НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Децентралізований месенджер з адаптивною маршрутизацією у нестабільних мережах»

Виконав:
студент IV курсу, групи ІС-13
Коноваленко Богдан Костянтинович _____

Керівник:
доцент кафедри ІСТ, к.т.н., доцент,
Писаренко Андрій Володимирович _____

Рецензент:
доцент кафедри ІПІ, к.т.н., доцент,
Ліщук Катерина Ігорівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту
Коноваленку Богдану Костянтиновичу

1. Тема проєкту «Децентралізований месенджер з адаптивною маршрутизацією у нестабільних мережах», керівник проєкту Писаренко Андрій Володимирович, к. т. н., доцент, затверджені наказом по університету від «23» травня 2025 р. № 1705-с

2. Термін подання студентом проєкту: 9 червня 2025 р.

3. Вихідні дані до проєкту: система яка підтримує функціонування у мережах від 3 до 50 вузлів без деградації продуктивності, час обробки маршруту повідомлення не перевищує 1000 мс за наявності зв’язку.

4. Зміст пояснювальної записки: опис предметної області, проєктування інформаційної системи, математичне забезпечення, реалізація компонентів системи, тестування та верифікація, реалізація інформаційної системи.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо): діаграма компонентів, діаграма послідовностей, діаграма класів, блок-схеми алгоритмів.

7. Дата видачі завдання: 12.03.2025.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення літератури	30.03.2025	
2	Аналіз існуючих методів розв’язання задачі	20.04.2025	
3	Постановка та формалізація задачі	28.04.2025	
4	Розроблення програмного забезпечення	11.05.2025	
5	Налагодження програми	16.05.2025	
6	Оформлення пояснювальної записки	25.05.2025	
7	Подання ДП на попередній захист	26.05.2025	
8	Подання ДП на основний захист	09.06.2025	

Студент Богдан КОНОВАЛЕНКО

Керівник Андрій ПИСАРЕНКО

АНОТАЦІЯ

Децентралізований месенджер з адаптивною маршрутизацією у нестабільних мережах.

Проєкт містить 83с. тексту, 25 рисунків, 1 таблиця, посилання на 31 літературних джерел, 2 додатки та 4 конструкторських документів.

ДЕЦЕНТРАЛІЗОВАНА МЕРЕЖА, ДИНАМІЧНА МЕРЕЖА, ДЕЦЕНТРАЛІЗОВАНІ АЛГОРИТМИ МАРШРУТИЗАЦІЇ, ДЕЦЕНТРАЛІЗОВАНИЙ МЕСЕНДЖЕР, МОБІЛЬНА МЕРЕЖА, КОМП'ЮТЕРНА МЕРЕЖА.

Об'єктом розроблення є децентралізовані комунікаційні системи з нестатичною топологією.

Мета розроблення – підвищення ефективності доставки повідомлень у децентралізованих мережах з нестабільною топологією шляхом застосування адаптивного алгоритму маршрутизації у прототипі месенджера.

Предмет дослідження – алгоритми адаптивної маршрутизації для підвищення надійності та ефективності доставки повідомлень в умовах децентралізованих мереж з нестабільною топологією.

У дипломному проєкті розроблена система обміну повідомленнями у децентралізованій мережі, а саме: алгоритми маршрутизації повідомлень із врахуванням нестабільної топології, система встановлення та підтримування однорангової децентралізованої мережі, а також підсистема обміну текстовими повідомленнями. Проведений аналіз та вибір алгоритмів маршрутизації для вирішення задач маршрутизації повідомлень. Виконане порівняння ефективності двох підходів у маршрутизації з урахуванням змінюваності топології і зв'язків, вибору методу побудови мережі для злагодженої роботи системи.

Отримані результати можуть бути корисними при реалізації проєктів з розподілених систем, організації їх комунікації.

SUMMARY

Decentralized messenger with adaptive routing in unstable networks.

The project contains 83 pages. text, 25 figures, 1 table, links to 31 literary sources, 2 annexes and 4 design documents.

Keywords: decentralized network, dynamic network, decentralized routing algorithms, decentralized messenger, mobile network, computer network.

The objective of the development is to enhance the efficiency of message delivery in decentralized networks with unstable topology through the implementation of an adaptive routing algorithm in a messenger prototype.

The subject of research is adaptive routing algorithms aimed at improving the reliability and efficiency of message delivery in decentralized networks with unstable topology.

Within the framework of this thesis project, a messaging system for decentralized networks has been developed. Specifically, it comprises: routing algorithms adapted to dynamic topologies, a system for establishing and maintaining a peer-to-peer decentralized network, as well as a subsystem for textual message exchange. An analysis and selection of routing algorithms for solving message routing tasks has been conducted. A comparative evaluation of the efficiency of two routing approaches was performed, taking into account topology dynamics and connection volatility, as well as the selection of an appropriate network construction method to ensure coherent system operation.

The obtained results may be beneficial for the implementation of distributed systems and the organization of their communication infrastructure.

№ рядка	Формат	Позначення	Найменування	Кіл. аркушів	№ екз.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІС13.080БАК.005 ПЗ	Децентралізований месенджер з	83		
6			адаптивною маршрутизацією у			
7			нестабільних мережах.			
8			Пояснювальна записка			
9	А3	ІС13.080БАК.005 Д1	Децентралізований месенджер з	1		
10			адаптивною маршрутизацією у			
11			нестабільних мережах.			
12			Діаграма компонентів.			
13	А3	ІС13.080БАК.005 Д2	Децентралізований месенджер з	1		
14			адаптивною маршрутизацією у			
15			нестабільних мережах.			
16			Діаграма послідовностей.			
17	А3	ІС13.080БАК.005 Д3	Децентралізований месенджер з	1		
18			адаптивною маршрутизацією у			
19			нестабільних мережах.			
20			Діаграма класів.			
21	А3	ІС13.080БАК.005 Д4	Децентралізований месенджер з	1		
22			адаптивною маршрутизацією у			
23			нестабільних мережах.			
24			Блок-схеми алгоритмів.			
25						
26						
27						
28						

Зм.	Лист	№ докум.	Підпис		ІС13.080БАК.005 ТП		
Розробив		Коноваленко			Децентралізований месенджер з адаптивною маршрутизацією у нестабільних мережах. Відомість дипломного проєкту	Літ.	Арк. 1 / Аркушів 1
Перевірив		Писаренко				Т	
						КПІ ім. Ігоря Сікорського Група ІС-13	
Затв.							

Пояснювальна записка
до дипломного проєкту
на тему: «Децентралізований месенджер з адаптивною маршрутизацією у нестабільних мережах»

Київ – 2025 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

MANET (Mobile Ad-hoc Network) – клас децентралізованих бездротових мереж із динамічною топологією, в яких вузли автономно встановлюють з'єднання між собою без використання фіксованої інфраструктури.

VANET (Vehicular Ad-hoc Network) – підклас MANET, що складається з рухомих транспортних засобів як вузлів мережі, з використанням V2V (vehicle-to-vehicle) та V2I (vehicle-to-infrastructure) комунікацій.

DTN (Delay-Tolerant Network) – мережа з високою затримкою або частими розривами зв'язку, у якій передача даних відбувається за принципом “зберігай–перенось–пересилай” (store-carry-forward).

P2P (Peer-to-Peer) – архітектурний підхід, за якого кожен вузол виконує функції клієнта і сервера, забезпечуючи обмін даними без централізованого посередника.

Mesh-мережа – тип топології, де кожен вузол може напряму зв’язуватись з кількома іншими вузлами, що забезпечує багатошляховість і підвищену відмовостійкість.

Epidemic Routing – алгоритм маршрутизації, який передбачає багатоадресне дублювання повідомлень усім доступним вузлам для забезпечення максимальної ймовірності доставки у DTN-середовищах.

PRoPHET (Probabilistic Routing Protocol using History of Encounters and Transitivity) – ймовірнісний протокол маршрутизації, що використовує історію контактів вузлів для прогнозування ймовірності доставки.

TTL (Time-To-Live) – параметр, що визначає граничну тривалість або кількість передач для повідомлення, після чого воно вважається недійсним.

MVP (Minimum Viable Product) – мінімально життєздатна версія продукту, яка реалізує основну функціональність та використовується для первинної перевірки концепції.

HeuristicRouter – реалізація маршрутизатора, що використовує евристичні критерії вибору вузлів для передачі повідомлень з урахуванням історії контактів і поточного стану вузлів.

Trust Table – таблиця локальної довіри до сусідів, яка формується на основі статистики успішних/невдалих передач для покращення адаптивної маршрутизації.

MessageMetadata – структура метаданих повідомлення, яка містить поля TTL, унікальний ідентифікатор, час створення, відправника і отримувача.

Store-Carry-Forward – принцип маршрутизації у DTN, за яким вузол зберігає повідомлення до моменту появи можливості передачі іншому вузлу.

Summary Vector – структура, що містить ідентифікатори вже отриманих повідомлень; використовується для уникнення дублювання в алгоритмах епідемічної маршрутизації.

ВСТУП

У сучасних умовах розвитку мобільних обчислювальних систем та збільшення кількості інтелектуальних пристроїв, що функціонують у складі розподілених обчислювальних середовищ, виникає потреба у розробці нових мережевих рішень, здатних забезпечити безперервну, надійну і адаптивну маршрутизацію інформаційних потоків. Розвиток рішень маршрутизації у централізованих та статичних мережах є значним завдяки попиту, адже вони інтегруються в інфраструктуру світової мережі Інтернет. Водночас, рішень для децентралізованих та динамічних мереж є відносно небагато, оскільки вони використовуються у рідкісних і специфічних випадках [6, 7].

У контексті децентралізованих середовищ, де мережа формується на основі тимчасових, спонтанних і часто нестабільних з'єднань між пристроями через різноманітні технології (Wi-Fi Direct, Bluetooth LE, LoRa, ZigBee), важливо вирішити комплекс задач, пов'язаний із підтримкою зв'язності, забезпеченням доставки повідомлень та мінімізацією навантаження на обмежені ресурси. Такі умови притаманні не лише звичайним сценаріям «інтернету речей», але й надзвичайним ситуаціям: польовим операціям, бойовим діям, рятувальним місіям, де відсутність стандартної централізованої інфраструктури зв'язку не дозволяє застосувати класичні мережеві рішення [8, 9].

Маршрутизація в мобільних MANET мережах залишається складною науково-технічною проблемою з огляду на потребу в швидкій адаптації до змін у топології та обмежену пропускну здатність. Традиційні протоколи, такі як Destination-Sequenced Distance-Vector (DSDV) [2], Ad hoc On-Demand Distance Vector (AODV) [3], Dynamic Source Routing (DSR) [4], розроблялися з урахуванням стабільніших каналів передачі даних, що вже на етапі тестування показали свої обмеження у високодинамічних сценаріях. Наприклад, DSDV, як проактивний протокол, генерує значний обсяг службового трафіку, що призводить до перевантаження каналів і затримок у мережі з низькою пропускною здатністю.

AODV і DSR, хоч і є реактивними, демонструють зростання часу затримки у разі зміни топології або при виникненні частих розривів маршрутів. Це підтверджується низкою емпіричних досліджень, що виявили зниження ефективності цих підходів у мережах з високою динамікою [5].

Одним із напрямів подолання вказаних обмежень є застосування біоінспірованих методів, що використовують аналогії з поведінкою природних систем. Особливе місце серед них займають алгоритми колонії мурах (Ant Colony Optimization, ACO), які завдяки децентралізованому збору інформації та адаптації до зміни середовища дозволяють формувати маршрути з урахуванням історії переміщень, якості зв'язку та поточної ситуації в мережі. Концепція AntNet полягає у тому, що мобільні агенти, аналогічні розвідникам у колонії мурах, переміщуються мережею, збираючи інформацію про затримки, пропускну здатність і загальну ефективність маршрутів [1]. Результати свідчать про зменшення службового трафіку і стабільність маршрутів при постійній зміні топології. Крім того, система зберігає адаптивність без потреби централізованої координації – властивість, яка важлива у динамічному бездротовому середовищі.

Іншим підходом, який знаходить застосування у вузькосмугових та енергозалежних мережах, є кероване флудування (Managed Flooding). На відміну від класичного флудування, яке спричиняє експоненційне зростання кількості пакетів у мережі, керовані схеми передбачають введення механізмів TTL (time-to-live), фільтрації дублікатів, обмеження за джерелом тощо. Ці заходи дозволяють зменшити кількість службових повідомлень, зберігаючи при цьому високу ймовірність доставки повідомлення до всіх потенційних вузлів. За правильного налаштування параметрів TTL та локальної буферизації, кероване флудування може демонструвати ефективність, співмірну з реактивними протоколами, навіть у складних польових умовах [5, 7]. Саме тому сучасні проєкти на зразок Meshtastic [8], goTenna [9], Briar [10] використовують гібридні методики, що поєднують елементи флудування з контекстно-адаптивною маршрутизацією.

Мета цього проєкту полягає у розробленні системи комунікації у децентралізованій мережі в умовах нестабільності мережі та змінюваності топології.

Для досягнення мети вирішені наступні задачі:

– охарактеризована очікувана поведінка мережі за визначених обставин;

– сформовані вимоги до методів маршрутизації повідомлень;

– визначена та описана топологія мережі, яку має утворювати система;

– спроєктовані два методи маршрутизації повідомлень для системи;

– проаналізована та порівняна ефективність розроблених методів;

– побудовані графіки результатів та зроблені висновки.

Об’єктом розроблення є децентралізовані комунікаційні системи з нестатичною топологією.

Предмет дослідження – алгоритми адаптивної маршрутизації для підвищення надійності та ефективності доставки повідомлень в умовах децентралізованих мереж з нестабільною топологією.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасні децентралізовані комунікаційні системи

Децентралізовані комунікаційні системи – це мережі, що функціонують без централізованої інфраструктури або єдиного керуючого вузла. У таких системах кожен вузол може виступати як джерелом, так і ретранслятором даних, а взаємодія здійснюється на основі рівноправності вузлів. Класичним прикладом є мобільні ad hoc-мережі (MANET – Mobile Ad Hoc Network), у яких мережа формується динамічно самими пристроями без попередньо налаштованих маршрутизаторів чи серверів. Подібні мережі набули актуальності завдяки застосуванням у надзвичайних ситуаціях, військових операціях та системах Інтернету речей (IoT), де відсутня можливість використання стаціонарної інфраструктури. До сучасних децентралізованих мереж також відносяться:

– бездротові ad hoc-мережі;

– бездротові mesh-мережі;

– peer-to-peer (P2P) мережі;

– мережі із затримками (DTN) та опортуністичні мережі.

Ad hoc-мережі є повністю децентралізованими бездротовими мережами, в яких вузли (пристрої) автономно встановлюють з’єднання один з одним. Такі мережі характеризуються динамічною топологією (вузли можуть довільно переміщуватися), гнучкістю і самоконфігурованістю. Відсутність центру керування означає, що маршрути прокладаються колективно самими вузлами без надійного центрального вузла. До цієї категорії належать MANET мережі, а також їх спеціалізації – зокрема, VANET мережі, FANET мережі та бойові мережі. Такі мережі здатні забезпечити зв’язок в польових умовах: під час стихійного лиха, в бойових діях або в сільській місцевості, де відсутня телекомунікаційна інфраструктура.

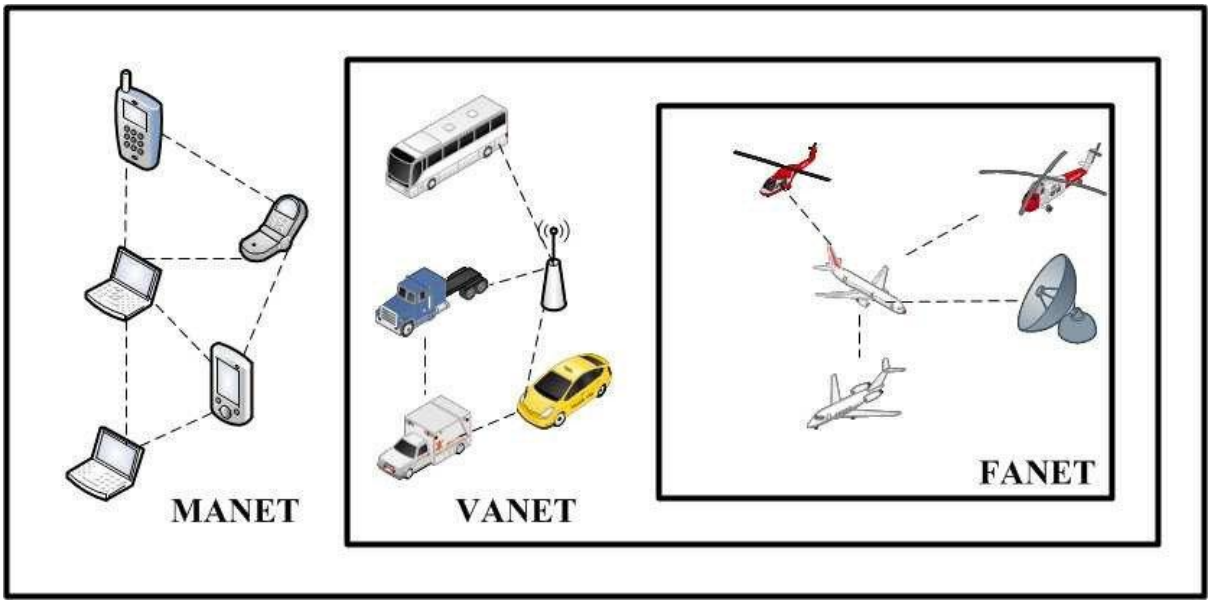

Рисунок 1.1 – Схематичне порівняння MANET, VANET і FANET мереж [31]

Багатохопові mesh-мережі – приклад децентралізованих систем, у яких кожен вузол (наприклад, Wi-Fi маршрутизатор або мобільний пристрій) діє як рівноправний ретранслятор. Відомі проєкти на кшталт Freifunk або mesh-мережі для розумних міст використовують протоколи на зразок OLSR або B.A.T.M.A.N. для розподіленої маршрутизації без центрального маршрутизатора. Mesh-мережі забезпечують стійкість до відмов окремих вузлів і автоматичне перепланування маршрутів при зміні топології.

Децентралізація характерна не лише для бездротових мереж – сучасні P2P-системи в Інтернеті (наприклад, файлообмінні мережі типу BitTorrent або розподілені системи на кшталт IPFS та блокчейни) теж не мають єдиного центру. В них вузли напряму взаємодіють один з одним для обміну даними і побудови маршрутів у підмережі всередині Інтернету. Так, у мережі BitTorrent використовується розподілена hash-таблиця Kademlia для маршрутизації запитів до вузлів, що зберігають потрібні ресурси. Протокол Kademlia став стандартним рішенням для багатьох децентралізованих P2P-мереж завдяки ефективному масштабованому алгоритму маршрутизації на основі XOR-відстані між ідентифікаторами вузлів. Хоча P2P-мережі фізично спираються на існуючу

інтернет-інфраструктуру, їх логічна топологія є децентралізованою і динамічно змінюваною (вузли можуть в будь-який час приєднуватися або виходити, що нагадує “нестатичну” поведінку мережі).

Мережі із затримками (DTN) та опортуністичні мережі – це системи, де постійне end-to-end з’єднання не гарантоване. Вузли можуть епізодично з’єднуватися один з одним (наприклад, супутникові мережі або сенсорні мережі в екстремальних середовищах). Маршрутизація тут також децентралізована: дані передаються “естафетно”, коли виникає можливість зв’язку, а збереження повідомлень у проміжних вузлах (store-carry-forward) компенсує розриви з’єднання. Хоч такі мережі і відрізняються від MANET за моделлю доставки, вони теж належать до децентралізованих систем, що не спираються на фіксовану інфраструктуру.

Децентралізовані нестатичні мережі набули значення завдяки своїй здатності забезпечувати зв’язок у ситуаціях, де традиційні мережі малопридатні, за рахунок гнучкості використання та високій відмовостійкості. Зростання кількості мобільних пристроїв, розвиток Інтернету речей та потреба у зв’язку в надзвичайних умовах підвищують актуальність досліджень таких мереж. Відсутність центру та динамічність топології, з одного боку, надають мережам гнучкості і стійкості до відмов (немає єдиної “точки відмови”), але з іншого – породжують складні науково-технічні проблеми, головна з яких – маршрутизація. Маршрутизація у децентралізованій мережі означає, що самі вузли колективно визначають шляхи доставки пакетів, адаптуючись до змін у топології.

1.2 Маршрутизація в нестабільних мережах

Маршрутизація в мережах з динамічно змінною топологією є складним завданням через низку обмежень і факторів, відсутніх у стаціонарних мережах. Далі наведено основні проблеми, що ускладнюють побудову ефективних маршрутів у децентралізованих нестатичних мережах.

Динамічність топології та часті розриви з’єднань є першою очевидною проблемою у нестатичних мережах. У рухомих ad hoc-мережах положення вузлів може змінюватися довільно і непередбачувано. Внаслідок цього встановлені маршрути швидко втрачають актуальність: лінії зв’язку перериваються, виникають нові сусідства між вузлами. Протокол маршрутизації повинен адаптуватися до таких змін, що потребує регулярного оновлення маршрутної інформації або швидкого відновлення шляхів. Забезпечити стійку доставку даних при частих перегрупуваннях мережі – одне з головних завдань. Наприклад, в MANET мережах кожен вузол може вийти з зони досяжності сусідів в будь-який момент, тому підтримання стабільного маршруту протягом довшого часу проблематичне.

У випадках коли використовуються подібні мережі часто обмежені в ресурсі вузлів таких як енергія, обчислювальна потужність, пам’ять. Багато децентралізованих вузлів – це мобільні пристрої або датчики, що живляться від батарей та акамуляторів. Кожне пересилання пакетів “чужого” трафіку витрачає енергію вузла-ретранслятора. Таким чином, маршрутизуючи трафік, вузол скорочує власний час автономної роботи. Це ставить вимоги енергоефективності до протоколів маршрутизації – мінімізувати зайві пересилання, зменшувати обсяг службових (сигнальних) повідомлень тощо. Крім того, обмеження пам’яті і обчислювальних потужностей впливають на складність алгоритмів: протоколи мають бути достатньо “легкими”, щоб працювати на малопотужних пристроях.

Також важливим фактором є обмежена пропускна здатність і надійність бездротового середовища. Бездротові канали зв’язку мають нижчу пропускну спроможність та більшу схильність до перешкод, ніж дротові. Якщо маршрутизувати пакети по багатьох хопах, сумарна пропускна здатність шляху падає, а затримки зростають. Часті колізії та втрата пакетів через радіоперешкоди можуть призводити до деградації роботи алгоритму маршрутизації (напр., багаторазове повторне відкриття маршрутів). Протоколи мають враховувати цю ненадійність каналу і, по можливості, обмежувати навантаження мережі службовим трафіком.

Відсутність централізованого управління також можна вважати за обмеження, адже в ситуації децентралізованих мереж задача підтримання мережі та маршрутизації пакетів є відповідальністю не лише одного спеціалізованого пристрою, а всіх пристроїв учасників. Маршрути повинні визначатися децентралізовано, тобто через обмін інформацією між вузлами-сусідами. Немає центрального маршрутизатора, який мав би глобальне бачення топології; кожен вузол “бачить” тільки локальних сусідів. Це ускладнює побудову оптимальних маршрутів і потребує розроблення алгоритмів розподіленого узгодження. Наприклад, проактивні протоколи розсилають інформацію про структуру мережі від кожного вузла до всіх інших, щоб кожен міг обчислити маршрути – але такий процес створює великий трафік і не масштабуються для великих мереж. Реактивні протоколи, навпаки, намагаються шукати маршрути “на вимогу” відправника, що зменшує зайве розповсюдження даних, але натомість призводить до затримок при встановленні нового з’єднання. Балансування між цими двома підходами – нетривіальна задача.

Навіть якщо успішно запустити мережеву систему, то постає питання підтримки якості обслуговування (QoS) у динамічних умовах. У багатьох застосуваннях важливо забезпечити достатню пропускну здатність, низькі затримки або надійність доставки (наприклад, для потокового відео чи критичних датчиків). В нестатичних мережах гарантувати QoS важче, оскільки маршрути, оптимальні на момент встановлення, можуть швидко деградувати. Протокол маршрутизації повинен або періодично перемаршрутизувати трафік з урахуванням актуального стану мережі, або підтримувати мультишляхові резервні маршрути, чи мати інші механізми забезпечення надійності (пере-передача, кодування тощо). Кожен з цих підходів, втім, підвищує складність і службовий трафік в мережі.

Перелічені проблеми означають, що побудова ефективної маршрутизації в децентралізованих нестатичних мережах – складне завдання, що залишається предметом активних досліджень. Як зазначено в роботі, динамічний характер MANET спричиняє постійні зміни мережевої структури, ускладнюючи

підтримання стабільного зв’язку; обмеження батареї та пропускної здатності лише доповнюють ці обмеження. На сьогодні запропоновано багато алгоритмів, кожен з яких намагається розв’язати зазначені проблеми по-своєму. Але не усі водночас, тому і стає важливим визначитись з найбільш та найменш пріоритетними проблемами, які має вирішувати алгоритм маршрутизації.

1.3 Порівняння алгоритмів маршрутизації

Алгоритми маршрутизації для ad hoc-мереж та інших нестатичних систем можна класифікувати за різними ознаками. Найбільш усталеною є класифікація на проактивні, реактивні та гібридні протоколи маршрутизації. Окремо виділяють категорію географічних (позиційних) протоколів, що використовують координати вузлів. Далі наведено стислий огляд найважливіших протоколів кожного типу, а у таблиці 1.1 представлено їх порівняння.

Проактивні протоколи (табличні). Протоколи цього класу підтримують актуальну інформацію про маршрути до всіх вузлів у кожному вузлі, незалежно від того, потрібні ці маршрути в даний момент чи ні. Вони періодично розсилають службові пакети мережою для оновлення маршрутних таблиць. Завдяки цьому, маршрут до будь-якого вузла відомий заздалегідь, що мінімізує затримки при передачі пакета – шлях вже зберігається в таблиці. Класичним прикладом є DSDV (Destination-Sequenced Distance-Vector) – один з перших алгоритмів для MANET, запропонований Перкінсом і Бхагватом у 1994 році [2]. Протокол DSDV базується на класичному дистанційно-векторному алгоритмі Беллмана-Форда, але вводить нумерацію маршрутів (sequence numbers) для усунення петель і неузгодженостей. Кожен вузол DSDV періодично розсилає свій список досягненних адрес із вказанням відстані та порядкового номера; інші вузли оновлюють відповідно свої таблиці. Перевага DSDV – простота і гарантія актуальності маршрутів (за умови достатньої частоти оновлень), недолік – відносно великий обсяг службового

трафіку навіть тоді, коли мережевий трафік відсутній або топологія не змінюється часто.

Ще один відомий проактивний протокол – OLSR (Optimized Link State Routing), стандартизований IETF у 2003 році (RFC 3626) [11]. OLSR є оптимізованим варіантом алгоритму стану зв'язків (link state) для бездротових ad hoc-мереж. Головне нововведення OLSR – використання мульти-точкових ретрансляторів (MPRs), спеціально обраних вузлів, які розсилають топологічну інформацію замість всіх вузлів. Це значно скорочує обсяг широкомовних повідомлень у порівнянні з класичним алгоритмом Лінка-Стейта, де кожен вузол "флудує" всю мережу. В OLSR кожен вузол через MPR-ретранслятори повідомляє про сусідів, що його обрали, а не про всі з'єднання. Таким чином досягається менший службовий трафік, зберігаючи при цьому актуальність маршрутів. OLSR забезпечує дуже малі затримки доставки (маршрут відомий одразу) і оптимальні за кількістю хопів шляхи, що робить його ефективним у відносно стабільних мережах. Обмеженням OLSR, як і інших проактивних протоколів, є зниження масштабованості: у великих або дуже рухомих мережах часті оновлення таблиць створюють значне навантаження і можуть перевантажувати канал.

На противагу проактивним протоколам існують реактивні протоколи. Протоколи цього типу діють протилежно проактивним: вони не зберігають постійно маршрути, а натомість знаходять маршрут лише тоді, коли вузол має дані для відправлення до певного призначення. Тобто маршрут встановлюється "на вимогу" у момент передачі, а не підтримується наперед. Такий підхід значно зменшує обсяг службових повідомлень в мережі, особливо коли трафік низький або нерегулярний, адже немає потреби постійно розсилати оновлення. Натомість недоліком є початкова затримка: перший пакет мусить зачекати, поки протокол виявить маршрут до адресата. Одним із найпоширеніших реактивних алгоритмів є AODV (Ad Hoc On-Demand Distance Vector), запропонований Ч. Перкінсом та співавторами [3]. AODV знаходить маршрут шляхом розповсюдження запиту (RREQ) по мережі: запит широкомовно передається від вузла до вузла, поки не

досягне потрібного адресата; той у відповідь надсилає підтвердження (RREP) назад уздовж знайденого шляху. Така схема дозволяє вузлам не тримати глобальних таблиць – інформація збирається динамічно. AODV успішно адаптується до високої мобільності, оскільки у разі розриву існуючого маршруту він може за потреби знову виконати пошук і відновити шлях. Протокол не використовує надмірного трафіку у стабільні моменти, що вигідно для енергозбереження. Недоліки AODV: можливі затримки при частих пошуках та певний обсяг широкомовних повідомлень під час фаз Route Request.

Іншим реактивним протоколом є DSR (Dynamic Source Routing), розроблений Д. Джонсоном і співавторами [4]. DSR застосовує принцип маршрутизації з явним джерелом: кожен пакет містить повний список вузлів від відправника до отримувача. Маршрут також шукається через розсилку запиту, але особливість DSR – вузли кешують знайдені маршрути для можливого подальшого використання. Якщо якийсь вузол вже знає маршрут до потрібного призначення (напряму або через кеш), він може відповісти замість адресата і доставити інформацію швидше. Перевага DSR – відсутність необхідності в періодичних повідомленнях: якщо трафіку нема, протокол не генерує навантаження. Крім того, використання кешу може зменшити кількість запитів при повторних відправленнях до тих самих вузлів. До обмежень DSR відносять: зростання розміру заголовку пакетів у міру подовження маршруту (що збільшує службовий трафік) та потенційні проблеми узгодженості кешованих даних при швидких змінах топології (можливе використання застарілого маршруту). Загалом AODV та DSR показують кращу ефективність у динамічних сценаріях, ніж проактивні протоколи, особливо у мережах з відносно невеликим числом одночасних з’єднань.

Слід згадати також розширення й варіації реактивних підходів. Наприклад, AOMDV (Ad Hoc OnDemand Multipath Distance Vector) розвиває ідеї AODV, знаходячи одразу кілька альтернативних маршрутів для підвищення живучості мережі; TORA (Temporally-Ordered Routing Algorithm) застосовує алгоритм зворотних зв’язків (link reversal) для швидкої реакції на зміни топології тощо.

Гібридні протоколи є певним балансом між особливостями проактивних та реактивних підходів. Жоден з підходів – ані проактивний, ані реактивний – не є універсально оптимальним для всіх сценаріїв. Тому розроблені гібридні алгоритми, що поєднують переваги обох схем. Ідея типового гібридного протоколу полягає в застосуванні проактивної маршрутизації на близьких відстанях (локально) і реактивного пошуку для далеких вузлів. Таким чином, досягається компроміс: у межах невеликого околу маршрути відомі наперед, а при зверненні до далеких сегментів мережі протокол уникає глобального flood-оновлення, користуючись запитом на вимогу. Найвідоміший приклад – Zone Routing Protocol (ZRP), запропонований З. Хаасом та співавторами [12]. ZRP ділить мережу на зони визначеного радіусу (наприклад, k хопів): всередині кожної зони вузол підтримує маршрути до всіх інших вузлів цієї зони проактивно, а для зв'язку з вузлами поза зоною використовує реактивний запит через прикордонні вузли зони. Правильно підібраний радіус зони дозволяє знизити обсяг повідомлень у порівнянні з чисто проактивним підходом, одночасно уникнувши великих затримок для близьких сусідів. Недоліком гібридних протоколів є ускладнення алгоритму (потрібно керувати двома режимами одночасно) та необхідність оптимального налаштування параметрів (наприклад, розміру зони у ZRP) під конкретну мережу.

Окремо варто зазначити протоколи, що використовують географічну інформацію. До гібридних інколи відносять позиційні алгоритми, хоча за принципом роботи вони відносяться до окремої категорії. Географічні протоколи, такі як GPSR (Greedy Perimeter Stateless Routing) [6], спираються на координати вузлів, отримані через GPS або інші засоби, для прийняття рішень про пересилання. Ідея GPSR полягає в тому, що пакет на кожному кроці відправляється на вузол, який найближчий до цільового місця призначення (жадібна передача). Якщо ж до цілі напряму рухатися неможливо (наприклад, вузол опинився в "ямі" без сусідів ближче до адресата), протокол перемикається в режим обходу периметру, щоб знайти шлях навколо "ями". Географічні алгоритми не підтримують глобальних маршрутних таблиць – рішення приймаються локально на основі положення

сусідів і цілі. Це дає високу масштабованість та швидку реакцію на зміни: при переміщенні вузлів нові маршрути обчислюються миттєво на основі оновлених координат, без глобального перебудування таблиць. Однак, очевидними обмеженнями є вимога знати місцеположення всіх вузлів (потребує наявності GPS або іншої системи координат) та проблеми на кшталт ситуацій, коли жадібний підхід заходить у глухий кут (необхідні додаткові алгоритми обходу перешкод). Позиційні протоколи добре підходять для мереж типу VANET, де вузли (автомобілі) здебільшого знають свої координати і можуть ними обмінюватися.

Проаналізувавши всі вищезазначені приклади, найважливіші характеристики розглянутих протоколів зведені у таблиці 1.1. У ній наведено тип протоколу, використовуваний підхід до маршрутизації та основні сильні/слабкі сторони:

Проактивні протоколи забезпечують найменшу затримку, але генерують постійний службовий трафік. Реактивні – економлять ресурси у спокійні періоди, проте додають затримку при встановленні з’єднання. Гібридні та географічні протоколи намагаються подолати ці обмеження шляхом комбінування підходів або використання додаткової інформації (позиційної) для прийняття рішень.

Таблиця 1.1 – Основні протоколи маршрутизації для децентралізованих нестатичних мереж та їх особливості

Протокол	Категорія	Переваги	Недоліки
DSDV	Проактивний (Distance-Vector)	Миттєва готовність маршрутів	Високий службовий трафік
OLSR	Проактивний (Link-State)	Зменшує flood, забезпечує низькі затримки	Складний та затратний у великих мережах
AODV	Реактивний (Distance-Vector on-demand)	Не потребує постійного оновлення таблиць	Має затримки при встановленні шляху

DSR	Реактивний (Source Routing)	Відсутність таблиць у проміжних вузлах	Зростання заголовка при довгих маршрутах
ZRP	Гібридний (Zone-based)	Комбінує переваги проактивних та реактивних, гарно маштабується	Потребує тонкого налаштування параметрів
GPSR	Географічний (позиційний)	Не потребує таблиць, швидкий	Залежить від точності позиційних даних

Висновки до розділу 1

У цьому розділі проведений огляд сучасних децентралізованих комунікаційних систем з нестатичною топологією, проаналізовані основні проблеми маршрутизації в таких мережах та розглянуті основні алгоритми маршрутизації. Децентралізовані нестатичні мережі (MANET, mesh-мережі, P2P та опортуністичних системи) є актуальними для застосувань, що потребують автономного мережевого взаємозв’язку без фіксованої інфраструктури. Вони характеризуються динамічністю, гнучкістю і відмовостійкістю, проте динамічна природа топології й відсутність центру породжують значні обмеження для забезпечення ефективної маршрутизації трафіку. Після ознайомлення з проблемою та всіх її аспектів можна прийти до висновку, що в умовах частих змін структури мережі, обмежених енергетичних та пропускних ресурсів вузлів, а також за відсутності глобального координатора, розроблення маршрутних протоколів вимагає нетривіальних і, зазвичай, комбінованих рішень.

Основними підходами до маршрутизації в таких системах є проактивна, реактивна та гібридна схеми, кожна з яких має свої переваги та недоліки. Проактивні протоколи забезпечують швидкий доступ до маршрутів і низькі затримки, проте страждають від великого службового навантаження, особливо в

великих або високодинамічних мережах. Реактивні протоколи зменшують зайвий трафік, виконуючи встановлення маршрутів “на вимогу”, що робить їх більш масштабованими і енергоощадними для великого числа вузлів. Їхній головний недолік – початкова затримка пошуку шляху і потенційна нестабільність, якщо мережа дуже швидко змінюється (маршрут може не встигнути “побудуватись”, як вже стане неактуальним). Гібридні рішення прагнуть об’єднати сильні сторони обох підходів, але потребують ретельного налаштування і складні в реалізації. Географічні алгоритми демонструють, що залучення зовнішньої інформації (координат) дозволяє ефективно маршрутизувати без зберігання повного стану мережі, проте такі алгоритми застосовні тільки за умов можливості отримання й використання позиційних даних вузлів.

Таким чином, на сьогодні не існує універсального протоколу, що однаково добре підходив би для всіх сценаріїв децентралізованих нестатичних мереж. Вибір алгоритму завжди є компромісом між вимогами до швидкості реакції на зміни, службовими витратами, масштабованістю та енергоефективністю. Проте в результаті активних досліджень в цій галузі створені гібридні підходи які оптимізовані під конкретні задачі. Це може свідчити, що найбільш ймовірно найкращим рішенням для конкретного випадку мережі і вимог до неї буде створити вузькооптимізований алгоритм під конкретну задачу.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Розроблювана інформаційна система призначена для забезпечення обміну повідомленнями між окремими пристроями, що формують децентралізовану динамічну мережу. Основною задачею є забезпечення надійної доставки повідомлень за допомогою спеціального агенту маршрутизації, здатного працювати в умовах відсутності централізованої інфраструктури. Кожен вузол системи реалізує базовий функціонал передачі й прийому повідомлень, а також інтегрує модуль маршрутизації, що самостійно приймає рішення про подальшу передачу пакетів залежно від поточної топології мережі.

2.1 Вимоги до системи

Проєктована інформаційна система призначена для функціонування у складі динамічної одноранговой мережі, що складається з декількох незалежних вузлів. Кожен вузол реалізує базовий обмін повідомленнями через середовище передачі даних із змінною пропускною здатністю, зокрема на основі технологій Bluetooth, Wi-Fi Direct або інших нестабільних бездротових каналів зв’язку. Центральним компонентом системи є модуль маршрутизації, який забезпечує прийняття рішень щодо вибору шляху доставки повідомлення у разі відсутності сталого маршруту.

Функціональні вимоги:

а) передача повідомлень між вузлами:

1) кожен користувач повинен мати можливість надіслати повідомлення, вказавши ідентифікатор отримувача (ID) та текст повідомлення;

2) повідомлення мають зберігатися у черзі передачі у разі відсутності прямого підключення до отримувача;

б) модуль маршрутизації повідомлень:

1) реалізація маршрутизатора, що приймає рішення про перенаправлення пакету на основі локальної таблиці маршрутів, часових міток, статистики з'єднань або евристик (залежно від вибраного алгоритму);

2) підтримка адаптації до змін у топології мережі в реальному часі;

в) самоідентифікація вузла:

1) кожен пристрій повинен мати змогу встановити власний ідентифікатор (у вигляді символьного або числового значення), який використовується для адресації в межах системи;

2) кожен ідентифікатор користувача у мережі має бути унікальним;

г) журналювання подій:

1) логування подій маршрутизації, зміни топології та відправки/отримання повідомлень для налагодження і дослідження.

Нефункціональні вимоги:

а) надійність:

1) збереження повідомлень до успішної доставки або до завершення TTL (time-to-live);

2) повідомлення не повинні бути втрачені при частковому збої з'єднання;

б) масштабованість:

1) система повинна підтримувати функціонування у мережах від 3 до 50 вузлів без деградації продуктивності;

в) реалізаційна незалежність:

1) архітектура повинна дозволяти реалізацію на різних пристроях з різними можливостями (наприклад, мікроконтролери ESP32, ПК або мобільні пристрої);

г) продуктивність:

1) час обробки маршруту повідомлення не повинен перевищувати 1000 мс при наявності зв’язку;

д) простота реалізації:

1) обмежена кількість UI-елементів, простий протокол міжвузлового обміну, що зменшує службові витрати.

2.2 Архітектура системи

2.2.1 Логічна структура

Логічна структура проєктованої інформаційної системи ґрунтується на моделі однорангової динамічної мережі (peer-to-peer dynamic network), у якій кожен вузол є автономною сутністю, здатною приймати, обробляти, пересилати або ініціювати передачу повідомлень. Всі вузли функціонують на основі єдиного програмного ядра, що поєднує модулі інтерфейсу, маршрутизації, комунікації та зберігання.

Кожен вузол містить такі компоненти:

а) користувацький інтерфейс (UI) забезпечує взаємодію користувача з системою. Складається з таких елементів:

1) поле введення ідентифікатора отримувача – дозволяє користувачу вказати однозначний ID цільового вузла;

2) поле введення повідомлення – текстове поле довільної довжини;

3) поле введення власного ідентифікатора вузла – використовується під час першого запуску;

4) інтерфейс історії повідомлень – перегляд вхідних/вихідних повідомлень з часовими мітками. Інтерфейс не зберігає повідомлення постійно, лише відображає їх, отримуючи дані з журналу;

б) модуль черги повідомлень – відповідає за буферизацію вихідних повідомлень, які не можуть бути негайно передані. Черга підтримує:

1) мітку часу створення;

2) ідентифікатор отримувача;

3) час життя повідомлення (TTL);

4) статус (нове / в обробці / доставлено / втрачено);

5) повідомлення можуть залишатися в черзі до моменту появи відповідного каналу або завершення TTL;

в) модуль маршрутизації – компонент логіки мережі. Його функції включають:

1) аналіз топологічної інформації (сусіди, попередній досвід передачі);

2) вибір наступного вузла для пересилки на основі евристик або алгоритму (наприклад, epidemic routing, spray-and-wait, PROPHET, delay-tolerant routing);

3) оновлення локальної таблиці маршрутів (за потреби);

4) визначення умов повторної передачі або відмови;

5) цей компонент є модульним і може підтримувати зміну алгоритму маршрутизації без зміни всієї системи;

г) комунікаційний модуль – здійснює передачу даних на фізичному рівні. Він забезпечує встановлення каналів зв'язку з іншими вузлами; приймання пакетів і передавання їх маршрутизатору; обробку сесій та підтвердження доставки. Він є єдиним компонентом, що взаємодіє із зовнішніми пристроями. Працює з низькорівневими API інтерфейсів типу:

1) Bluetooth/BLE (наприклад, у режимі broadcast та GATT-серверів);

2) Wi-Fi Direct / SoftAP;

3) Serial-over-USB або інші варіанти;

д) Модуль зберігання (Storage / Logs). Цей модуль веде постійне збереження:

1) журналів подій (часові мітки, спроби передачі, маршрути);

2) історії повідомлень (вхідні/вихідні з результатом доставки);

3) кешу маршрутної інформації (наприклад, статистика затримок, активність вузлів);

4) використовується також для відображення історії у UI.

Для візуального представлення цієї логічної структури розроблено діаграму компонентів яка продемонстрована на кресленику ІС13.080БАК.005 Д1, яка

показує взаємозв’язки між модулями одного вузла інформаційної системи. На діаграмі компонентів зображена схема взаємодії компонентів у системи:

– користувач взаємодіє з інтерфейсом, вводячи повідомлення;

– інтерфейс передає його у чергу повідомлень, яка зберігає повідомлення до моменту обробки;

– модуль маршрутизації перевіряє стан мережі й передає повідомлення у комунікаційний модуль;

– якщо пакет надходить ззовні через “Комунікаційний модуль”, він передається в “Модуль маршрутизації”, який вирішує, чи повідомлення слід доставити локально чи переслати далі;

– модуль “Лог“ зберігає повний журнал повідомлень, який доступний для користувача через користувацький інтерфейс.

2.2.2 Функціональна структура

Функціональна модель інформаційної системи описує динамічну поведінку окремих компонентів при виконанні сценаріїв використання. Основна функціональність системи полягає у формуванні, передачі, маршрутизації та доставці повідомлень між вузлами у динамічному мережевому середовищі. Нижче подано опис основних процесів функціонування, з урахуванням затримок, втрат з’єднання та повторної передачі.

Сформовані три сценарії які описують очікувану поведінку від інформаційної системи у масштабі одного вузла. Ці сценарії відповідають опису процесу взаємодії компонентів продемонстрованому на діаграмі послідовності на кресленику ІС13.080БАК.005 Д2.

Сценарій 1 – надсилання повідомлення локальним користувачем:

– користувач запускає застосунок і задає власний ідентифікатор, якщо це перший запуск;

– користувач вводить ідентифікатор отримувача та текст повідомлення;

– повідомлення передається з інтерфейсу до черги повідомлень;

– маршрутизатор ініціює перевірку локальної маршрутної таблиці;

– якщо з’єднання з отримувачем наявне, повідомлення передається миттєво через комунікаційний модуль;

– якщо ні – маршрутизатор обирає інший сусідній вузол для передачі пакета з метою подальшої ретрансляції;

– всі дії логуються (створення повідомлення, спроба передачі, успіх/невдача).

Сценарій 2 – приймання повідомлення вузлом:

– комунікаційний модуль приймає пакет від іншого вузла;

– пакет передається маршрутизатору;

– якщо поточний вузол є отримувачем – повідомлення записується до журналу та передається UI;

– якщо ні – перевіряється TTL та контекст мережі. За потреби – повідомлення повторно додається в чергу для подальшої маршрутизації.

Сценарій 3 – релеєва пересилка повідомлення через проміжний вузол:

– проміжний вузол отримує пакет, який не призначений йому;

– маршрутизатор визначає, чи варто пересилати пакет далі;

– якщо так – повідомлення пересилається найбільш підходящому вузлу;

– у разі відсутності підходящих сусідів повідомлення зберігається до наступної можливості пересилки.

2.3 Вибір технологій

У процесі реалізації проєктованої інформаційної системи виникає низка обмежень, пов’язаних із фізичним розгортанням мережі на основі мобільних пристроїв. Повноцінне моделювання динамічної топології, з використанням реальних бездротових інтерфейсів (таких як Wi-Fi або Bluetooth), потребує наявності великої кількості фізичних вузлів, синхронізованого середовища та

постійного контролю за станом з’єднань, що істотно ускладнює процес розроблення та верифікації.

З огляду на це прийняте технічно доцільне рішення реалізувати комунікаційну модель у вигляді змодельованого віртуального мережевого шару, де кожен вузол взаємодіє з іншими через конфігуровані логічні агенти зв’язку, а сам обмін повідомленнями виконується за допомогою бібліотеки ZeroMQ. Такий підхід дозволяє моделювати довільну структуру мережі, задавати параметри маршрутизації, і динамічно змінювати топологію з мінімальними затратами ресурсів, зберігаючи при цьому модульність архітектури системи.

У якості основної мови реалізації обрана мова програмування Python [18]. Причинами такого вибору є висока швидкість розроблення, наявність широкого спектра бібліотек для обробки мережевих з’єднань, тестування, побудови користувацьких інтерфейсів та асинхронного програмування.

Також важливо, що Python має сильну підтримку середовищ тестування (pytest), асинхронних моделей (asyncio, trio) і інтерфейсів (streamlit, gradio), що дозволяє розглядати його як універсальну платформу для побудови модульних систем, зокрема у сфері мережевих експериментів [13].

Оскільки система є моделлю розподіленої мережі без фактичного використання фізичних каналів передачі даних (Wi-Fi, Bluetooth), необхідно впровадити віртуальний комунікаційний шар, що імітує маршрутизацію між вузлами.

У цьому контексті обрана ZeroMQ – високопродуктивна бібліотека обміну повідомленнями між процесами, яка надає абстракцію над сокетами та дозволяє будувати складні топології з мінімальними службовими витратами. Вона підтримує шаблони типу PAIR, PUB/SUB, REQ/REP, що дає змогу моделювати як симетричні, так і асиметричні канали зв’язку [14].

ZeroMQ також дозволяє використовувати один або кілька логічних процесів для моделювання вузлів, забезпечуючи легке масштабування, гнучкість у налаштуванні топології (наприклад, шляхом читання topology.json), та ізоляцію

логіки маршрутизації від фізичного рівня. Інтеграція з Python забезпечується через офіційну обгортку pyzmq, яка активно підтримується спільнотою [15].

Для реалізації простого, інтуїтивно зрозумілого інтерфейсу користувача, який дозволяє вводити ідентифікатор отримувача, текст повідомлення та переглядати історію комунікації обраний Streamlit — Python-фреймворк для швидкої побудови веб-інтерфейсів.

Streamlit дає змогу реалізувати повноцінний користувацький інтерфейс без необхідності застосування HTML/CSS/JS або сторонніх фреймворків. Згідно з офіційною документацією [16], створення прототипу форми з кількома полями займає менше ніж 20 рядків коду. Інструмент підтримує гаряче оновлення, роботу з session_state, асинхронні події, а також гнучке налаштування відображення логів, що є корисним при реалізації журналу повідомлень.

Таким чином, Streamlit обрано як найбільш збалансоване рішення між швидкістю реалізації, простотою розгортання і функціональністю, необхідною для MVP.

Для забезпечення контролю коректності функціонування системи, включно з маршрутизатором, чергами повідомлень та вхідними/вихідними інтерфейсами, обрана бібліотека pytest.

Pytest є сучасною системою автоматизованого тестування для Python, що дозволяє створювати модульні, параметризовані та інтеграційні тести з мінімальним обсягом шаблонного коду. Завдяки своїй архітектурі, pytest інтегрується із системами CI/CD, а також підтримує перевірку асинхронних функцій (через pytest-asyncio) та створення ізольованих тестів з використанням мок-об’єктів (unittest.mock) [17].

Крім того, простота запуску (pytest tests/) та зрозумілість повідомлень про помилки робить цей інструмент ефективним під час розроблення та рефакторингу.

Для прискорення розроблення і тестування на ранніх етапах не використовуються контейнерні технології типу Docker. Замість цього застосована структура автономних процесів Python, які запускаються зі спільною топологією,

визначеною у зовнішньому JSON-файлі. Такий підхід забезпечує максимальну прозорість запуску, швидке редагування сценаріїв і можливість інтерактивного налагодження.

У перспективі, для масштабування або перенесення на інші системи, система може бути розгорнута в Docker-контейнерах із використанням docker-compose, однак на стадії створення MVP це створює зайві службові витрати та уповільнює цикл розроблення.

2.4 Проєктування механізмів маршрутизації

2.4.1 Загальні вимоги до маршрутизації

У змодельованій одноранговій мережі, яка функціонує в умовах нестабільного або тимчасового з’єднання вузлів, маршрутизатор має забезпечувати гарантовану (по можливості) доставку повідомлень без наявності постійного глобального з’єднання. До вимог до механізмів маршрутизації належать:

– незалежність від фізичного каналу передачі;

– можливість роботи в асинхронному режимі;

– підтримка TTL (Time-to-Live) для кожного повідомлення;

– ведення локальної таблиці доставлених та втрачених пакетів;

– обмежене споживання ресурсів (буфер, обчислення);

– можливість роботи з динамічною топологією без централізованого контролера.

На основі проведеного аналізу сформульовані наступні пріоритети, яким має відповідати механізм маршрутизації в межах розроблюваної системи:

– гарантованість доставки повідомлень у межах топологічно доступної мережі.

– обмеження службового трафіку – система не повинна створювати надлишкове навантаження через дублювання або неконтрольоване поширення службових повідомлень;

– адаптивність до змін у топології та стійкість до збоїв – маршрутизатор має автоматично адаптуватися до зникнення, появи або переміщення вузлів у середовищі;

– напрямна ефективність – навіть за відсутності ідеального маршруту система має забезпечувати хоча б спрямований рух повідомлення в бік адресата.

Зважаючи на пріоритети перераховані вище, обрані два механізми маршрутизації:

– модифікований Epidemic Routing як класичну DTN-реалізацію;

– авторський гібридний евристичний алгоритм, що поєднує аналіз локальної взаємодії та структури спостережуваної топології.

2.4.2 Модифікований алгоритм Epidemic Routing

Epidemic Routing – протокол багатоадресного дублювання, розроблений для мереж з високою ймовірністю відсутності з'єднання у будь-який момент часу. При зустрічі двох вузлів вони обмінюються масивами ідентифікаторів повідомлень, що вже оброблені вузлом, та передають один одному лише ті, що сусід ще не отримував. Такий підхід забезпечує широке поширення повідомлень, навіть у динамічних мережах з обмеженими каналами зв'язку.

Особливості модифікованої реалізації:

– введення обмеження TTL (максимальної кількості пересилок);

– уникнення повторних передач через хеш-таблицю отриманих ID;

– обмеження буфер повідомлень;

– запис статистики успішних передач для подальшого аналізу.

Цей алгоритм використаний як референтна модель, що дозволяє досягти максимальної імовірності доставки при високих витратах. Його перевагою є відсутність потреби в глобальному знанні топології або параметрах з'єднання, однак ця універсальність досягається ціною надлишкових копій і швидкого заповнення буфера, що вимагає додаткових механізмів обмеження.

2.4.3 Гібридний евристичний алгоритм маршрутизації

Другим варіантом обрано гібридне рішення, створене шляхом інтеграції трьох підходів: обмеженого flooding, евристичної оцінки доцільності передачі та локального адаптивного навчання на основі історії спостережень. Цей механізм має на меті покращити ефективність використання мережевих ресурсів і зменшити кількість зайвих пересилок без втрати загальної надійності доставки.

Основні компоненти алгоритму:

– локальна таблиця довіри до сусідів – для кожного доступного вузла ведеться оцінка на основі статистики попередніх успішних та невдалих пересилок. Це дозволяє формувати динамічний рейтинг, який бере участь у прийнятті рішень.

– евристична функція оцінки корисності передачі – кожне повідомлення оцінюється з точки зору доцільності передачі конкретному сусіду з урахуванням його надійності та положення щодо цільового вузла. Цей елемент дозволяє уникнути безглуздого дублювання;

– механізм обмеження дублювання – застосовується локальне правило TTL та унікальна мітка повідомлення. Передача дозволена лише один раз кожному вузлу, окрім випадків суттєвого оновлення маршрутної інформації;

– пріоритетне обслуговування – у разі переповнення буфера повідомлення відсіюються згідно з метрикою, що враховує TTL та рейтинг вузла, через який має відбутися передача.

Завдяки цим елементам гібридний алгоритм забезпечує баланс між надійністю та ефективністю. Він не вимагає глобальної координації, однак дозволяє за рахунок локальної евристики зменшити обсяг переданих даних та підвищити якість маршрутизації в умовах змінної топології.

Висновки до розділу 2

У цьому розділі здійснене проєктування інформаційної системи для передачі повідомлень у динамічному розподіленому середовищі. Спочатку визначені функціональні та нефункціональні вимоги до системи, сформульовані основні принципи її побудови та описана логічна і функціональна структура. У результаті моделювання мережевого середовища із змінною топологією обрані технології, що дозволяють досягти максимальної гнучкості у розробці та верифікації системи: Python як мову програмування, ZeroMQ як механізм віртуального з’єднання між вузлами, Streamlit для інтерфейсу користувача, а також pytest для тестування модулів.

Особлива увага приділена проєктуванню механізмів маршрутизації. Розроблені два підходи: перший – модифікована версія Epidemic Routing, що дозволяє досягти максимальної гарантованості доставки; другий – гібридний алгоритм, що забезпечує адаптивність, економію ресурсів і ефективне реагування на зміни у мережевому середовищі. Обидва варіанти відповідають встановленим пріоритетам системи маршрутизації: надійність доставки, мінімізація службового трафіку, адаптація до змін та базова ефективність.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Моделювання динамічної топології мережі

3.1.1 Постановка задачі

Динамічна топологія мережі означає, що множина з'єднань між вузлами змінюється в часі: з'являються нові лінки при зближенні вузлів та зникають при розриві зв'язку. У відсутності централізованого контролера кожен вузол діє автономно, а передача повідомлень можлива навіть за відсутності одночасного прямого шляху між відправником і отримувачем, а саме за рахунок проміжного зберігання й ретрансляції (асинхронні «store-carry-forward» контакти). Задача полягає у побудові строгого математичного уявлення такої мережі, яке описує: множину вузлів та їх властивості; множину з'єднань (контактів) як функцію часу; змінність структури мережі (топології) у часовому вимірі; асинхронність зв'язків, тобто можливість відкладеної доставки даних.

Така модель має дозволити аналіз досяжності вузлів, затримок передачі та забезпечити основу для розроблення алгоритмів маршрутизації у середовищі з непостійними зв'язками.

3.1.2 Математична формалізація задачі

Для моделювання використовуємо апарат теорії графів із введенням часової залежності на ребрах. Нехай V – множина вузлів мережі. Динамічну топологію можна представити як темпоральний граф $G_T = (V, E_T)$, де $E_T \subseteq V \times V \times T$ – множина темпоральних ребер. Кожне темпоральне ребро є трійкою (u, v, t), що означає наявність прямого зв'язку (контакту) між вузлами u і v у момент часу t. Іншими словами, граф змінюється з часом: множина ребер $E(t)$ в кожний момент t різна. Така модель відповідає підходу, де динамічна мережа розглядається як орієнтований граф з часовими мітками на ребрах [28]. Кожне ребро постачене міткою $\lambda(e)$ – часом встановлення зв'язку між відповідними вузлами [29]. За

потреби модель можна розширити, задавши кожному ребру додаткові атрибути, наприклад пропускну здатність лінку (кількість пакетів, що може бути передано за контакт) [29]. Для спрощення подальшого аналізу вважатимемо, що один контакт дозволяє обмінятися щонайбільше одним повідомленням (пакетом) між вузлами.

Множину подій контакту можна визначити як $C = \{(i, j, t_k)\}$, де (i, j, t_k) означає подію зв'язку між вузлами i та j в момент t_k. При цьому допускається, що $i = j$ немає (контакт вузла із самим собою тривіальний) і контакти неупорядковані: подія (i, j, t) тотожна (j, i, t). На основі множини C можна побудувати послідовність «знімків» топології: наприклад, дискретизувавши час, розглядаємо серію статичних графів $G(t_1), G(t_2), \dots, G(t_N)$, кожен з яких містить ребро (u, v) якщо був хоча б один контакт між u і v у відповідному інтервалі часу. Альтернативним підходом є побудова просторово-часового графа: вводиться множина вершин $V \times T$ (кожна пара (вузол, час)), а ребра з'єднують пари, між якими відбувається безпосередній контакт. На такому графі класичні алгоритми пошуку шляхів дають маршрут передачі з урахуванням часу [28, 29].

У рамках темпорального графа визначається поняття часозалежного маршруту (або time-respecting path). Часозалежний маршрут з вузла a до вузла b – це скінченна послідовність ребер $(a = v_0 \to v_1, t_1), (v_1 \to v_2, t_2), \dots, (v_{k-1} \to v_k = b, t_k)$, де $(v_{i-1}, v_i, t_i) \in E_T$ і часові мітки зростають: $t_1 < t_2 < \cdots < t_k$. Інтуїтивно це означає, що повідомлення може бути передане від a до b через посередників $v_1, \dots, v_{k-1}$, якщо кожен наступний контакт відбувається після завершення попереднього. Вказане визначення еквівалентне наступному: маршрут існує, якщо вузол a контактує з деяким v_1 у час t_1, потім v_1 контактує з v_2 у пізніший момент $t_2 > t_1$, і так далі, аж доки v_{k-1} контактує з b в $t_k > t_{k-1}$. Таким чином, класичний шлях у статичному графі замінюється шляхом у просторі-часі, що задовольняє часовим обмеженням [29]. Якщо ж для двох вузлів не існує жодної послідовності контактів з незменшуваними часовими мітками, то вони тимчасово

нероз’єднані (не має способу доставити повідомлення в рамках доступних контактів).

3.1.3 Обґрунтування методу розв’язання

Представлення мережі у вигляді темпорального графа є сучасним підходом до моделювання мобільних ad hoc та delay-tolerant мереж. На відміну від статичних графів, де шляхи існують незалежно від часу, темпоральний граф дозволяє строго врахувати інтервальну природу зв’язків і їхню асинхронність. Наприклад, навіть якщо два вузли ніколи не перебувають у прямому контакті одночасно, повідомлення може бути доставлено опосередковано через третій вузол у відповідні моменти. Така модель застосована в архітектурі DTN, що побудована на принципі проміжного зберігання даних у вузлах за відсутності наскрізного шляху. У DTN-вузлах передбачене використання пам’яті для буферизації повідомлень доти, доки не з’явиться можливість їх пересилання – мережа здатна працювати навіть якщо жодного повного маршруту в даний момент не існує [22]. Таким чином, асинхронність зв’язків компенсується за рахунок механізму «store-carry-forward»: вузол-носій може тривалий час зберігати повідомлення та переносити його, поки не зустріне іншого вузла, якому можна передати дані.

Темпоральна модель графа дає можливість застосувати математичний апарат теорії графів та стохастичних процесів до аналізу нестабільних мереж. Зокрема, можна визначати показники надійності мережі: темпоральну досяжність вузлів (ймовірність або можливість існування time-respecting маршруту між довільною парою вузлів за певний інтервал часу), темпоральний діаметр (максимальна затримка доставки між будь-якими двома вузлами, якщо існує шлях) тощо. На основі цієї моделі можливе обґрунтування алгоритмів маршрутизації – наприклад, доведення того, що за нескінчено великого часу епідемічна маршрутизація здатна доставити повідомлення за умови зв’язності темпорального графа (принаймні один time-respecting шлях існує). Альтернативні підходи до моделювання – наприклад,

моделі марковських процесів для опису руху вузлів – можуть доповнювати графовий підхід, але саме модель графа з часозалежними ребрами є зручною для подальшого алгоритмічного розроблення. Це підтверджується широким використанням темпоральних графів у науковій літературі з DTN [29].

3.1.4 Опис моделі рішення

Темпоральний граф $G_T = (V, E_T)$ слугує базовою моделлю. Множина V відповідає вузлам месенджера (пристроям, що пересилають повідомлення). Множина темпоральних ребер E_T формується на основі контактів між вузлами: для кожної події прямого зв'язку (i, j, t) до E_T додається орієнтоване ребро $i \to j$ з часовою міткою t і, аналогічно, $j \to i$ з міткою t (оскільки зв'язок двосторонній) [29]. Кожне таке ребро означає можливість передачі повідомлення від i до j у час t. Якщо контакт триває певний інтервал часу Δt, його можна моделювати як кілька ребер з однаковим індексом часу або як одне ребро із зазначенням інтервалу, достатньо фіксувати момент початку контакту, оскільки саме він визначає часове впорядкування передач.

Для довільного фіксованого моменту t можна розглянути миттєвий граф зв'язності $G(t) = \big(V, E(t)\big)$, де $E(t) = \{(u, v) : \exists\, (u, v, t') \in E_T,\ t' \leq t\}$ – множина всіх зв'язків, що відбулися до часу t. Очевидно, $G(t)$ монотонно не спадає за включенням (зв'язки накопичуються). Практичний інтерес має аналіз на інтервалах: наприклад, для кожного повідомлення з часом генерації t_{orig} можна розглянути граф $G\big(t_{orig}, t_{deadline}\big)$, що містить ребра лише в межах інтервалу відправлення повідомлення до граничного часу доставки. Темпоральний маршрут від джерела до отримувача в цьому підграфі відповідає успішній доставці вчасно.

Таким чином, сформульована модель охоплює всі необхідні елементи: вузли (з їх можливим рухом чи відмовами, що відображено у динаміці контактів) і зв'язки (із зазначенням часу існування). На основі цієї моделі далі будуватимуться

алгоритми маршрутизації, які використовують локальну інформацію про сусідів та історію контактів для прийняття рішень про передачу повідомлень.

3.1.5 Висновок до підзадачі

В результаті розв’язання цієї задачі сформований строгий математичний опис нестабільної мережі як темпорального графа. Запропоноване моделювання дозволяє врахувати часову динаміку топології та асинхронність зв’язків: введене поняття темпорального (часозалежного) шляху, досяжності вузлів у часі, можливість відкладеної передачі через проміжне зберігання. Ця модель створює основу для аналізу та порівняння алгоритмів маршрутизації у середовищах з непостійною зв’язністю, що є характерним для децентралізованого месенджера в умовах DTN. Отримані формалізми використані в наступних підзадачах для строгого опису алгоритмів розповсюдження повідомлень.

3.2 Модифікований алгоритм Epidemic Routing

3.2.1 Постановка задачі

Epidemic Routing – це протокол маршрутизації, в якому повідомлення розповсюджуються по мережі шляхом максимального можливого дублювання: кожен вузол, що має копію повідомлення, при зустрічі з іншим вузлом передає йому цю копію (якщо той ще її не отримав). Така епідемічна передача гарантує, що за достатнього часу і пам’яті повідомлення досягне всіх куточків мережі, а отже і отримувача (якщо той хоча б періодично з’являється на зв’язку). Перевагою є мінімально можливий середній час доставки (наближається до найкоротшого можливого шляху), недоліком – надзвичайно високі витрати ресурсів: кількість копій повідомлення експоненційно зростає, виникають перевантаження буферів вузлів і витрати пропускної спроможності на дублювання. У реальних нестабільних мережах ресурси обмежені, тому ставиться задача модифікувати алгоритм

Epidemic Routing: обмежити неконтрольоване розмноження копій та нескінченне життя повідомлень у мережі. Зокрема, передбачимо такі обмеження: кожне повідомлення має час життя (TTL), після закінчення якого воно більше не передається; кожне повідомлення має унікальний ідентифікатор (хеш), що дозволяє вузлам розпізнавати дублікати і не приймати другу копію того ж самого; буфер кожного вузла обмежений за обсягом, тому протокол має враховувати політику видалення старих або менш пріоритетних повідомлень при заповненні пам'яті. Мета – зберегти надійність доставки, близьку до 100%, яка притаманна епідемічному підходу, але зробити алгоритм придатним для реалізації на реальних вузлах із кінцевими ресурсами.

3.2.2 Математична постановка задачі

Формалізуємо мережу і повідомлення для алгоритму Epidemic Routing. Нехай маємо множину вузлів V (які відповідають вершинам темпорального графа з підзадачі 3.1). Введемо множину повідомлень M; кожне повідомлення $m \in M$ характеризується такими параметрами: джерело $src(m) \in V$, призначення $dst(m) \in V$, унікальний ідентифікатор (хеш) $id(m)$, розмір (наприклад, в байтах) $size(m)$ та Time-To-Live $TTL(m)$ – величина, що обмежує максимальну кількість ретрансляцій або максимальний час перебування повідомлення в мережі. Для спрощення припустимо, що TTL заданий у термінах максимального числа контактів (hops), через які може пройти копія повідомлення. Наприклад, якщо $TTL = 1$, то повідомлення може бути передане лише один раз – безпосередньо від джерела до отримувача і не далі [30]. Якщо $TTL = 2$, повідомлення може мати не більше двох хопів (джерело → проміжний вузол → отримувач), і т.д.

Кожен вузол v має обмежений обсяг пам'яті під буфер, який можна задати як $B(v)$ – максимальне число повідомлень (або байт), що можуть одночасно зберігатися на вузлі. Вводимо стан буфера: $Q_v(t) \subseteq M$ – множина повідомлень, що зберігаються на вузлі v в момент часу t. Початковий стан: коли джерело генерує

нове повідомлення m у час t_0, m розміщується в буфер $Q_{src(m)}(t_0)$; при цьому його $TTL(m)$ ініціалізується деяким початковим значенням (наприклад, L).

Необхідним елементом алгоритму є ідентифікація копій. Кожне повідомлення має унікальний $id(m)$, який використовується для перевірки, чи отримував вузол цю копію раніше. Для цього кожен вузол v підтримує структуру даних, що називається summary vector – множину ідентифікаторів повідомлень $H_v(t) = \{id(m) : m \in Q_v(t)\}$, тобто реєстр отриманих повідомлень. Цей summary vector передається сусіднім вузлам під час встановлення контакту [30].

Правила передачі формалізуємо так. Нехай у час t відбувся контакт між двома вузлами a і b (існування такого контакту моделюється наявністю темпоральних ребер (a, b, t) і (b, a, t) у графі підзадачі 1). В процесі встановлення зв'язку виконується обмін службовою інформацією між a та b: кожен вузол передає іншому свій summary vector $H_a(t)$ і $H_b(t)$. Отримавши список ідентифікаторів від сусіда, вузол може визначити, які повідомлення відсутні у сусіда. Формально, вузол a обчислює множину $\Delta_a = \{m \in Q_a(t) : id(m) \notin H_b(t)\}$ – тобто повідомлення, які є в a, але відсутні в b. Аналогічно b визначає $\Delta_b = \{m \in Q_b(t): id(m) \notin H_a(t)\}$. На етапі обміну даними вузли надсилають один одному копії повідомлень зі списків відсутніх. Вузол a надсилає на b всі повідомлення з Δ_a, але при цьому виконується кілька важливих умов:

– передається тільки якщо $TTL(m) > 0$, інакше повідомлення вважається "простроченим" і не підлягає подальшій передачі;

– якщо буфер Q_b заповнений ($|Q_b| \geq B(b)$), тоді приймаючий вузол b повинен застосувати певну стратегію звільнення місця – наприклад, видалити з буфера найстаріше повідомлення або повідомлення з найменшим залишковим TTL, щоб прийняти нове;

– копії повідомлення m, що передається, призначаються тільки для читання вузлом b – тобто b отримує повну копію (включно з полями id та TTL). Після успішного прийому b додає m до свого буфера ($Q_b := Q_b \cup \{m\}$) і до свого реєстру

H_b; вузол a може залишити m у себе (тобто копія не знищується при передачі). Аналогічні дії виконує b для повідомлень з Δ_b. Завдяки обміну summary vectors жодне повідомлення не передається двічі в одному контакті: вузол не запитує те, що вже має [30]. Таким чином усувається зайва передача дублікатів між двома вузлами.

Окремо визначаємо оновлення TTL: щоразу при передачі копії повідомлення його лічильник TTL зменшується на 1 (або відповідно зменшується час життя, якщо TTL інтерпретується як часовий інтервал). Формально, якщо a надсилає m вузлу b, встановлюємо $TTL_{new}(m) = TTL_{old}(m) - 1$ для копії, що продовжує циркуляцію мережею. Якщо після цього оновлення $TTL(m)$ досягає 0, копія m більше не передається далі (хоча може залишатися в буфері поточного вузла до завершення контакту або видалення за політикою звільнення пам'яті). Нарешті, якщо вузол отримав повідомлення, де він сам є отримувачем ($v = dst(m)$), тоді воно доставляється адресату (наприклад, передається на застосунок) і може бути видалене з мережі. В простому варіанті протоколу вузол-отримувач може не інформувати інших про успішну доставку (відсутнє розповсюдження ACK), тому інші копії m можуть ще певний час циркулювати мережею до спливання TTL. Більш оптимізований варіант – впровадження механізму підтверджень отримання: коли $dst(m)$ отримав повідомлення, він сам починає розповсюджувати коротке повідомлення-підтвердження (ACK) епідемічним шляхом, яке, досягнувши інших вузлів, сигналізує їм видалити копії m як уже непотрібні. Такі розширення виходять за рамки базового алгоритму, але згадуються для повноти.

3.2.3 Обґрунтування вибору та коректності алгоритму

Епідемічна маршрутизація доцільна для умов фрагментованих мобільних мереж, де відсутні стабільні маршрути [30]. Її ідея – забезпечити максимальну ймовірність доставки, поширюючи копії повідомлення подібно до зараження вірусом популяції: зрештою повідомлення має охопити всю мережу, щоб знайти

отримувача [30]. У чистому вигляді цей алгоритм генерує надлишковий трафік і забиває пам’ять вузлів: «повідомлення розсилаються на всю мережу, щоб досягти одного адресата, що створює конкуренцію за буферний простір» [30]. Введені обмеження (TTL, унікальні ідентифікатори, ліміти буфера) покликані зменшити службові витрати і зробити алгоритм масштабованим. Зокрема, обмеження TTL локалізує епідемічне розповсюдження: вибираючи менше TTL, можна скоротити максимальну відстань поширення копій, зменшивши навантаження, хоч і ціною можливої втрати шансів доставки на дальні відстані. Налаштування TTL та розміру буфера дозволяє збалансувати затримку доставки і відсоток успішної доставки: менший TTL і менший буфер знижують затримку, але і зменшують ймовірність доведення повідомлення до адресата, тоді як більші значення підвищують гарантованість доставки, але ціною більших затримок і навантаження [30]. Таким чином, системний адміністратор мережі або сам протокол можуть «тюнінгувати» параметри TTL та політики буфера під конкретні вимоги (продуктивність vs надійність).

Використання summary vectors ідентифікаторів забезпечує відсутність зайвих копій: кожен вузол знає, які повідомлення вже бачив сусід, і не надсилає їх повторно [30]. Це гарантує, що алгоритм не зациклиться і не генеруватиме нескінченні повторення однієї і тієї ж копії між двома вузлами. Обмежений буфер і політика видалення запобігають «зависанню» застарілих повідомлень: наприклад, якщо повідомлення дуже старе або має малий залишковий TTL, протокол може видалити його першим, звільнивши місце для новіших або важливіших. Це підвищує стійкість до переповнення: при великому потоці нових повідомлень мережа поступово «забуває» старі, що не знайшли отримувача.

Сукупно, описані модифікації зберігають властивість епідемічного алгоритму – максимізацію ймовірності доставки. Якщо ресурси дозволяють, протокол все одно поширить копії на більшість вузлів, тому шанс, що хоча б одна копія досягне призначення, близький до одиниці. З іншого боку, контроль параметрів дає можливість стримувати надмірне дублювання. За потреби,

епідемічна схема може слугувати еталонним рішенням (benchmark), з яким порівнюють продуктивність більш вибіркових алгоритмів маршрутизації [30]. Враховуючи це, у цьому проєкті модифікований Epidemic Routing використаний як базовий рівень маршрутизації, на який накладатимуться додаткові евристики (див. підзадачу 3.3).

3.2.4 Опис алгоритму

Модифікований алгоритм Epidemic Routing можна записати псевдокодом або описати кроками. Наведемо покроковий опис з точки зору окремого вузла v:

а) структури даних вузла. Кожен вузол має буфер Q_v ємністю $B(v)$ повідомлень, та summary vector $H_v = \{, id(m) : m \in Q_v, \}$, що оновлюється автоматично при надходженні або видаленні повідомлень;

б) обробка нового повідомлення на джерелі. Якщо застосунок вузла v генерує нове повідомлення m з отримувачем $dst(m)$, то:

1) полю $id(m)$ призначається унікальний хеш, встановлюється початкове $TTL(m) = L$ (певне задане число hop'ів або часовий інтервал життя);

2) m поміщається в буфер Q_v (можливо, із витісненням якогось старого повідомлення, якщо $|Q_v| = B(v)$);

3) ідентифікатор додається до H_v. Якщо $dst(m) = v$ (випадок, коли вузол відправляє повідомлення сам собі), то воно відразу вважається доставленим;

в) при встановленні контакту з сусіднім вузлом u:

1) вузол v отримує summary-вектор сусіда H_u і надсилає йому свій H_v (сеанс обміну метаданими);

2) визначається набір повідомлень для відправлення: $\Delta_v = \{ m \in Q_v : id(m) \notin H_u \wedge TTL(m) > 0 \}$. Це ті повідомлення, які має v і яких немає в u, і що ще не втратили TTL;

3) аналогічно сусід визначає $\Delta_u = \{ m \in Q_u : id(m) \notin H_v \wedge TTL(m) > 0 \}$;

4) вузол v надсилає вузлу u копії всіх повідомлень з Δ_v. Перед передачею для кожного m встановлюється $TTL(m) := TTL(m) - 1$ (зменшуємо лічильник hop'ів на 1). Вузол u приймає копії, додає їх до свого буфера Q_u (виконує видалення старих, якщо необхідно) і реєстру H_u;

5) сусід u одночасно виконує дзеркальні дії: передає v копії всіх повідомлень з Δ_u з декрементом TTL, які v отримує та зберігає;

6) таким чином, відбувається двостороння синхронізація буферів: після контакту v і u володіють однаковим набором копій повідомлень (об'єднаним з їх попередніх копій) за винятком повідомлень, чиї TTL вичерпалися;

г) обробка отриманого повідомлення. Якщо вузол v отримав у кроці 3 копію повідомлення m і виявилося, що $\mathrm{dst(m)} = \mathrm{v}$ (цей вузол є адресатом), то вузол позначає повідомлення як доставлене. Подальша передача m іншим вузлам може не виконуватися (в простій реалізації вузол просто не включає його в Δ_v при наступних контактах, оскільки мета досягнута). Опціонально вузол може поширити ACK-повідомлення про доставку, але базовий алгоритм цього не вимагає;

д) видалення за часом життя. Періодично (або під час контактів) вузол перевіряє повідомлення в буфері і видаляє ті, чий $TTL \leq 0$ або чий час життя перевищив допустимий (наприклад, повідомлення «прострочене»). Це запобігає нескінченному зберіганню застарілих копій.

Такий алгоритм реалізований в емуляторах DTN. Наприклад, у модулі Epidemic Routing для NS-3 реалізований аналогічний підхід: вузли обмінюються списками ідентифікаторів повідомлень і пересилають один одному неотримані

повідомлення, зберігаючи поле hop count (аналог TTL) для кожного повідомлення та зменшуючи його при передачі [30].

3.2.5 Висновок до підзадачі

Модифікований алгоритм Epidemic Routing, як показано, забезпечує надлишкове розповсюдження повідомлень для досягнення високої вірогідності доставки у динамічному середовищі, водночас вводячи контроль за ресурсами. TTL обмежує радіус і тривалість «епідемії» повідомлення, унікальні ідентифікатори та обмін summary-векторами гарантують відсутність зайвих дублікатів, а кінцевий обсяг буфера на вузлах не перевищується завдяки політикам витіснення. Хоча цей алгоритм не знижує суттєво обсяг трафіку в найгіршому випадку (все ще у гіршому разі кожне повідомлення розсилається до кожного вузла, якщо TTL достатньо великий), він створює основу для оптимізації. Отримана формальна схема буде базовою точкою порівняння: у наступній підзадачі буде зменшений обсяг передач, використовуючи додаткову інформацію про мережу, але при цьому результати Epidemic Routing з обмеженнями (доля доставлених, середня затримка) слугуватимуть бенчмарком. В цілому, підзадача дала математично строгий опис алгоритму епідемічної маршрутизації в умовах нестабільної мережі та показала, як параметри TTL і буфера впливають на ефективність (через компроміс швидкості проти надійності) [30].

3.3 Гібридний евристичний алгоритм маршрутизації

3.3.1 Постановка задачі

Попередній алгоритм (епідемічний) забезпечує високу надійність ціною великого навантаження. У практичних сценаріях доцільно спрямувати передачу повідомлень вибірково – тільки до тих сусідів, які дійсно підвищують ймовірність доставки. Ідея полягає в тому, щоб використати локальний контекст вузла і

історичні дані про зустрічі вузлів, аби оцінити корисність пересилання повідомлення кожному з можливих сусідів. У цій підзадачі розробляється гібридний евристичний алгоритм маршрутизації, запропонований автором, який поєднує:

– локальну інформацію про сусідів та стан мережі;

– історію попередніх передач і зустрічей вузлів (статистичні оцінки ймовірності доставки);

– таблиці надійності сусідів (метрики «довіри» чи delivery predictability кожного контакту);

– евристичну функцію корисності, що агрегує ці фактори і допомагає прийняти рішення – чи передавати повідомлення певному сусіду, чи утриматись.

Алгоритм називається гібридним, оскільки поєднує підхід прямого розповсюдження (як в епідемічному алгоритмі) з інтелектуальним ранжуванням потенційних ретрансляторів (як у детермінованих протоколах). Евристичний – тому що не гарантує абсолютно оптимального рішення, але прагне до нього через набір обґрунтованих правил і метрик. Постановка задачі: необхідно формально визначити метрики надійності вузлів, спосіб їх оновлення, вигляд евристичної функції корисності і сам алгоритм передачі повідомлень; довести, що такий алгоритм покращує показники (скорочує кількість копій, зберігаючи високу ймовірність доставки) у порівнянні з чисто епідемічним.

3.3.2 Математична формалізація

На цьому етапі важливо ввести формальні поняття, які ляжуть в основу алгоритму.

Delivery predictability. Нехай для кожної пари вузлів (A, B) визначено значення $P_A(B)$ – ймовірність доставки повідомлення до вузла B за участі вузла A як проміжного. Цю величину можна інтерпретувати так: якщо вузол A має повідомлення для вузла B, то $P_A(B)$ від 0 до 1 показує, наскільки «надійним» є A

як носій цього повідомлення (0 – зовсім не має шансів зустріти B, 1 – напевно доставить сам). У протоколі PRoPHET (Probabilistic Routing Protocol using History of Encounters and Transitivity) подібна метрика називається delivery predictability і ініціалізується для всіх пар на невеликі значення, після чого поступово збільшується при частих зустрічах вузлів [23]. Позначимо її $P(A,B)$ для пари A (вузол-носій) і B (потенційний отримувач або проміжний вузол на шляху до отримувача).

Таблиці надійності сусідів. Кожен вузол X зберігає у себе таблицю $\{P_X(Y): Y \in V\}$ оцінки ймовірності доставки від себе до кожного іншого вузла Y. Ці оцінки оновлюються на основі історії контактів за такими правилами:

– якщо X зустрівся безпосередньо з Y, тобто встановлено контакт, то їхня близькість підвищується: $P_X(Y)$ збільшується. Типова формула оновлення: $P_X(Y) := P_X(Y) + (1 - P_X(Y)) \cdot \alpha$, де α – константа (коефіцієнт приросту при контакті) [23]. Такий закон означає, що при першій зустрічі ($P_X(Y)$ мало) приріст буде значним (наблизиться до α), а при багаторазових частих зустрічах $P_X(Y)$ вже велике і збільшиться ненабагато (ефект насичення до 1). Параметр α слід обирати на основі характерної частоти зустрічей у мережі. За відсутності попередніх даних приймають $\alpha = 0.5..0.7$ [23], тобто перша ж зустріч дає $\sim 50-70\%$ шанс, що вузли ще зустрінуться;

– при відсутності контактів між X і Y протягом тривалого часу їхня delivery predictability повинна зменшуватись (старі контакти «забуваються»). Для цього вводиться коефіцієнт старіння: з часом $P_X(Y)$ експоненційно зменшується. Формула: $P_X(Y) := P_X(Y) \cdot \gamma^{\Delta t}$, де Δt – час з моменту останнього оновлення, $\gamma \in (0,1)$ – параметр старіння (чим ближче до 1, тим повільніше забування) [23]. Наприклад, $\gamma = 0.98$ означає $\sim 2\%$ втрати «довіри» за одиницю часу без зустрічей;

– транзитивне оновлення. Якщо X контактує з Z і дізнається від нього, що Z часто зустрічає Y (має високе $P_Z(Y)$), то після контакту X може підвищити свою

оцінку $P_X(Y)$ навіть якщо сам X ніколи не бачив Y. Цей механізм покращує шанси доставки через декілька перескоків. Формула:

$$P_X(Y) := \max\{ P_X(Y),\ P_X(Z) \cdot P_Z x\,(Y) \cdot \beta \}$$ [23], де $\beta \in (0,1)$;

– коефіцієнт транзитивного послаблення (наскільки менше довіряємо транзитивній інформації порівняно з прямою). Типове значення $\beta \approx 0.25 – 0.5$.

В результаті кожна таблиця надійності $P_X(\cdot)$ постійно еволюціонує і відображає останні тенденції: з ким вузол X часто контактує напряму та опосередковано. Ці таблиці оновлюються розподілено: при кожному реальному контакті вузли обмінюються своїми таблицями (або релевантною їх частиною) і після обміну перераховують значення згідно правил вище [23]. Таким чином, у кожного вузла завжди є локальна оцінка, через кого краще відправити повідомлення до того чи іншого адресата.

Локальний контекст. Окрім історичних ймовірностей доставки, вузол може враховувати поточний стан сусідів і мережі – це і є локальний контекст. До нього входять, зокрема:

– залишковий розмір буфера сусіда (скільки місця має сусід для прийому нових повідомлень);

– залишковий заряд батареї сусіда (на випадок, якщо вузли на батареї – пристрій з низьким зарядом може не дожити до доставки);

– присутність в мережі: чи підключений сусід до якоїсь інфраструктури (наприклад, до інтернет-шлюзу);

– географічна позиція або глобальні тенденції руху: якщо відомо, куди рухається вузол (напрямок, швидкість), це може підказати, наближається він до отримувача чи віддаляється;

– «популярність» вузла – наскільки багато контактів він має з іншими (наприклад, ступінь вузла в соціальному графі контактів). Більш популярний вузол потенційно краще як ретранслятор, бо охоплює більше сусідів.

Перелічені фактори можна формалізувати як змінні або функції на множині сусідів. Наприклад, нехай для вузла X введено: $buf(X)$ – частка вільного буферу

(0=немає місця, 1=повністю вільний); $pow(X)$ – рівень енергії (0=батарея розряджена, 1=повністю заряджений); $pop(X)$ – популярність (можна визначити як середню кількість контактів за день, нормовану на максимум в мережі). Місцеположення можна врахувати, ввівши функцію $dist_X(Y)$ – оцінка "наближеності" маршруту від X до Y за географією (чим менша, тим ближче – напр., нормована відстань чи ранг за відстанню). Ці величини $buf, pow, pop, dist$ складають локальний контекст, відомий вузлу або отриманий шляхом обміну з сусідами.

Евристична функція корисності. Тепер визначимо функцію U (utility), яка оцінює доцільність передачі повідомлення певному сусіду. Щоб визначити, чи варто вузлу A передавати повідомлення для отримувача D сусіду B, алгоритм обчислює $U_{A\to B}(D)$ – чим більше це значення, тим більша користь від передачі. Функція U має врахувати як історичні дані, так і поточний контекст. Один із можливих підходів – лінійна згортка метрик (зважена сума). Наприклад, можна задати:

$$U_{A\to B}(D) = w_1 \cdot P_B(D) + w_2 \cdot P_B^{max} + w_3 \cdot buf(B) + w_4 \cdot pop(B) + w_5 \cdot pow(B) + w_6 \cdot f\big(dist_B(D)\big),$$

де w_i – вагові коефіцієнти, $P_B(D)$ – прогнозована ймовірність доставки від B до отримувача D, $P_B^{max} = max_{Z\in V} P_B(Z)$ – максимальна «здатність доставки» вузла B до будь-якого вузла (показник загальної надійності B як ретранслятора), а $f\big(dist_B(D)\big)$ – деяка функція відстані між B і D (наприклад, f спадає від 1 до 0, коли відстань зростає).

Наведена формула – лише концептуальна; на практиці набір факторів та їх ваг підбирають емпірично або методами машинного навчання. В літературі зустрічаються приклади, коли до метрики корисності включали одразу кілька параметрів. Зокрема, в протоколі PRoPHET+ автори використали саме такий підхід: «deliverability» вузла визначається як зважена сума розміру буфера, рівня батареї, геолокації, популярності та значення ймовірності доставки» [27]. Іншими

словами, кожному з цих аспектів призначено вагу, і їх лінійна комбінація дає числовий рейтинг вузла. При зустрічі двох вузлів один з одним вони порівнюють свої значення deliverability: якщо значення для вузла-отримувача вище за певний поріг, то повідомлення передається; якщо декілька сусідів доступні одночасно, копія повідомлення надсилається тому, в кого найбільше значення корисності [27]. Така стратегія гарантує, що повідомлення не дублюється на надто багатьох вузлах – воно передається лише перспективним ретрансляторам.

Гібридність підходу. Алгоритм названо гібридним тому, що він поєднує поширення повідомлень декількома копіями (як багатокопійні протоколи на кшталт епідемічного) зі строгим відбором найкращих кандидатів (як детерміністичні протоколи). Можна навести аналогію з існуючими розробками: наприклад, покращений PRoPHET-протокол міг діяти так – «спочатку у ранній фазі доставки розповсюдити повідомлення епідемічно для швидкого охоплення мережі, а на пізніших етапах обмежити копіювання, вимагаючи виконання умови високої ймовірності доставки» [27]. Алгоритм, що запропонований, дотримується схожої ідеї, але без явного поділу на фази: він на кожному кроці вирішує, передавати чи ні, виходячи з евристичної оцінки. Якщо мережа розріджена і хороших кандидатів немає, алгоритм може деградувати до епідемічного (бо всі сусіди рівнопогані – можливо, все одно передасть хоча б одному, щоб не втратити шанс). Якщо ж у мережі є чіткі «центри» і «мости» (вузли з високою популярністю чи ймовірністю доставки), то основний трафік піде через них, а зайві копії не розповсюджуватимуться.

3.3.3 Обґрунтування методу

Підхід з використанням історичної інформації про контакти значно знижує службові витрати в DTN порівняно з наївним flood-маршрутизуванням. Базовим представником цього класу є згаданий протокол PRoPHET. Він саме і мотивується тим, що рух вузлів не є цілком випадковим; часто існують повторювані шаблони

(наприклад, люди щодня ходять схожими маршрутами, транспорт рухається за розкладом тощо). PRoPHET замінив сліпе дублювання на ймовірнісну маршрутизацію: кожен вузол поступово обчислює метрику $P(A,B)$ для всіх відомих вузлів і використовує її, щоб вирішити, кому передати повідомлення. У результаті протокол будує не один фіксований маршрут, як класичні протоколи, а скоріше «обрізає дерево епідемічного поширення», відкидаючи гілки, що мають низьку ймовірність успіху. Наприклад, якщо A зустрів B і знає, що B має більший шанс зустріти отримувача D, ніж A сам, то A передасть копію B і, можливо, не передаватиме іншим, менш перспективним вузлам. Це значно знижує кількість копій у мережі. PRoPHET, таким чином, є однією з основ для цього алгоритму; він розвинутий шляхом додавання іншої евристики (буфер, енергія тощо).

Удосконалення PRoPHET саме в такому напрямі вже пропонувалися: зокрема, у PRoPHET+ показано, що врахування додаткових факторів (популярність, залишок пам’яті, батарея) дозволяє підвищити Delivery Probability і знизити Average Delay проти базового PRoPHET. Альтернативні підходи включають використання соціальних показників (протоколи SimBet, BubbleRap – де вузли оцінюються за центральністю в соціальному графі та належністю до «спільнот») і навіть залучення методів штучного інтелекту. Алгоритм можна розглядати як частково соціальний (через фактор популярності), частково історичний (через $P(A,B)$). Також існує клас біо-натхненних алгоритмів, де маршрути знаходяться шляхом імітації поведінки тварин. Яскравий приклад – алгоритм AntNet, що імітує роботу колонії мурах: спеціальні агентпакети («мурахи») розповсюджуються мережею і накопичують інформацію про затримки і якість шляхів, залишаючи в вузлах «феромонні мітки». На основі цих міток вузли будують ймовірнісні маршрути, які адаптуються до змін мережі. AntNet добре працює у стаціонарніших мережах (наприклад, дротових або слабо мобільних), даючи близькі до оптимальних маршрути за критерієм затримки і пропускної здатності. Однак у DTN з довготривалими розривами зв’язку застосування AntNet у чистому вигляді утруднене (агенти можуть не мати можливості швидко

повернутися зі зворотньою інформацією через відсутність маршрутів). Тим не менш, ідея агентів, що збирають статистику, співзвучна з підходом збирання історії контактів, який ми використовуємо. Отже, сучасні дослідження підтверджують ефективність гібридних евристичних підходів: вони успадковують високу надійність від багатокопійних схем, але різко скорочують зайві передачі завдяки використанню накопиченої інформації [27]. Алгоритм узгоджується з цими тенденціями і розширює їх, комбінуючи кілька критеріїв в єдину метрику корисності.

Висновок до розділу 3

У цьому розділі виконані три пов’язані етапи, що закладають теоретичну основу для побудови децентралізованого месенджера з адаптивною маршрутизацією в нестабільних мережах.

По-перше, розроблена математична модель динамічної топології у вигляді темпорального графа, яка відображає змінність з’єднань у часі та асинхронність зв’язків. Ця модель дозволила формально визначити критерії досяжності вузлів у середовищі без гарантованого маршруту і підготувала ґрунт для аналізу алгоритмів маршрутизації.

По-друге, строго описаний алгоритм Epidemic Routing з внесеними обмеженнями (поля TTL, унікальні ідентифікатори повідомлень, межі буферів). Показано, що такі обмеження зберігають гарантії доставки, проте запобігають неконтрольованому зростанню мережевих витрат; проведений аналіз впливу параметрів TTL і розміру буфера на характеристики алгоритму.

По-третє, представлений гібридний евристичний алгоритм, який на основі локального і історичного контексту приймає рішення про передачу повідомлень. Цей алгоритм формалізований через введення метрик ймовірності доставки та допоміжних показників (надійності сусідів, популярності тощо) і задання евристичної функції корисності. Зроблене обґрунтування, що такий підхід значно

зменшує кількість зайвих копій у мережі, наближаючи поведінку системи до оптимальної без втрати надійності.

Узагальнюючи результати, можна констатувати: в межах цього розділу отримане цілісне математичне підґрунтя для побудови адаптивної маршрутизації в децентралізованому месенджері. Модель динамічної мережі дала можливість формально описати середовище функціонування, епідемічний алгоритм забезпечив відправну точку з максимальними можливостями доставки, а розроблений евристичний алгоритм – механізм інтелектуального вибору найкращих шляхів. Ці напрацювання створюють основу для реалізації прототипу месенджера та його подальшого аналізу в наступних розділах роботи, зокрема для оцінки ефективності маршрутизації на реальних або змодельованих даних. В результаті виконання поставлених підзадач забезпечена математична підтримка проєкту у вигляді суворих формулювань і обґрунтувань алгоритмів, що оптимально працюють в умовах нестабільної мережевої топології.

4 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

Діаграма класів розробленого застосунку продемонстрована на кресленику ІС13.080БАК.005 Д3.

Прототип розробленої інформаційної системи реалізований базуючись на підходах об’єктно-орієнтованого програмування.

4.1 Модуль користувацького інтерфейсу

Реалізація модулю інтерфейсу для взаємодії з користувачем розбита на декілька підкласів та відповідні поля та методи:

а) ChatUI – основний клас для відображення інтерфейсу та інструментів взаємодії з застосунком. Потребує ідентифікатор користувача для коректної роботи;

1) username – збережений ідентифікатор користувача, якщо він не заданий, то система самостійно визначає та присвоює випадковий ідентифікатор;

2) run() – метод запускук відображення інтерфейсу;

3) send_message_via_router(message: Message) – метод взаємодії з модулем маршрутизації, приймає повідомлення та направляє його на обробку маршрутизатором;

4) receive_external_messages() – є допоміжним методом, призначеним для оновлення історії отриманих повідомлень.

б) Message – є класом-контейнером для повідомлення;

1) from_user – поле для збереження ідентифікатора відправника;

2) to_user – поле для збереження ідентифікатора отримувача;

3) text – поле для збереження самого повідомлення;

4) metadata: MessageMetadata – слугує полем для класу, що зберігати всі необхідні метадані для повідомлень (пакетів);

в) ChatState – клас, що є інкапсулятором для даних користувача, виконує роль модуля історії повідомлень;

1) get_messages() – є методом для отримання повного списку повідомлень;

2) add_message() – є методом, що викликається для додавання повідомлень до історії.

4.2 Модуль маршрутизації

Реалізація модулю інтерфейсу для маршрутизації пакетів розбита на декілька підкласів та відповідні поля і методи:

а) RoutingAgent – основний клас для модулю маршрутизації, взаємодіє з іншими модулями системи;

1) router: BaseRouter – приймає метод маршрутизації при ініціалізації, це дає можливість створити та/або замінити поточний метод маршрутизації пакетів за потреби;

2) send(message: Message) – передає повідомлення комунікатору;

3) pull() – передати повідомлення інтерфейсу користувача;

б) BaseRouter – інтерфейс для створення методів маршрутизації, декларує необхідні методи для функціонування;

1) route_message(message: Message) – передача повідомлень іншим вузлам;

2) fetch_incoming() – отримання вхідного повідомлення з мережі;

в) DummyRouter – маршрутизатор, що нічого не робить, створений для тестування.

4.3 Методи маршрутизації

Реалізація методів відбувалась на базі класу-інтерфейсу BaseRouter. Схеми алгоритмів маршрутизації можна знайти на діаграмах діяльності на кресленику ІС13.080БАК.005 Д4.

Для реалізації двох методів маршрутизації повідомлень розроблені наступні класи та їх поля і методи:

а) MessageMetadata – клас, що зберігає у собі всі метадані для певного пакета з повідомленням, використовуються для ідентифікації та TTL:

1) message_id – унікальний хеш або UUID повідомлення;

2) ttl – кількість дозволених пересилок;

3) created_at – час створення (для пріоритетів, TTL);

4) source_id – ID відправника;

5) destination_id – ID отримувача;

6) creation_time – зберігає час надсилання повідомлення;

б) NodeBuffer – додатковий службовий клас для реалізації черги (буферу) повідомлень:

1) Capacity – максимальна кількість повідомлень у буфері;

2) messages – збережені повідомлення парами ключ-значення (ключ: message_id);

3) add(message) – додає повідомлення до буфера або відкидає;

4) remove(message_id) – видаляє повідомлення;

5) evict_policy() – застосовує політику видалення (наприклад, за TTL або пріоритетом);

в) EpidemicRouter(BaseRouter) – реалізація модифікованого алгоритму багатоадресного дублювання:

1) buffer – локальний буфер повідомлень;

2) seen_messages – множина вже отриманих повідомлень;

3) ttl_limit – глобальне обмеження TTL;

4) delivery_stats – журнал успішних доставок (статистика);

5) route_message(message) – додає повідомлення до буфера, передає новим сусідам;

6) fetch_incoming() – обробляє прийняті повідомлення;

7) synchronize_metadata(peer_ids) – обмінюється ID повідомлень із сусідами;

8) select_messages_for_peer(peer_id) – обирає нові повідомлення для відправки;

9) log_delivery_result(...) – записує інформацію про успішну доставку;

г) HeuristicRouter(BaseRouter) – реалізація гібридний евристичниго алгоритму маршрутизатора з локальною оцінкою:

1) buffer – буфер повідомлень;

2) trust_table – довіра до сусідів (рейтинг на основі історії);

3) transmission_history – історія передачі (<message_id>, <peer_id>);

4) heuristic_threshold – поріг передавання (0..1);

5) max_ttl – глобальне TTL обмеження;

6) route_message(message) – приймає рішення про передачу на основі оцінки;

7) fetch_incoming() – приймає нові повідомлення;

8) evaluate_utility(messageб, peer_id) – евристична оцінка доцільності передачі;

9) update_trust(peer_id, success) – оновлює довіру до вузла;

10) filter_for_peer(peer_id) – відбирає повідомлення для пересилки;

11) evict_policy() – вибір повідомлень на видалення (TTL + надійність цільового вузла);

д) VirtualNode – є основним класом вузла який інкапсулює у собі усі функціональні компоненти системи, мережева комунікація здійснюється безпосередньо на рівні цього класу:

1) node_id – унікальний ідентифікатор вузла;

2) context – контекст ZeroMQ для створення сокетів;

3) router_agent – модуль маршрутизації, інтегрований як ядро маршрутизаційної логіки;

4) incoming_queue – вхідна черга повідомлень;

5) connections – словник активних з’єднань з іншими вузлами (peer_id → сокет);

6) receiver – ZeroMQ REP-сервер для прийому повідомлень;

7) listen_thread – окремий потік для прослуховування вхідних повідомлень;

8) running – прапорець активності вузла;

9) config – збережена конфігурація вузла (при наявності);

10) __init__(node_id, config_json) – ініціалізація вузла, сокетів, конфігурації;

11) start() – запуск сервера та фонової обробки;

12) stop() – завершення роботи вузла, очищення сокетів;

13) get_port() – розрахунок TCP-порту на основі ID вузла (наприклад, 10000 + int(node_id));

14) load_config(config_json) – завантаження JSON-конфігурації (наприклад, список підключень);

15) connect_to_node(peer_id) – створення нового outbound-з’єднання до вузла-сусіда;

16) disconnect_node(peer_id) – розрив з’єднання;

17) send_message(message) – передача повідомлення через відповідний сокет;

18) listen_loop() – цикл прослуховування вхідних повідомлень, кладе їх у incoming_queue;

е) poll_incoming() – витягує повідомлення з черги, передає у router_agent.send(...).

Висновки до розділу 4

У межах цього розділу здійснене проєктування та модульна реалізація програмних компонентів системи динамічної маршрутизації повідомлень, що функціонує в умовах нестабільного з’єднання та обмежених мережевих ресурсів. Усі розроблені компоненти побудовані за принципами об'єктно-орієнтованого програмування, із забезпеченням гнучкості, масштабованості та інтеграційної придатності.

Створений модуль користувацького інтерфейсу, реалізований засобами фреймворку Streamlit, що забезпечує базову взаємодію користувача з системою. Інтерфейс дозволяє здійснювати відправлення, перегляд та обробку повідомлень у структурованому вигляді, підтримуючи модель "один-вузол — один клієнт". Архітектура інтерфейсу передбачає використання стандартного вхідного поля, стрічки повідомлень та підтримку авторизації за унікальним ідентифікатором вузла. Компонент реалізований у формі класу ChatUI, що реалізує також функціональність споживача повідомлень (відповідно до протоколу MessageConsumer).

Окремо реалізований модуль маршрутизації, що інкапсулює логіку обробки повідомлень, взаємодію з інтерфейсом користувача та можливість використання різних стратегій доставки. Центральним об'єктом у даному модулі є клас RoutingAgent, який отримує повідомлення з каналу комунікації, передає їх зареєстрованим споживачам (UI, логер тощо), а також делегує відправлення повідомлень конкретному реалізованому маршрутизатору, що відповідає інтерфейсу BaseRouter. Така модульна структура дозволила реалізувати та інтегрувати одразу дві різні маршрутизуючі стратегії: модифікований Epidemic Routing (EpidemicRouter) та гібридний евристичний алгоритм (HeuristicRouter), кожна з яких втілює різну модель поведінки в умовах обмеженої та мінливої топології.

Модифікований алгоритм Epidemic Routing забезпечує гарантоване поширення повідомлень у мережі без централізованого контролю, але зазнає перевантаження через надлишкове дублювання. Додане обмеження TTL, унікальну ідентифікацію повідомлень та обмеження буфера, що дозволило суттєво зменшити обсяг мережевого трафіку при збереженні властивої даному підходу надійності доставки. Крім того, реалізований механізм збирання статистики доставки, що дозволить у подальшому здійснювати аналіз ефективності мережевої поведінки та приймати зважені рішення щодо параметрів протоколу.

Гібридний алгоритм, у свою чергу, поєднує елементи евристичної оцінки, локальної адаптації та обмеженого дублювання. Його інновацією є використання таблиці довіри (TrustTable) – локального рейтингу вузлів, який формується на основі статистики попередніх успішних або невдалих пересилок. Такий підхід дозволяє зменшити надлишковість передач, сприяє більш цілеспрямованому використанню ресурсу мережі та забезпечує адаптивність у реальних умовах змінної доступності з'єднань.

Також розроблений модуль віртуалізованої комунікації (VirtualNode), який моделює поведінку окремого вузла в умовній одноранговій мережі. Передача повідомлень здійснюється засобами ZeroMQ, що дозволяє уникнути складних мережевих протоколів на етапі моделювання, зберігаючи при цьому реалістичну поведінку процесів обміну, очікування відповіді та формування динамічних з’єднань між вузлами. Вузол підтримує запуск у фоновому режимі, опитування вхідної черги повідомлень, з’єднання та розрив зв’язків із сусідніми вузлами, а також конфігурацію через JSON-опис.

Система побудована таким чином, що кожен компонент є незалежним, але взаємопов’язаним, з чіткими інтерфейсами взаємодії. Внаслідок цього реалізація є розширюваною: дозволяє підключення додаткових протоколів маршрутизації, розширення користувацького інтерфейсу або підключення реальних мережевих модулів замість ZeroMQ, без необхідності повної перебудови структури.

5 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ

Для забезпечення функціональної коректності, стійкості до відмов та відповідності заявленим вимогам, усі основні компоненти розробленої інформаційної системи піддані модульному тестуванню з використанням підходу мок-тестування. Основна мета тестування – верифікувати логіку взаємодії між модулями без прив’язки до реального мережевого середовища або складної інфраструктури, використовуючи програмно створені об’єкти.

Тестування охоплює аспекти функціонування: від базової обробки повідомлень, роботи маршрутизаторів, черг та буферів до мережевих взаємодій між віртуальними вузлами та графічного інтерфейсу користувача. Такий підхід обраний через те, що він дозволяє ізолювати помилки, виявити неконсистентну логіку в реалізації алгоритмів маршрутизації та гарантувати, що система правильно поводиться у граничних випадках. Нижче наведений перелік мок-тестів, які реалізовані для забезпечення повного покриття усіх модулів та компонентів системи:

а) тести класу Message:

1) test_message_creation – перевіряє правильність ініціалізації об'єкта повідомлення;

2) test_message_id_uniqueness – перевіряє, що різні повідомлення мають унікальні ідентифікатори;

б) тести буфера NodeBuffer:

1) test_buffer_add_within_capacity – перевіряє успішне додавання повідомлень у буфер при нормальному навантаженні;

2) test_buffer_add_exceeds_capacity_evicts_oldest – перевіряє коректну роботу політики витіснення при переповненні;

3) test_buffer_remove_message – тестує видалення повідомлення з буфера;

4) test_buffer_duplicate_message_not_added – перевіряє, що дублікати не додаються;

в) тести таблиці довіри TrustTable:

1) test_trust_initial_value – перевіряє, що довіра до нового вузла стартує з початкового значення;

2) test_trust_increases_on_success – перевіряє підвищення рейтингу після успішної передачі;

3) test_trust_decreases_on_failure – перевіряє зниження рейтингу після невдалої передачі;

4) test_trust_bounds_clamping – перевіряє, що рейтинг не виходить за межі [0.0, 1.0];

г) тести EpidemicRouter:

1) test_epidemic_route_adds_unseen_message – перевіряє додавання нового до буфера;

2) test_epidemic_route_ignores_duplicate_message – перевіряє, що повторне повідомлення не дублюється;

3) test_epidemic_ttl_decrement – перевіряє зменшення TTL після маршрутизації;

4) test_epidemic_fetch_returns_buffered_messages – перевіряє отримання повідомлень з буфера;

5) test_epidemic_delivery_logging – перевіряє запис результатів доставки;

д) тести HeuristicRouter:

1) test_heuristic_skips_low_utility_peers – перевіряє, що вузли з низькою довірою не отримують повідомлення;

2) test_heuristic_routes_to_high_trust_peer – перевіряє відправлення до вузлів з високим рейтингом;

3) test_heuristic_prevents_duplicate_transmission – перевіряє, що повідомлення не надсилається двічі одному вузлу;

4) test_heuristic_evict_policy_applies – перевіряє виклик політики видалення при переповненні буфера;

5) test_heuristic_utility_function_threshold – тестує порогове значення функції корисності;

е) тести RoutingAgent:

1) test_agent_registers_consumer – перевіряє реєстрацію споживача;

2) test_agent_sends_message_via_router – перевіряє делегування надсилання маршрутизатору;

3) test_agent_dispatches_incoming_messages – перевіряє розсилку отриманих повідомлень усім підписаним споживачам;

ж) тести VirtualNode:

1) test_node_start_and_stop – перевіряє запуск та коректне завершення вузла;

2) test_node_connects_to_peer – перевіряє створення з’єднання з іншим вузлом;

3) test_node_disconnects_peer – перевіряє розрив з’єднання;

4) test_node_send_message_to_peer – перевіряє надсилання повідомлення до конкретного вузла;

5) test_node_receives_message_and_buffers – перевіряє отримання повідомлення та його обробку;

6) test_node_load_config_creates_connections – перевіряє, що з JSON-конфігурації створюються з’єднання;

7) test_node_poll_incoming_routes_to_agent – перевіряє, що вхідні повідомлення передаються агенту маршрутизації;

з) тести ChatUI (із застосуванням моків):

1) test_ui_receives_message_and_renders – перевіряє, що UI коректно обробляє вхідне повідомлення;

2) test_ui_send_message_creates_valid_message – перевіряє створення об’єкта повідомлення через інтерфейс.

Усі результати тестування, включаючи логи виконання, кількісні показники покриття та зафіксовані винятки, зібрані та структуровані у звіт з тестування, що наводиться у Додатку Б до цієї роботи.

Висновки до розділу 5

Проведене тестування програмних компонентів інформаційної системи засвідчило їхню функціональну коректність, відповідність заданим очікуванням, а також підтвердило стабільність поведінки системи при різних типах вхідних даних і сценаріях взаємодії. Основна увага зосереджена на забезпеченні модульного покриття основних елементів архітектури, включно з механізмами маршрутизації, обробки повідомлень, буферизації, довірчих таблиць, логіки маршрутизаторів, роботи віртуальних вузлів і базової користувацької взаємодії через UI.

Спроєктовані мок-тести дозволили імітувати міжвузлову взаємодію, моделювати передачу повідомлень у віртуалізованому середовищі, верифікувати функціональність алгоритмів маршрутизації (як модифікованого Epidemic, так і евристичного гібридного) без прив’язки до реальної мережі, а також перевірити, як вузли реагують на перевантаження буфера, зміну параметрів TTL, дублювання повідомлень або некоректні сценарії. Такий підхід забезпечив можливість ізольованої перевірки кожного модуля системи за визначених умов та дозволив локалізувати потенційні дефекти на ранніх етапах розроблення.

Згідно з результатами запуску тестів (описані у Додатку Б), усі протестовані функції успішно проходять перевірку у межах своїх сценаріїв використання. Всі об’єкти системи та їх функціонал які підлягають тестуванню успішно проходять перевірку і показують поведінку, яка запрограмована у мок-тести як очікувана поведінка від компонентів системи. Компоненти, що реалізують моделювання мережевого шару мають незначні в контексті системи затримки, тому всі затримки які будуть у готовому застосунку будуть повністю контрольовані логікою системи.

6 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

У цьому розділі описана процес створення прототипу (MVP), принцип демонстрації його роботи, а також здійснена первинна оцінка продуктивності системи на основі віртуального моделювання.

6.1 Розроблення прототипу

Заключним етапом розроблення стало інтегрування усіх компонентів інформаційної системи у єдиний застосунок, який виконує повний цикл обробки повідомлень у змодельованій децентралізованій мережі.

З огляду на складність проведення повноцінного експерименту в умовах реальної мережі з обмеженим з’єднанням, комунікаційний модуль реалізований у вигляді віртуального шару. Це рішення дало змогу здійснювати запуск віртуальних вузлів на одному фізичному пристрої, незалежно змінювати топологію з’єднань та контролювати стан мережі в динаміці.

На момент створення MVP усі важливі модулі системи (агент маршрутизації, модуль доставки повідомлень, черга повідомлень, базовий інтерфейс обміну) реалізовані та протестовані окремо. З урахуванням цього, процес збирання прототипу зводився до:

– впровадження та модифікація модуля VirtualNode для моделювання логіки фізичних вузлів та зручної ініціалізації;

– контейнеризації компонентів (черга, маршрутизатор, логи, інтерфейс) у межах кожного вузла ізольовані з відповідним VirtualNode;

– реалізації механізму ініціалізації мережі з довільною кількістю вузлів;

– встановлення протоколу взаємодії між віртуальними вузлами через змодельовані канали.

Додатково приділена увага створенню користувацького інтерфейсу, який дозволяє контролювати стан мережі, запускати та зупиняти передачу повідомлень, а також переглядати журнали подій кожного вузла окремо.

6.2 Демонстрація роботи

Перша дія необхідна для роботи та взаємодії з системою – запуск та встановлення зв’язків. Для цього сформований JSON-файл з базовою конфігурацією мережі та вузлів. До налаштувань які можна задати належать:

– топологія мережі, початковий набір вузлів та зв’язки між ними;

– довжина буферу за замовченням;

– максимальна кількість зв’язків між вузлами;

– алгоритм маршрутизації, що запускається на вузлах системи;

– time-to-live за замовченням для повідомлень кожного вузла;

– специфічні параметри для цільового вузла.

net_config.json > ...

```json
{
    "defaults": {
        "buffer_size": 100,
        "max_connections": 4,
        "routing_algorithm": "epidemic",
        "message_ttl": 60
    },
    "topology": [
        ["111", "222"],
        ["222", "333"],
        ["222", "444"],
        ["333", "444"],
        ["444", "555"]
    ],
    "111": {
        "max_connections": 2,
        "buffer_size": 500,
        "message_ttl": 5
    }
}
```

Рисунок 6.1 – Приклад JSON-файлу початкової конфігурації системи

Наступним кроком після підготовки файлу конфігурації є початкова ініціалізація системи. Вона виконується однією командою у терміналі, яка викликає виконання основного скрипту системи, ініціалізує окремі вузли майбутньої мережі та покроково утворює зв'язки між ними згідно наданої конфігурації. Виглядає вона аналогічно до прикладу на рисунку 6.2. В результаті успішної ініціалізації у терміналі має бути результат зображений на рисунку 6.3.

Рисунок 6.2 – Команда для ініціалізації системи з початковою конфігурацією описаною у файлі "./net_config.json"

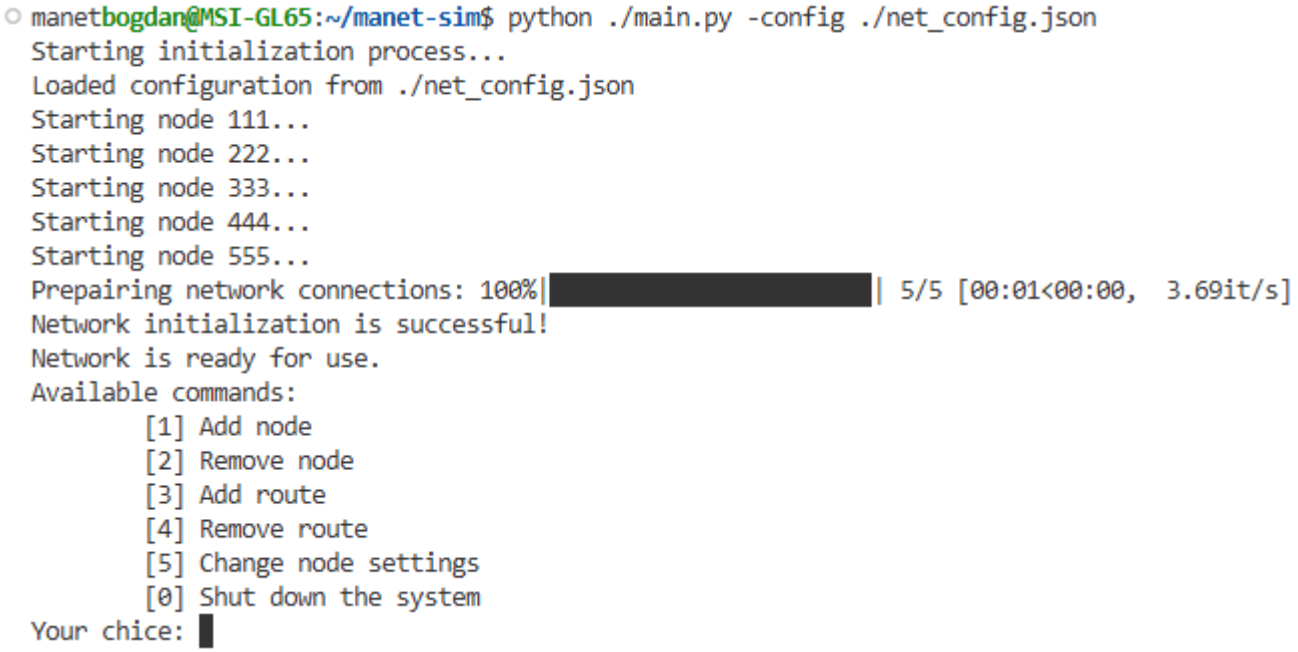

Рисунок 6.3 – Результат ініціалізації системи з заданою початковою конфігурацією у файлі "./net_config.json"

Після успішної ініціалізації у користувача є можливість користуватися системою та її мережею через меню терміналу. Для цього треба обрати опцію з запропонованих системою. Першою є опція перегляду списку вузлів, запущених у системі. Приклад її використання зображений на рисунку 6.4.

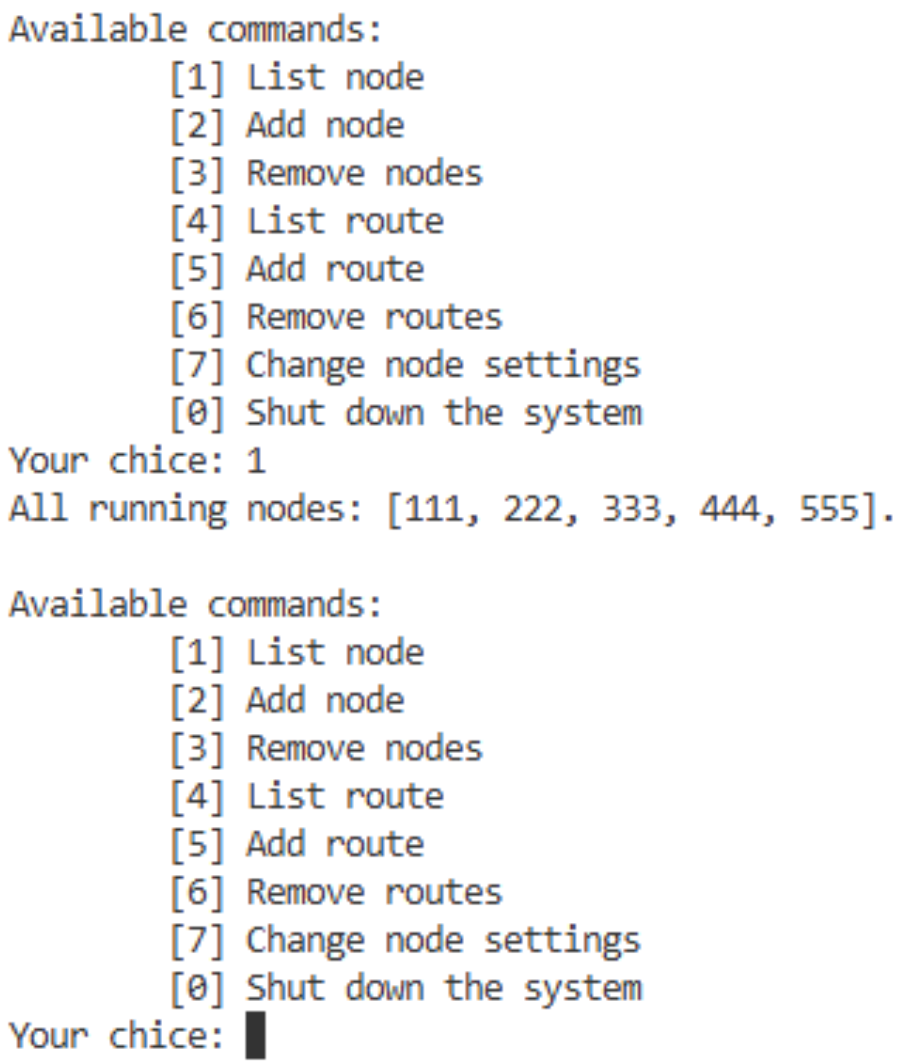

Рисунок 6.4 – Демонстрація використання команди для перегляду списку запущених вузлів

Наступною командою, доступною користувачеві, є додавання вузлів. Користувач обирає опцію в меню, система запитує індекс нового вузла, після вводу номеру вузла система валідує його та повідомляє про успішність операції. Якщо є помилка в номері введеному користувачем – результатом буде переривання операції з повідомленням про помилку (рисунок 6.5). Якщо введений номер існуючого вузла, то система проінформує що введений номер існує та перерве операцію (рисунок 6.6). Якщо введений номер не співпадає з будь-яким існуючим у системі, то система проінформує про те, що операція пройшла успішно (рисунок 6.7).

```
Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice: 2
Type node number first: qwerty
ERROR: Wrong input! The node numben needs to be integer value in range 100-999!

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.5 – Демонстрація додавання вузла з не валідним номером

```
Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice: 2
Type node number first: 111
ERROR: This node is already exists! The node number needs to be unique and do not match with any existing one!

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.6 – Демонстрація додавання вузла з номером вже існуючого вузла

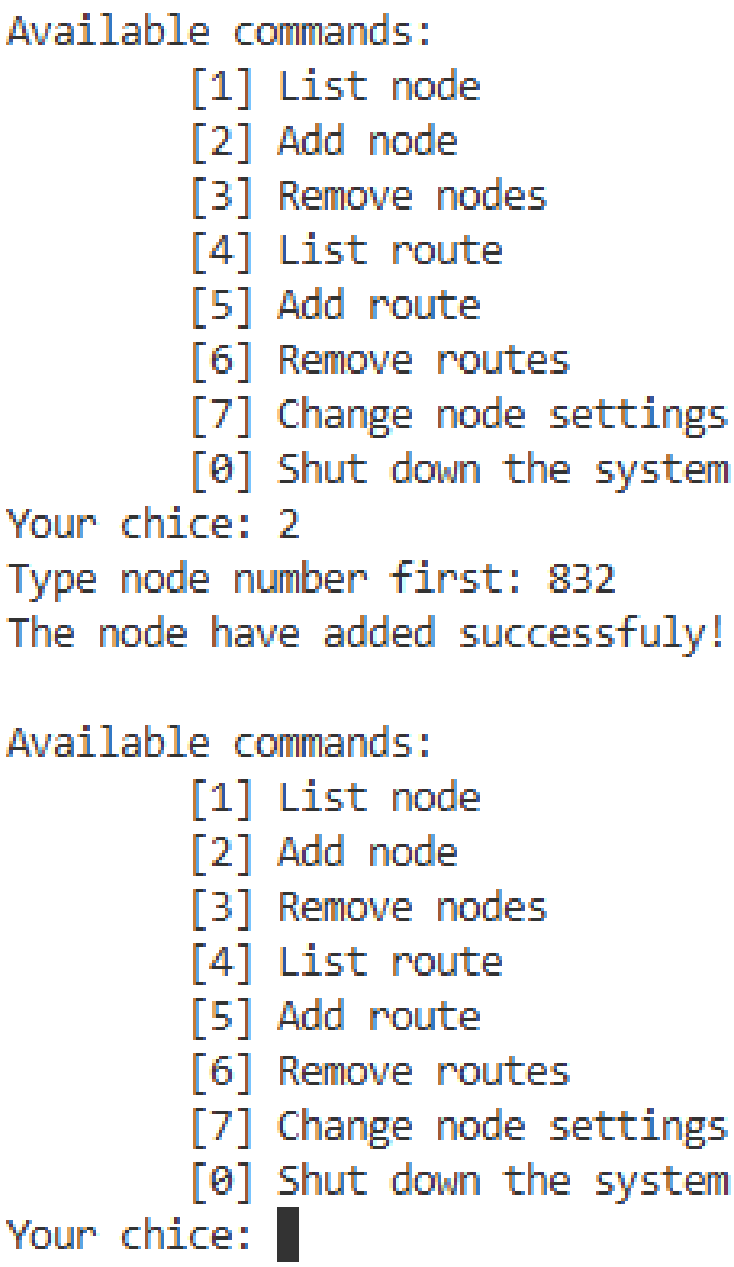

Рисунок 6.7 – Демонстрація успішного додавання вузла до мережі

Окрім функції додавання вузлів, реалізована функція їх видалення. Ця функція дає можливість видаляти наявні вузли з мережі. Вона не обмежує видалення вузла навіть якщо він має зв’язки з іншими вузлами системи. В разі видалення такого вузла система спочатку розриває зв’язки з сусідніми вузлами, а після цього видаляє і вузол. Результат таких дій зображений на рисунку 6.8. Якщо зазначеного вузла не існує, результат буде аналогічний тому, що на рисунку 6.9.

```
Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice: 3
Type node number first: 333
The node was removed successfuly!

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.8 – Демонстрація видалення вузла з мережі

```
Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice: 3
Type node number first: 999
ERROR: Node with number 999 do not exist! Try to list existing nodes.

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.9 – Демонстрація спроби видалити неіснуючий вузол

Наступною функцією є відображення списку зв’язків. Ця функція повертає список зв’язків між вузлами мережі (рисунок 6.10).

```
Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice: 4
All links in the network: [
        111 - 222
        222 - 333
        222 - 444
        333 - 444
        444 - 555
]

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.10 – Демонстрація відображення усіх зв’язків у мережі

Наступною функцією є функція додавання зв’язку. Для цього потрібно ввести два номери вузлів, щоб утворити зв’язок між ними. Якщо одного з вузлів не існує, результат буде аналогічний рисунку 6.11. Якщо введений вже існуючий зв’язок, система перерве операцію (рисунок 6.12). Аналогічно операція буде неуспішна, якщо користувач спробує зв’язати вузол сам з собою (рисунок 6.13). Якщо все введено вірно, система повідомить про успіх (рисунок 6.14).

```
Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice: 5
Type first node number: 111
Type second node number: 999
ERROR: Cannot connect this two nodes! Node with number 999 do not exist!

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.11 – Демонстрація спроби з’єднати існуючий та неіснуючий вузли у мережі

```
Available commands:
        [1] List nodes
        [2] Add node
        [3] Remove node
        [4] List routes
        [5] Add route
        [6] Remove route
        [7] Change node settings
        [0] Shut down the system
Your chice: 5
Type first node number: 111
Type second node number: 222
ERROR: Those nodes are already connected! Try to list existing routes to proof it

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.12 – Демонстрація спроби додавання вже існуючого з’єднання

```
Available commands:
        [1] List nodes
        [2] Add node
        [3] Remove node
        [4] List routes
        [5] Add route
        [6] Remove route
        [7] Change node settings
        [0] Shut down the system
Your chice: 5
Type first node number: 111
Type second node number: 111
ERROR: You are trying to connect node to itself! Not very smart!

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.13 – Демонстрація спроби зв’язати вузол сам із собою

```
Available commands:
        [1] List nodes
        [2] Add node
        [3] Remove node
        [4] List routes
        [5] Add route
        [6] Remove route
        [7] Change node settings
        [0] Shut down the system
Your chice: 5
Type first node number: 111
Type second node number: 333
Nodes are connected successfully!

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.14 – Демонстрація успішної спроби з’єднання двох вузлів

Наступна функція меню є функція видалення зв’язків. Аналогічно з додаванням вузлів, потрібно ввести номери відповідних вузлів. Якщо намагатися прибрати неіснуючий зв’язок (в тому числі зв’язок вузла сам до себе або з неіснуючим вузлом) система буде повідомляти, що такого зв’язку не існує (рисунки 6.15 та 6.16). В разі успішності операції, результат буде як на рисунку 6.17.

```
Available commands:
        [1] List nodes
        [2] Add node
        [3] Remove node
        [4] List routes
        [5] Add route
        [6] Remove route
        [7] Change node settings
        [0] Shut down the system
Your chice: 6
Type first node number: 111
Type second node number: 555
There are no such route!

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.15 – Демонстрація спроби видалити неіснуючий зв’язок

```
Available commands:
        [1] List nodes
        [2] Add node
        [3] Remove node
        [4] List routes
        [5] Add route
        [6] Remove route
        [7] Change node settings
        [0] Shut down the system
Your chice: 6
Type first node number: 111
Type second node number: 111
There are no such route!

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.16 – Демонстрація спроби прибрати зв'язок вузла самого до себе

```
Available commands:
        [1] List nodes
        [2] Add node
        [3] Remove node
        [4] List routes
        [5] Add route
        [6] Remove route
        [7] Change node settings
        [0] Shut down the system
Your chice: 6
Type first node number: 222
Type second node number: 333
Route deleted successfully!

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.17 – Демонстрація успішного видалення зв’язку між вузлами

Наступна функція меню відповідає за модифікацію налаштувань вузла. При виборі даного пункту меню система просить задати номер вузла який треба переналаштувати. При вводі неправильного номеру вузла буде відповідне повідомлення як на рисунку 6.18. У іншому випадку буде доступний перелік параметрів з нумеруванням для вибору пункту для редагування (рисунок 6.19). При вводі нового значення система повідомить про успішно проведену операцію (рисунок 6.19).

```
Available commands:
        [1] List nodes
        [2] Add node
        [3] Remove node
        [4] List routes
        [5] Add route
        [6] Remove route
        [7] Change node settings
        [0] Shut down the system
Your chice: 7
Type node number: 999
Such node do not exist!

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice:
```

Рисунок 6.18 – Демонстрація вводу неіснуючого вузла у виборі вузла для зміни налаштувань вузла

```
Available commands:
        [1] List nodes
        [2] Add node
        [3] Remove node
        [4] List routes
        [5] Add route
        [6] Remove route
        [7] Change node settings
        [0] Shut down the system
Your chice: 7
Type node number: 222
Node 222 settings:
{
        [1] "buffer_size": 100,
        [2] "max_connections": 4,
        [3] "routing_algorithm": "epidemic",
        [4] "message_ttl": 60
}
What to change (0 - nothing): 3
Type new value: heuristic
Setting "routing_algorithm" successfully changed to "heuristic" for node 222!

Available commands:
        [1] List node
        [2] Add node
        [3] Remove nodes
        [4] List route
        [5] Add route
        [6] Remove routes
        [7] Change node settings
        [0] Shut down the system
Your chice: 1
```

Рисунок 6.19 – Демонстрація успішного вибору налаштування та зміни значення вибраного параметра

Останньою функцією меню ядра системи є його вимкнення. Коли ця функція вибрана, система починає поступово закривати всі зв’язки, потім вимикати всі вузли і після цього повністю зупиняється. Приклад того, як відбувається вимкнення системи зображений на рисунку 6.20.

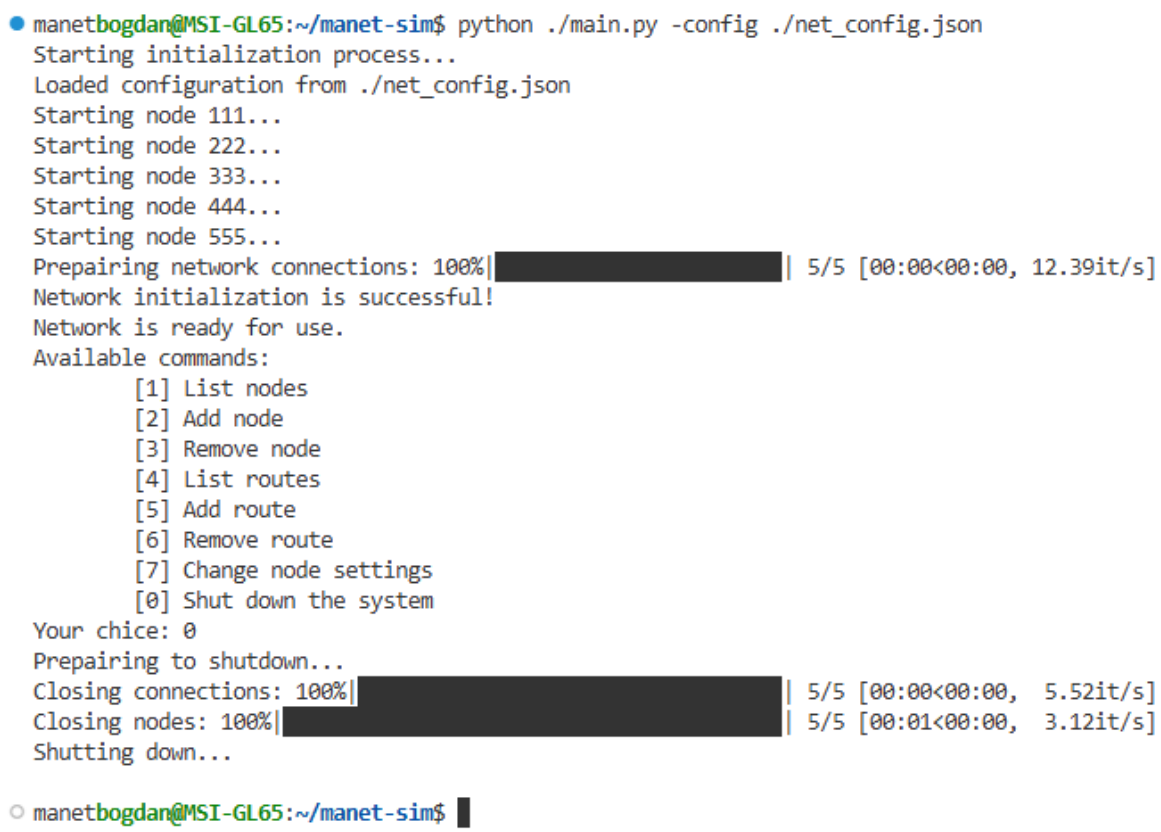

Рисунок 6.20 – Демонстрація вимкнення системи

Наступним компонентом, з яким взаємодіє користувач – це інтерфейс користувача. Інтерфейс користувача відкривається як веб-сторінка у браузері. Інтерфейс користувача, який не отримував та не відсилав жодного повідомлення виглядає як на рисунку 6.21.

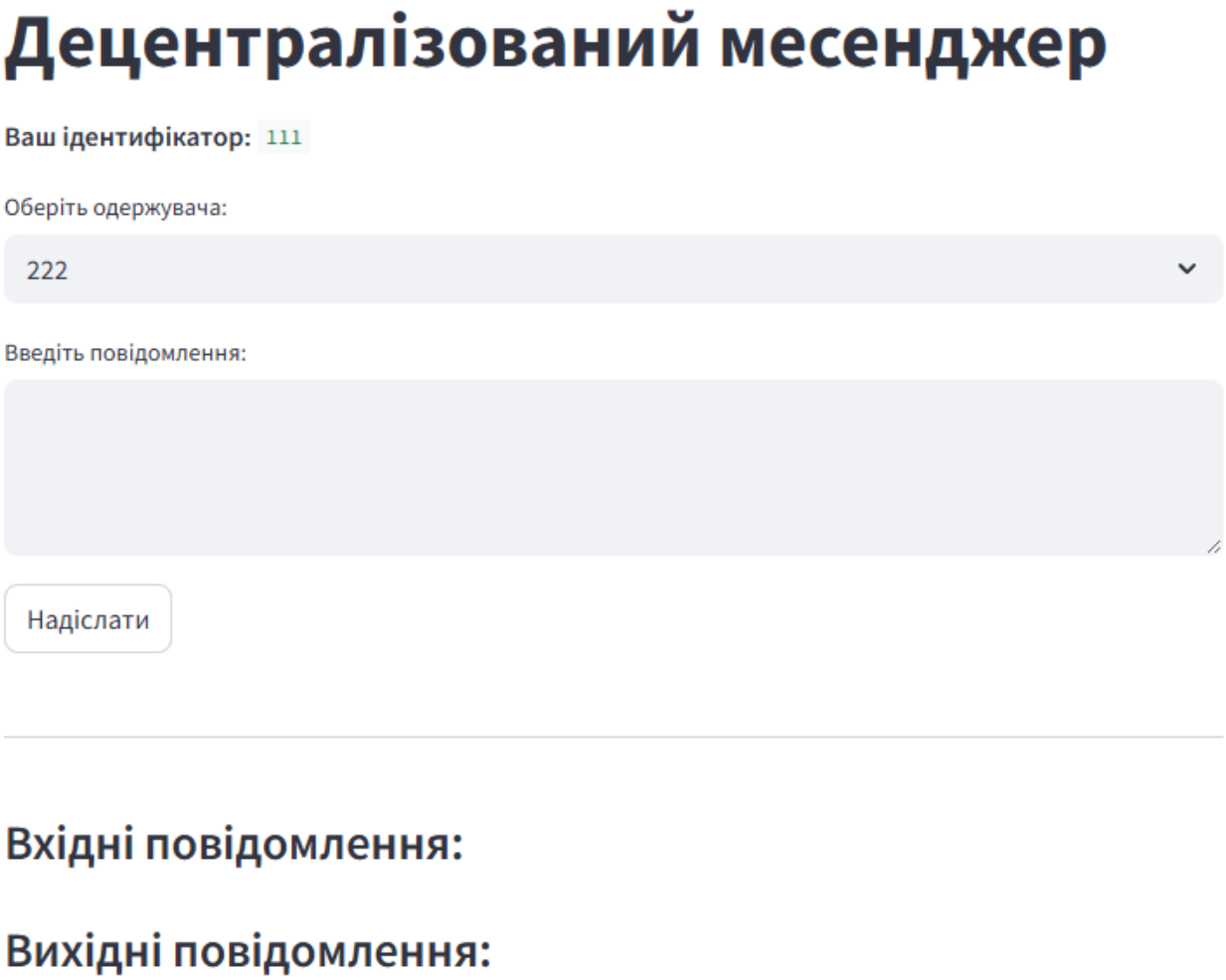

Рисунок 6.21 – Скриншот інтерфейсу користувача чату

Інтерфейс користувача містить: номерний ідентифікатор користувача, поле для вибору ідентифікатору одержувача повідомлення, текстове поле для повідомлення, кнопку «Надіслати» та розділи вхідних та вихідних повідомлень.

Щоб надіслати повідомлення користувач вибирає у випадаючому списку «Оберіть одержувача» ідентифікатор того користувача, якому користувач бажає надіслати повідомлення. Далі користувач вводить повідомлення у текстове поле та натискає кнопку «Надіслати». Після надсилання, повідомлення з’являється в історії надісланих повідомлень, як показано на рисунку 6.22.

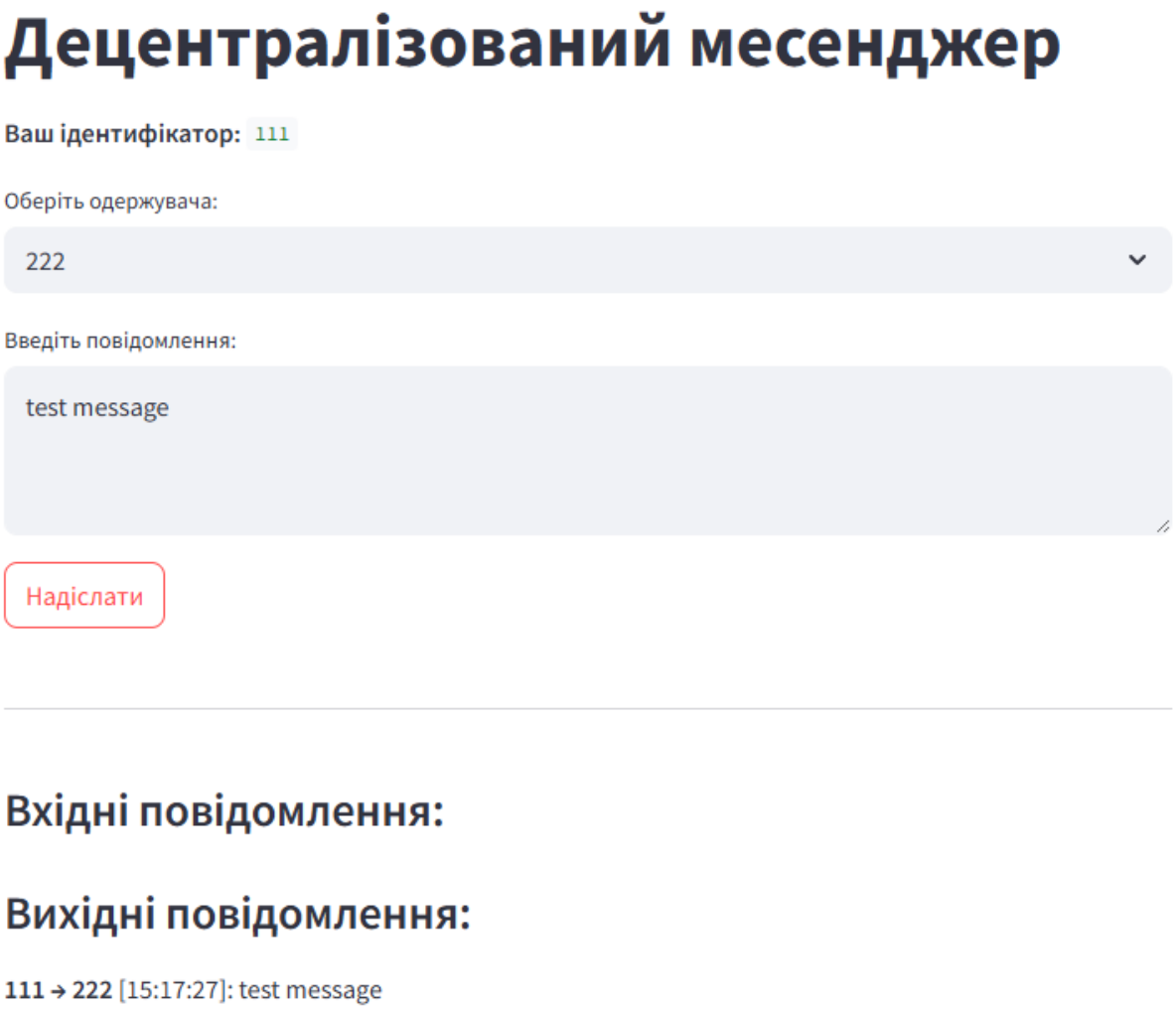

Рисунок 6.22 – Демонстрація надісланого повідомлення

Після відправки повідомлення проходить шлях до отримувача. Прослідкувати пакет можна за логами, що виконують вузли при кожній події. На рисунку 6.23 зображено приклад логування відправки повідомлення по мережі від вузла 111 до вузла 555. При отриманні повідомлення з’являється в історії вхідних повідомлень як на рисунку 6.24.

111-555-msg-log.json > ...


```
[
  {"step": 1,"node": "111","action": "send","to": "222","message": {"from": "111","to": "555","ttl": 60}
  },
  {"step": 2,"node": "222","action": "receive","from": "111","message": {"from": "111","to": "555","ttl": 59},
    "routing_decision": ["333", "444"],
    "router": "epidemic"
  },
  {"step": 3,"node": "444","action": "receive","from": "222","message": {"from": "111","to": "555","ttl": 58},
    "routing_decision": ["555"],
    "router": "epidemic"
  },
  {"step": 4,"node": "333","action": "receive","from": "222","message": {"from": "111","to": "555","ttl": 58},
    "routing_decision": ["444"],
    "router": "epidemic",
    "result": "ignored"
  },
  {"step": 5,"node": "555","action": "receive","from": "444","message": {"from": "111","to": "555","ttl": 57},
    "result": "delivered"
  }
]
```


Рисунок 6.23 – Приклад логів надсилання пакета і маршрутизації його методом епідемічної маршрутизації

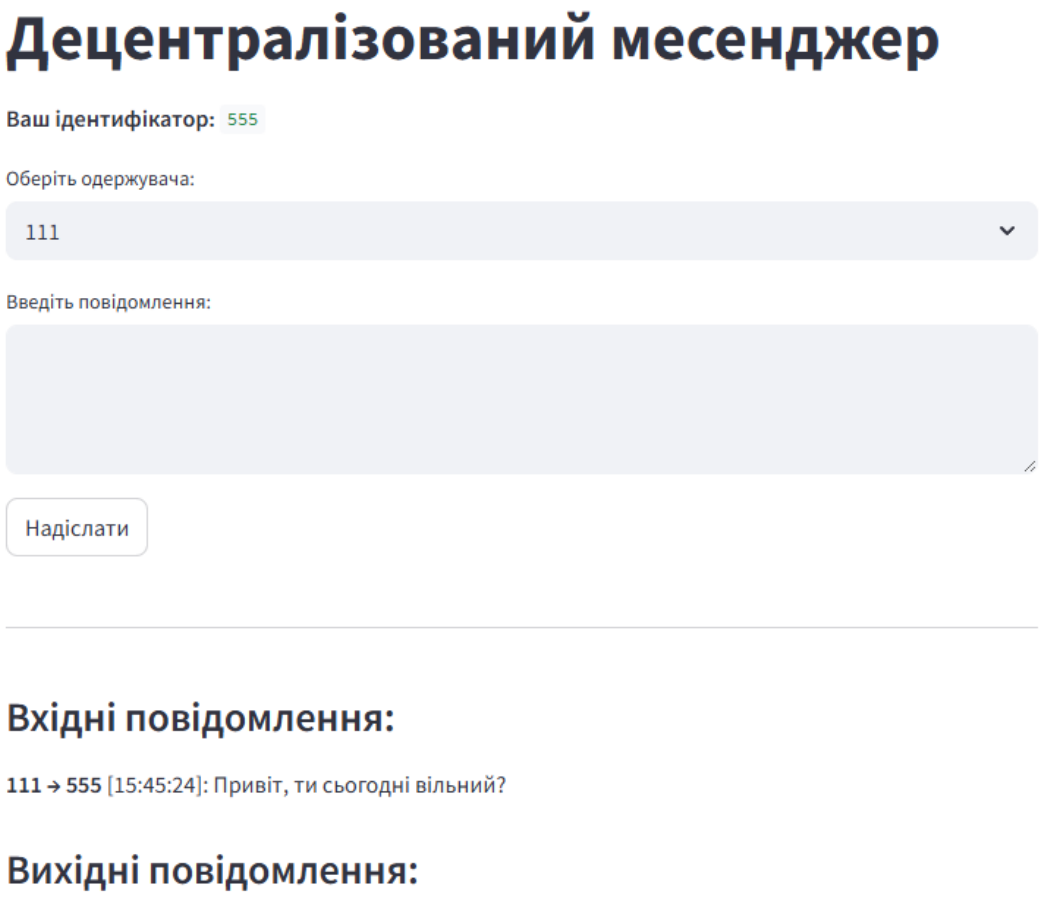

Рисунок 6.24 – Демонстрація отримання повідомлення користувачем

ВИСНОВКИ

У межах виконаного дипломного проєкту виконані дослідження, проєктування, математичне обґрунтування, програмна реалізація та тестування інформаційної системи децентралізованого обміну повідомленнями з адаптивною маршрутизацією у нестабільних мережах.

На етапі аналізу предметної області окреслена актуальність і складність задач маршрутизації в умовах нестатичних динамічних мереж. Дана класифікація алгоритмів маршрутизації та здійснений порівняльний аналіз сучасних підходів, таких як DSDV, AODV, DSR, OLSR, ZRP, GPSR. Виявлено, що жоден з існуючих протоколів не є універсальним, а отже для специфічного застосування потрібні адаптовані алгоритми з урахуванням обмежень середовища.

У розділі проєктування розроблена архітектура інформаційної системи, що охоплює логічну та функціональну структуру однорангової мережі вузлів. Виокремлені ключові компоненти: модуль маршрутизації, черга повідомлень, інтерфейс користувача, віртуальний мережевий модуль. Система побудована з урахуванням вимог масштабованості, адаптивності до змін, незалежності від фізичного носія передачі та спрощення для MVP-реалізації.

Особлива увага приділена математичному обґрунтуванню алгоритмів. Побудована темпоральна модель мережі, формалізований модифікований варіант алгоритму Epidemic Routing та розроблений гібридний евристичний алгоритм маршрутизації. Обидва алгоритми реалізують підходи з високою імовірністю доставки повідомлень у складних умовах, але при цьому суттєво різняться філософією функціонування та ефективністю використання ресурсів.

Здійснена реалізація MVP системи у вигляді модульного застосунку з використанням технологій Python, ZeroMQ, Streamlit. Усі компоненти системи протестовані із застосуванням сучасних підходів до unit- і integration-тестування (бібліотека pytest). Результати верифікації свідчать про функціональну коректність системи, її стійкість до типових відмов та узгодженість з поставленими вимогами.

Тестові логи, які зібрано у Додатку Б, демонструють високу якість реалізації й охоплення ключових сценаріїв.

Щодо маршрутизаторів, слід зазначити, що обидва реалізовані алгоритми показали себе як ефективні рішення у своїх класах. Модифікований Epidemic Routing має перевагу у максимально можливій гарантованості доставки в умовах повної невизначеності та є кращим вибором для надзвичайно простих автономних платформ із обмеженими комунікаційними можливостями. У свою чергу, гібридний евристичний алгоритм має більший потенціал у складніших мережах: він демонструє адаптацію до зміни топології, використовує логіку пріоритетного пересилання через надійні вузли, скорочує дублювання, і в умовах обмежень на пропускну здатність, енергоспоживання або низьку ймовірність доставки – показує помірну перевагу. Попри це, він не потребує настільки складних розрахунків, сучасні платформи з помірними обчислювальними ресурсами або з акцентом на автономність та високу самостійність мають достатньо запасу потужності для запуску такого маршрутизаційного модулю.

В майбутній перспективі запропонована система може стати основою для реального месенджера, придатного до використання в умовах відсутності інфраструктури – для рятувальників, військових, волонтерів, екологів тощо. У майбутньому розроблена система може бути адаптована під вбудовані платформи (ESP32, Raspberry Pi), інтегрована з мобільними застосунками або спеціалізованими пристроями. Перспективним напрямом є розширення системи в бік побудови самоконфігурованих мереж (self-forming mesh), що підтримують обмін повідомленнями навіть у надзвичайних умовах або при відсутності покриття. Також можливим є використання SNN (спайкових нейронних мереж) або RL (підкріпленого навчання) для подальшого інтелектуального вибору маршрутів у реальному часі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Di Caro, G., & Dorigo, M. AntNet: A Mobile Agents Approach to Adaptive Routing: Technical Report IRIDIA/97-12. Université Libre de Bruxelles, 1997. URL: https://www.gm.th-koeln.de/~hk/lehre/ala/ws0506/Praktikum/Projekt/B_gelb/Prim%E4rliteratur%20-%20AntNet.pdf (дата звернення: 16.06.2025).
2. Perkins, C. E., & Bhagwat, P. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers // ACM SIGCOMM Computer Communication Review. 1994. Vol. 24, № 4. С. 234–244. URL: https://dl.acm.org/doi/pdf/10.1145/190809.190336 (дата звернення: 16.06.2025).
3. Perkins, C., Belding-Royer, E., & Das, S. Ad hoc On-Demand Distance Vector (AODV) Routing: RFC 3561. 2003. URL: https://datatracker.ietf.org/doc/rfc3561/ (дата звернення: 16.06.2025).
4. Johnson, D. B., & Maltz, D. A. Dynamic Source Routing in Ad Hoc Wireless Networks // Mobile Computing. Springer, 1996. С. 153–181.
5. Djenouri, D., Bagaa, M., Doudou, M., & Djenouri, Y. Network Routing Protocols for Wireless Sensor Networks: A Review // International Journal of Ad Hoc and Ubiquitous Computing. 2013. Vol. 12, № 2. С. 78–93.
6. Fall, K. A Delay-Tolerant Network Architecture for Challenged Internets // ACM SIGCOMM. 2003.
7. Merz, R., & Burgstaller, M. A Survey of Mesh Routing Protocols for Low-Power and Lossy Networks // Computer Communications. 2022. Vol. 189. С. 16–33.
8. Meshtastic Project Documentation [Електронний ресурс]. – Режим доступу: https://meshtastic.org/docs (дата звернення: 16.06.2025).
9. goTenna Mesh Developer Guide [Електронний ресурс]. – Режим доступу: https://gotenna.com/ (дата звернення: 16.06.2025).
10. Briar Project code repository [Електронний ресурс]. – Режим доступу: https://code.briarproject.org/briar/briar (дата звернення: 16.06.2025).

11. Clausen, T., Jacquet, P. Optimized Link State Routing Protocol (OLSR) : IETF RFC 3626. 2003. 75 с.

12. Haas, Z., Pearlman, M., Samar, P. The Zone Routing Protocol (ZRP) for Ad Hoc Networks : IETF Internet Draft, draft-ietf-manet-zone-zrp-04. 2002. 25 с.

13. van Rossum, G., Drake, F. L. The Python Language Reference Manual. Network Theory Ltd., 2009. 140 с.

14. Hintjens, P. ZeroMQ: Messaging for Many Applications. O'Reilly Media, 2013. 384 с.

15. PyZMQ documentation [Електронний ресурс]. – Режим доступу: https://pyzmq.readthedocs.io/ (дата звернення: 16.06.2025).

16. Streamlit documentation [Електронний ресурс]. – Режим доступу: https://docs.streamlit.io/ (дата звернення: 16.06.2025).

17. Krekel, H. pytest documentation [Електронний ресурс]. – Режим доступу: https://docs.pytest.org/en/stable/ (дата звернення: 16.06.2025).

18. Python documentation [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/ (дата звернення: 16.06.2025).

19. pytest documentation [Електронний ресурс]. – Режим доступу: https://docs.pytest.org/en/stable/ (дата звернення: 16.06.2025).

20. Vahdat, A., & Becker, D. (2000). Epidemic Routing for Partially-Connected Ad Hoc Networks [Електронний ресурс] / Technical Report CS-2000-06. – Durham, NC: Duke University. – Режим доступу: https://cs.duke.edu/~vahdat/papers/epidemic.pdf (дата звернення: 16.06.2025).

21. Lindgren, A., Doria, A., & Schelén, O. (2003). Probabilistic Routing in Intermittently Connected Networks // ACM SIGMOBILE Mobile Computing and Communications Review. – 2003. – Vol. 7, № 3. – С. 19–20. – DOI: 10.1145/961268.961272

22. RFC 4838: Cerf, V. G. et al. Delay-Tolerant Networking Architecture [Електронний ресурс]. – Internet Engineering Task Force (IETF), 2007. – 37 с. – Режим доступу: https://www.rfc-editor.org/rfc/rfc4838.html (дата звернення: 16.06.2025).

23. RFC 6693: Lindgren, A. et al. Probabilistic Routing Protocol for Intermittently Connected Networks (PRoPHET) [Електронний ресурс]. – Internet Engineering Task Force (IETF), 2012. – 45 с. – Режим доступу: https://www.rfc-editor.org/rfc/rfc6693.html

24. Spyropoulos, T., Psounis, K., & Raghavendra, C. S. (2005). Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks // Proceedings of ACM SIGCOMM Workshop on Delay-Tolerant Networking. – Philadelphia: ACM Press, 2005. – С. 252–259. – DOI: 10.1145/1067170.1067174

25. Di Caro, G. A., & Dorigo, M. (1998). AntNet: Distributed Stigmergetic Control for Communications Networks // Journal of Artificial Intelligence Research. – Vol. 9. – С. 317–365. – DOI: 10.1613/jair.530

26. Balasubramanian, A., Levine, B. N., & Venkataramani, A. (2007). DTN Routing as a Resource Allocation Problem // ACM SIGCOMM. – 2007. – С. 373–384. – DOI: 10.1145/1282380.1282427

27. Su, J., & Ma, H. (2015). An Enhanced PRoPHET Routing Protocol Based on Multiple Decision Factors // Sensors. – 2015. – Vol. 15. – С. 7116–7136. – DOI: 10.3390/s150407116

28. Kempe, D., Kleinberg, J., & Kumar, A. (2000). Connectivity and Inference Problems for Temporal Networks // Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC 2000). – Portland: ACM Press, 2000. – С. 504–513. – DOI: 10.1145/335305.335387

29. Casteigts, A., Flocchini, P., Quattrociocchi, W., & Santoro, N. Time-Varying Graphs and Dynamic Networks : Research Report RR-7277 [Електронний ресурс] / INRIA. – 2010. – 41 с. – Режим доступу: https://inria.hal.science/inria-00494473/PDF/RR-7277.pdf

30. Jain, S., Fall, K., & Patra, R. (2004). Routing in a Delay Tolerant Network // Chapter in: Delay-Tolerant Networks. – Temple University. – [Електронний ресурс]. – Режим доступу: https://cis.temple.edu/~jiewu/routing-DTN-chapter%5b1%5d.pdf

31. On the Routing in Flying Ad hoc Networks - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/MANET-VANET-and-FANET_fig1_300337987