоритму (1-4): ")
    G = load_graph_from_file()

    if choice == "1":
        visualize_graph(G, "spring")
    elif choice == "2":
        visualize_graph(G, "circular")
    elif choice == "3":
        visualize_graph(G, "spectral")
    elif choice == "4":
        visualize_graph(G, "kamada-kawai")
    else:
        print("Некоректний вибір, використовується spring layout.")
        visualize_graph(G, "spring")
```

Файл simulate_traffic.py основної програми

```
import networkx as nx
import numpy as np
import json
import random
import matplotlib.pyplot as plt
from collections import Counter

def load_graph_from_file(filename="adjacency_matrix.json"):
    with open(filename, "r") as f:
        adjacency_matrix = json.load(f)
    G = nx.from_numpy_array(np.array(adjacency_matrix))
    return G

def shortest_path_routing(G, source, target):
    try:
        path = nx.shortest_path(G, source=source, target=target, weight=None)
        return path
    except nx.NetworkXNoPath:
        return None

def simulate_traffic(G, num_packets=10, traffic_model="poisson"):
    traffic_data = []
    nodes = list(G.nodes())

    for _ in range(num_packets):
        src, dst = random.sample(nodes, 2)
        path = shortest_path_routing(G, src, dst)

        if path is None:
            continue

        if traffic_model == "poisson":
            delay = np.random.poisson(5)
        elif traffic_model == "bursty":
            delay = np.random.exponential(2)
        else:
            delay = random.uniform(1, 5)

        traffic_data.append((src, dst, path, delay))

    return traffic_data

def analyze_and_visualize(G, traffic_data):
    pos = nx.spring_layout(G)
    delays = []
    path_lengths = []
    node_load = Counter()

    for i, (src, dst, path, delay) in enumerate(traffic_data):
        plt.figure(figsize=(8, 6))

        senders = [src]
        receivers = [dst]

        node_colors = ["red" if node in senders else "green" if node in
receivers else "lightblue" for node in G.nodes()]
        nx.draw(G, pos, with_labels=True, node_size=500, node_color=node_colors,
edge_color='gray')

        path_edges = list(zip(path, path[1:]))
        nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color='blue',
width=2)
        mid_node = path[len(path) // 2]
        plt.text(pos[mid_node][0], pos[mid_node][1], f"{delay:.1f}s",
fontsize=9, color='blue')
```

```
        plt.title(f"Ітерація {i+1}: {src} -> {dst}, Затримка: {delay:.1f}s")
        plt.show()

        print(f"Ітерація {i+1}: {src} -> {dst} через {path}, затримка:
{delay:.1f}s")

        delays.append(delay)
        path_lengths.append(len(path))
        for node in path:
            node_load[node] += 1

    avg_delay = np.mean(delays)
    max_delay = np.max(delays)
    avg_path_length = np.mean(path_lengths)
    max_node_load = max(node_load.values()) if node_load else 0
    busiest_nodes = [node for node, load in node_load.items() if load ==
max_node_load]

    print("\n--- АНАЛІЗ МЕРЕЖІ ---")
    print(f"Середня затримка: {avg_delay:.2f} с")
    print(f"Максимальна затримка: {max_delay:.2f} с")
    print(f"Середня довжина маршруту: {avg_path_length:.2f} вузлів")
    print(f"Найбільш завантажені вузли: {busiest_nodes} (Обробили
{max_node_load} запитів)")

    if avg_delay > 10:
        print("⚠ Високі затримки в мережі! Рекомендація: Оптимізуйте
маршрутизацію або додайте додаткові канали.")
    if max_node_load > len(G.nodes()) / 2:
        print("⚠ Деякі вузли перевантажені! Рекомендація: Додайте балансування
навантаження або резервні вузли.")
    if avg_path_length > len(G.nodes()) / 3:
        print("⚠ Довгі маршрути передачі! Рекомендація: Оптимізуйте топологію
мережі для зменшення відстані між вузлами.")

    # Побудова аналітичних графіків
    plt.figure(figsize=(10, 5))
    plt.hist(delays, bins=5, color='blue', alpha=0.7)
    plt.xlabel("Затримка (s)")
    plt.ylabel("Частота")
    plt.title("Розподіл затримок у мережі")
    plt.show()

    plt.figure(figsize=(10, 5))
    plt.hist(path_lengths, bins=5, color='green', alpha=0.7)
    plt.xlabel("Довжина маршруту")
    plt.ylabel("Частота")
    plt.title("Розподіл довжин маршрутів")
    plt.show()

    plt.figure(figsize=(10, 5))
    plt.bar(node_load.keys(), node_load.values(), color='purple')
    plt.xlabel("Вузли")
    plt.ylabel("Завантаження")
    plt.title("Завантаженість вузлів у мережі")
    plt.show()

if __name__ == "__main__":
    G = load_graph_from_file()

    print("Оберіть модель передачі трафіку:")
    print("1: Poisson Traffic Model (випадковий потік)")
    print("2: Bursty Traffic Model (нерівномірний потік)")
    print("3: Random Traffic Model (випадкові затримки)")

    choice = input("Введіть номер моделі (1-3): ")

    if choice == "1":
```

```
        traffic_model = "poisson"
    elif choice == "2":
        traffic_model = "bursty"
    elif choice == "3":
        traffic_model = "random"
    else:
        print("Некоректний вибір, використовується Poisson Model.")
        traffic_model = "poisson"

    num_packets = int(input("Введіть кількість ітерацій (відправок пакетів): "))
    traffic_data = simulate_traffic(G, num_packets=num_packets,
traffic_model=traffic_model)
    analyze_and_visualize(G, traffic_data)
```

Файл analyze_graph_network.py основної програми

```
import networkx as nx
import numpy as np
import json
import matplotlib.pyplot as plt
from collections import Counter

def load_graph_from_file(filename="adjacency_matrix.json"):
    with open(filename, "r") as f:
        adjacency_matrix = json.load(f)
    G = nx.from_numpy_array(np.array(adjacency_matrix))
    return G

def analyze_centrality(G):
    degree_centrality = nx.degree_centrality(G)
    betweenness_centrality = nx.betweenness_centrality(G)
    closeness_centrality = nx.closeness_centrality(G)
    eigenvector_centrality = nx.eigenvector_centrality(G, max_iter=1000)

    print("\n--- АНАЛІЗ ЦЕНТРАЛЬНОСТІ ВУЗЛІВ ---")
    print("Вузли з найвищою центральністю:")
    high_degree = max(degree_centrality, key=degree_centrality.get)
    high_betweenness = max(betweenness_centrality,
key=betweenness_centrality.get)
    high_closeness = max(closeness_centrality, key=closeness_centrality.get)
    high_eigenvector = max(eigenvector_centrality,
key=eigenvector_centrality.get)

    print(f"Ступенева центральність: {high_degree}")
    print(f"Посередницька центральність: {high_betweenness}")
    print(f"Близькість: {high_closeness}")
    print(f"Eigenvector Centrality: {high_eigenvector}")

    # Рекомендації на основі центральності
    print("\n--- РЕКОМЕНДАЦІЇ ЗА ЦЕНТРАЛЬНІСТЮ ---")
    print("1. Вузли з високою ступеневою центральністю (Degree Centrality) часто
є точками високого навантаження.")
    print(f"   -> Вузол {high_degree} може стати перевантаженим. Розгляньте
додавання резервних з'єднань.")
    print("2. Вузли з високою посередницькою центральністю (Betweenness
Centrality) є ключовими маршрутами.")
    print(f"   -> Вузол {high_betweenness} є критичним для багатьох маршрутів.
Його відмова може сильно вплинути на мережу.")
    print("   -> Додайте альтернативні шляхи, щоб знизити ризики.")
    print("3. Вузли з високою близькістю (Closeness Centrality) забезпечують
швидкий доступ до всієї мережі.")
сокою близькістю (Closeness Centrality) забезпечують швидкий доступ до всієї
мережі.")
    print(f"   -> Вузол {high_closeness} важливий для швидкої доставки пакетів.
Його потрібно захистити від перевантаження.")
    print("4. Вузли з високою eigenvector centrality мають великий вплив у
мережі.")
    print(f"   -> Вузол {high_eigenvector} пов'язаний із ключовими вузлами.
Варто забезпечити його надійне з'єднання.")

    return degree_centrality, betweenness_centrality, closeness_centrality,
eigenvector_centrality

def find_bottlenecks(G):
    articulation_points = list(nx.articulation_points(G))
    bridges = list(nx.bridges(G))

    print("\n--- ВУЗЬКІ МІСЦЯ В МЕРЕЖІ ---")
    print(f"Точки артикуляції (видалення яких розриває мережу):
{articulation_points}")
    print(f"Мости (ключові з'єднання між частинами мережі): {bridges}")
```

```
    # Рекомендації щодо вузьких місць
    print("\n--- РЕКОМЕНДАЦІЇ ПО ВРАЗЛИВИХ МІСЦЯХ ---")
    if articulation_points:
        print("1. Вузли-артикуляції є критичними точками.")
        print(f"   -> Захистіть або дублюйте з'єднання для вузлів
{articulation_points}.")
    if bridges:
        print("2. Мости – це єдині з'єднання між частинами мережі.")
        print(f"   -> Додайте резервні канали замість {bridges}.")

    return articulation_points, bridges

def analyze_connectivity(G):
    num_components = nx.number_connected_components(G)
    largest_component = max(nx.connected_components(G), key=len)

    print("\n--- АНАЛІЗ ЗВ'ЯЗНОСТІ ---")
    print(f"Кількість компонент зв'язності: {num_components}")
    print(f"Розмір найбільшої компоненти: {len(largest_component)} вузлів")

    # Рекомендації з покращення зв’язності
    print("\n--- РЕКОМЕНДАЦІЇ ПО ЗВ'ЯЗНОСТІ ---")
    if num_components > 1:
        print("1. Мережа містить ізольовані підграфи.")
        print("   -> Розгляньте додавання нових з'єднань для об'єднання
підграфів.")
    print("2. Забезпечте резервні шляхи для ключових з'єднань.")

    return num_components, len(largest_component)

def visualize_graph(G, critical_nodes=[], critical_edges=[]):
    plt.figure(figsize=(8, 6))
    pos = nx.spring_layout(G)

    node_colors = ["red" if node in critical_nodes else "lightblue" for node in
G.nodes()]
    edge_colors = ["red" if edge in critical_edges else "gray" for edge in
G.edges()]

    nx.draw(G, pos, with_labels=True, node_size=500, node_color=node_colors,
edge_color=edge_colors)
    plt.title("Візуалізація мережі з критичними вузлами")
    plt.show()

if __name__ == "__main__":
    G = load_graph_from_file()

    # Аналіз центральності
    degree_centrality, betweenness_centrality, closeness_centrality,
eigenvector_centrality = analyze_centrality(G)

    # Визначення вузьких місць
    articulation_points, bridges = find_bottlenecks(G)

    # Аналіз зв’язності
    num_components, largest_component_size = analyze_connectivity(G)

    # Візуалізація графа з критичними вузлами та ребрами
    visualize_graph(G, critical_nodes=articulation_points,
critical_edges=bridges)
```

Файл node_failure_analysis.py основної програми

```
import networkx as nx
import numpy as np
import json
import random
import matplotlib.pyplot as plt

def load_graph_from_file(filename="adjacency_matrix.json"):
    with open(filename, "r") as f:
        adjacency_matrix = json.load(f)
    G = nx.from_numpy_array(np.array(adjacency_matrix))
    return G

def random_node_failure(G, failure_rate=0.2):
    num_failures = int(len(G.nodes()) * failure_rate)
    failed_nodes = random.sample(list(G.nodes()), num_failures)
    G_removed = G.copy()
    G_removed.remove_nodes_from(failed_nodes)

    num_components = nx.number_connected_components(G_removed)
    largest_component = max(nx.connected_components(G_removed), key=len,
default=set())
    largest_component_size = len(largest_component)

    print("\n--- АНАЛІЗ ВИПАДКОВИХ ВІДМОВ ВУЗЛІВ ---")
    print(f"Кількість видалених вузлів: {num_failures}")
    print(f"Кількість компонент після відмов: {num_components}")
    print(f"Розмір найбільшої компоненти після відмов:
{largest_component_size}")

    return G_removed, failed_nodes, num_components, largest_component_size

def targeted_node_attack(G, attack_fraction=0.2):
    centrality = nx.betweenness_centrality(G)
    sorted_nodes = sorted(centrality, key=centrality.get, reverse=True)
    num_attacks = int(len(G.nodes()) * attack_fraction)
    attacked_nodes = sorted_nodes[:num_attacks]
    G_removed = G.copy()
    G_removed.remove_nodes_from(attacked_nodes)

    num_components = nx.number_connected_components(G_removed)
    largest_component = max(nx.connected_components(G_removed), key=len,
default=set())
    largest_component_size = len(largest_component)

    print("\n--- АНАЛІЗ НАЦІЛЕНИХ АТАК НА КРИТИЧНІ ВУЗЛИ ---")
    print(f"Кількість атакованих вузлів: {num_attacks}")
    print(f"Кількість компонент після атак: {num_components}")
    print(f"Розмір найбільшої компоненти після атак: {largest_component_size}")

    return G_removed, attacked_nodes, num_components, largest_component_size

def visualize_network(G, removed_nodes, title):
    plt.figure(figsize=(8, 6))
    pos = nx.spring_layout(G)

    node_colors = ["red" if node in removed_nodes else "lightblue" for node in
G.nodes()]

    nx.draw(G, pos, with_labels=True, node_size=500, node_color=node_colors,
edge_color='gray')
    plt.title(title)
    plt.show()

if __name__ == "__main__":
    G = load_graph_from_file()
```

```
    print("Виберіть тип відмовостійкості для аналізу:")
    print("1: Випадкові відмови вузлів")
    print("2: Націлені атаки на важливі вузли")

    choice = input("Введіть номер (1-2): ")

    if choice == "1":
        failure_rate = float(input("Введіть частку вузлів для відмови
(наприклад, 0.2): "))
        G_failed, failed_nodes, num_components, largest_component_size =
random_node_failure(G, failure_rate)
        visualize_network(G_failed, failed_nodes, "Мережа після випадкових
відмов вузлів")
    elif choice == "2":
        attack_fraction = float(input("Введіть частку вузлів для атаки
(наприклад, 0.2): "))
        G_attacked, attacked_nodes, num_components, largest_component_size =
targeted_node_attack(G, attack_fraction)
        visualize_network(G_attacked, attacked_nodes, "Мережа після націлених
атак на критичні вузли")
    else:
        print("Некоректний вибір, завершення програми.")
```

Файл network_advanced_analysis.py основної програми

```
import networkx as nx
import numpy as np
import json
import matplotlib.pyplot as plt

def load_graph_from_file(filename="adjacency_matrix.json"):
    with open(filename, "r") as f:
        adjacency_matrix = json.load(f)
    G = nx.from_numpy_array(np.array(adjacency_matrix))
    return G

def analyze_mst(G):
    mst = nx.minimum_spanning_tree(G)
    print("\n--- АНАЛІЗ МІНІМАЛЬНОГО КІСТЯКОВОГО ДЕРЕВА (MST) ---")
    print(f"MST містить {len(mst.edges())} ребер із {len(G.edges())} у вихідному
графі.")

    return mst

def analyze_network_center(G):
    center_nodes = nx.center(G)
    print("\n--- АНАЛІЗ ЦЕНТРУ МЕРЕЖІ ---")
    print(f"Оптимальні центральні вузли: {center_nodes}")

    return center_nodes

def analyze_diameter(G):
    try:
        diameter = nx.diameter(G)
        print("\n--- АНАЛІЗ ДІАМЕТРА ГРАФА ---")
        print(f"Діаметр мережі: {diameter}")
        return diameter
    except nx.NetworkXError:
        print("Діаметр не визначений (граф не є зв'язним).")
        return None

def analyze_load_balancing(G):
    betweenness = nx.betweenness_centrality(G)
    max_node = max(betweenness, key=betweenness.get)
    print("\n--- АНАЛІЗ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ ---")
    print(f"Вузол {max_node} обробляє найбільше маршрутів (найвища betweenness
centrality: {betweenness[max_node]:.4f})")

    return max_node

def analyze_critical_edges(G):
    critical_edges = list(nx.bridges(G))
    print("\n--- АНАЛІЗ КРИТИЧНИХ РЕБЕР ---")
    print(f"Ключові з'єднання (мости) у мережі: {critical_edges}")

    return critical_edges

def generate_recommendations(G, diameter, center_nodes, mst, max_load_node,
critical_edges):
    print("\n--- РЕКОМЕНДАЦІЇ ---")
    if diameter and diameter > len(G.nodes()) / 2:
        print("1. Високий діаметр мережі вказує на довгі маршрути.")
        print("   -> Рекомендація: Додати більше прямих з'єднань між вузлами.")
    if max_load_node:
        print(f"2. Вузол {max_load_node} сильно завантажений.")
        print("   -> Рекомендація: Додати альтернативні маршрути або розподілити
трафік.")
    if critical_edges:
        print("3. Деякі ребра є критичними для зв’язності.")
        print(f"   -> Рекомендація: Забезпечити резервні зв’язки для
{critical_edges}.")
```

```
    if mst and len(mst.edges()) < len(G.edges()):
        print("4. Мінімальне кістякове дерево має менше з'єднань, ніж поточна
мережа.")
        print("   -> Рекомендація: Оптимізувати структуру для мінімізації
витрат.")
    if center_nodes:
        print(f"5. Оптимальні центральні вузли: {center_nodes}.")
        print("   -> Рекомендація: Використовувати ці вузли як основні точки
маршрутизації.")

def visualize_graph(G, highlight_nodes=None, highlight_edges=None,
title="Мережевий граф"):
    plt.figure(figsize=(8, 6))
    pos = nx.spring_layout(G)

    node_colors = ["red" if node in highlight_nodes else "lightblue" for node in
G.nodes()] if highlight_nodes else "lightblue"
    edge_colors = ["red" if edge in highlight_edges else "gray" for edge in
G.edges()] if highlight_edges else "gray"

    nx.draw(G, pos, with_labels=True, node_size=500, node_color=node_colors,
edge_color=edge_colors)
    plt.title(title)
    plt.show()

if __name__ == "__main__":
    G = load_graph_from_file()

    # Виконання аналізу
    mst = analyze_mst(G)
    center_nodes = analyze_network_center(G)
    diameter = analyze_diameter(G)
    max_load_node = analyze_load_balancing(G)
    critical_edges = analyze_critical_edges(G)

    # Генерація рекомендацій
    generate_recommendations(G, diameter, center_nodes, mst, max_load_node,
critical_edges)

    # Візуалізація мережі з критичними вузлами та ребрами
    visualize_graph(G, highlight_nodes=center_nodes,
highlight_edges=critical_edges, title="Критичні елементи мережі")
```

Файл virus_spread_simulation.py основної програми

```
import networkx as nx
import numpy as np
import json
import matplotlib.pyplot as plt
import random

def load_graph_from_file(filename="adjacency_matrix.json"):
    with open(filename, "r") as f:
        adjacency_matrix = json.load(f)
    G = nx.from_numpy_array(np.array(adjacency_matrix))
    return G

def visualize_network(G, infected_nodes=set(), removed_nodes=set(), title="Стан
мережі"):
    plt.figure(figsize=(8, 6))
    pos = nx.spring_layout(G)

    node_colors = []
    for node in G.nodes():
        if node in infected_nodes:
            node_colors.append("red")  # Інфіковані вузли
        elif node in removed_nodes:
            node_colors.append("gray")  # Видалені вузли
        else:
            node_colors.append("lightblue")  # Здорові вузли

    nx.draw(G, pos, with_labels=True, node_size=500, node_color=node_colors,
edge_color='gray')
    plt.title(title)
    plt.show()

def simulate_virus_spread(G, num_iterations=5, infection_rate=0.3,
recovery_rate=0.2):
    nodes = list(G.nodes())
    infected_nodes = set(random.sample(nodes, 1))  # Початкове зараження
    removed_nodes = set()

    print("\n--- ПОЧАТОК СИМУЛЯЦІЇ ---")
    for i in range(1, num_iterations + 1):
        new_infected = set()
        new_removed = set()

        for node in infected_nodes:
            if random.random() < recovery_rate:
                new_removed.add(node)  # Видалений з інфекції
            else:
                for neighbor in G.neighbors(node):
                    if neighbor not in infected_nodes and neighbor not in
removed_nodes and random.random() < infection_rate:
                        new_infected.add(neighbor)  # Нове зараження

        infected_nodes.update(new_infected)
        infected_nodes.difference_update(new_removed)
        removed_nodes.update(new_removed)

        print(f"Ітерація {i}: {len(infected_nodes)} заражених,
{len(removed_nodes)} видалених.")
        visualize_network(G, infected_nodes, removed_nodes, title=f"Ітерація
{i}: Поширення вірусу")

    print("\n--- СИМУЛЯЦІЯ ЗАВЕРШЕНА ---")
    print(f"Фінальний стан: {len(infected_nodes)} заражених,
{len(removed_nodes)} видалених.")
    visualize_network(G, infected_nodes, removed_nodes, title="Фінальний стан
мережі")
```

```
if __name__ == "__main__":
    G = load_graph_from_file()
    print("Мережу успішно завантажено з файлу adjacency_matrix.json.")
    visualize_network(G, title="Початковий стан мережі")

    num_iterations = int(input("Введіть кількість ітерацій моделі: "))
    simulate_virus_spread(G, num_iterations)
```