МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Факультет: **ННІ Каразінський банківський інститут**
Кафедра: **Інформаційних технологій та математичного моделювання**
Спеціальність: **122 Комп'ютерні науки**
Освітня програма: **Комп'ютерні науки та інформаційні технології в бізнесі**

Група: АК-21М **денна форма навчання**

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«МАТЕМАТИЧНІ ТА КОМП'ЮТЕРНІ МОДЕЛІ СИМУЛЯЦІЇ ОНЛАЙН ГРОШОВИХ ПЕРЕКАЗІВ P2P-СИСТЕМ» ЗА НАКАЗОМ № 4601-5/3045 ВІД 25 ВЕРЕСНЯ 2024 РОКУ

здобувача вищої освіти **Балациря Олександра Михайловича**

Робота допущена до захисту в ЕК
протокол кафедри ІТММ №4 від 30.11.2024 р.

Завідувач кафедри
к. п. н., доцент
_____ Н. І. Стяглик

Науковий керівник
д.т.н., професор
_____ О.І.Морозова

м. Харків 2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В. Н. Каразіна

Факультет навчально-науковий інститут "Каразінський банківський інститут"
Кафедра інформаційних технологій та математичного моделювання
Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ **Н. І. Стяглик**
Підпис ініціали, прізвище

"25" вересня 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЄКТ)

Балациря Олександра Михайловича
(прізвище, ім'я, по батькові студента)

1. Тема роботи «МАТЕМАТИЧНІ ТА КОМП'ЮТЕРНІ МОДЕЛІ СИМУЛЯЦІЇ ОНЛАЙН ГРОШОВИХ ПЕРЕКАЗІВ P2P-СИСТЕМ»

керівник роботи д.т.н., професор Морозова О.І.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "25" вересня 2024 року № 4601-5/3045

2. Строк подання студентом роботи 20 листопада 2024 року

3. Перелік питань, які потрібно розробити:
У розділі 1: дослідити архітектуру та математичні моделі P2P-систем, визначити переваги та недоліки.
У розділі 2: дослідити комп'ютерне моделювання розподілених P2P-систем, визначити їх переваги та недоліки.
У розділі 3: виконати імітаційне моделювання середовищ, використовуючи P2P-симулятори.

4. План роботи

№ з/п	Назви етапів роботи
1	Вибір здобувачем теми кваліфікаційної магістерської роботи
2	Затвердження плану і завдання кваліфікаційної магістерської роботи
3	Здача кваліфікаційної магістерської роботи керівнику
4	Підпис кваліфікаційної магістерської роботи керівника
5	Підпис кваліфікаційної магістерської роботи у нормоконтролера
6	Допуск завідувачем кафедри до захисту кваліфікаційної магістерської роботи
7	Захист кваліфікаційної магістерської роботи

5. Дата видачі завдання 25 вересня 2024 року

Студент _____ О.М.Балацир
підпис ініціали, прізвище

Керівник роботи _____ О.І.Морозова
підпис ініціали, прізвище

РЕФЕРАТ
НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ
«МАТЕМАТИЧНІ ТА КОМП'ЮТЕРНІ МОДЕЛІ СИМУЛЯЦІЇ ОНЛАЙН ГРОШОВИХ ПЕРЕКАЗІВ P2P-СИСТЕМ»
Балациря Олександра Михайловича

Кваліфікаційна магістерська робота містить: 63 сторінки, 6 таблиць, 26 рисунків, 10 формул, список літератури з 53 найменувань.

Об'єкт дослідження: математичні та комп'ютерні моделі симуляції P2P-систем.

Предмет дослідження: методи та алгоритми математичних та комп'ютерних симуляцій P2P-систем.

Мета кваліфікаційної магістерської роботи: аналіз та практичне застосування методів математичної та комп'ютерної симуляції онлайн грошових переказів P2P-систем.

Завдання кваліфікаційної магістерської роботи:

- у першому розділі дослідити архітектуру та математичні моделі P2P-систем, визначити переваги та недоліки;

- у другому розділі дослідити комп'ютерне моделювання розподілених P2P-систем, визначити їх переваги та недоліки;

- у третьому розділі виконати імітаційне моделювання середовищ, використовуючи P2P-симулятори.

Актуальність дослідження полягає для покращення роботи P2P-систем в умовах нестабільних або динамічних мережевих середовищ, що дозволяє спрогнозувати поведінку мережі, визначити можливі вразливості та оптимізувати навантаження, що покращить якість обслуговування користувачів.

За результатами дослідження сформульовані імітаційні моделі середовищ Chord та Gnutella на симуляторах PeerSim та NeuroGrid відповідно.

Практична новизна: полягає в дослідженні математичних та комп'ютерних моделей P2P-систем, результати яких дозволять оптимізувати роботу P2P-систем.

Одержані результати можуть бути використані в тестуванні та оптимізації протоколів розподілених P2P-систем.

КЛЮЧОВІ СЛОВА: CHORD, GNUTELLA, NEUROGRID, P2P-СИСТЕМИ, PEERSIM, ІМІТАЦІЙНЕ МОДЕЛЮВАННЯ, КОМП'ЮТЕРНА МОДЕЛЬ, МАТЕМАТИЧНА МОДЕЛЬ.

ABSTRACT
AT QUALIFICATION MAGISTER WORK

«MATHEMATICAL AND COMPUTER MODELS OF THE SIMULATION OF ONLINE MONEY TRANSFERS OF P2P SYSTEMS»

Oleksandr Balatsyr

The master's thesis contains 63 pages, 6 tables, 26 figures, 10 formulas, a list of literature from 53 titles..

The object of the study is mathematical and computer simulation models of P2P systems.

The subject of the study is methods and algorithms of mathematical and computer simulations of P2P systems.

The purpose of the qualifying master's thesis: analysis and practical application of methods of mathematical and computer simulation of online money transfers of P2P systems.

The tasks of the master's qualification work are:

- in the first section to explore the architecture and mathematical models of P2P systems, determine advantages and disadvantages;
- in the second section to investigate computer modeling of distributed P2P systems, determine their advantages and disadvantages;
- in the third section to perform simulation modeling of environments using P2P simulators.

The relevance of the study to improve the operation of P2P systems in unstable or dynamic network environments, which allows predicting the behavior of the network, identifying possible vulnerabilities and optimizing the load, which will improve the quality of user service.

According to the results of the research, simulation models of the Chord and Gnutella environments were formulated using the PeerSim and NeuroGrid simulators, respectively.

Main theoretical provisions on the topic of the practical relevance of the consists in the research of mathematical and computer models of P2P systems, the results of which will allow optimizing the operation of P2P systems.

The results obtained can be used in testing and optimization of protocols of distributed P2P systems.

KEYWORDS: CHORD, GNUTELLA, NEUROGRID, P2P SYSTEMS, PEERSIM, SIMULATION, COMPUTER MODEL, MATHEMATICAL MODEL.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧОК, СИМВОЛІВ І ТЕРМІНІВ

BitTorrent – бітовий потік

DAG – Directed Acyclic Graph, орієнтовний ациклічний граф

DeFi – Decentralized Finance, децентралізовані фінансові системи

DHT – Distributed Hash Table, хеш-таблиця

IPTV – Internet Protocol Television

QoS – Quality of Service

SHA-1 – Secure Hash Algorithm 1

TCP/IP – Transmission Control Protocol/Internet Protocol

TTL – Time-to-Live

P2P-системи – системи Peer-to-Peer

ВСТУП

Із розвитком сучасних технологій та зростанням об’ємів даних виникла необхідність створення мережевих систем, здатних ефективно розподіляти ресурси та оброблювати інформацію без єдиної точки відмови. Однієї з таких архітектур є розподілені системи peer-to-peer (P2P-системи). Вони забезпечують децентралізовану взаємодію між учасниками мережі, оминаючи централізовані сервери. У таких системах кожен користувач виконує роль як клієнта, так і серверу. Це робить P2P-мережі більш гнучкими та масштабованими у порівнянні з традиційними клієнт-серверними архітектурами [15, 38, 42, 52].

Сьогодні розподілені P2P-системи знайшли широке застосування в різних областях. Вони використовуються від файлового обміну до децентралізованих фінансових систем (DeFi). Популярність цих систем пояснюється їх високою стійкістю до відмовлень, можливістю функціонувати в умовах обмеженої пропускної здатності та можливістю до адаптації та саморегуляції пари зміненні числа клієнтів. Тим не менш, P2P-мережі стикаються з певними проблемами, такими як забезпечення безпеки, балансування навантаження, оптимізація маршрутизації та захист від атак злочинців. Одним із ефективних способів вирішення цих проблем є використання математичних моделей і комп’ютерних симуляцій. Такі підходи дозволяють досліджувати поведінку P2P-систем у різних сценаріях експлуатації, аналізувати вплив різних факторів на їх продуктивність і розроблювати методи оптимізації роботи мережі. Математичне моделювання дозволяє не тільки зрозуміти принципи роботи розподілених систем, а й спрогнозувати їх поведінку в умовах реальних мережевих навантажень, тим самим спрощуючи розробку більш ефективних протоколів [10, 14, 48, 51].

Актуальність дослідження полягає в покращенні роботи P2P-систем в умовах нестабільних або динамічних мережевих середовищ.

Мета дослідження полягає в аналізі та наведенні практичного

застосування методів математичної та комп’ютерної симуляції онлайн грошових переказів P2P-систем.

Завданнями дослідження є наступні:

- дослідити архітектуру P2P-систем;
- дослідити математичні моделі P2P-систем (розподілення файлу в пірінговий файл обмінній мережі, Gnutella та система кабельного мовлення);
- дослідити комп’ютерне моделювання розподілених P2P-систем (універсальне середовище SimGrid, симулятори PeerSim і NeuroGrid);
- виконати імітаційні моделювання середовищ Chord та Gnutella на симуляторах PeerSim і NeuroGrid відповідно;
- виконати порівняльний аналіз імітацій та оцінити їх ефективність.

Об’єктом дослідження є математичні та комп’ютерні моделі симуляції P2P-систем.

Предметом дослідження є методи та алгоритми математичних та комп’ютерних симуляцій P2P-систем.

Робота складається із вступу, трьох розділів і висновків.

У першому розділі проводиться дослідження архітектури та математичної моделі P2P-систем, визначаються переваги та недоліки.

У другому розділі проводиться дослідження комп’ютерного моделювання розподілених P2P-систем, визначаються їх переваги та недоліки.

У третьому розділі виконується опис імітаційного моделювання середовищ, використовуючи P2P-симулятори.

Висновки надають результати проведеної роботи, оцінюють ефективність виконаної імітації, підкреслюють найкращу з них і надають шлях до розвитку виконаного дослідження в подальшому.

РОЗДІЛ 1.
ДОСЛІДЖЕННЯ МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ РОЗПОДІЛЕНИХ P2P-СИСТЕМ

1.1. Дослідження архітектури P2P-систем

За багатьма джерелами інформації мережа P2P є розподілена та децентралізована мережева архітектура, в якій користувачі або піри напряму спілкуються один з одним і не потребують допомоги централізованого серверу або органу влади. Кожен пір в мережі типу «рівний-рівному» має однаковий статус та може сумісно використовувати та споживати ресурси [18, 19, 29].

На рисунку 1.1 наведена схема роботи архітектури P2P.

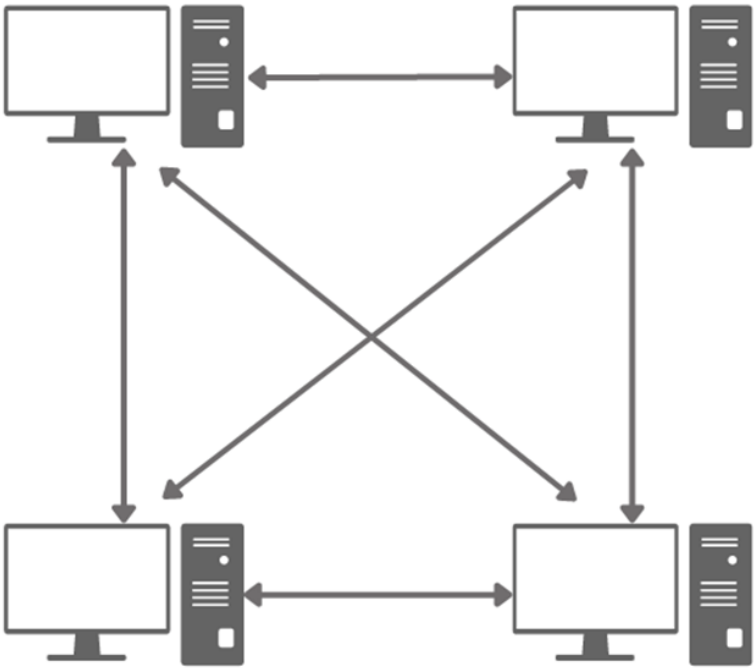

Рис. 1.1. Схема роботи архітектури P2P

Існує три види P2P-мереж [18, 19, 29]:

- неструктуровані;
- структуровані;

- гібридні.

У неструктурованих однорангових мережах вузли не розташовуються в певному порядку, тобто зв’язок між ними є випадковим. Такий тип мережі підходить для випадків з високою активністю, таких як соціальні мережі, де користувачі можуть регулярно приєднуватися до мережі або виходити з неї. Перевагами є такі фактори як відсутність центрального керування, вузли взаємодіють напряму, стійкість без єдиної точки відмови та одним з варіантів використанням є застосунки для обміну файлів. Однак, неструктуровані P2P-мережі мають свої недоліки. Для їх правильної роботи необхідний значний об’єм процесору та пам’яті. Апаратне забезпечення повинно підтримувати максимальну кількість мережевих транзакцій, щоб усі вузли могли взаємодіяти один з одним у будь-який час. Це може бути проблемою, особливо якщо мережа велика та має великий об’єм активності [4, 6].

Структуровані мережі протилежні до вже згаданих. Тобто, вузли мають здатність спосіб взаємодіяти один з одним більш організованим чином. Цього можна досягнути з використанням гарно організованої архітектури, яка дозволяє клієнтам знаходити та використовувати файли більш ефективно, не шукаючи їх випадковим чином. Перевагами є такі аспекти як керування центральним сервером зв’язками між вузлами, більший контроль та безпека, ефективне керування задачами та можливість використання в онлайн-іграх, чатах тощо. Хоч і структуровані мережі є більш ефективними, вони мають деяку централізацію через організованість. Це також означає, що вони можуть бути більш коштовними в обслуговуванні та налаштуванні, ніж неструктуровані. Однак, усе ж таки структуровані P2P-мережі більш стабільні, ніж неструктуровані [6, 37].

Гібридні P2P-мережі об’єднують попередні два види. Можуть мати підтримку серверної індексації, суперпіри або трекери. Такі компоненти можуть допомогти досягти біль високої продуктивності та надійності за рахунок оптимізації розподілення ресурсів, покращення продуктивності пошуку та координації комунікації. Це також може використовувати основний

принцип P2P-мереж шляхом децентралізації компонентів для певних функцій. Наприклад, зберігання та сумісне використанням даних. Гібридні мережі можуть бути біль масштабованими, ніж чисті системи. У залежності від конкретних вимог можна використовувати різні ступені централізації. Таким чином, забезпечується гнучкість за рахунок створення між перевагами централізованої та децентралізованої архітектури [46].

Отож, можна визначити характеристики архітектури P2P. Вони наведені в таблиці 1.1.

Таблиця 1.1

Характеристики P2P-архітектури

Характеристика	Опис
1	2
Децентралізація	Відсутність центрального серверу, що дозволяє кожному вузлу діяти як клієнт і сервер одночасно. Це збільшує відмовостійкість, так як мережа не залежить від однієї точки відмови
Масштабованість	По мірі додавання нових вузлів збільшуються доступні ресурси мережі, включаючи обчислювальну потужність та сховище даних
Розподілення ресурсів	Усі вузли діляться своїми ресурсами такими як файли, обчислювальні потужності або пропускна здібність мережі, що сприяє оптимальному використанню системі
Стійкість до відмов	Системи стійкі до відмов через продовжність функціонування мережі навіть при виході з ладу окремих вузлів
Відсутність єдиної точки контролю	Керування та контроль розподілені між вузлами, що ускладнює цензуру або втручання зі сторони третіх осіб, що робить мережу автономною

Продовження таблиці 1.1

1	2
Ефективне розподілення навантаження	При збільшенні числа учасників, робота в мережі розподіляється між вузлами, що дозволяє одночасно виконати множину операцій, таких як завантаження
Безпека	Кожен вузол відповідає за свою безпеку. Однак, децентралізація може ускладнити керування правами доступу та підвищити ризик уразливостей, пов’язаних з відсутністю центрального контролю

Джерело: розробка автора

Перераховані характеристики є популярними для децентралізованих застосунків, файлообміну, блокчейн технологій та інших областей, де необхідна незалежність від централізованих рішень.

За описаними тезами можна виділити наступні переваги P2P-архітектури [25, 35]:

- відсутність єдиної точки відмови через розподілення даних між вузлів, що підвищує стійкість до збоїв та атак;

- легкість додавання нових вузлів, що робить архітектуру масштабованою без значних витрат на інфраструктуру;

- передача даних напряму між користувачами, що забезпечує великий ступінь приватності та анонімності;

- зниження перевантаження серверів шляхом рівномірного розподілення передачі даних між всіма учасниками системи;

- збільшення загальної продуктивності мережі за рахунок можливості в користувачів ділитися своїми обчислювальними потужностями, пам’яттю та пропускною здатністю.

Але, як і в інших архітектур, P2P має свій перелік недоліків [25, 35]:

- відсутність центрального органу, який має відповідати за контроль і

захист;

- залежність продуктивності від швидкості та стабільності з’єднання окремих учасників;

- виникнення проблеми координації дій між користувачами та оновлення системи у випадку використання крупної мережі;

- можливість розповсюдження нелегального контенту, що може призвести до юридичних проблем для користувачів мережі;

- зниження ефективності системи через споживання більшої кількості ресурсів деякими вузлами;

- повільний час відгуку через децентралізований характер системи.

Таким чином, P2P-архітетура залишається актуальною завдяки децентралізації, масштабованості та безпеці, але все ж таки зберігає складність із керуванням, безпекою та ефективністю в мережі з нерівномірними ресурсами. Для вирішення наявних проблем можуть допомогти математичні та комп’ютерні моделі P2P-систем. Вони дозволяють глибше зрозуміти та покращити їх роботу. Ці моделі допомагають спрогнозувати поведінку мережі, аналізувати маршрутизацію, розподілення навантаження та стійкість до збоїв.

1.2. Математичні моделі P2P-систем

Математичні моделі P2P-систем призначені для опису та аналізу взаємодій вузлів, розподілення даних та забезпечення надійності мережі. Існує безліч видів таких моделей, але виділяють ключові наступні [11, 44]:

1) графові моделі, в яких вузли є вершинами, а канали зв’язку між ними – ребрами, що дозволяє моделювати топологію мережі та досліджувати зв’язність, стійкість до збоїв та розподілення навантаження;

2) стохастичні моделі описують невизначеності в роботі затримок при передачі даних та доступність вузлів;

3) епідемічні моделі описують процес розподілення даних у мережі за аналогією з розподіленням інфекції; тобто, вузол, що отримав файл, передає його іншим, що моделює швидкість і найдінність доставлення інформації;

4) моделі, засновані на диференціальних рівняннях використовуються для опису динаміки змін параметрів системи за часом, таких як завантаження мережі або розподілення навантаження;

5) ігрові моделі або теорія ігор використовуються для аналізу стратегічної поведінки вузлів;

6) моделі на основі економічних принципів аналізують вартість ресурсів та поведінку вузлів у контексті обміну ресурсами, що корисно для аналізу розподілення ресурсів та стимулювання вузлів для підтримки мережі.

Прикладами використання математичних моделей є такі [13, 41, 43]:

- модель розподілення файлу в файл обмінних системах можна описати через епідемічні або стохастичні моделі, важливими аспектами яких є швидкість розповсюдження файлу та рівномірність розподілення навантаження між вузлами;

- модель мережі Gnutella не має центрального серверу, а пошук у мережі здійснюється через затоплення, що моделюється як процес пошуку у випадковому графі;

- модель мережі Chord є розподіленою хеш-таблицею (DHT), де вузли організовані в кільцеву структуру, а маршрутизація запитів виконується через систему пальців, яка допомагає знаходити вузли з логарифмічною складністю.

Описані моделі допомагають аналізувати такі параметри мережі, як швидкість пошуку файлів, балансування навантаження, стійкість до збоїв і ефективне розподілення даних.

1.2.1. Математична модель розподілення файлу в пірінговій файл обмінній мережі

Системи P2P, такі як BitTorrent (бітовий потік), забезпечують ефективне розподілення великих файлів за рахунок паралельного завантаження різних частин файлу від різних вузлів. Файл розподіляється на N частин, де кожна може бути ідентифікована як $P_i, i = 1,2, \dots, N$. Вузли мережі представляють собою n учасників, кожен з яких може виконувати функції завантаження та роздачі. Кожен вузол U_j має свою швидкість завантаження $v_{download,j}$ та роздачі $v_{upload,j}$, де $j = 1,2, \dots, n$ [13, 41].

Першим кроком алгоритму математичної моделі є роздача частин файлу P_i вузлами іншим учасникам мережі. Далі на кожному кроці час передачі частин файлу визначається за такою формулою [13]:

$$t_{ij} = \frac{S_i}{(v_{upload,j}, v_{download,j})}, \tag{1.1}$$

де: t_{ij} – час передачі частини файлу P_i від вузла i до вузла j;

S_i – розмір частини файлу P_i.

Вірогідність отримання частини файлу P_i вузлом j на кроці t можна змоделювати з використанням марковських ланцюгів, де відстань s_i представляє чи присутня частина P_i у вузла j у момент часу t. Нехай $p_{ij}(t)$ – вірогідність передачі частини P_i від вузла i до вузла j, тоді [13]:

$$p_{ij}(t) = 1 - exp\ exp\left(-\lambda_{ij} \cdot t\right), \tag{1.2}$$

де: λ_{ij} – інтенсивність передачі між вузлами.

Схема роботи такої системи наведена на рисунку 1.2. За рисунком, вузли

мережі взаємодіють, обмінюючись частинами файлу, позначеними як, наприклад, $P_1, P_2, P_3, \dots, P_i$. Стрілки показують процес двостороннього завантаження та роздачі частин між різними вузлами. Система організована децентралізовано, без центрального серверу, де кожен вузол може одночасно завантажувати та роздавати частини файлу іншим учасникам.

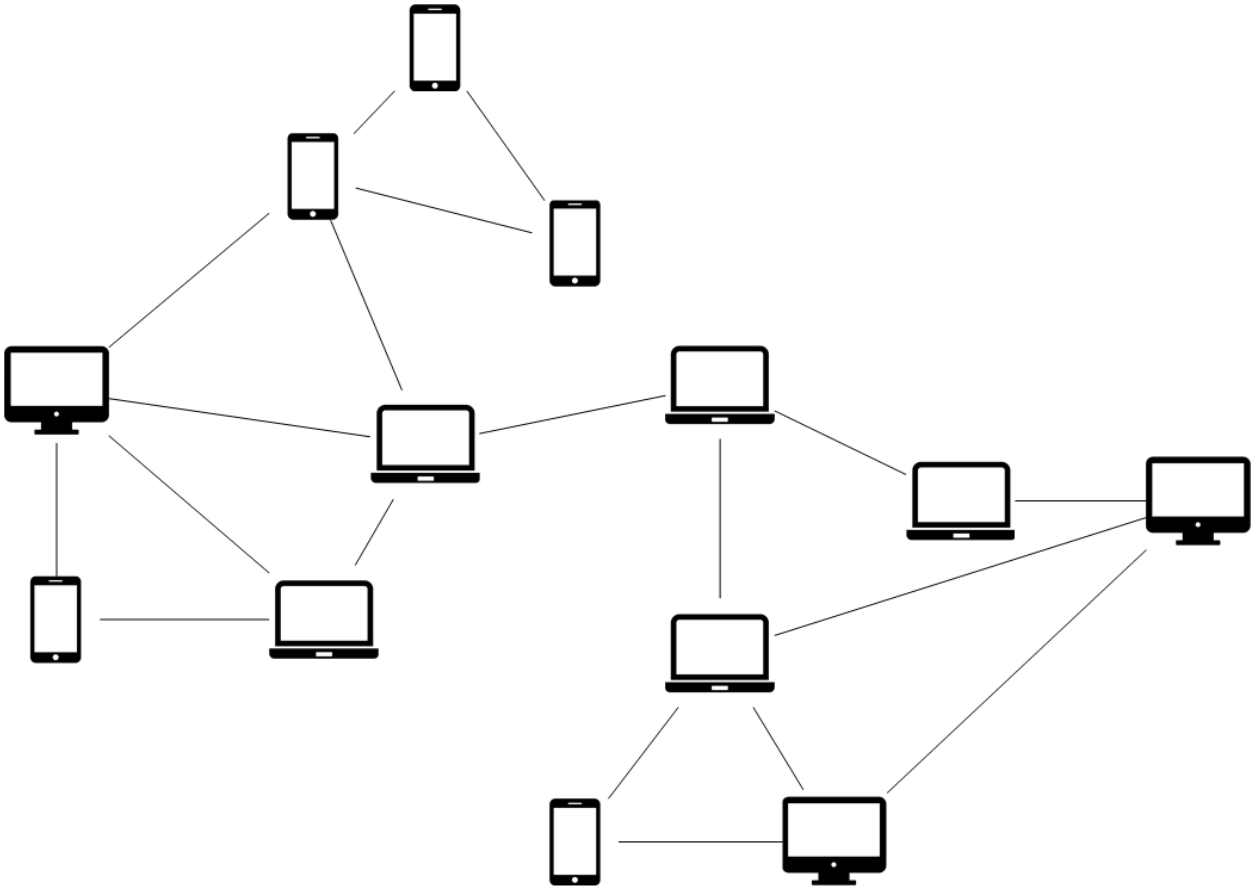

Рис. 1.2. Приклад роботи моделі розподілення файлу в пірінговій файл обмінній мережі [13]

Математичні моделі пірінгових файл обмінних мереж дозволяють аналізувати розподілення файлів, оптимізацію трафіку та стійкість до збоїв. Вони дають представлення про взаємодію вузлів, які обмінюються даними та допомагають оцінити ефективність передачі інформації в мережі.

1.2.2. Модель мережі Gnutella

Мережа Gnutella є децентралізованою P2P-мережею, що була розроблена для обміну файлами та вона виділяється інших мереж відсутністю єдиної центральної точки відмови [32].

Основними характеристика мережі Gnutella є наступні аспекти [7, 32]:

- не використання центральних серверів для координації обміну файлів; усі вузли в мережі рівні та взаємодіють один з одним напряму;

- запити на пошук файлу розсилаються широкомовно, що призводить до затопленню мережі;

- запити надсилаються всім сусіднім вузлам, а ті передають їх далі поки запит не досягне потрібного вузла;

- для запобігання перевантаження мережі використовуються time-to-live (TTL) – максимальна кількість вузлів, через які може пройти запит, а після досягнення межі TTL запит вмирає;

- вузли можуть кешувати популярні файли, що покращення швидкість та знижує навантаження на мережу та активність вузлів;

- вузли в мережі використовують спеціальний протокол для передачі повідомлень, типи яких наведені в таблиці 1.2.

За таблицею 1.2 можна скласти наступний алгоритм роботи мережі Gnutella [31]:

1) при підключенні новий вузол надсилає повідомлення Ping для пошуку інших вузлів у мережі;

2) вузли, що отримують Ping, відповідають повідомленням Pong з інформацією про себе та файлах, які вони зберігають;

3) вузол надсилає запит Query, який передається через сусідні вузли;

4) кожен вузол перевіряє свої файли на відповідність запиту та, у разі не знаходження відповідних, надсилає зворотну

відповідь QueryHit;

5) знаходження вузлом необхідного файлу та встановлює пряме з’єднання з вузлом, що зберігає файл;

6) починається завантаження через Transmission Control Protocol/Internet Protocol (TCP/IP) з’єднання.

Таблиця 1.2

Типи повідомлень протоколу зв’язку мережі Gnutella

Повідомлення	Опис
Широкомовні повідомлення	
Ping	Широкомовний запит для пошуку інших вузлів
Query	Запит пошуку конкретних файлів у мережі
Повідомлення відповідь	
Pong	Відповідь на запит Ping, що містить інформацію про наявність файлів та про сам вузол
Query Hit	Відповідь на запит Query, вказуючий про не існування файлу
Повідомлення між вузлами	
Push	Повідомлення, що ініціює передачу файлу від одного вузла до іншого

Джерело: [1, 7, 31]

Для наочності було створено алгоритм у вигляді графів. Було взято для прикладу сім вузлів та виконаний вище описаний алгоритм дій. Він наведений на рисунку 1.3.

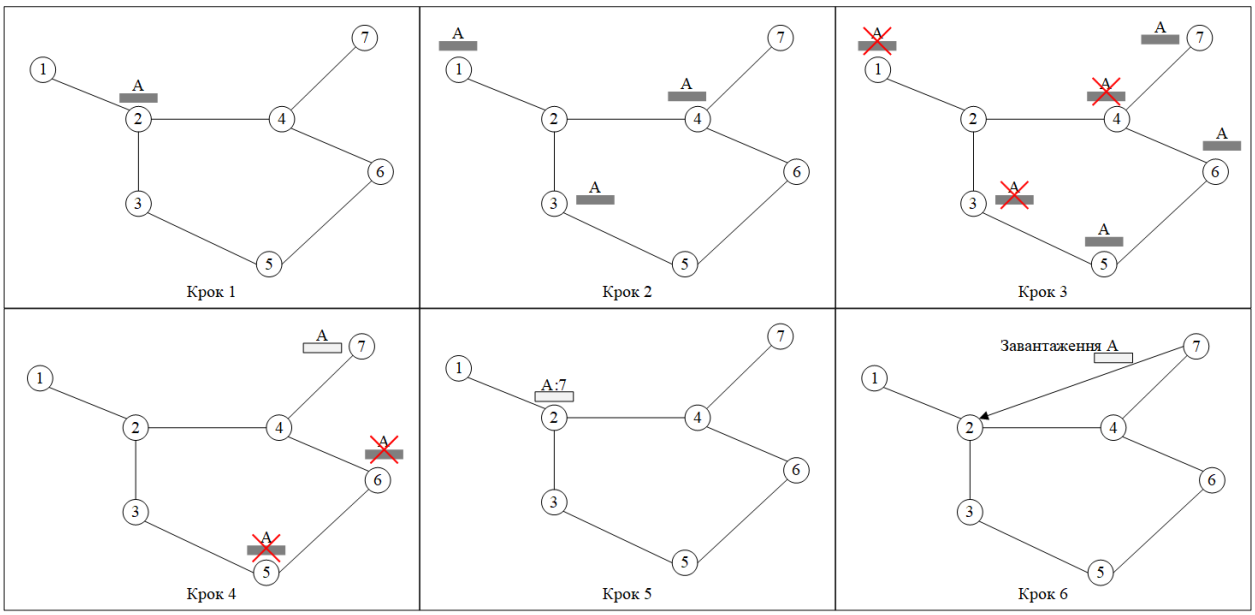

Рис. 1.3. Алгоритм роботи математичної моделі мережі Gnutella

Деякі кроки алгоритму можуть повторюватися, як наприклад другий, третій та четверті кроки, адже вузлів може бути набагато більше, аніж у зазначеному прикладі.

Модель мережі Gnutella є прикладом децентралізованої P2P-мережі, де кожен вузол працює автономно, а дані передаються напряму між користувачами. Однак, архітектура такої мережі стикається з проблемами масштабованості та перевантажень, що призводить до розробки кращих P2P-мереж.

1.2.3. Модель мережі Chord

Мережа Chord є розподіленим протоколом та алгоритмом пошуку даних у децентралізованій P2P-мережі. На відміну від Gnutella, в якій пошук проводиться через широкомовні запити, Chord використовує метод хешування для ефективного розподілення та пошуку даних за вузлами мережі. Така методика робить його більш масштабованим та ефективним [12, 47].

Основними характеристиками мережі Chord є такі як кільцева топологія, хешування та логарифмічний час пошуку. Вони наведені в таблиці 1.3.

Таблиця 1.3

Основні характеристики мережі Chord

Характеристика	Опис
1	2
Кільцева топологія	Усі вузли в мережі Chord організовані в кільце, де кожний вузол має свій унікальний ідентифікатор, отриманий через хешування. Використання хеш-функції, такої як Secure Hash Algorithm 1 (SHA-1), щоб рівномірно розподілити ідентифікатори за кільцем
Консистентне хешування	Розподіляє ключі за вузлами таким чином, щоб мінімізувати зміни в мережі при додаванні або видаленні вузлів. Кожен вузол у мережі відповідає за визначений діапазон ключів
Логарифмічний час пошуку	Забезпечення пошуку ключів за логарифмічним часом за кількістю вузлів у мережі. Тобто пошук потребує $O(log\ log\ N)$ кроків, де N – кількість вузлів

Джерело: [12, 40, 47]

Кільцева топологія наведена рисунку 1.4.

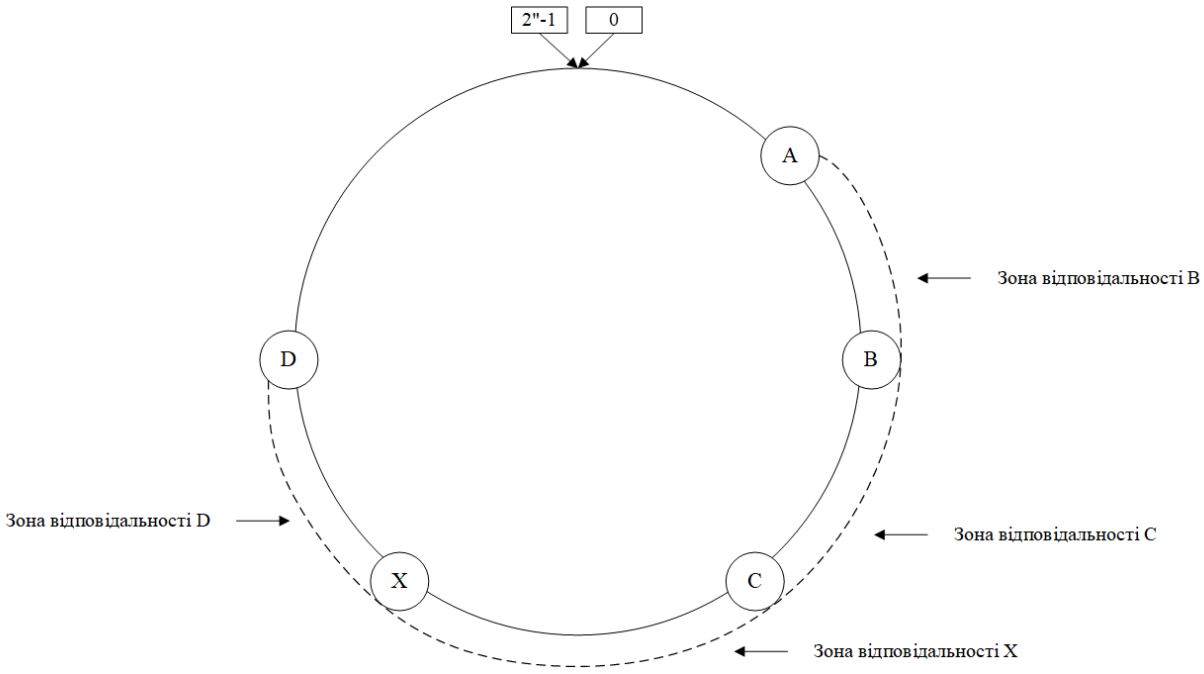

Рис. 1.4. Кільцева топологія моделі мережі Chord

За таблицею 1.3 та рисунком 1.4 можна сформувати принцип роботи мережі Chord.

Кожен вузол та кожен ресурс (наприклад, файл) у мережі має унікальний ідентифікатор, який обчислюється хеш-функцією. Наприклад, якщо використовується SHA-1, ідентифікатор буде 160-бітним числом. Отож, нехай ідентифікатор вузла ID_{node} знаходиться в діапазоні від 0 до $2^m - 1$, де m – кількість бітів у хеш-функції. Кожен ресурс також має свій унікальний ключ ID_{key}, який також підпадає в той же діапазон. Кожен вузол відповідає за ресурси, ідентифікатори яких менше його власного ідентифікатора, але більше ідентифікатора попередника в кільці. Тобто, якщо ключ знаходиться в інтервалі $(ID_{popered}, ID_{node}]$, вузол буде відповідати за цей ключ [40, 53].

Коли вузол хоче знайти ресурс с певним ключем ID_{key}, він використовує наступний алгоритм пошуку [40, 53]:

1) якщо ID_{key} знаходиться в діапазоні відповідності поточного вузла, то ресурс знайдений;

2) якщо ні, вузол перенаправляє запит наступному вузлу в кільці або використовує спеціальну таблицю, в якій зберігається інформація про

розташування вузлів у мережі (finger table).

Кожен вузол зберігає інформацію тільки про деякі вузли в мережі. Ця інформація про приближені вузли, які знаходяться на відстанях, кратних ступеням двійки від поточного вузла. Це й є finger table. Вона дозволяє вузлу не пересилати запити через всі вузли підряд, а одразу перескочити через декілька, що прискорює пошук. Розмір таблиці дорівнює кількості бітів в ідентифікаторі вузла (m). Кожен вузол зберігає до m записів, кожна з них відповідає за вузол на відстані 2^i, де i – номер запису [40, 53].

Отож, наприклад, вузол із ідентифікатором 10 хоче знайти ключ 42. Всього є 8 вузлів, ідентифікатори яких розподілені від нуля до 15. Кожен вузол отримує свій унікальний ідентифікатор через хешування. Наприклад, нехай ключі ресурсів також хешуються та мають свої ідентифікатори. Кожен вузол відповідає за ключі, ідентифікатори яких менше його власного, але більше ідентифікатора за його попередника, як було сказано вище. Алгоритм дій:

- вузол починає порівнювати його з діапазоном відповідному власному;
- якщо ключ знаходиться за діапазоном, він надсилає запит ближньому вузлу, що знаходиться на шляху до ключу 42 (за finger table);
- цей процес повторюється доки ключ не буде знайденим.

Наочно робота алгоритму наведена на рисунку 1.5. Схема є розгорнутим колом, для кращого сприяння.

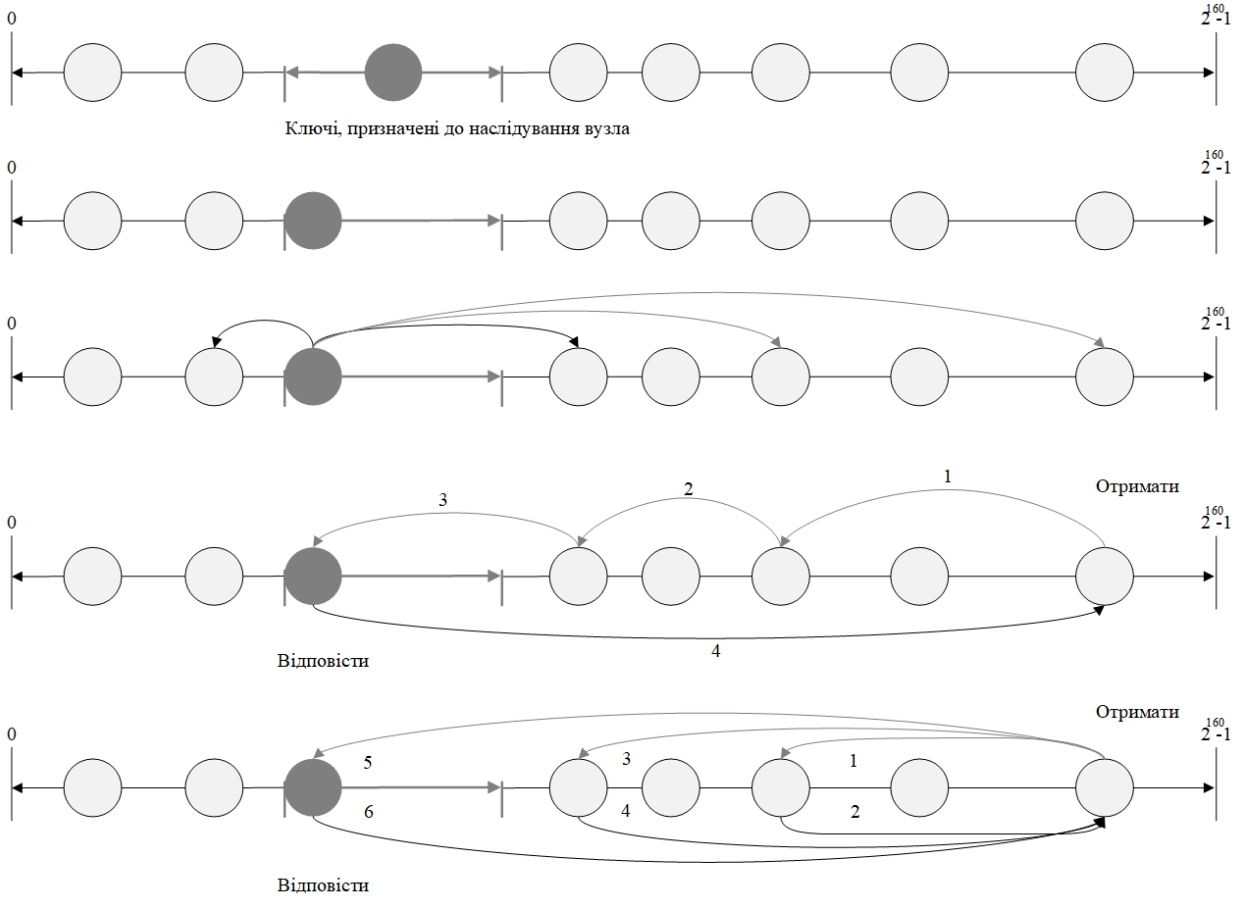

Рис. 1.5. Алгоритм роботи математичної моделі Chord (розгорнуті кільця)

Також, додатковою функцією мережі Chord є додавання та видалення вузлів. Коли новий вузол додається, він займає своє місце в кільці та переймає частину відповідальності за ключі від ближнього сусіднього вузла. Коли вузол видаляється, його вузли передаються сусідам, що мінімізує вплив на мережу [53].

Таким чином, мережа Chord забезпечує ефективне розподілення даних, швидке знаходження ресурсів та простоту керування вузлами, що робить її ідеальною для використання в децентралізованих застосунках. Процес додавання та пошуку ресурсів оптимізований завдяки концепціям консистентного хешування та finger table. Це забезпечує масштабованість та відмово стійкість.

1.2.4. Математична модель системи кабельного мовлення IPTV

Математична модель системи кабельного мовлення Internet Protocol Television (IPTV) у контексті P2P-мережі будується на сумісності потокової передачі мультимедіа даних через традиційні канали та взаємодії між рівноправними вузлами мережі. У такої моделі акцент робиться на розподіленні трафіку між користувачами, ефективною передачею відео контенту, а також на надійності та масштабованості системи [27, 31].

Схема роботи IPTV наведена на рисунку 1.6.

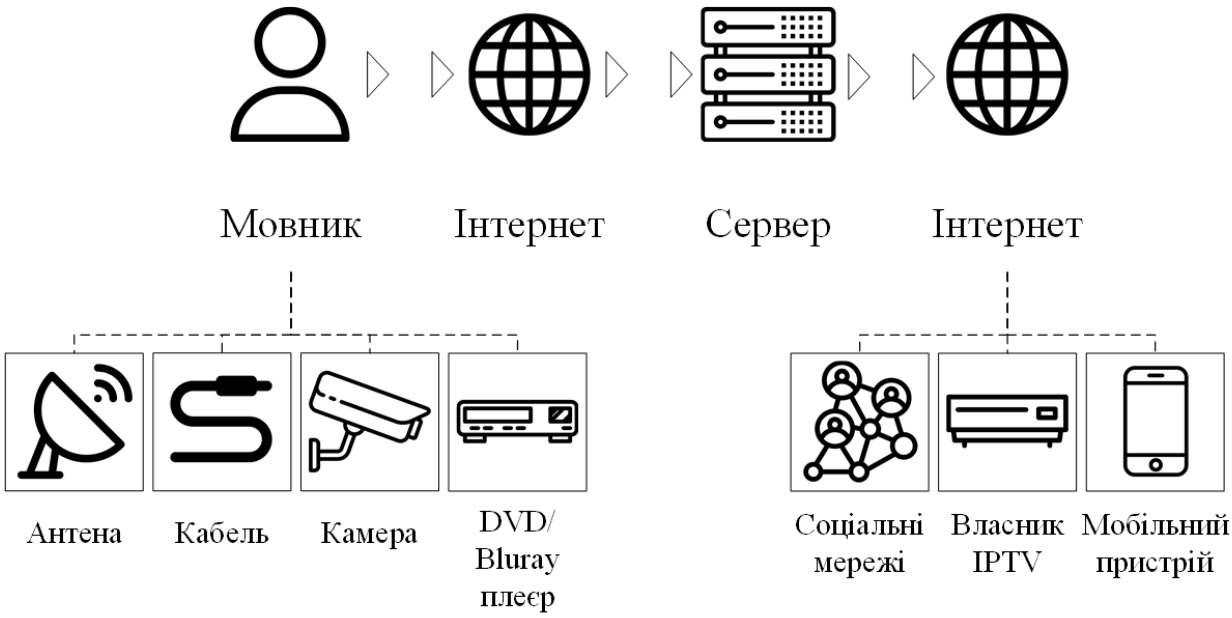

Рис. 1.6. Схема роботи моделі системи кабельного мовлення IPTV [31]

P2P-мережу можна розглядати як децентралізовану мережу, де всі вузли рівноправні та можуть як отримувати, так і передавати дані. У цій системі центральний сервер не є єдиним джерелом контенту, що суттєво знижує навантаження на інфраструктуру. Граф $G(V, E)$ описує структуру мережі, де кожний вузол може з'єднуватися з одним або декількома іншими вузлами для передачі даних. Це дає гнучкість у маршрутизації трафіку та робить систему масштабованою. Для IPTV-трафіка, який потребує високої пропускної здатності, топологія мережі повинна забезпечувати мінімальні затримки та високу пропускну здатність. Це потребує аналізу щільності зв'язку та стійкості мережі до змін. У реальних P2P-мережах використовується гібридна

архітектура, де частина контенту передається через центральні сервери, а мультимедійні дані розповсюджуються через піри [27, 31].

Кожен вузол у мережі отримує не тільки дані, але й може передавати їх іншим вузлам. Контент розбивається на невеликі блоки даних $C_1, C_2, \dots, C_n$, які передаються між вузлами. Цей процес називається chunk-based передачею даних, де вузли завантажують та одночасно надсилають іншим вузлам різні частини відеофайлу. Кожен вузол може передавати певну кількість даних за одиницю часу. Це обмежує його пропускну. Здатність. Об'єм даних, який вузол передає іншим вузлам у момент часу t, можна описати наступною формулою [31]:

$$B(t) = \sum_{i=1}^{n} \quad S(i,t), \tag{1.3}$$

де: $S(i,t)$ – об'єм даних, що передається вузлом i у момент часу t.

Величина $S(i,j,t)$ може змінюватися в залежності від умов мережі, таких як завантаженість каналів, що потребує динамічного контролю та розподілення ресурсів для мінімізації затримок. Розподілення навантаження P2P-мережі важливе через різну пропускну здатність вузлів. Наприклад, вузли з більш високою пропускною здатністю можуть передавати більший об'єм даних, аніж вузли з обмеженими ресурсами. Балансування навантаження полягає в тому, щоб рівномірно розподілити навантаження серед усіх вузлів. Таким чином можна уникнути ситуації перевантаження одних вузлів і бездіяльність інших. Функція навантаження $L(i,t)$ може включати в себе параметри кількості активних з'єднань, об'єм переданих даних та час відгуку вузла, як наведено в формулі нижче [31, 36]:

$$L(i,t) = \alpha \cdot connections(i,t) + \beta \cdot B(i,t) + \gamma \cdot delay(i,t), \tag{1.4}$$

де: α, β, γ – коефіцієнти, що відображують важливість кожного з параметрів.

В ідеалі балансування навантаження дозволяє системі залишатися стійкою навіть при значному збільшенні числа користувачів. Це також знижує вірогідність затримок та втрат даних, оскільки трафік розподіляється мережею рівномірно. Затримки визначають якість обслуговування (quality of service, QoS). Високі затримки призводять до погіршення якості відео, що неприйнятно для систем потокової передачі. Затримка $D(i,j)$ може виникати через декілька факторів [31, 36]:

- перевантаження мережі;
- відстань між вузлами;
- обмежена пропускна здатність каналів.

Затримку можна змоделювати наступним чином:

$$D(i,j) = d_{static}(i,j) + d_{dynamic}(i,j), \tag{1.5}$$

де: $d_{static}(i,j)$ – статична затримка, пов'язана з фізичною відстанню між вузлами;

$d_{dynamic}(i,j)$ – динамічна затримка, викликана поточним станом мережі.

Для мінімізації затримки в P2P-мережах можуть застосовуватися алгоритми вибору найменшого маршруту, такі алгоритми як Дейкстри або Беллмана-Форду.

У будь-якій реальній мережі існує вірогідність втрати пакетів, особливо в умовах нестальної мережі, де вузли можуть підключатися та відключатися від мережі. Втрата пакетів може виникнути через перезавантаження вузлів, помилок у каналі зв'язку, а також через затримок в маршрутизації. Втрата пакетів $P_{loss}(i,j,t)$ може бути змодельована як випадкова величина, яка залежить від багатьох факторів [31, 36]:

$$P_{loss}(i,j,t) = f\big(load(i), errors(j), latency(i,j)\big), \tag{1.6}$$

де: $load(i)$ – навантаження вузла i;

$errors(j)$ – помилки вузла j;

$latency(i,j)$ – затримка між вузлами.

Втрата пакетів особливо критична для потокових відео, оскільки втрата навіть невеликої кількості даних може суттєво погіршити якість відео. Для боротьби з цією проблемою застосовують алгоритми відновлення пакетів та механізмів дублювання даних, які дозволяють компенсувати втрату інформації. При дублюванні кожен блок контенту C_i зберігається на декількох вузлах. Нехай $R(i)$ – кількість копій даних, переданих вузлом i, тоді система може залишатися працездатною навіть при виході з ладу частини вузлів. Кількість копій можна описати наступним чином [5, 36]:

$$R(i) = \sum_{j \in N(i)} \quad D(j), \tag{1.7}$$

де: $N(i)$ – множина вузлів, які зберігають дані, передані вузлом i.

Важливим аспектом є мінімізація надмірності дублювання задля не перевантаження мережі зайвими даними, забезпечуючи надійність.

Алгоритми маршрутизації грають важливу роль у передачі даних P2P-мережі. На відміну від централізованих систем, де існує фіксований маршрут для передачі даних, у P2P-мережах маршрути можуть динамічно змінюватися в залежності від стану мережі. Одним з найбільш розповсюджених підходів є маршрутизація з найменшою затримкою. Вибір такого маршруту для передачі даних, який мінімізує час затримки $R(i,j)$. Однак, в умовах перевантаженості мережі даних підхід може бути недостатньо ефективним. Тоді застосовується маршрутизація з мінімальною втратою пакетів, яка обирає маршрути, що мінімізують вірогідність втрати даних. Іншими варіантами є використання алгоритмів з урахуванням балансу навантаження, де маршрути обираються не

тільки на основі затримок або втрат, але з урахуванням поточної завантаженості вузлів [5].

Модель IPTV у контексті P2P-мереж дає детальне представлення про те, як функціонує система передачі мультимедійного контенту в децентралізованій мережі.

1.3. Висновки до розділу 1

За розділом були досліджені архітектура системи P2P, описані її основні характеристики. Були досліджені математичні моделі P2P-мережі. Описана математична модель файл обмінної мережі. Досліджені моделі мереж Gnutella та Chord. Наведені їх алгоритми роботи. Проаналізована робота IPTV, описані її проблеми та шляхи їх вирішення.

РОЗДІЛ 2.
ДОСЛІДЖЕННЯ КОМП'ЮТЕРНОГО МОДЕЛЮВАННЯ РОЗПОДІЛЕНИХ P2P-СИСТЕМ

2.1. Комп'ютерне моделювання P2P-систем

Комп'ютерне моделювання P2P-систем є дослідженням та аналізом децентралізованих мереж, де вузли або користувачі напряму обмінюються ресурсами та даними без центрального серверу. Такі системи широко використовуються для обміну файлами, стрімінгу, розподілених обчислень та криптовалютних систем [2, 17, 45].

Ключові аспекти характеристик комп'ютерного моделювання P2P-систем наведені в таблиці 2.1.

Таблиця 2.1

Характеристики комп'ютерного моделювання P2P-систем

Характеристика	Опис
Архітектура мережі	Неструктуровані мережі, вузли яких випадково підключаються один до одного та пошуку даних виконується через широкомовні запити. Структуровані мережі, вузли яких організовані за певними правилами, що дозволяє більш ефективний пошук даних
Алгоритми маршрутизації	Набори правил, які визначають, як дані передаються між вузлами. Моделювання алгоритмів пошуку, балансування навантаження та відмовостійкості задля розуміння взаємодії вузлів та обміну даними
Масштабованість	Здатність системи ефективно збільшувати кількість вузлів
Динаміка мережі	Висока динамічність через увімкнення вимкнення вузлів

Продовження таблиці 2.1

1	2
Надійність і стійкість до відмов	Оцінка стійкості мережі до виходу з ладу окремих вузлів та груп вузлів
Аналіз пропускної здатності та затримок	Моделювання затримок при передачі даних між вузлами та розподілення ресурсів

Джерело: [28]

Прикладами використання комп'ютерного моделювання P2P-систем є наступні [3, 16, 49]:

- PeerSim використовується для моделювання великих P2P-мереж і фокусується на масштабованості та моделюванні з динамічними змінами;

- NS-3 використовується для симуляцій мереж, підтримуж роботу з різними мережевими протоколами та топологіями;

- OMNeT++ моделює мережі з різними протоколами, підходить для аналізу та симуляції як традиційних мереж, так і P2P-систем;

- SimGrid призначений для моделювання розподілених обчислювальних систем, таких як кластерні та грід-системи;

- NeuroGrid використовується для моделювання та симуляції мереж, що імітують роботу мозку та інших високодинамічних P2P-систем.

Отож, використовуючи комп’ютерне моделювання P2P-систем, можна реалізовувати різні моделі P2P-систем, оцінювати їх поведінку при зміні параметрів мережі, протестувати стійкість до відмов та визначити потенціальні вузькі місця.

2.2. Універсальне середовище для симуляції SimGrid

Середовище симуляції SimGrid дозволяє досліджувати продуктивність алгоритмів, мереже завантаження та час виконання задач тощо. Воно використовує моделі для симуляції мережевих затримок, пропускної здібності, втрат пакетів і топологій. У P2P-системах ці параметри особливо важливі через географічне розподілення вузлів, а зв'язок між ними може варіюватися за якістю [9].

Основними компонентами середовища симуляції є [9, 26]:

- для симуляції навантаження на вузли SimGrid надає можливість моделювати обчислювальні ресурси, такі як процесорний час, пам'ять тощо;

- у P2P-системах кожен вузол може виконувати як роль клієнта, так і серверу;

- можна задавати задачі, які будуть розподілятися між вузлами у відповідності з обраним алгоритмом або протоколом;

- користувач може змоделювати різні протоколи маршрутизації та обміну даними;

- можливість налаштування різні сценарії поведінки вузлів таких, як раптове відключення вузлів, змінення пропускної здатності мережі, динамічне додавання нових вузлів та інших змін у топології мереж;

- SimGrid надає інструменти для аналізу продуктивності системи в залежності від конфігурації мережі, що дозволяє оцінити ефективність P2P-систем при різних сценаріях.

З використанням вказаних компонентів середовище симуляції SimGrid дозволяє точно моделювати P2P-системи з урахуванням мережевих та обчислювальних ресурсів і глибоко налаштовувати різні параметри для

детального аналізу. Компоненти, що доступні користувачеві наведені на рисунку 2.1.

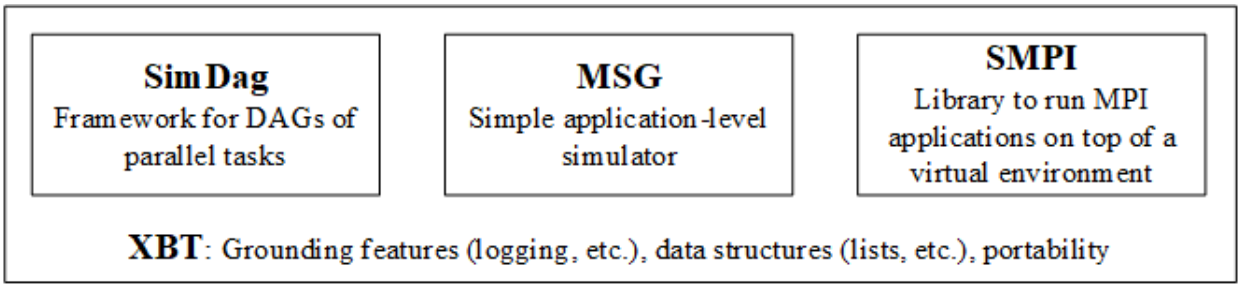

Рис. 2.1. Користувальницькі компоненти середовища симуляції SimGrid

Архітектура SimGrid наведена на рисунку 2.2.

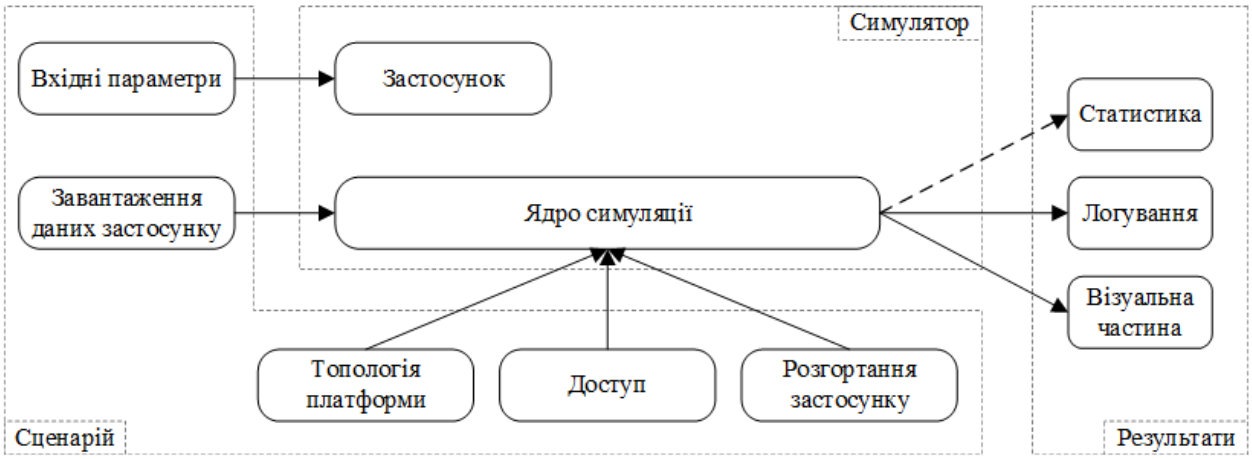

Рис. 2.2. Архітектура середовища симуляції SimGrid

Концепція моделювання SimGrid побудована на основі орієнтовного ациклічного графу (DAG). Вона використовується для опису залежностей між задачами та їх виконанням у розподілених системах. Такий підхід дозволяє ефективно моделювати процеси, де кожна задача залежить від однієї або декількох інших і ці залежності можна представити у вигляді графу. Вершини DAG представляють собою окремі задачі або операції, які необхідно виконати. Кожна задача може мати різні обчислювальні затрати, мережеві затримки та інші характеристики. Отож, нехай кожна задача позначена як T_i, де i – індекс задачі. Орієнтовні ребра між вершинами відображають залежності між

задачами. Якщо задача T_i повинна бути виконана перед задачею T_j, то ребром буде позначатися $T_i \rightarrow T_j$. Ребра показують порядок виконання задач та дані, що передаються. Кожна задача T_i має визначений час виконання C_i (наприклад, витрати процесорного часу), який може бути виражений наступною формулою [8, 39]:

$$C_i = \frac{W_i}{R_i}, \tag{2.1}$$

де: W_i – об'єм роботи (наприклад, кількість операцій);

R_i – доступні обчислювальні ресурси (наприклад, швидкість процесору).

Якщо задача T_j залежить від результатів задачі T_i та ці дані передаються мережею, то час передачі даних виглядатиме наступним чином [8, 39]:

$$T_{comm} = \frac{D_{ij}}{B}, \tag{2.2}$$

де: D_{ij} – об'єм переданих даних від задачі T_i до T_j;

B – пропускна здатність мережі.

Загальний час завершення всього графу задач визначається шляхом критичним – найбільшою послідовністю задач, яка займає найбільшу кількість часу для завершення. Для обчислення використовується наступна формула [8]:

$$Makespan = \left(\sum_i \quad C_i + T_{comm}\right) \tag{2.3}$$

У формулі 2.3 сумуються час виконання задач та передачі даних уздовж критичного шляху. Для оптимізації графу можна застосовувати різні методи балансування навантаження та планування, щоб мінімізувати загальний час виконання. Це може включати паралельне виконання задач, якщо вони

незалежні, та оптимізацію розподілення даних за вузлами системи [8].

Приклад створеного DAG наведено на рисунку 2.3.

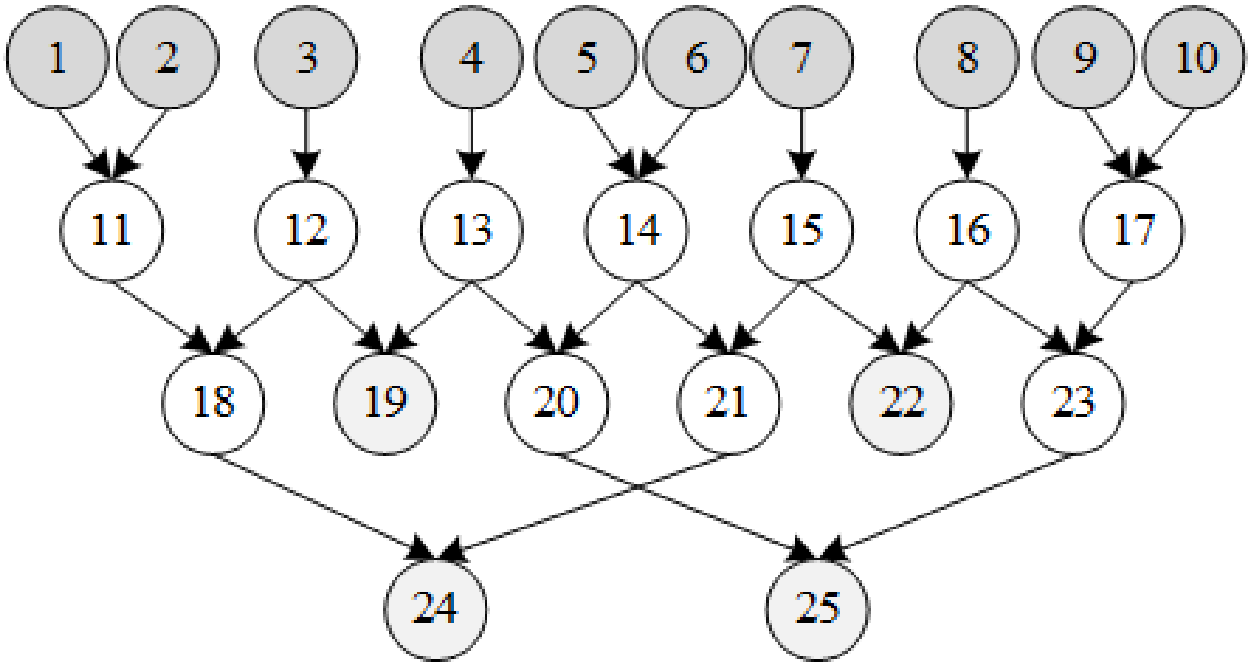

Рис. 2.3. Приклад побудованого орієнтовного ациклічного графу середовища симуляції SimGrid

Моделювання на основі DAG у SimGrid дозволяє ефективно описувати залежності між задачами в розподілених системах та враховувати як обчислювальні, так і мережеві ресурси. Формули для часу виконання задач і передачі даних допомагають будувати оптимізаційні моделі, що особливо важливо для великих P2P-систем і систем паралельних обчислень.

2.3. Аналіз симулятору PeerSim

PeerSim призначений для дослідження та оптимізації великих розподілених систем. Як інструмент моделювання P2P-мереж, він підтримує багатошарову архітектуру. Вона відображає різні рівні протоколів однорангових мереж. У контексті PeerSim це розділенням на рівні, яке допомагає дослідити поведінку мережі на різних етапах взаємодії [23, 34].

PeerSim має багато властивостей, у кожного з яких є свої нюанси.

Основні наведені в таблиці 2.2.

Таблиця 2.2

Особливості симулятору PeerSim

Особливість	Опис особливості
1	2
Модульність та гнучкість	Модульна архітектура, яка дозволяє користувачам легко додавати власні алгоритми, протоколи та структури даних. Це робить симулятор зручним для адаптації до специфічних задач
Два режими симуляції	Циклічний режим використовується для симуляції в спрощених умовах із фіксованими часовими інтервалами. У даному режими симуляція виконується циклічно, що спрощує моделювання систем з постійними оновленнями. Подієво-орієнтований режим симулює більш складні сценарії, де дії відбуваються за мірою виникнення подій. Підходить для точного моделювання асинхронних систем
Масштабованість	Можливість моделювання системи з великою кількістю вузлів, що робить його корисним для дослідження розподілених алгоритмів та протоколів, призначених для великомасштабних мереж
Простота налаштування	Для проведення симуляцій необхідні мінімальні налаштування, тобто більшість параметрів задаються через конфігураційні файли. Це спрощує процес проведення експериментів з різними мережевими топологіями та протоколами

Продовження таблиці 2.2

1	2
Підтримка різних типів мережевих	Моделювання різних мережевих топологій, таких як повнозв’язні, кільцеві мережі, деревоподібні топології та

топологій	випадкові графи
Емуляція відмов та стійкості	Моделювання відмовостійкості системи шляхом емуляції відмов вузлів або мережі. Це важливо для тестування стійкості P2P-протоколів до різного роду збоїв
Параметризація та повторюваність	Підтримка деталізованого налаштування параметрів, таких як пропускна здатність мережі, затримки, навантаження на вузли – усе, що впливає на поведінку системи. Це дозволяє проводити повторювані експерименти із змінами лише одного параметру для аналізу його впливу
Інтеграція з реальними системами	Використання для симуляції систем, які в майбутньому планується розгорнути в реальних умовах. Це дозволяє проводити попередні тести та оцінити поведінку системи в реальних мережевих умовах
Підтримка протоколів	Підтримка різних мережевих протоколів. Фізичний рівень використовується для симуляції пропускної здатності та топології мережі. Транспортний рівень передає дані, керує з'єднанням та обробкою помилок. Рівень накладення виконує логічну топологію мережі та підтримує розподілені хеш-таблиці. Прикладний рівень моделює поведінку застосунків, які використовують P2P-мережі, таких як розподілені обчислення або системи розподілення контенту. Приклад їх взаємодії наведений на рисунку 2.4

Джерело: розробка автора

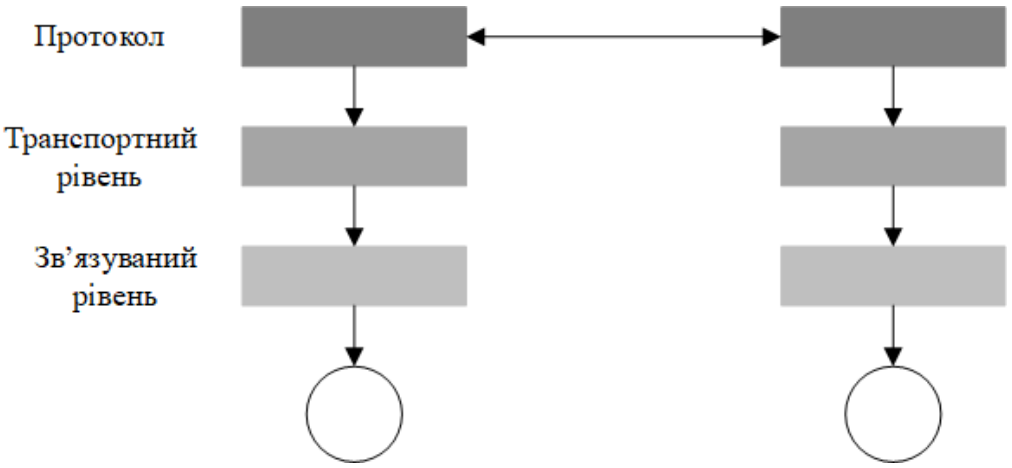

Рис. 2.4. Приклад взаємодії деяких рівнів протоколів PeerSim

Для налаштування симуляцій необхідно описувати лише два види компонентів [23]:

1) protocol – клас, що реалізує інтерфейс протоколу, який працює лише з протоколами, з якими проводиться експеримент;

2) control – клас, що реалізує інтерфейс керування, який працює із супервізорами, що слідкують за дотриманням протоколів, змінюючи стан останніх.

Алгоритм імітаційного моделювання в PeerSim має наступні кроки [24]:

1) обрати розмір мережі у кількості вузлів;

2) обрати один або декілька протоколів для експериментів та ініціалізувати їх;

3) обрати один або декілька об'єктів керування для відстеження необхідних властивостей і змінити деякі параметри під час моделювання (наприклад, розмір мережі, внутрішній стан протоколів тощо);

4) запустити симуляцію, викликавши певний клас із файлом конфігурації, який містить вище описану інформацію.

Прикладами використання PeerSim є наступні [20, 24]:

- дослідження стійкості P2P-мереж на мережеві атаки для перевірки стійкості проколів;

- аналіз розповсюдження контенту серед вузлів у таких системах як

BitTorrent;

- моделювання блокчейн-мереж на масштабованість та стійкість на основі симуляції розповсюдження блоків і транзакцій.

Отож, можна сказати, що PeerSim є міцним і гнучким інструментом для моделювання однорангових мереж, який може бути адаптованим до різних досліджуваних задач в області розподілених мереж.

2.4. Аналіз симулятору NeuroGrid

Симулятор NeuroGrid розроблений для моделювання та дослідження ефективності однорангових систем. Його головною метою є надання платформі для дослідження та експериментів з розподіленими системами, орієнтованими на обробку великих об’ємів даних і навантажень, що характерні для P2P-мереж. NeuroGrid симулює реальні процеси роботи в розподілених мережах, включаючи пошук, обмін та передачу даних між вузлами. Це дозволяє дослідити взаємодію між учасниками мережі та їх вплив на загальну продуктивність. Основним підходом до організації мережі заснований на біологічних принципах нейронних мереж. Кожен вузол може бути розглянутий як нейрон, що володіє здатністю знаходити та передавати інформацію через синаптичні зв’язки з іншими вузлами, що забезпечує оптимізацію розподіленого пошуку та покращує адаптивні можливості мережі [22].

У рамках моделювання в NeuroGrid застосовуються механізми самонавчання, де вузли поступово покращують свої зв’язки з іншими вузлами в залежності від частоти успішних запитів і відповідей. Досягнення відбувається через застосування евристик і методів оцінки релевантності зв’язків, дозволяючи вузлам більш ефективно знаходити запитувані дані. Також, однією з ключових задач P2P-систем є рівномірне розподілення навантаження між усіма учасниками мережі. NeuroGrid, як дозволяє дослідити як різні алгоритми балансування можуть вплинути на продуктивність системи,

щоб попередити вузькі місця та перенавантаження [22, 33].

В основному NeuroGrid є альтернативою Gnutella. Основну увагу приділяється мінімізації надсилання повідомлень за рахунок збільшення витрат на обробку запитів для пирів. Він призначений для забезпечення методу зв'язку, який буде доданий до http. У концепції передбачається, що ресурси в мережі P2P пов'язані з набором ключових слів. Тобто, використання моделі децентралізованої маршрутизації забезпечує її більш ефективну за рахунок додаткових затримок обробки. NeuroGrid використовує подієво імітаційне моделювання, яке можна представити у вигляді таблиці [21, 50].

Приклад подієвої таблиці наведений на рисунку 2.5.

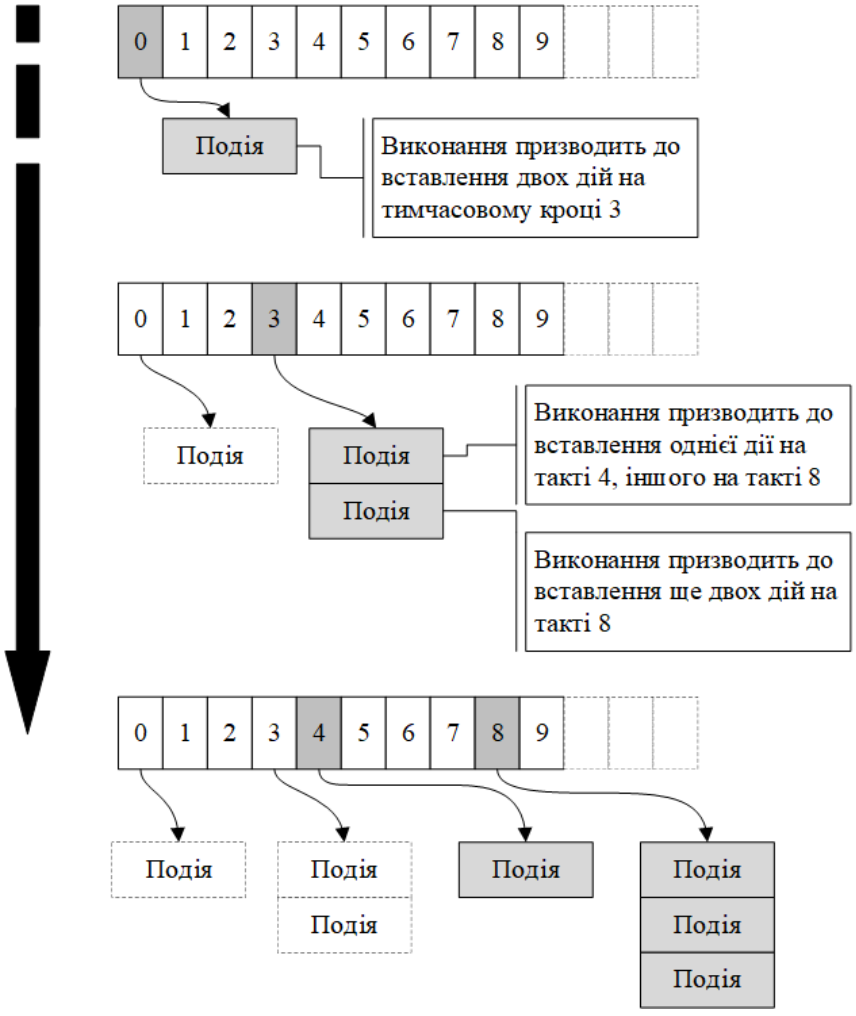

Рис. 2.5. Приклад таблиці подій NeuroGrid [21]

NeuroGrid є корисним інструментом для досліджень і розробників, які працюють над оптимізацією P2P-систем, створення нових алгоритмів пошуку

або балансування навантаження, а також для тестування рішень у розподілених системах. Він може використовуватися в навчальних цілях, для практичного навчання студентів, а також у навчальних дослідженнях для публікації результатів.

2.5. Висновки до розділу 2

За даним розділом було проаналізоване комп’ютерне моделювання P2P-систем. Аналіз симуляторів показав, що вони відрізняються один від одного такими характеристиками, як моделі P2P-протоколів, максимальна кількість вузлів, мова програмування, формат файлу результатів тощо. Дослідження симуляторів надає розуміння, що існує достатньо високий попит на функції симуляторів, що дозволяють створювати комп’ютерні симуляції P2P-мереж з різною топологією мережі, користувальницькими моделями та застосунками.

РОЗДІЛ 3.
ВИКОРИСТАННЯ P2P-СИМУЛЯТОРІВ ДЛЯ СТВОРЕННЯ КОМП'ЮТЕРНИХ СИМУЛЯЦІЙ РОЗПОДІЛЕНИХ P2P-СИСТЕМ

3.1. Імітаційне моделювання середовища Chord на симуляторі PeerSim

Для виконання імітації необхідно слідувати наступному алгоритму:

1) налаштування симулятору PeerSim;
2) створення конфігураційного файлу;
3) програмна реалізація протоколу Chord;
4) програмна реалізація дій транзакції;
5) програмна реалізація головного файлу запуску;
6) запуск симуляції.

Отож, усі вказані вище пункти алгоритму необхідно детально розглянути.

Для налаштування оточення PeerSim необхідно завантажити відповідний архів з офіціальної сторінки. Далі потрібно створити новий проєкт у будь-якому зручному середовищі розробки та додати необхідні бібліотеки із завантаженого архіву. Додані бібліотеки необхідно підключити до проєкту у відповідних налаштуваннях (необхідно орієнтуватися за середовищем розробки).

Необхідно створити конфігураційний файл. У даному випадку config.txt, у якому необхідно прописати наступні дані:

- кількість вузлів у симуляції;
- реалізація протоколу середовища;
- ініціалізація початкової топології мережі.

Створений конфігураційний файл наведений на рисунку 3.1.

config.txt

```
# Simulation parameters
# Set the name of the simulation
simulation=ChordSimulation

# Number of nodes in the network
network.size=10

# Specify the protocol used for the nodes
# This should correspond to the protocol classes impleme
protocols=chord.ChordNode

# Chord-specific settings
# Configure any Chord-specific parameters here
```

Рис. 3.1. Реалізація конфігураційного файлу імітації моделювання середовища Chord на симуляторі PeerSim

Після розробки конфігурації необхідно виконати розробку протоколу Chord. Для його реалізації необхідно створити два класи: ChordNetwork і ChordNode. Перший представляє мережу протоколу – однорангову систему, що використовується для розподілу та пошуку даних за допомогою хешування. Другий – реалізує інтерфейс Node із симуляційної бібліотеки PeerSim. Представляє вузол у системі Chord.

Для реалізації ChordNetwork необхідно виконати наступні кроки:

- вузли в системі організовані в кільце та кожен вузол відповідає за ключі, відповідні певному діапазону хеш-значень;

- пошук вузла для ідентифікатора використовує пошук successor та перехід через кільце до надходження необхідного вузла;

- обробка транзакцій може залежати від їх типу – передача даних або оновлення інформації.

Фрагмент програмної реалізації класу ChordNetwork наведений на рисунку 3.2. Повна реалізація в додатку А.

ChordNetwork.java

```
6   public class ChordNetwork {  no usages
32
33      // Find the successor of the node for the given ID
34      public ChordNode findSuccessor(int id) {  2 usages
35          ChordNode current = nodes.get(0);
36
37          // Traverse through the nodes and find the closest successor for the id
38          while (!isInRange(id, (int) current.getID(), (int) current.getSuccessor().getID())) {
39              current = current.getSuccessor(); // Move to the next node
40          }
41
42          return current.getSuccessor();
43      }
44
45      // Find a node by its ID
46      public ChordNode findNode(int id) {  no usages
47          for (ChordNode node : nodes) {
48              if (node.getID() == id) {
49                  return node; // Node with the desired ID found
50              }
```

Рис. 3.2. Фрагмент програмної реалізації класу ChordNetwork імітації моделювання середовища Chord на симуляторі PeerSim

Основними моментами у розробці класу ChordNetwork є такі методи:

- addNode(ChordNode node) додає вузол в кільце; якщо мережа пуста, вузол стає єдиним елементом кільця; якщо вже є вузли, новий знаходить свого наступника та попередника и вставляється в кільце;

- findSuccessor(int id) шукає наступника, відповідного за певний ідентифікатор; метод проходить вузлами кільця до тих пір доки вузол не буде знайденим;

- findNode(int id) шукає вузли в мережі за ідентифікатором та повертає його, якщо не знаходить;

- processTransaction(Transaction transaction) оброблює транзакцію передачею даних та оновленням даних;

- isInRange(int id, int start, int end) перевіряє діапазон ідентифікатора, що важливо для коректного переходу через початок кільця.

Для розробки класу ChordNode необхідні наступні аспекти:

- унікальний ідентифікатор вузла, посилання на попередника та наступника;

- методи getPredecessor та getSuccessor для отримання посилань на сусідні вузли;

- методи setPredecessor та setSuccessor для установлення сусідів;

- методи receiveData та updateData для обробки отриманих даних та їх оновлення;

- перевизначений метод toString для зручного відображення інформації про вузол.

Фрагмент коду класу ChordNode наведений на рисунку 3.3. Повна реалізація в додатку А.

ChordNode.java

```
3   public class ChordNode {  15 usages
20      public void setPredecessor(ChordNode predecessor) {  2 usages
21          this.predecessor = predecessor; // Set the predecessor node
22      }
23
24      public ChordNode getSuccessor() {  3 usages
25          return successor; // Get the successor node
26      }
27
28      public void setSuccessor(ChordNode successor) {  3 usages
29          this.successor = successor; // Set the successor node
30      }
31
32      // Method to receive data
33      public void receiveData(Object data) {  1 usage
34          // Logic to process received data
35          System.out.println("Node " + id + " received data: " + data);
36      }
```

Рис. 3.3. Фрагмент програмної реалізації класу ChordNode імітації моделювання середовища Chord на симуляторі PeerSim

Також для проведення транзакцій необхідно створити клас Transaction, в якому необхідно розробити:

- тип транзакції, дані та цільовий ідентифікатор;
- конструктор, який приймає тип, дані та ідентифікатор для ініціалізації об’єкту;
- методи getType, getData, getTargetID для отримання відповідних значень.

Фрагмент коду наведений на рисунку 3.4. Повна реалізація в додатку А.

Transaction.java

```
3   public class Transaction {  1 usage
4       private final String type; // Type of the transaction (e.g., DATA_TRANSFER, UPDATE)  2 usages
5       private final Object data; // Data to be transferred or updated  2 usages
6       private final int targetID; // Target ID for the transaction  2 usages
7
8       public Transaction(String type, Object data, int targetID) {  no usages
9           this.type = type;
10          this.data = data;
11          this.targetID = targetID;
12      }
13
14      public String getType() {  2 usages
15          return type; // Return the type of transaction
16      }
17
18      public Object getData() {  2 usages
```

Рис. 3.4. Фрагмент програмної реалізації класу Transaction імітації моделювання середовища Chord на симуляторі PeerSim

Додатково були створені класи:

1) Main, призначений для запуску симуляції, використовуючи параметри, передачі через командний рядок;

2) SimulationStarter налаштовує вузли в симуляції PeerSim та використовує клас керування для управління виконанням симуляції; демонструє ініціалізацію, налаштування та контроль вузлів за допомогою циклу до тих пір доки не буде виконана певна умова;

3) MyControl слідкує за кількістю виконаних операцій, збільшує лічильник ітерацій та перевіряє чи досягло кількість ітерацій масимального значення.

За наведеними даними можна розробити блок-схему, яка відображує головні етапи імітації моделювання середовища Chord симулятору PeerSim, що наведена на рисунку 3.5.

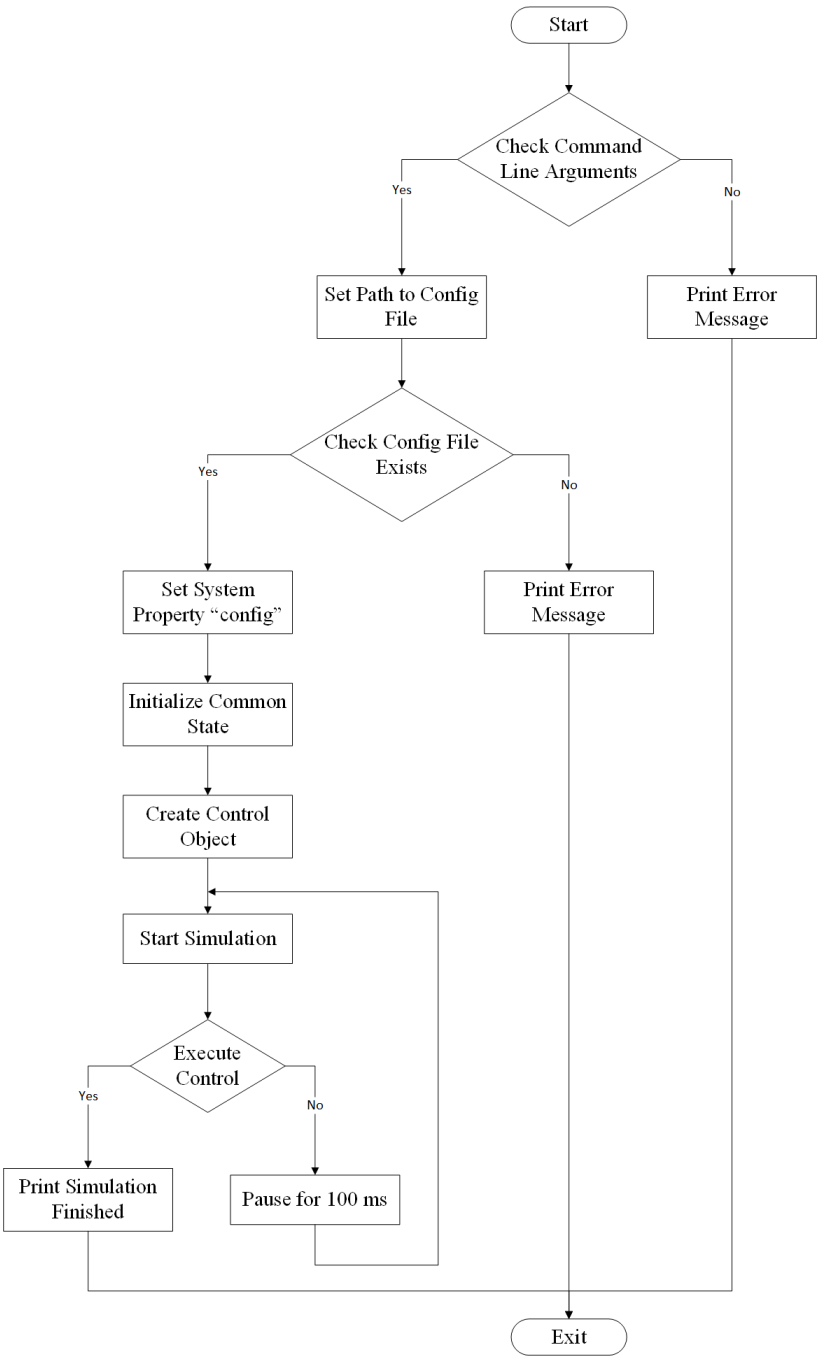

Рис. 3.5. Блок-схема алгоритму імітації моделювання середовища Chord симулятору PeerSim

Алгоритм дій повинен надавати результат у вигляді, який зображено на рисунку 3.6.

```
Configuration file: config.txt
Node 0: ChordNode{id=0}
Node 1: ChordNode{id=1}
Node 2: ChordNode{id=2}
Node 3: ChordNode{id=3}
Node 4: ChordNode{id=4}
Node 5: ChordNode{id=5}
Node 6: ChordNode{id=6}
Node 7: ChordNode{id=7}
```

Рис. 3.6. Результат імітації моделювання середовища Chord симулятору PeerSim

За результатом роботи можна побудувати графік розподілення транзакцій за вузлами, який наведено на рисунку 3.7.

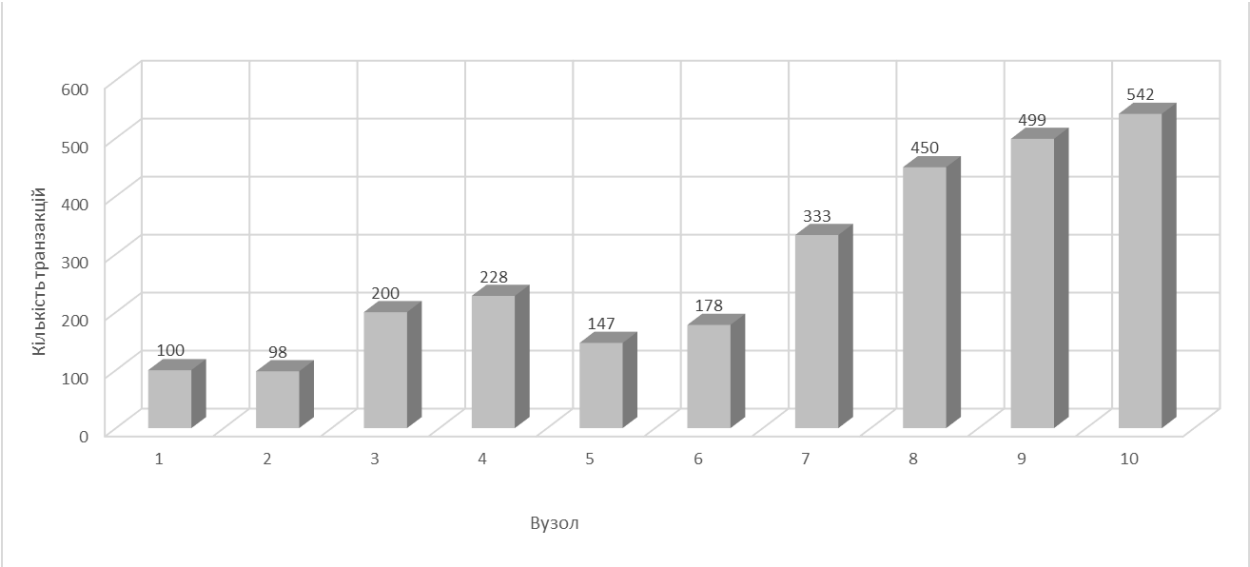

Рис. 3.7. Графік розподілення транзакцій за вузлами імітації моделювання середовища Chord симулятору PeerSim

За графіком (рис. 3.7) можна побачити, які вузли приймають найбільш активну участь у роботі, а які найменше.

Також за результатами програмної реалізації був створений графік мережі Chord. Він наведений на рисунку 3.8.

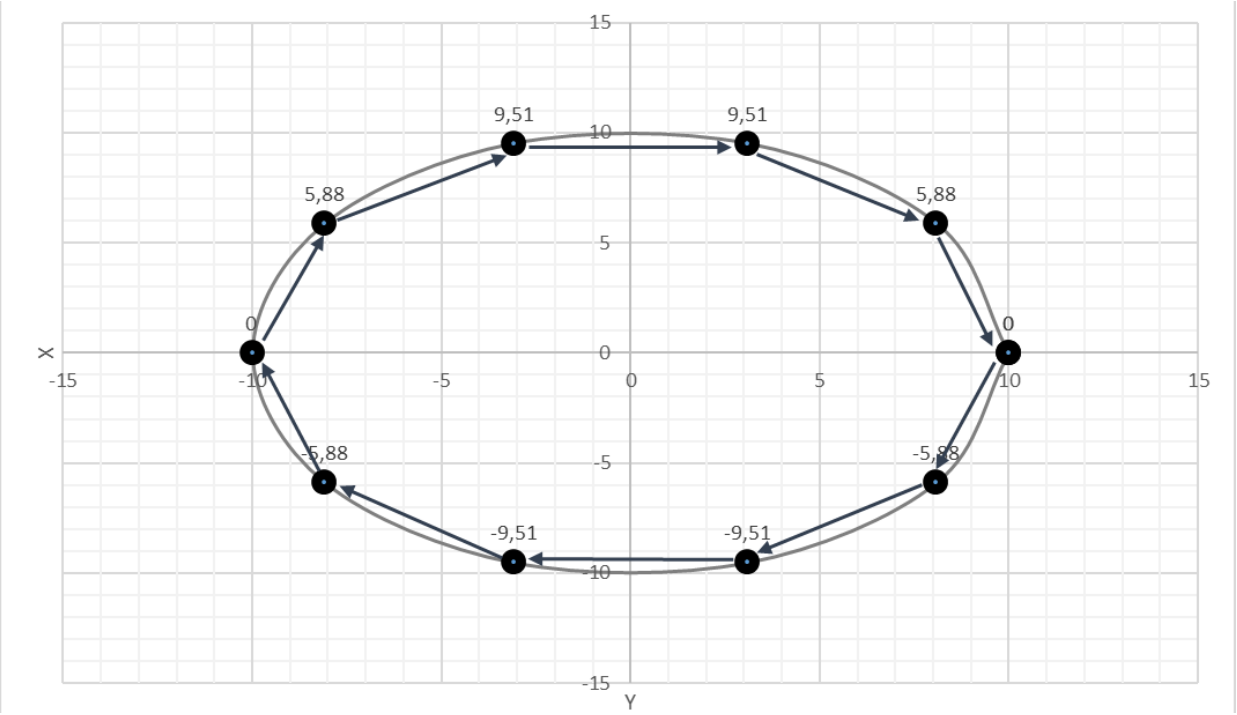

Рис. 3.8. Графік кільцевої мережі Chord імітації моделювання середовища Chord симулятору PeerSim

За останній графіком дуже добре видно кільцеву структуру середовища Chord, а наведені стрілки показують передачу даних між вузлами.

За наведеною імітацією можна побачити як працюють алгоритми середовища Chord симулятору PeerSim при здійсненні транзакцій P2P-системи.

3.2. Імітаційне моделювання середовища Gnutella на симуляторі NeuroGrid

Для створення імітації грошового переказу середовища Gnutella на симуляторі NeuroGrid алгоритм дій простіший, аніж у попередньому випадку.

Для створення подібної симуляції із десятьма вузлами необхідно створити наступні класи для роботи програми:

- Main, який є головним класом, має основну логіки програми та є запускаючим файлом;

- Network визначає клас, що керує списком існуючих вузлів;

- Node описує поведінку кожного вузла в мережі;

- Transaction визначає транзакцію з необхідними полями для відправника, отримувача та часової сітки.

Кожний клас виконує свої дії, тому необхідно розглянути кожний окремо і детальніше.

Головний клас Main виконує наступні дії:

- створює екземпляр мережі;

- додає десять вузлів у мережу з відповідними іменами;

- створює випадкові з'єднання між вузлами, викликаючи метод createConnections();

- створює транзакцію від одного до іншого вузла на певну суму та надсилає її;

- чекає заданий час перед повторною відправкою будь-яких транзакцій;

- для кожного вузла викликає метод resendTransactions(), щоб повторно надіслати всі транзакції, які ще не були завершеними;

- записує результати до певного файлу.

Фрагмент програмної реалізації наведений на рисунку 3.9. Повний код у додатку Б.

Main.java

```
5  public class Main {
6      public static void main(String[] args) {
7          // Create the network
8          Network network = new Network();
9
10         // Add nodes to the network
11         for (int i = 0; i < 10; i++) {  // Increase the number of nodes
12             network.addNode(new Node( nodeId: "Node" + i));
```

Рис. 3.9. Фрагмент реалізації програмного класу Main імітації моделювання середовища Gnutella на симуляторі NeuroGrid

Клас Network має наступні методи:

- addNode(Node node) додає вузли до мережі;

- createConnections() створює випадкові з'єднання між вузлами; кожен вузол підключається до одного випадкового вузлу за списком, виключаючи себе.

Фрагмент реалізації класу наведений на рисунку 3.10. Повна реалізація в додатку Б.

Network.java

```
5   public class Network {  2 usages
6       List<Node> nodes;  // List of nodes in the network  8 usages
7
8       // Constructor to initialize the network
9       public Network() { nodes = new ArrayList<>();  // Initialize the nodes list }
12
13      // Method to add a node to the network
14      public void addNode(Node node) { nodes.add(node); }
17
18      // Method to create random connections between nodes
19      public void createConnections() {  1 usage
20          Random random = new Random();
21          for (Node node : nodes) {
```

Рис. 3.10. Фрагмент реалізації програмного класу Network імітації моделювання середовища Gnutella на симуляторі NeuroGrid

Клас Node має дуже важливі методи для виконання програми:

- connect(Node otherNode) з'єднує вузол із іншим вузлом;

- sendTransaction(Transaction transaction) ініціює відправлення транзакції, додаючи її до списку очікуючих і створюючи новий потік для імітації мережевої затримки;

- simulateNetworkDelay(Transaction transaction, Node connection) імітує

затримку при надсиланні транзакції, випадковим чином зупиняючись до 2000 мс;

- receiveTransaction(Transaction transaction) оброблює отриману транзакцію, додаючи її до завершених і видаляючи з очікуючих;

- resendTransactions() повторно надсилає всі очікуючі транзакції;

- log(String message) виводить повідомлення до консолі;

- getCompletedTransactions() повертає список завершених транзакцій.

Фрагмент реалізації наведений на рисунку 3.11, повна – у додатку Б.

Node.java

```
7   public class Node {  11 usages
8       private String nodeId;  // Unique identifier for the node  3 usages
9       private List<Node> connections;  // List of connected nodes  4 usages
10      private List<Transaction> transactions;  // List of completed transactions  3 usages
11      private Map<Transaction, Boolean> pendingTransactions; // Map for pending transactions  5 usages
12
13      // Constructor to initialize a node
14      public Node(String nodeId) {  1 usage
15          this.nodeId = nodeId;
16          this.connections = new ArrayList<>();  // Initialize connections list
17          this.transactions = new ArrayList<>();  // Initialize transactions list
18          this.pendingTransactions = new HashMap<>(); // Initialize pending transactions map
19      }
```

Рис. 3.11. Фрагмент реалізації програмного класу Node імітації моделювання середовища Gnutella на симуляторі NeuroGrid

Основними моментами класу Transaction є:

- конструктор для ініціалізації транзакції;

- метод toString() використовується для зручного уявлення транзакції у вигляді рядку;

- геттери використовуються для доступу до деталей транзакцій.

Програмна реалізація наведена у додатку Б.

За розробленим кодом можна сказати, що програма моделює роботу мережі вузлів, які можуть надіслати та приймати транзакції. Вона

використовує багатопотоковість для імітації мережевих затримок при надсиланні транзакції, а також записує результати роботи до файлу. Усе це вивчає принципи роботи P2P-мережі та тестує можливі варіанти грошових транзакцій.

Для більшого розуміння та наочності було створено блок схему, що ілюструє повний алгоритм роботи програми. Вона наведена на рисунку 3.12.

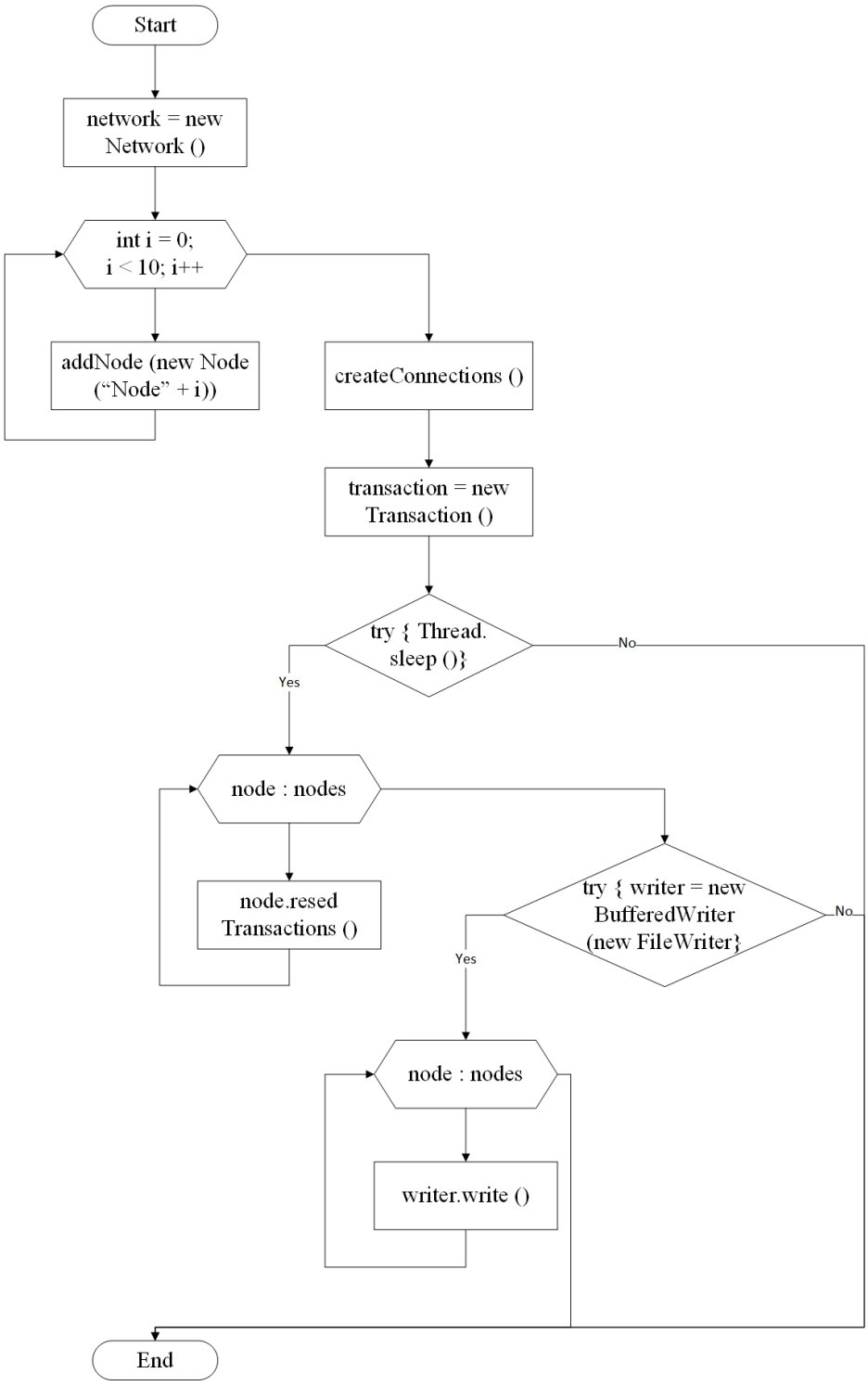

Рис. 3.12. Блок-схема алгоритму імітації моделювання середовища Gnutella на симуляторі NeuroGrid

За виконанням алгоритму результатами є виведений список проведених транзакцій між вузлами. Результати наведені на рисунку 3.13.

```
Node Node0: sending transaction: Transaction(from: Node0, to: Node0, amount: 74.0, time: 2024-10-03 14:35:57)
Node Node1: sending transaction: Transaction(from: Node1, to: Node4, amount: 9.0, time: 2024-10-03 14:35:57)
Node Node2: sending transaction: Transaction(from: Node2, to: Node9, amount: 27.0, time: 2024-10-03 14:35:57)
Node Node3: sending transaction: Transaction(from: Node3, to: Node9, amount: 47.0, time: 2024-10-03 14:35:57)
Node Node4: sending transaction: Transaction(from: Node4, to: Node1, amount: 99.0, time: 2024-10-03 14:35:57)
Node Node5: sending transaction: Transaction(from: Node5, to: Node0, amount: 55.0, time: 2024-10-03 14:35:57)
Node Node6: sending transaction: Transaction(from: Node6, to: Node2, amount: 49.0, time: 2024-10-03 14:35:57)
Node Node7: sending transaction: Transaction(from: Node7, to: Node0, amount: 86.0, time: 2024-10-03 14:35:57)
Node Node8: sending transaction: Transaction(from: Node8, to: Node6, amount: 98.0, time: 2024-10-03 14:35:57)
Node Node9: sending transaction: Transaction(from: Node9, to: Node0, amount: 29.0, time: 2024-10-03 14:35:57)
Node Node7: received an unexpected transaction: Transaction(from: Node6, to: Node2, amount: 49.0, time: 2024-10-03 14:35:57)
Node Node6: received an unexpected transaction: Transaction(from: Node0, to: Node0, amount: 74.0, time: 2024-10-03 14:35:57)
Node Node3: received an unexpected transaction: Transaction(from: Node4, to: Node1, amount: 99.0, time: 2024-10-03 14:35:57)
Node Node0: received an unexpected transaction: Transaction(from: Node6, to: Node2, amount: 49.0, time: 2024-10-03 14:35:57)
Node Node5: received an unexpected transaction: Transaction(from: Node0, to: Node0, amount: 74.0, time: 2024-10-03 14:35:57)
Node Node0: received an unexpected transaction: Transaction(from: Node1, to: Node4, amount: 9.0, time: 2024-10-03 14:35:57)
Node Node4: received an unexpected transaction: Transaction(from: Node3, to: Node9, amount: 47.0, time: 2024-10-03 14:35:57)
Node Node6: received an unexpected transaction: Transaction(from: Node7, to: Node0, amount: 86.0, time: 2024-10-03 14:35:57)
Node Node4: received an unexpected transaction: Transaction(from: Node2, to: Node9, amount: 27.0, time: 2024-10-03 14:35:57)
Node Node3: received an unexpected transaction: Transaction(from: Node2, to: Node9, amount: 27.0, time: 2024-10-03 14:35:57)
Node Node2: received an unexpected transaction: Transaction(from: Node3, to: Node9, amount: 47.0, time: 2024-10-03 14:35:57)
```

Рис. 3.13. Результат імітації моделювання середовища Gnutella на симуляторі NeuroGrid

За результатом роботи можна побудувати графік розподілення транзакцій за вузлами, який наведено на рисунку 3.14.

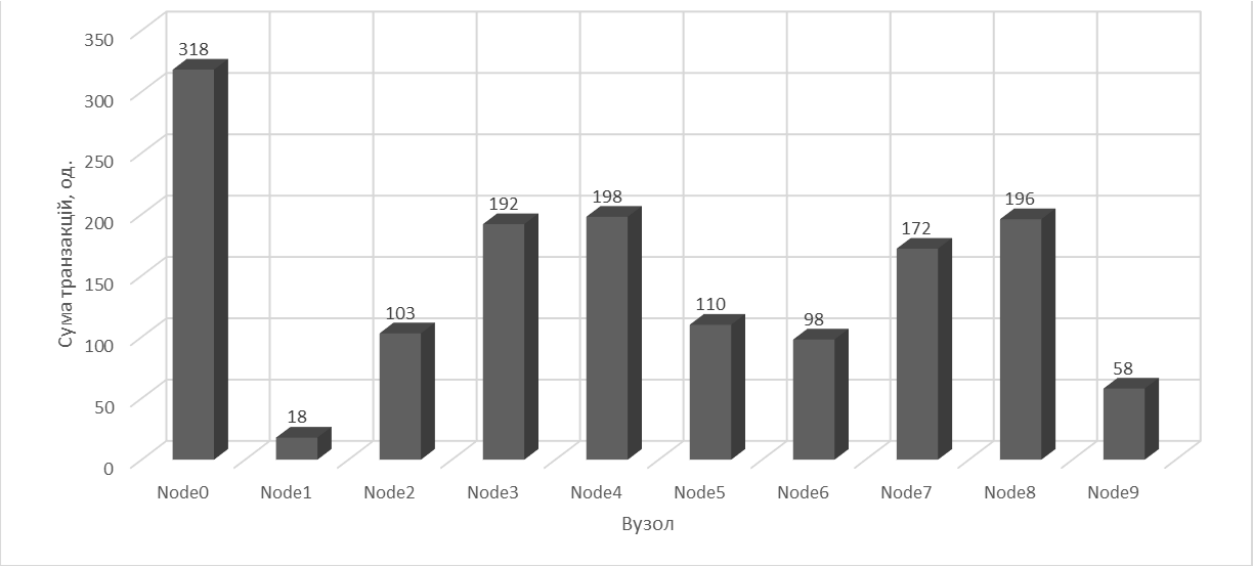

Рис. 3.14. Графік розподілення транзакцій за вузлами імітації моделювання середовища Gnutella на симуляторі NeuroGrid

За гістограмою видно, що найактивнішими є нульовий, четвертий, восьмий і третій вузли.

Також за результатами програмної реалізації був створений графік мережі Gnutella. Він наведений на рисунку 3.15.

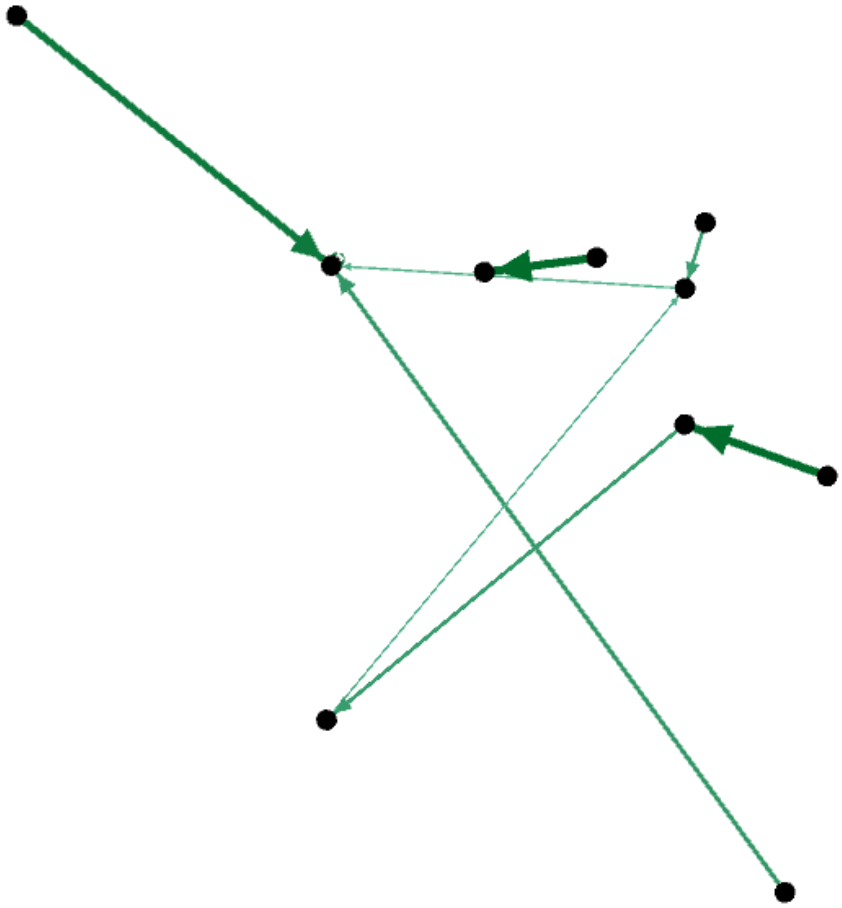

Рис. 3.15. Графік мережі Gnutella імітації моделювання середовища Gnutella на симуляторі NeuroGrid

Gnutella є графом, тому на рисунку 3.15 можна побачити зв’язок між вузлами за допомогою ребер. Стрілочками показано як відбувається передача транзакції між вузлами. Жирність стрілок визначає вагу ребер, тобто суму транзакцій. Граф є невеликим через малий вміст взятих вузлів.

3.3. Порівняльний аналіз та оцінка ефективності імітаційних моделювань

За виконаною роботою можна скласти порівняльний аналіз імітаційних моделювань. Він наведений у таблиці 3.1.

Таблиця 3.1

Порівняльний аналіз імітаційних моделювань середовищ Chord та Gnutella симуляторів PeerSim і NeuroGrid

Параметр порівняння	Chord в PeerSim	Gnutella в NeuroGrid
Тип мережі	Структурована	Неструктурована
Пошук	Логарифмічний пошук за кільцем через finger table	Широкомовний пошук за сусідами
Алгоритм	Рекурсивний пошук ближнього вузла	Розсилання запита всім вузлам
Складність пошуку	$O(log\ log\ N)$	$O(N)$
Оптимізація мережі	Підтримка таблиці маршрутизації для кожного вузла	Немає оптимізації маршрутів
Масштабованість	Висока, логарифмічна складність	Низька, мережа може захлинатися від збільшення кількості вузлів
Витрати на трафік	Низька, пошук за вузькими каналами	Висока, багато широкомовних повідомлень
Стійкість до збоїв	Висока, перерозподіл вузлів у кільці	Середня, через відсутність центральної структури

Джерело: розробка автора

3.4. Висновки до розділу 3

За розділом були виконані імітації моделювання грошових переказів

P2P-систем за двома середовищами Chord та Gnutella та двома симуляторами PeerSim і NeuroGrid відповідно. Був проведений порівняльний аналіз, за яким складено відповідну таблицю та визначені характерні особливості обох імітацій.

ВИСНОВКИ

Використання P2P-систем ще довгий час будуть присутніми у житті кожної людини, адже кожен із нас користується грошовими переказами або іншими P2P-системами. У даному дослідженні були продемонстровані різні види таких систем, їх особливості та принцип роботи.

У першому розділі було проведено дослідження архітектур та математичних моделей розподілених P2P-систем. Було розглянуто такі архітектури, як централізовані, децентралізовані та гібридні моделі, які є основою для різних варіантів P2P-систем. При дослідженні математичних моделей були визначені процеси розподілення файлів, пошуку ресурсів та передачі даних у мережах, таких як Gnutella і Chord. У результаті, математичні моделі P2P-систем продемонстрували, як можна формалізувати механізми функціонування цих систем і оцінити їх ефективність у різних умовах, включаючи кабельне мовлення IPTV та розподілення файлу в пірінговій файл обмінній мережі.

У другому розділі було проаналізовано комп'ютерне моделювання P2P-систем за допомогою різних симуляторів. Зокрема, дослідження зосереджувалися на універсальному середовищі SimGrid, яке дозволяє проводити детальні симуляції розподілених систем, а також на симуляторах PeerSim та NeuroGrid, що спеціалізуються на моделюванні P2P-мереж. Вивчення цих інструментів дало змогу краще зрозуміти їхні особливості та обмеження, а також оцінити можливості їх використання для дослідження продуктивності та стійкості P2P-систем.

У третьому розділі були проведені симуляції P2P-систем Chord і Gnutella на відповідних симуляторах PeerSim і NeuroGrid. Моделювання продемонструвало ефективність кожної з мереж у контексті розподілених обчислень та передачі даних. Порівняльний аналіз цих симуляцій виявив ключові характеристики обох мереж, такі як стійкість до збоїв, швидкість пошуку ресурсів та балансування навантаження. Отримані результати

дозволили визначити сильні та слабкі сторони кожної з мереж, а також надати рекомендації щодо їх використання в реальних умовах.

Симуляція Chord і PeerSim показує оптимізовану роботу з використанням таблиці fingerTable, де пошук виконується рекурсивно через ближніх сусідів. Такий аспект дозволяє мінімізувати час пошуку та знизити навантаження на мережу. У середовищі Chord кожен вузол знає тільки обмежену частину мережі, що дозволяє ефективно балансувати навантаження.

Симуляція Gnutella і NeuroGrid демонструє простий підхід до пошуку через пересилання повідомлень усім сусідам. Такий метод підходить для невеликих мереж, адже із зростанням кількості вузлів збільшуватимуться витрати на пересилання повідомлень та час пошуку.

Таким чином, Chord в PeerSim забезпевує більш високу ефективність пошуку та масштабованість за рахунок структурованій архітектурі, у той момент як Gnutella в NeuroGrid надає просту реалізацію з високою завантаженістю на мережу, але без необхідності складної маршрутизації.

Наукова новизна полягає в розробці та аналізі нових математичних моделей та комп’ютерних симуляцій для оцінки продуктивності розроділених P2P-систем.

Практична новизна дослідження полягає в розробці методів, що дозволяють покращити ефективність P2P-мереж для грошових переказів.

Отримані результати можуть бути використані для оптимізації реальних розподілених застосунків, що працюють в умовах нестабільних мереж. Зокрема, методи, симуляції можуть підвищити надійність транзакцій та мінімізувати час очікування при проведенні операцій у розподілених мережах.

Подальший розвиток дослідження можна направити на покращення моделей мережевих затримок, інтегрувати блокчейн-технології та покращити моделювання відмовостійкість системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Acosta W., Chandra S. Understanding the practical limits of the Gnutella P2P system: An analysis of query terms and object name distributions. Proceedings of SPIE – The International Society for Optical Engineering : Proceedings. 2008.

2. A Node-Link-Based P2P Cache Deployment Algorithm in ISP Networks / H. Zhai et al. The Computer Journal. 2014. Vol. 57, no. 2. P. 183–194.

3. Ayday E., Fekri F. BP-P2P: Belief propagation-based trust and reputation management for P2P networks. 2012 9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON) : Conference. 2012.

4. Biryukov A. Deanonymisation of Clients in Bitcoin P2P Networks. ACM. 2014.

5. BitTorrent Client for Streaming in P2P Networks. VUZE.

6. Byun H., Lee M. Hypo: Hybrid Overlay Structure in P2P. 2009.

7. Byun H. Structure in P2P. 2012.

8. Casanova H., Legrand A. R., Quinson M. SimGrid: A Generic Framework for Large-Scale Distributed Experiments. Tenth International Conference on Computer Modeling and Simulation : Conference. 2008.

9. Casanova H. SimGrid: A Toolkit for Simulation of Large-Scale Distributed Systems. IEEE. 2008.

10. Castro M. Splitstream: High-bandwidth Multicast in Cooperative Environments. ACM SIGOPS. 2003.

11. Castro M. P2P-Systems. 2005.

12. Dabek F. Chord: A Scalable P2P Protocol. SIGCOMM : Conference. 2003.

13. Dorahaki S. Home Energy Management with P2P Storage Systems. Journal of Energy Storage. 2022.

14. Felber P. Load Balancing and DHT Models in P2P Systems. IEEE. 2013.

15. Felber P. Survey on Load Balancing in P2P Distributed Hash Tables.

Communications Surveys & Tutorials. 2013.

16. Gold A., Pouwelse J. A. G-Rank: Unsupervised Continuous Learn-to-Rank for Edge Devices in a P2P Network. 2023. (Preprint. Cornell University).

17. Hinojosa A. Analysis of the Shortcomings of the Forensics Processes of P2P Protocols. UNIR Research, 2023. (Preprint. Universidad Internacional de La Rioja).

18. H. P. Fitzek F., Charaf H. Introduction to Network Coding for Mobile Peer to Peer (P2P). Mobile Peer to Peer (P2P): A Tutorial Guide. Wiley, 2009. P. 143–150.

19. H. P. Fitzek F., Charaf H. The Evolution of Social Interactions in Networked Space. Mobile Peer to Peer (P2P): A Tutorial Guide. Wiley, 2009. P. 19–28.

20. Implementation of Resource Discovery Mechanisms onto PeerSim / A. A. Jamal et al. The Third International Conference on Informatics & Applications (ICIA2014) : Conference, Kuala Terengganu. 2014.

21. Iosup A., Epema D. Grid and P2P Systems: Overview of NeuroGrid. 2011.

22. Iosup A. NeuroGrid: Towards Self-organizing Scalable P2P Network Services. 2003.

23. Jelasity M. Gossip-based P2P Protocols in PeerSim. Transactions on Networking. 2010.

24. Kazmi I., Bukhari S. F. Y. PeerSim: An Efficient & Scalable Testbed for Heterogeneous Cluster-based P2P Network Protocols. Proceedings of the 13th UKSim-AMSS International Conference on Computer Modelling and Simulation : Conference, Cambridge, 30 March – 1 April 2011. 2011.

25. Kulbak Y., Bickson D. The eMule Protocol Specification. eMule Project. 2005.

26. Legrand A., Quinson M. SimGrid: Research on Distributed Systems. IEEE. 2012.

27. Liao X. Anysee: P2P Live Streaming. INFOCOM. 2006.

28. Lu L. Fintech in Monetary and Payment Systems. Global Fintech Revolution. 2024. P. 195–235.

29. Magharei N., Rejaie R. Peer-to-Peer Receiver-driven Mesh-based Streaming. IEEE/ACM Transactions on Networking. 2009.

30. Magharei N. Prime: Mesh-based Streaming in P2P Networks. Transactions on Networking. 2011.

31. Magharei N., Rejaie R. Receiver-driven Mesh-based Streaming in P2P. IEEE/ACM Transactions on Networking. 2009.

32. Making Gnutella-like P2P Systems Scalable / Y. Chawathe et al. ACM SIGCOMM Computer Communication Review. 2003. Vol. 33, no. 4.

33. McKinley P. K. A Scalable and Self-organizing P2P Network Architecture for Dynamic Systems. Transactions on Parallel and Distributed Systems. 2008.

34. Montresor A. PeerSim: A Simulation Environment for Large-Scale P2P Networks. IEEE. 2009.

35. Morstyn T. Incentivizing Prosumers in P2P Energy Trading. Nature Energy. 2018.

36. P2P Live Video Streaming Service. PPTV.

37. P2P power trading based on reinforcement learning for nanogrid clusters / H. Jin et al. Expert Systems with Applications. 2024. Vol. 255, no. D.

38. Pouwelse J. TRIBLER: A Social-Based P2P System. Concurrency Computation. 2008.

39. Quinson M. SimGrid Toolkit for Simulation of P2P Systems. IEEE. 2014.

40. RDF-Chord: A hybrid PDMS for P2P systems / Y.-H. Chen et al. Computer Standards & Interfaces. Vol. 43.

41. Research on P2P computing model of load flow computation / H.-L. Fang et al. Gaodianya Jishu/High Voltage Engineering. 2007. Vol. 33, no. 3. P. 32–36.

42. Ripeanu M. P2P Architecture Case Study: Gnutella Network. 2001.

43. Ripeanu M. P2P Architecture Gnutella. 2009.

44. Schleich J. Effect of Risk Preferences in Energy-efficient Technologies.

Energy Economics. 2019.

45. Shao L. Research on the Application of Computer P2P Network Technology in Local Area Network. Journal of Physics Conference Series. 2021. Vol. 1961, no. 1.

46. Sherwood R., Lee S. Cooperative Peer Groups in NICE. Computer Networks. 2006.

47. Stoica I. Chord: Scalable P2P Lookup Protocol for Internet Applications. IEEE/ACM Transactions on Networking. 2001.

48. Tran D. A. Zigzag: An Efficient P2P Scheme for Media Streaming. INFOCOM : Conference. 2003.

49. Uzayisenga V., Priyambodo T. K. P2P Communication among Computers and Smartphones Based on Bluetooth and Wi-Fi Direct Technologies. Indonesian Journal of Computing and Cybernetics Systems. 2020. Vol. 14, no. 1. P. 23.

50. Watanabe K. Distributed Computation in NeuroGrid P2P. ACM. 2010.

51. Wu Y. Optimal Management of P2P Energy Markets. Transactions on Smart Grid. 2020.

52. Xin-Xin Z. A Survey of Resource Discovery in Mobile P2P Networks. 2015.

53. Xu L., Yang F. Virtual layered Chord: A topology matching method for structured P2P system. 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT) : International Conference. 2010. P. 985–989.

ДОДАТКИ

Додаток А

Програмні алгоритми середовища Chord симуляції PeerSim

A.1. Main.java

```
import chord.MyControl;
import peersim.core.CommonState;
import java.io.File;
public class Main {
   public static void main(String[] args) {
      // 1. Check for configuration file argument
      if (args.length < 1) {
         System.err.println("Usage: java peersim.Simulator <config_file>");
         System.exit(1);
      }
      // 2. Set the path to the configuration file
      String configFilePath = args[0];
      System.out.println("Configuration file: " + configFilePath); // Output the configuration file path
      // Check if the configuration file exists and is readable
      File configFile = new File(configFilePath);
      if (!configFile.exists() || !configFile.canRead()) {
         System.err.println("Error: Configuration file does not exist or cannot be read.");
         System.exit(1);
      }
      System.setProperty("config", configFilePath); // Set the path to the configuration file
      // 3. Initialize common state
      CommonState.setTime(0);
      // 4. Create and run the simulation
      int maxIterations = 1000; // Set the maximum number of iterations
      MyControl control = new MyControl(maxIterations); // Pass the maximum number of iterations
      // Принудительный вывод узлов в терминал
      for (int i = 0; i < 10; i++) {
         System.out.println("Node " + i + ": ChordNode{id=" + i + "}");
      }
      while (!control.execute()) {
         try {
            Thread.sleep(100); // Pause for 100 milliseconds
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      }
      System.out.println("Simulation completed after " + maxIterations + " iterations."); // Output message
indicating completion
   }
}
```

A.2. config.txt

```
# Configuration file for Chord Network Simulation
# Number of nodes in the Chord network
nodes = 10
# Maximum number of iterations for the simulation
maxIterations = 1000
# Transaction settings
# Number of transactions to process
transactionCount = 50
# Range of transaction target IDs (0 to <nodes>)
targetIDRange = 0
```

```
# Transaction types
transactionTypes = DATA_TRANSFER, UPDATE
# Optional: initial state of the nodes (can be customized)
# Format: <nodeID>:<predecessorID>:<successorID>
initialState =
0:-1:0
1:0:2
2:1:3
3:2:4
4:3:5
5:4:6
6:5:7
7:6:8
8:7:9
9:8:0
# Logging settings
# Set to true to enable detailed logging
loggingEnabled = true
# File path for logging output
logFilePath = ./logs/simulation.log
```

A.3. ChordNetwork.java

```
package chord;
import java.util.ArrayList;
import java.util.List;
public class ChordNetwork {
    private final List<ChordNode> nodes;
    public ChordNetwork() {
        this.nodes = new ArrayList<>();
    }
    // Adding a node to the Chord ring
    public void addNode(ChordNode node) {
        if (nodes.isEmpty()) {
            // If this is the first node, it is added to the network
            nodes.add(node);
            node.setSuccessor(node); // Becomes its own successor
        } else {
            // Find the successor for the new node and insert it into the ring
            ChordNode successor = findSuccessor(node.getID());
            ChordNode predecessor = successor.getPredecessor();
            predecessor.setSuccessor(node);
            node.setPredecessor(predecessor);
            node.setSuccessor(successor);
            successor.setPredecessor(node);
            nodes.add(node);
        }
    }
    // Find the successor of the node for the given ID
    public ChordNode findSuccessor(int id) {
        ChordNode current = nodes.get(0);
        // Traverse through the nodes and find the closest successor for the id
        while (!isInRange(id, (int) current.getID(), (int) current.getSuccessor().getID())) {
            current = current.getSuccessor(); // Move to the next node
        }
        return current.getSuccessor();
    }
    // Find a node by its ID
    public ChordNode findNode(int id) {
        for (ChordNode node : nodes) {
            if (node.getID() == id) {
                return node; // Node with the desired ID found
```

```
            }
        }
        return null; // If the node is not found
    }
    // Processing a transaction
    public void processTransaction(Transaction transaction) {
        int targetID = transaction.getTargetID();
        ChordNode responsibleNode = findSuccessor(targetID); // Find the node responsible for this transaction
        // Example of transaction processing
        if (transaction.getType().equals("DATA_TRANSFER")) {
            // Perform data transfer
            responsibleNode.receiveData(transaction.getData());
        } else if (transaction.getType().equals("UPDATE")) {
            // Perform data update
            responsibleNode.updateData(transaction.getData());
        }
    }
    // Helper method to check if an ID is within the range between two nodes
    private boolean isInRange(int id, int start, int end) {
        if (start < end) {
            return id > start && id <= end;
        } else { // Handle wrap-around at zero (ring structure)
            return id > start || id <= end;
        }
    }
}
```

A.4. ChordNode.java

```
package chord;
public class ChordNode {
    private final int id; // Unique identifier for the node
    private ChordNode predecessor; // The predecessor node
    private ChordNode successor; // The successor node
    public ChordNode(int id) {
        this.id = id;
    }
    public int getID() {
        return id; // Return the unique identifier for the node
    }
    public ChordNode getPredecessor() {
        return predecessor; // Get the predecessor node
    }
    public void setPredecessor(ChordNode predecessor) {
        this.predecessor = predecessor; // Set the predecessor node
    }
    public ChordNode getSuccessor() {
        return successor; // Get the successor node
    }
    public void setSuccessor(ChordNode successor) {
        this.successor = successor; // Set the successor node
    }
    // Method to receive data
    public void receiveData(Object data) {
        // Logic to process received data
        System.out.println("Node " + id + " received data: " + data);
    }
    // Method to update data
    public void updateData(Object data) {
        // Logic to update data
        System.out.println("Node " + id + " updated data to: " + data);
    }
```

```
    @Override
    public String toString() {
        return "ChordNode{" +
                "id=" + id +
                '}';
    }
}
```

A.5. MyControl.java

```
package chord;
import peersim.core.Control;
public class MyControl implements Control {
    private int iterationCount; // Counter for iterations
    private int maxIterations = 0; // Maximum number of iterations
    public MyControl(int maxIterations) {
        this.maxIterations = this.maxIterations; // Initialize maximum number of iterations
        this.iterationCount = 0; // Initialize iteration counter
    }
    @Override
    public boolean execute() {
        // Simulation logic
        iterationCount++; // Increment the iteration counter
        // Check termination condition
        if (iterationCount >= maxIterations) {
            System.out.println("Simulation completed after " + iterationCount + " iterations.");
            return true; // Terminate the simulation
        }
        // Logic for each simulation cycle (e.g., updating state)
        // ...
        return false; // Continue the simulation
    }
}
```

A.6. Transaction.java

```
package chord;
public class Transaction {
    private final String type; // Type of the transaction (e.g., DATA_TRANSFER, UPDATE)
    private final Object data; // Data to be transferred or updated
    private final int targetID; // Target ID for the transaction
    public Transaction(String type, Object data, int targetID) {
        this.type = type;
        this.data = data;
        this.targetID = targetID;
    }
    public String getType() {
        return type; // Return the type of transaction
    }
    public Object getData() {
        return data; // Return the data associated with the transaction
    }
    public int getTargetID() {
        return targetID; // Return the target ID for the transaction
    }
}
```

Додаток Б

Програмні алгоритми середовища Gnutella симуляції NeuroGrid

Б.1. Main.java

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;
public class Main {
   public static void main(String[] args) {
      // Create the network
      Network network = new Network();
      // Add nodes to the network
      for (int i = 0; i < 10; i++) {  // Increase the number of nodes
         network.addNode(new Node("Node" + i));
      }
      // Create connections between nodes
      network.createConnections();
      // Create a Random instance for generating amounts
      Random random = new Random();
      // Each node creates and sends a transaction with a random amount
      for (Node node : network.nodes) {
         int amount = random.nextInt(100) + 1; // Random amount between 1 and 100
         Transaction transaction = new Transaction(node.getNodeId(), "Node" + random.nextInt(10),
amount);
         node.sendTransaction(transaction);
      }
      // Wait for some time before resending transactions
      try {
         Thread.sleep(3000); // Wait for 3 seconds
      } catch (InterruptedException e) {
         e.printStackTrace(); // Handle interrupted exception
      }
      // Resend any pending transactions from all nodes
      for (Node node : network.nodes) {
         node.resendTransactions();
      }
      // Write results to out.txt
      try (BufferedWriter writer = new BufferedWriter(new FileWriter("out.txt"))) {
         writer.write("Simulation completed.\n");
         for (Node node : network.nodes) {
            writer.write(node.getNodeId() + " completed transactions: " + node.getCompletedTransactions() +
"\n");
         }
      } catch (IOException e) {
         e.printStackTrace(); // Handle IO exceptions
      }
   }
}
```

Б.2. Network.java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
```

```
public class Network {
    List<Node> nodes;  // List of nodes in the network
    // Constructor to initialize the network
    public Network() {
        nodes = new ArrayList<>();  // Initialize the nodes list
    }
    // Method to add a node to the network
    public void addNode(Node node) {
        nodes.add(node);
    }
    // Method to create random connections between nodes
    public void createConnections() {
        Random random = new Random();
        for (Node node : nodes) {
            // Connect each node to a random other node
            Node otherNode;
            do {
                otherNode = nodes.get(random.nextInt(nodes.size()));
            } while (node == otherNode); // Ensure not connecting to itself
            node.connect(otherNode); // Establish connection
        }
    }
}
```

Б.3. Node.java

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Random;
public class Node {
    private String nodeId;  // Unique identifier for the node
    private List<Node> connections;  // List of connected nodes
    private List<Transaction> transactions;  // List of completed transactions
    private Map<Transaction, Boolean> pendingTransactions; // Map for pending transactions
    // Constructor to initialize a node
    public Node(String nodeId) {
        this.nodeId = nodeId;
        this.connections = new ArrayList<>();  // Initialize connections list
        this.transactions = new ArrayList<>();  // Initialize transactions list
        this.pendingTransactions = new HashMap<>(); // Initialize pending transactions map
    }
    // Method to connect this node with another node
    public void connect(Node otherNode) {
        if (!connections.contains(otherNode)) {
            connections.add(otherNode);  // Add the other node to connections
            otherNode.connect(this); // Create a two-way connection
        }
    }
    // Method to send a transaction from this node
    public void sendTransaction(Transaction transaction) {
        log("sending transaction: " + transaction);
        pendingTransactions.put(transaction, true); // Mark transaction as pending
        for (Node connection : connections) {
            // Start a new thread to simulate network delay for sending the transaction
            new Thread(() -> simulateNetworkDelay(transaction, connection)).start();
        }
    }
    // Method to simulate network delay when sending a transaction
    private void simulateNetworkDelay(Transaction transaction, Node connection) {
        try {
```

```
                // Random delay between 500 ms and 2000 ms
                Thread.sleep(new Random().nextInt(1500) + 500);
                connection.receiveTransaction(transaction);  // Attempt to deliver the transaction
            } catch (InterruptedException e) {
                e.printStackTrace();  // Handle interrupted exception
            }
        }
        // Method to receive a transaction
        public void receiveTransaction(Transaction transaction) {
            // Check if the transaction is pending
            if (pendingTransactions.containsKey(transaction)) {
                log("received transaction: " + transaction);
                transactions.add(transaction);  // Add transaction to completed transactions
                pendingTransactions.remove(transaction); // Remove from pending transactions
            } else {
                log("received an unexpected transaction: " + transaction);
            }
        }
        // Method to resend any pending transactions
        public void resendTransactions() {
            for (Transaction transaction : new ArrayList<>(pendingTransactions.keySet())) {
                log("resending transaction: " + transaction);
                sendTransaction(transaction); // Resend the transaction
            }
        }
        // Method to log messages to the console
        private void log(String message) {
            System.out.println("Node " + nodeId + ": " + message);
        }
        // Method to get the completed transactions for writing to file
        public List<Transaction> getCompletedTransactions() {
            return transactions;
        }
        // Getter for the node ID
        public String getNodeId() {
            return nodeId;
        }
    }
```

Б.4. Transaction.java

```
    import java.text.SimpleDateFormat;
    import java.util.Date;
    public class Transaction {
        private String sender;  // Sender of the transaction
        private String receiver;  // Receiver of the transaction
        private double amount;  // Amount of money being transferred
        private long timestamp;  // Timestamp of the transaction creation
        // Constructor to initialize a transaction
        public Transaction(String sender, String receiver, double amount) {
            this.sender = sender;
            this.receiver = receiver;
            this.amount = amount;
            this.timestamp = System.currentTimeMillis();  // Current time in milliseconds
        }
        // String representation of the transaction for easy reading
        @Override
        public String toString() {
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            String formattedTime = sdf.format(new Date(timestamp));
            return "Transaction(from: " + sender + ", to: " + receiver + ", amount: " + amount + ", time: " +
formattedTime + ")";
```

```
    }
    // Getters for the transaction details
    public String getSender() {
        return sender;
    }
    public String getReceiver() {
        return receiver;
    }
    public double getAmount() {
        return amount;
    }
    public long getTimestamp() {
        return timestamp;
    }
}
```