ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «МЕРЕЖЕВИЙ ЕМУЛЯТОР ДЛЯ ДОСЛІДЖЕННЯ ПРОТОКОЛІВ БЕЗДРОТОВОГО ЗВ'ЯЗКУ»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми «Комп'ютерна інженерія»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Ростислав ТАТУСЬКО*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач(ка) вищої освіти гр. КІД-42
Ростислав ТАТУСЬКО
Ім'я, ПРІЗВИЩЕ

Керівник: Віталій ВОЛОХІН
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент:
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерна інженерія

Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Татусько Ростислав Ігорович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Мережевий емулятор для дослідження протоколів бездротового зв'язку»
керівник кваліфікаційної роботи: Віталій ВОЛОХІН, канд. техн. наук, доцент,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025р. №56
2. Строк подання кваліфікаційної роботи «05» червня 2025р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, стандарти та протоколи бездротового зв'язку (Wi-Fi, IEEE 802.11), архітектура мережевих емуляторів, особливості моделювання мереж у NS-3, методи збору та аналізу логів трафіку, сценарії симуляції та параметри тестування, інструменти візуалізації та оцінки результатів.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Теоретичні основи емуляції бездротових мереж
    2. Архітектура та реалізація дослідницької системи
    3. Аналіз, порівняння та перспективи розвитку
5. Перелік ілюстративного матеріалу: *презентація*
    1. Мета, об'єкт, предмет дослідження
    2. Актуальність роботи
    3. Загальна архітектура запропонованої системи
    4. Розгортання віртуального ізольованого середовища
    5. Демонстрація функціонування системи (CLI)
    6. Демонстрація функціонування системи (gnuplot)
    7. Виявленні недоліки під час тестування
    8. Методи покращення існуючої системи

9. Порівняння запропонованої системи зі альтернативами
10. Висновки

6. Дата видачі завдання «24» лютого 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	25.02-19.03.2025	
2	Обробка матеріалу	20.03-02.04.2025	
3	Розробка теоретичної частини	03.04-07.04.2025	
5	Архітектура та реалізація дослідницької системи	08.04-27.04.2025	
6	Аналіз, порівняння та перспективи розвитку	28.04-11.05.2025	
7	Висновки за результатами аналізу	12.05-15.05.2025	
8	Розробка презентації	16.05-26.05.2025	
9	Передзахист	27.05-29.05.2025	
10	Здача в деканат	30.05-05.06.2025	

Здобувач(ка) вищої освіти _____ Ростислав ТАТУСЬКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Віталій ВОЛОХІН
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 89 стор., 33 рис., 9 табл., 35 джерел.

Мета роботи – спрощення дослідження протоколів бездротового зв'язку за рахунок використання моделі контейнеризованого середовища емуляції на основі мережевого симулятора NS-3.

Об'єкт дослідження – процес моделювання бездротової взаємодії вузлів у мережевому середовищі.

Предмет дослідження – програмні засоби емуляції протоколів бездротового зв'язку, що реалізовані в інструментарії NS-3 у поєднанні з контейнерними технологіями.

Короткий зміст роботи: У роботі розглянуто архітектуру та функціональні можливості симулятора NS-3 для дослідження протоколів бездротового зв'язку. Описано процеси розгортання середовища емуляції у контейнерах Docker, налаштування базового Wi-Fi-сценарію, модифікації параметрів топології та збору логів. Проведено запуск симуляції, збереження результатів, їх попередній аналіз та візуалізацію. Особливу увагу приділено забезпеченню відтворюваності експериментів шляхом ізоляції середовища запуску та автоматизації серійних тестів.

Здійснено порівняння NS-3 з іншими інструментами дослідження мереж — OMNeT++, Mininet-WiFi, GNS3 — за критеріями підтримки протоколів, гнучкості сценаріїв, простоти використання. Окреслено потенційні напрямки вдосконалення симуляційного середовища: реалізація веб-інтерфейсу для користувача, запуск на хмарних платформах, створення шаблонів для освітніх задач і включення в CI/CD-процеси.

КЛЮЧОВІ СЛОВА: NS-3, Wi-Fi, LoRaWAN, БЕЗДРОТОВІ ПРОТОКОЛИ, ДОКЕР, СИМУЛЯЦІЯ, КОНТЕЙНЕРИЗАЦІЯ, МЕРЕЖЕВИЙ ЕМУЛЯТОР, ТОПОЛОГІЯ, СЦЕНАРІЙ, ВІЗУАЛІЗАЦІЯ, АНАЛІЗ, ПРОДУКТИВНІСТЬ.

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЕМУЛЯЦІЇ БЕЗДРОТОВИХ МЕРЕЖ 9
1.1 Основи протоколів бездротового зв’язку 9
1.2 Методи дослідження мереж: емуляція, симуляція, віртуалізація 17
1.3 Особливості мережевої емуляції в середовищі Docker 24
1.4 Архітектура та модулі NS-3 31
1.5 Можливості моделювання бездротових мереж в NS-3. 38
РОЗДІЛ 2. АРХІТЕКТУРА ТА РЕАЛІЗАЦІЯ ДОСЛІДНИЦЬКОЇ СИСТЕМИ 45
2.1 Загальна архітектура дослідницької системи 45
2.2 Налаштування та розгортання середовища NS-3 у Docker 53
2.3 Алгоритми проведення емуляцій протоколів Wi-Fi/LoRaWAN 59
2.4 Реалізація дослідного прикладу та демонстрація 67
РОЗДІЛ 3. АНАЛІЗ, ПОРІВНЯННЯ ТА ПЕРСПЕКТИВИ РОЗВИТКУ 74
3.1 Аналіз результатів та методи покращення моделі 74
3.2 Порівняння з іншими інструментами дослідження 81
3.3 Можливі напрями розвитку та удосконалення 89
ВИСНОВКИ 96
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 97
Додаток А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 100

ВСТУП

Сучасні бездротові мережі відіграють ключову роль у забезпеченні зв’язку між пристроями в різних сферах — від побутових IoT-систем до складних промислових рішень. Ефективність і надійність передачі даних значною мірою залежать від характеристик протоколів: їхньої адаптивності до середовища, масштабованості та енергоспоживання. У контексті стрімкого зростання кількості підключених пристроїв особливо актуальним стає попереднє моделювання та оцінка мережевої поведінки до фактичного розгортання інфраструктури.

Для аналізу бездротових протоколів у наукових і навчальних цілях активно застосовуються мережеві емулятори, що дозволяють створювати складні сценарії без потреби в дорогому обладнанні. Серед них вирізняється NS-3 — симулятор з відкритою архітектурою, підтримкою сучасних специфікацій та широкою функціональністю для моделювання мереж Wi-Fi, LoRaWAN, LTE й інших протоколів. Завдяки модульності та розширюваності NS-3 широко використовується для вивчення мережевої взаємодії на різних рівнях.

Втім, робота з такими платформами вимагає технічної підготовки та часу: складність інтерфейсу, відсутність шаблонів типових топологій, потреба у ручному налаштуванні параметрів і сценаріїв, труднощі зі збереженням результатів та їх відтворюваністю — усе це ускладнює використання в освіті та швидкому прототипуванні.

Це формує потребу у побудові рішень, які поєднують можливості симуляторів на кшталт NS-3 із сучасними підходами до автоматизації, ізоляції середовищ і візуалізації. Такі рішення дозволять знизити поріг входу, спростити налаштування та забезпечити ефективні повторювані експерименти для дослідження бездротових протоколів зв’язку.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ЕМУЛЯЦІЇ БЕЗДРОТОВИХ МЕРЕЖ

1.1 Основи протоколів бездротового зв'язку

Рівень зв'язку, що використовується в сучасному світі, включає як традиційні дротові технології, так і новітні бездротові рішення. Історично основа телекомунікацій будувалася на мідних кабелях і оптоволоконних лініях, що простягаються на тисячі кілометрів, формуючи магістралі, які досі залишаються важливою частиною інфраструктури. Вони забезпечують стабільне з'єднання як у міських, так і в домашніх чи офісних умовах. Проте паралельно з цим стрімко розвиваються бездротові технології, які вже не є новинкою, а перетворилися на стандарт у багатьох сферах життя. Зменшення розмірів пристроїв, орієнтація на мобільність і автономність, потреба уникати фізичних з'єднань і кабельних обмежень — усе це сприяє широкому впровадженню бездротових протоколів зв'язку. Вони стали необхідністю, особливо в умовах мініатюризації пристроїв, де застосування кабелів фізично неможливе або економічно невигідне. Замість цього використовуються ефірні канали, які забезпечують передачу інформації між пристроями без будь-якого фізичного контакту [1-5].

Основні стандарти бездротових протоколів (Wi-Fi, LoRaWAN, Zigbee). Кожен бездротовий протокол розробляється з урахуванням особливостей середовища використання, кількості пристроїв у мережі, вимог до енергоспоживання та обсягу даних, які потрібно передавати. Існує безліч стандартів, однак найпоширенішими і такими, що стали фундаментом для побудови сучасних IoT-систем і локальних мереж, є Wi-Fi, Zigbee та LoRaWAN. Їх використання визначається конкретними потребами: високошвидкісний доступ, енергоефективність, масштабованість або дальність дії (див. табл. 1.1).

Таблиця 1.1 – Порівняння основних протоколів бездротового зв’язку за характеристиками

Протокол	**Частота, МГц**	**Дальність, м**	**Швидкість передачі**	**Енергоспоживання**	**Призначення**
Wi-Fi	2400–5800	до 100	до 1 Гбіт/с	Високе	Високошвидкісний доступ
LoRaWAN	868 / 915	до 10000	до 50 Кбіт/с	Низьке	IoT, сенсори на великі відстані
Zigbee	2400	до 100	до 250 Кбіт/с	Дуже низьке	Розумний дім, автоматизація

Wi-Fi, створений на основі стандарту IEEE 802.11, є найбільш популярною технологією для локальних мереж з високою пропускною здатністю. Його переваги включають широку підтримку пристроями, простоту налаштування та інтеграції в існуючі мережі, а також високу швидкість передачі даних, яка у версії Wi-Fi 6 сягає 9,6 Гбіт/с. Він використовується практично скрізь: у будинках, офісах, громадських просторах, кав’ярнях. Його можна знайти у всіх сучасних смартфонах, ноутбуках, планшетах і навіть у багатьох моделях персональних комп’ютерів завдяки інтеграції модулів Wi-Fi у материнські плати. Проте високе енергоспоживання робить цю технологію непридатною для автономних енергонезалежних пристроїв, особливо в середовищах з обмеженим живленням [1-2].

У випадках, коли важлива мінімізація енергоспоживання, але при цьому потрібно під’єднати велику кількість сенсорів, доцільніше використовувати протокол Zigbee на основі стандарту IEEE 802.15.4. Цей протокол оптимізований для побудови мереж з великою кількістю пристроїв, що мають низьке енергоспоживання. Завдяки підтримці сітчастої топології (mesh), Zigbee дозволяє розширювати мережу без втрати стабільності, адже кожен пристрій може виконувати функцію ретранслятора, передаючи дані далі, що значно збільшує покриття. У центрі такої мережі зазвичай знаходиться координатор, який

забезпечує управління й маршрутизацію трафіку, тоді як інші пристрої можуть діяти автономно. Недоліком Zigbee є обмежена швидкість передачі даних — до 250 кбіт/с, що робить його непридатним для роботи з відео, аудіо чи іншими вимогливими типами трафіку, але цілком прийнятним для більшості сенсорів у системах розумного дому, промисловому моніторингу чи охоронних рішеннях [1-2].

Іншим протоколом, що також орієнтований на енергоефективність, є LoRaWAN — рішення для великих географічних покриттів із низьким енергоспоживанням, яке базується на модуляції LoRa. Його особливість полягає у здатності передавати невеликі обсяги даних на дуже великі відстані, що може сягати кількох кілометрів за умови прямої видимості, що дає змогу використовувати LoRaWAN у сільських районах, на підприємствах із розгалуженою інфраструктурою, у міських агломераціях з великою кількістю перешкод, де інші протоколи виявляються малоефективними. Хоча LoRaWAN не підтримує mesh-топологію, на відміну від Zigbee, він компенсує це високим радіусом дії та винятковою автономністю пристроїв, які можуть працювати на одній батареї роками. Основними недоліками LoRaWAN є ще нижча швидкість передачі даних порівняно з Zigbee, а також можливі затримки, обумовлені дальністю та низькою пропускною здатністю. Однак у контексті періодичної передачі телеметричних даних із сенсорів ці характеристики цілком прийнятні (див. рис. 1.1).



Рисунок 1.1 – Основні стандарти бездротового зв’язку

Кожен із цих протоколів є результатом ретельного технічного компромісу між швидкістю, дальністю, енергоспоживанням і складністю реалізації. Wi-Fi, завдяки своїй універсальності, залишається оптимальним вибором для сценаріїв з високими вимогами до швидкості й продуктивності, таких як передача медіа або забезпечення доступу до інтернету. Zigbee відзначається гнучкістю та надійністю у масштабованих IoT-мережах, орієнтованих на сенсори, керування освітленням, моніторинг температури тощо. LoRaWAN, у свою чергу, дозволяє створювати мережі на рівні міста чи навіть області, що охоплюють сотні пристроїв з мінімальними витратами енергії та інфраструктурними вимогами. Вибір конкретного протоколу завжди зумовлений специфікою задачі: від побудови домашньої системи автоматизації до моніторингу навколишнього середовища, контролю логістичних об’єктів чи управління інфраструктурними об’єктами [1-2].

Особливості передачі даних у бездротовому середовищі. Передача даних у бездротовому середовищі значно відрізняється від дротової, насамперед через високу залежність від зовнішніх факторів, які впливають на якість і стабільність зв’язку. Основними характеристиками такого середовища є підвищена вразливість до інтерференції, тобто до впливу перешкод, які можуть створюватися як іншими бездротовими пристроями, що працюють на тих самих частотах, так і фізичними об’єктами, які знаходяться в безпосередній близькості до передавача або приймача. Такий вплив часто призводить до втрати пакетів, зниження швидкості зв’язку або навіть до повного припинення передачі даних на певних ділянках, що в цілому негативно впливає на цілісність і достовірність інформації, яка передається між пристроями [3-4].

Крім цього, варто враховувати ще один важливий аспект – затухання сигналу, що особливо характерно для ситуацій, коли відстань між передавачем і приймачем збільшується. У таких випадках сигнал поступово слабшає, а фізичні перешкоди, такі як стіни або інші конструкції, здатні суттєво знижувати рівень сигналу до рівня, коли передача стає нестабільною або неможливою. Ще одним складним для обробки фактором є мультипроменевість, коли сигнал одночасно відбивається від різних поверхонь, утворюючи кілька копій того самого сигналу, які доходять до

приймача з незначним зсувом у часі, що може спричинити викривлення, а в деяких випадках навіть взаємне накладення хвиль, що додатково ускладнює обробку прийнятих даних. Проте у деяких протоколах, таких як Wi-Fi або Zigbee, існують механізми, які дозволяють частково використовувати багатопроменевість як перевагу, зберігаючи стійкість зв'язку в умовах приміщень за рахунок підвищеної кількості шляхів передачі сигналу [3-4].

Динаміка руху пристроїв, що передають або приймають інформацію, також є суттєвою особливістю бездротового середовища. У випадках, коли вузли мережі змінюють своє розташування, змінюється і якість каналу зв'язку, що потребує постійного моніторингу параметрів передачі та відповідного налаштування системи як у реальному використанні, так і в процесі моделювання або тестування. Не менш важливим фактором є обмеження енергоспоживання: більшість бездротових пристроїв працюють від автономних джерел живлення, що змушує обирати енергоефективні стратегії обміну даними. Для прикладу, у протоколі Zigbee використовується підтвердження доставки пакетів на канальному рівні для забезпечення надійності з мінімальними витратами енергії, тоді як у LoRaWAN застосовується адаптивна зміна швидкості передачі (ADR), що дозволяє зменшити енергоспоживання за рахунок подовження інтервалів передачі або зниження обсягу даних (див. рис. 1.2).

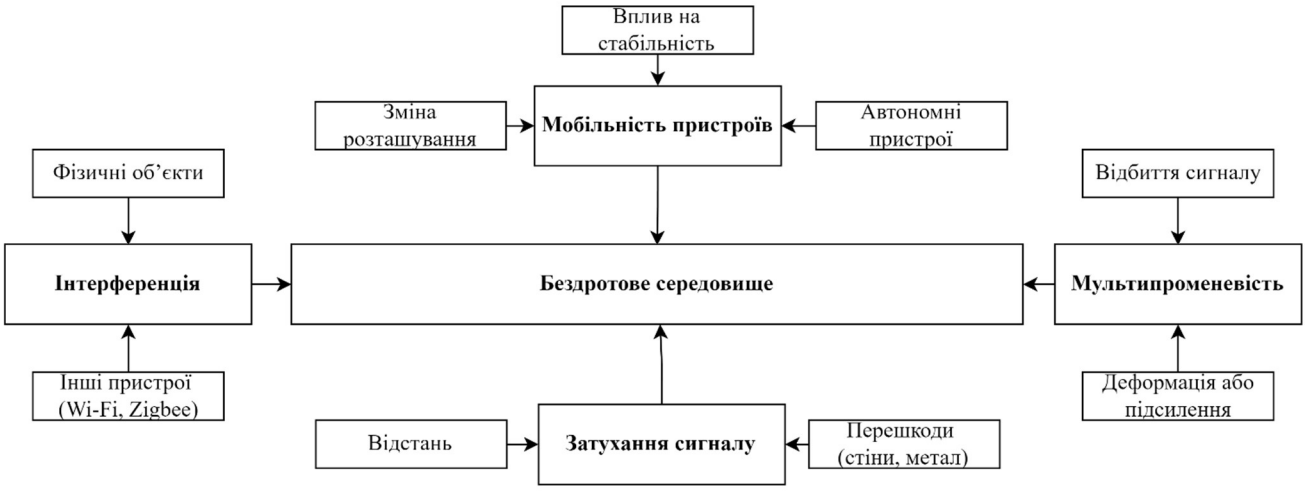


Рисунок 1.2 – Вплив факторів на бездротову передачу даних

Таким чином, ефективне використання бездротових технологій вимагає глибокого розуміння не лише логіки протоколів і структур даних, але й фізичних параметрів середовища, у якому відбувається обмін. Лише урахування таких характеристик дозволяє адекватно проектувати або досліджувати системи, зокрема за допомогою емуляторів на кшталт NS-3, що дають змогу максимально наблизити умови моделювання до реальних ситуацій і відтворити комплексний вплив перешкод, затухання, мультипроменевості та мобільності на поведінку протоколів [3-4].

Класифікація протоколів за рівнями OSI-моделі. Розглядаючи підхід до класифікації бездротових протоколів за рівнями моделі OSI (Open Systems Interconnection), слід зазначити, що дана модель є концептуальною основою, яка дозволяє структурувати функціонування протоколів у вигляді семи логічних рівнів. У бездротових мережах найчастіше реалізуються лише частина цих рівнів, зокрема фізичний та канальний. Фізичний рівень описує тип модуляції, частотний діапазон, параметри передачі, а також потужність сигналу. Наприклад, у Wi-Fi використовується ортогональне частотне мультиплексування (OFDM), тоді як LoRa застосовує модуляцію з розширеним спектром (CSS), що дозволяє забезпечити дальню та енергоефективну передачу в умовах низького рівня сигналу [5].

Канальний рівень (MAC) відповідає за організацію доступу до середовища передачі, контроль помилок та повторну передачу даних. У Zigbee реалізовано механізм CSMA/CA, який забезпечує уникнення конфліктів під час одночасного доступу кількох пристроїв до ефіру, тоді як у LoRaWAN передбачено класи вузлів A, B і C, що відрізняються за режимами прийому даних і енерговитратами, з урахуванням того, що більшість пристроїв спілкуються через шлюзи. На мережевому рівні реалізується маршрутизація: Zigbee може використовувати сітчасту (mesh) або зоряну структуру, а LoRaWAN частіше використовує централізовану топологію з мінімізацією проміжних вузлів [5].

На вищих рівнях – транспортному та прикладному – реалізація функцій значною мірою залежить від цільового застосування. Наприклад, у LoRaWAN свідомо уникається використання TCP через високе енергоспоживання, натомість

застосовуються легковагові протоколи типу UDP або спеціалізовані формати повідомлень, що дозволяє забезпечити передачу даних з мінімальним навантаженням на пристрій. У Wi-Fi, навпаки, транспортний і прикладний рівень реалізуються повністю, наприклад за допомогою стеку TCP/IP і протоколів на кшталт HTTP, які функціонують на стороні операційної системи або окремих сервісів (див. рис. 1.3).

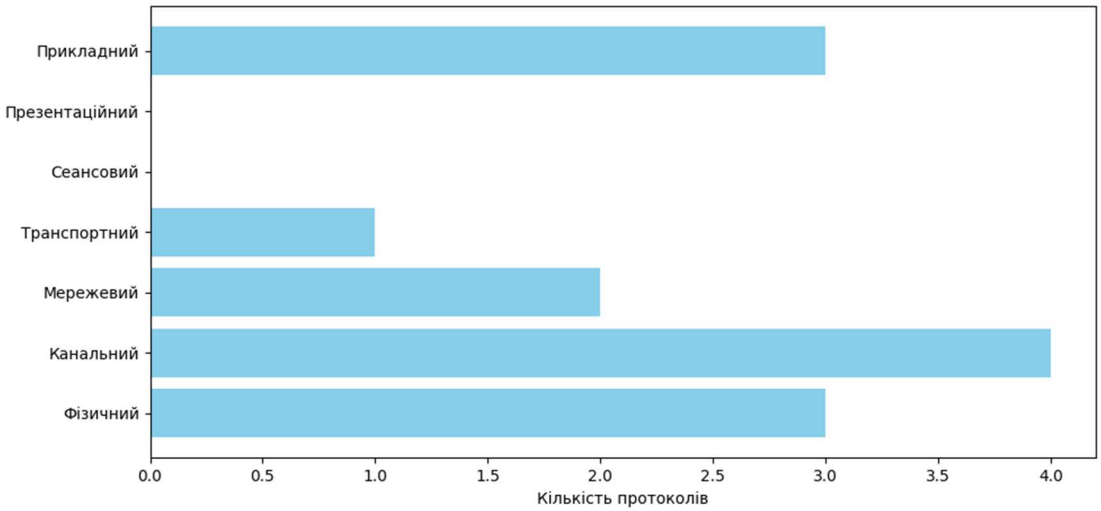


Рисунок 1.3 – Класифікація протоколів за рівнями OSI-моделі

Загалом, кожен протокол реалізує лише обрані рівні моделі OSI відповідно до свого призначення, енергобюджету, топології та вимог до надійності зв'язку. Наприклад, Wi-Fi зосереджений на високошвидкісній передачі в межах локальних мереж і покриває переважно фізичний та канальний рівень, передаючи інші функції на рівень ОС, тоді як LoRaWAN формує повноцінний комунікаційний стек із фізичним, MAC, мережевим і частково прикладним рівнями, що робить його автономною технологією з мінімальною залежністю від додаткових протоколів. Саме завдяки такій структурованості аналіз протоколів через призму OSI-моделі дозволяє точно визначити, які функції реалізуються на якому рівні та як це впливає на загальну архітектуру IoT-рішень [5].

Вимоги до тестування та аналізу бездротових протоколів. Будь-яке дослідження, що стосується бездротових протоколів, вимагає створення умов, максимально наближених до реального середовища, оскільки лабораторні умови

зазвичай є занадто ідеалізованими та не враховують впливу затримок, перешкод, втрат сигналу або непередбачуваної поведінки мережевих вузлів. У таких середовищах усе функціонує відповідно до теоретичних припущень, без врахування фізичних або інженерних обмежень, тому результати випробувань у подібних умовах не дають повної картини працездатності протоколу. Саме тому в дослідженнях бездротових технологій часто використовуються спеціалізовані інструменти моделювання, такі як емулятори або мережеві симулятори, що дозволяють створити реалістичні сценарії роботи мереж.

Однією з базових вимог до таких інструментів є наявність гнучких налаштувань параметрів середовища, зокрема можливості змінювати відстані між вузлами, моделювати різні рівні втрат сигналу, встановлювати різні швидкості передачі та створювати перешкоди або джерела інтерференції, які будуть впливати на стабільність зв’язку. Додатково необхідною є підтримка моделювання різноманітних топологій, таких як точка-точка, зірка, деревовидна структура або навіть складні динамічні сітчасті конфігурації, де пристрої можуть змінювати своє розташування або маршрутизацію в режимі реального часу.

Крім можливості варіювати структуру мережі, інструменти дослідження повинні включати функції аналізу продуктивності, які охоплюють показники затримки, втрат пакетів, пропускної здатності, а також оцінку навантаження на канал у різних умовах роботи, що дозволяє виявити, як протокол поводиться при пікових навантаженнях або у випадках нестабільного зв’язку, що особливо критично для систем реального часу або пристроїв з обмеженими енергетичними ресурсами.

Окремо важливою є масштабованість, тобто здатність досліджуваного інструменту підтримувати тестування мереж, які складаються не лише з кількох пристроїв, а десятків або навіть сотень вузлів, адже в реальних IoT-системах кількість елементів може зростати динамічно, що впливає як на маршрутизацію, так і на загальну продуктивність протоколу. Усе це доповнюється функціональністю візуалізації та логування процесів обміну даними, що дозволяє досліднику відстежувати поведінку кожного вузла, переглядати журнал подій і

виявляти помилки або неочікувану поведінку алгоритмів у розрізі конкретних умов. Таким чином, вимоги до тестування бездротових протоколів охоплюють широкий спектр технічних можливостей, які мають забезпечити повноцінне наближення до сценаріїв реального використання та забезпечити точність оцінки ефективності протоколу в умовах, максимально схожих на практичні.

1.2 Методи дослідження мереж: емуляція, симуляція, віртуалізація

Окрім базових принципів функціонування бездротових протоколів, дослідження мереж потребує розуміння методів, за допомогою яких здійснюється моделювання, оцінювання та аналіз поведінки мережевих технологій у змінених або контрольованих умовах, оскільки лише теоретичних знань недостатньо для реального впровадження нових рішень, що повинні пройти перевірку на відповідність вимогам продуктивності, масштабованості чи адаптивності. Для цього застосовуються спеціалізовані підходи, серед яких найчастіше розглядаються емуляція, симуляція та віртуалізація, які хоча й мають спільну мету створення контрольованого середовища для дослідження мереж, усе ж відрізняються за принципами реалізації, точністю результатів і сценаріями використання. Усі три методи можна об’єднати під поняттям емуляційних платформ, здатних здійснювати відтворення параметрів мережевої взаємодії, тестування нових протоколів або аналіз існуючих стандартів у безпечному середовищі до їхньої фактичної реалізації в інфраструктурі [6-9].

Порівняння методів – емуляція проти симуляції. Між методами симуляції та емуляції існує низка відмінностей, які визначають доцільність вибору кожного з них у певному дослідницькому контексті. Симуляція передбачає побудову математичної моделі, що абстрагує апаратні та програмні компоненти системи, використовуючи внутрішню логіку для відображення поведінки протоколів та пристроїв, без запуску реального коду. Подібний підхід дозволяє швидко

виконувати моделювання великомасштабних мереж або тестувати десятки варіантів конфігурацій з мінімальними витратами ресурсів, оскільки в таких інструментах, як NS-2 чи NS-3, усі пакети, події та таймери опрацьовуються на основі заздалегідь заданих сценаріїв. Симуляція забезпечує високу гнучкість, простоту налаштування та зручність при навчанні, однак через абстрактний характер моделювання її результати завжди мають певне відхилення від реальної поведінки системи [6].

На противагу цьому, емуляція передбачає використання реального коду мережевих протоколів або програм у віртуалізованому або ізольованому середовищі, дозволяючи запускати, наприклад, повноцінний TCP/IP стек або інші компоненти, з чим пов’язана вища достовірність результатів, наближених до умов реальної мережі. Такий підхід є корисним при перевірці працездатності існуючих реалізацій або при інтеграції мережевого протоколу з реальними застосунками, проте потребує значно більше обчислювальних ресурсів, має вищу складність конфігурації та обмежується можливостями операційної системи (див. рис. 1.4)

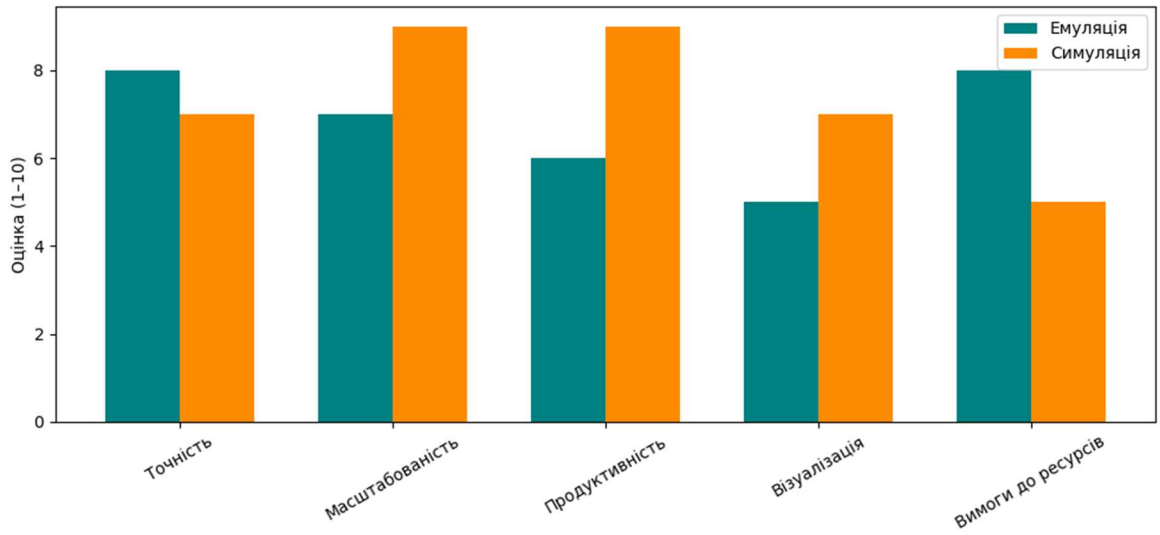


Рисунок 1.4 – Порівняння емуляції та симуляції за ключовими характеристиками

Зазвичай симуляція підходить для початкових етапів дослідження, коли йдеться про теоретичну перевірку гіпотез, тоді як емуляція є наступним кроком, який

дозволяє побачити, як протокол або система поведеться в умовах, максимально наближених до реального світу (див. табл. 1.2) [6].

Таблиця 1.2 – Порівняльна характеристика методів дослідження мереж

Метод	**Рівень деталізації**	**Реалістичність**	**Споживання ресурсів**	**Застосування**
Емуляція	Високий	Близька до реальної	Високе	Тестування мережевих протоколів
Симуляція	Середній	Теоретична	Низьке	Наукові дослідження, аналіз моделей
Віртуалізація	Залежно від гіпервізора	Реальна	Середнє	Побудова мереж у тестових середовищах

Порівняння характеристик цих методів дозволяє визначити ключові відмінності: за продуктивністю симуляція виграє, оскільки не виконує повний стек протоколів, реалістичність вища в емуляції завдяки використанню справжнього коду, гнучкість налаштувань є сильнішою стороною симуляції, тоді як емуляція обмежується середовищем виконання. Щодо придатності для навчання, симуляція простіша в освоєнні, тоді як емуляція вимагає глибших знань інфраструктури. Складність налаштування також нижча в симуляції, що дозволяє швидко розгортати сценарії дослідження без глибокого втручання в системні компоненти [6].

Особливості застосування симуляторів в освіті та науці. Симулятори, які застосовуються в дослідженнях мереж, особливо у сфері бездротових протоколів, набувають особливого значення в освітньому та науковому контексті, оскільки дозволяють відтворити повноцінну поведінку мереж без потреби у фізичній інфраструктурі, що особливо важливо в умовах обмежених ресурсів, коли

створення реального середовища з великою кількістю пристроїв є недоцільним або неможливим. У віртуалізованому або повністю симульованому середовищі можна досліджувати специфіку функціонування мережевих протоколів у різних ситуаціях, таких як підвищене завантаження каналу, втрати пакетів, варіативна затримка чи нестабільність зв'язку, що дозволяє сформувати цілісне уявлення про надійність і ефективність певного алгоритму або стеку технологій. Крім цього, симулятори допомагають засвоїти принципи маршрутизації, адресації, а також особливості транспортного рівня завдяки інтерактивним сценаріям, які можна як візуалізувати, так і проаналізувати у вигляді вихідного коду [7].

Під час експериментального підбору параметрів симулятор створює можливість практичного навчання оптимізації налаштувань мережі, а у сфері академічних досліджень надає інструментарій для перевірки ефективності нових алгоритмів маршрутизації або методів доступу до середовища передачі. Симулятори дозволяють створювати складні сценарії з великою кількістю вузлів, зокрема понад сотню, що фізично реалізувати або змоделювати у реальному середовищі вимагає непропорційних ресурсів. За рахунок багатократного запуску одного й того ж експерименту з варіаціями параметрів можна здійснити статистичний аналіз продуктивності або виявити слабкі місця в архітектурі протоколу. Завдяки відкритому вихідному коду такі програмні середовища як NS-3, OMNeT++ та інші дозволяють змінювати внутрішню логіку і в такий спосіб адаптувати симулятор до будь-якої наукової мети, що критично при побудові нових моделей і пропозицій щодо майбутніх стандартів у сфері комп'ютерних мереж [7].

Сильні та слабкі сторони програмної емуляції. Паралельно з цим, програмна емуляція забезпечує зовсім інший рівень дослідження, оскільки створюється віртуальне середовище, в якому функціонують справжні реалізації мережевого стека, наприклад TCP/IP. Використовуючи такі інструменти як GNS3, CORE Network Emulator або MiniNet Wi-Fi, можна запускати реальне програмне забезпечення, зокрема маршрутизатори, фаєрволи, шлюзи та інші мережеві компоненти, у цілком контрольованих умовах. Важливою перевагою є можливість гібридної взаємодії з хмарними середовищами або фізичними пристроями, що

розширює сферу експериментів та дає змогу перевірити поведінку реального трафіку, використовуючи такі інструменти як Wireshark чи TCPdump. Окрім цього, можливо вручну моделювати параметри мережі — затримку, втрати, пропускну здатність — через застосування віртуальних драйверів мережевого трафіку, таких як tc у Linux, що дозволяє максимально точно впливати на характеристики експерименту [8].

Однак слід ураховувати і певні недоліки емуляції, які стають помітними при масштабуванні досліджень. Кожен вузол, який створюється у віртуальному середовищі, споживає значну кількість системних ресурсів, тому запуск сценаріїв зі 100+ пристроями перетворюється на виклик з огляду на доступну обчислювальну потужність. Крім того, складність сценаріїв та обмеження, зумовлені конкретною операційною системою або конфігурацією середовища, можуть призводити до втрати гнучкості, а також до труднощів при повторенні досліду на іншому обладнанні або в іншій інсталяції (див. рис. 1.5).

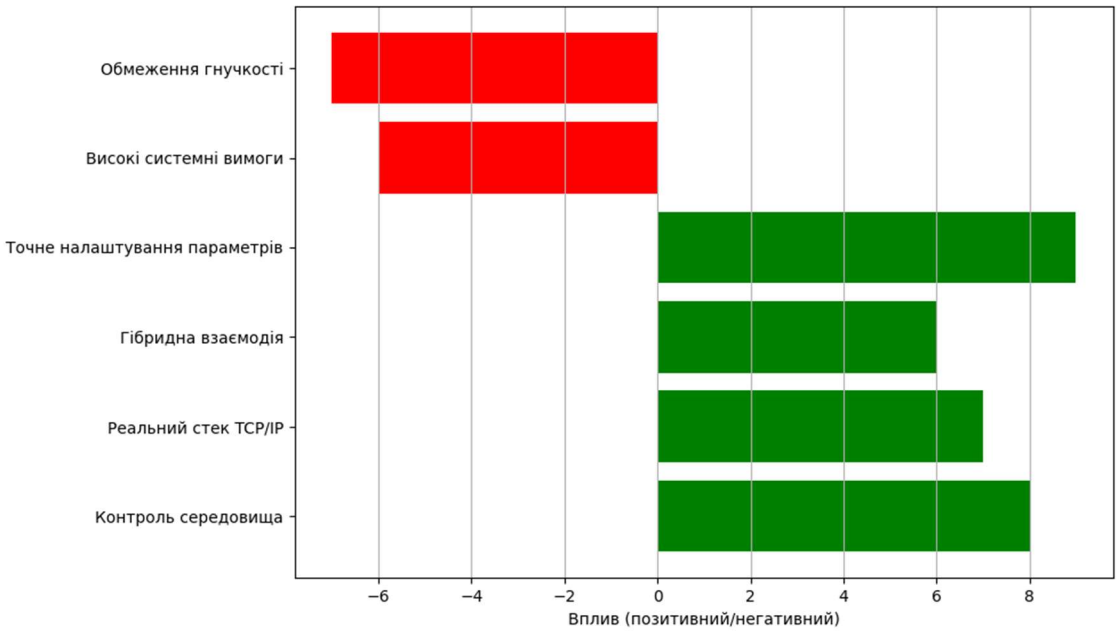


Рисунок 1.5 – Переваги та недоліки емуляції мереж

Таким чином, хоча емуляція є незамінним інструментом для тестування прототипів програмного забезпечення та перевірки його працездатності в умовах, максимально наближених до реальних, вона не завжди придатна для абстрактного або

математичного аналізу масштабних мережевих концепцій, зокрема тоді, коли дослідження вимагає повторюваності, широкої параметричної варіативності або симуляції складних топологій у великій кількості ітерацій [8].

Ключові інструменти для дослідження мереж. Серед наявних на сьогодні інструментів для дослідження мереж існує велика кількість рішень, які реалізують підхід симуляції або програмної емуляції з різним ступенем складності, точності та спрямованістю на практику чи теорію. Усі ці інструменти загалом мають схожі завдання, однак кожен із них реалізує власний підхід залежно від обраного сценарію використання, бажаного рівня деталізації та доступних обчислювальних ресурсів. Одним із перших широко використовуваних симуляторів можна назвати ns-2 — класичне рішення, що стало основою багатьох наукових робіт на початку 2000-х, зокрема завдяки підтримці аналізу бездротових мереж. Водночас, через складний синтаксис та застарілу архітектуру, інструмент втрачає актуальність, поступаючись місцем новішим рішенням [9].

Одним із таких сучасних симуляторів є NS-3, що побудований на C++ із можливістю інтеграції з Python. Його архітектура значно гнучкіша, а підтримка моделей таких протоколів, як Wi-Fi, LTE, 5G, LoRa та Zigbee, робить його універсальним інструментом для досліджень у галузі бездротового зв’язку. Крім того, NS-3 підтримує емуляцію реального стеку протоколів через механізм tap-інтерфейсів, а також можливість контейнеризації через Docker, що особливо важливо для відтворюваних експериментів. Іншим потужним симулятором є OMNeT++, який має модульну архітектуру та графічне середовище для побудови моделей, активно використовується у дослідженнях, де крім мережевого рівня важливо моделювати обчислювальні або навіть фізичні процеси. Цей інструмент містить низку бібліотек, орієнтованих на телекомунікаційні системи, зокрема INET Framework для TCP/IP-мереж [9].

Окрему категорію становлять інструменти, які реалізують саме програмну емуляцію, такі як GNS3 — популярне середовище, що дозволяє запускати повноцінні образи операційних систем маршрутизаторів, файрволів або шлюзів (наприклад, Cisco IOS, MikroTik, pfSense) у віртуалізованому оточенні. Цей

інструмент забезпечує надзвичайно високий рівень реалізму, тому є надзвичайно корисним для практичного навчання, тестування конфігурацій або перевірки сумісності різних компонентів мережі. Подібне застосування має Mininet-WiFi — розширення класичного Mininet, яке дозволяє емулювати бездротові SDN-сценарії з можливістю контролю мобільності, рівня сигналу та маршрутизації між точками доступу. Завдяки використанню Python API сценарії в цьому середовищі можна автоматизувати, що значно пришвидшує дослідження [9].

Кожен із згаданих інструментів має власну специфіку. Наприклад, симулятори на зразок NS-3 чи OMNeT++ орієнтовані переважно на теоретичні дослідження та математичне моделювання, дають змогу створювати масштабовані сценарії з десятками або сотнями вузлів без значного навантаження на систему, оскільки вони оперують абстракціями, а не реальними виконуваними образами. Водночас емулювати роботу мережевого стеку або поведінку повноцінної операційної системи, як це роблять GNS3 чи Mininet-WiFi, можна лише з використанням реального ПЗ, що вимагає більших ресурсів, але забезпечує точнішу відповідність до реального світу, що особливо цінно при тестуванні готових рішень (див. рис. 1.6).

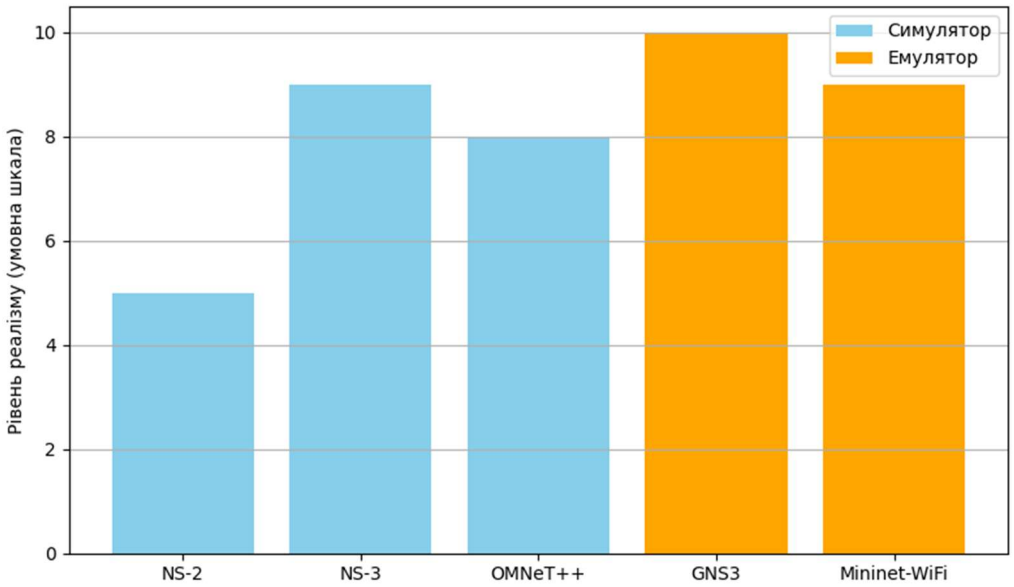


Рисунок 1.6 – Класифікація мережевих дослідницьких інструментів (за рівнем реалізму)

Поєднання підходів симуляції та емуляції, а також використання віртуалізації, формує гнучкий інструментарій для глибокого аналізу як логіки роботи протоколів, так і їх поведінки в змодельованих або реальних умовах. Такий підхід є особливо ефективним у сфері досліджень бездротових технологій нового покоління, де необхідно оцінити не лише продуктивність, але й стійкість, адаптивність до змінних умов середовища та потенціал до масштабування в майбутніх інфраструктурах [9].

1.3 Особливості мережевої емуляції в середовищі Docker

У контексті дослідження бездротових протоколів або мережевих технологій часто виникає потреба у створенні масштабованого, реалістичного середовища для тестування, при цьому існує обмеження в обчислювальних ресурсах або часу на налаштування великої кількості фізичних вузлів. Використання класичних емуляторів хоча й дозволяє перевірити поведінку окремих протоколів чи мережевих сервісів, однак не завжди забезпечує достатню гнучкість чи масштабованість, яка потрібна для повноцінного моделювання складних мережевих сценаріїв. Зі свого боку, симулятори здатні відтворити великі мережеві топології з абстрактним рівнем деталізації, але позбавлені точності у взаємодії з реальними реалізаціями програмного забезпечення та протоколів. Саме тому часто використовується комбінований підхід, де симуляція, емуляція та віртуалізація застосовуються разом як взаємодоповнюючі методи, що дозволяють отримати повнішу картину функціонування мережевих технологій [10-12].

У таких випадках ефективним рішенням стає мережева емуляція із застосуванням інструментів контейнеризації, які дозволяють значно спростити і прискорити процес розгортання експериментального середовища. Одним з найбільш універсальних інструментів у цьому напрямку виступає Docker, який забезпечує гнучке, легке та багаторазове використання налаштованих середовищ

для відтворення роботи мережевих вузлів, серверів, маршрутизаторів або навіть віртуальних точок доступу. У випадках, коли ресурси обмежені, розгортання повноцінної віртуальної інфраструктури може виявитися надто затратним, тоді як контейнеризоване середовище дозволяє реалізувати аналогічну функціональність за менших витрат і з високою швидкістю впровадження на будь-якому сумісному пристрої [13].

Поняття контейнеризації та ізоляції мережевих середовищ. Під контейнеризацією розуміється підхід, за якого додаток або набір процесів із усіма необхідними залежностями упаковується в окреме ізольоване середовище — контейнер. Відмінність від віртуальних машин полягає в тому, що контейнер не потребує повноцінної гостьової операційної системи, адже використовує ядро хостової системи, що суттєво знижує обсяг використаних ресурсів та пришвидшує запуск. Найпоширенішим інструментом реалізації такої технології є саме Docker, який дозволяє створювати, запускати та управляти контейнерами за допомогою зручного командного інтерфейсу та API, забезпечуючи ізоляцію не тільки файлової системи, але й мережевих стеків. Кожен контейнер у Docker може мати окремі мережеві інтерфейси, власні маршрути, правила IP tables, мережеву ізоляцію, що дозволяє створювати достовірні умови для дослідження взаємодії мережевих вузлів, навіть якщо вони фактично не існують як фізичні пристрої [10].

Docker пропонує кілька типів мережевих інтерфейсів, які можуть бути використані залежно від мети моделювання. За замовчуванням використовується Bridge Network — це ізольована підмережа для внутрішньої комунікації між контейнерами, де кожен контейнер має свою IP-адресу. Якщо потрібно, щоб контейнер працював напряму з мережею хостової машини, використовується режим Host Network, який дозволяє контейнеру бачити всі інтерфейси хоста, як свої. Для більш складних сценаріїв, де емулюється фізична мережа або тестуються специфічні протоколи рівня доступу, використовується інтерфейс Macvlan, який призначає кожному контейнеру окрему MAC-адресу, що дає змогу взаємодіяти з мережею як фізичному пристрою. При потребі побудови власної ізольованої топології з точним контролем маршрутизації та IP-адресації застосовуються

Custom User-Defined Networks, які створюються вручну з можливістю вказання всіх параметрів мережевої взаємодії (див. рис. 1.7) [10].

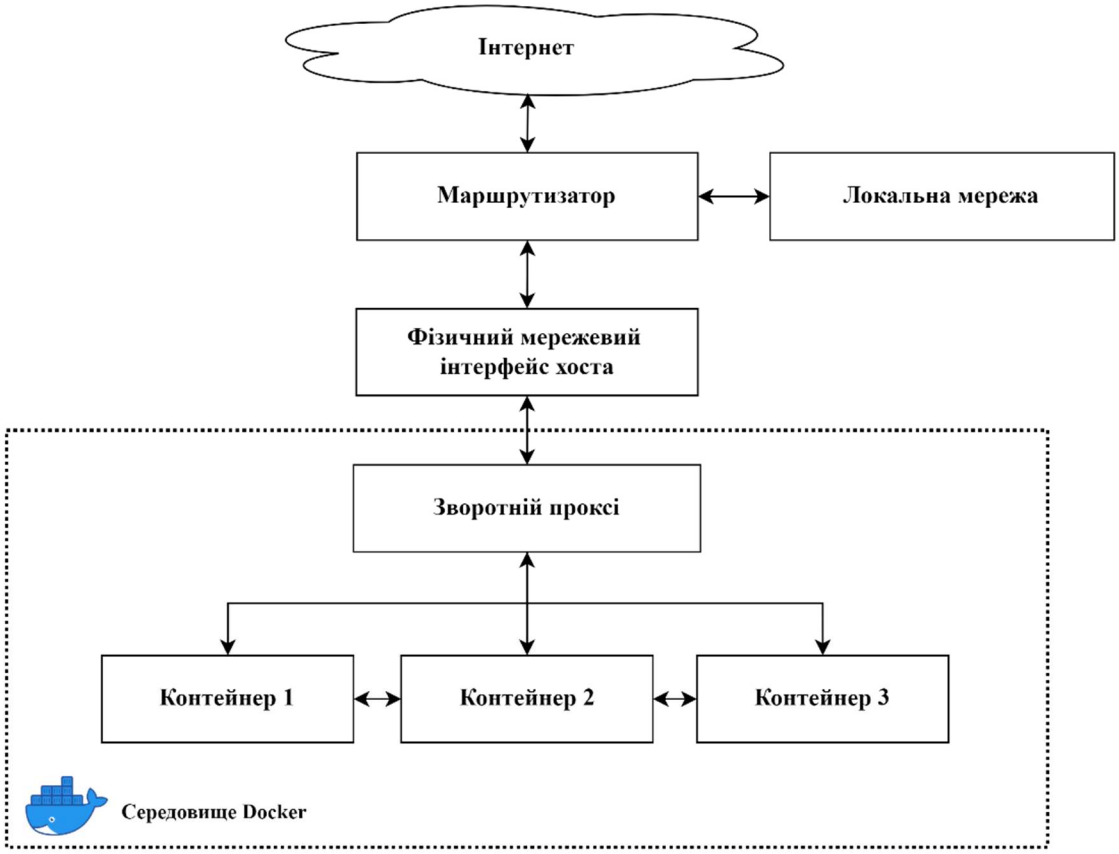


Рисунок 1.7 – Схема ізоляції мережевих середовищ у Docker

Переваги використання Docker для мережевих досліджень. Docker широко використовується в мережевих дослідженнях завдяки сукупності переваг, які роблять його зручним та ефективним інструментом для моделювання, симуляції та відтворення мережевих середовищ. Здобута популярність цього інструменту пояснюється тим, що він дозволяє швидко, точно й масштабовано реалізовувати різні мережеві сценарії, зокрема у контексті тестування мережевих протоколів, інтерфейсів і архітектур бездротового зв'язку, де часто потрібне тонке налаштування середовища. У випадку використання платформ-емуляторів чи симуляторів Docker виступає як додатковий рівень ізоляції й керованості, що спрощує запуск дослідів, створення віртуальних топологій і управління залежностями (див. рис. 1.8) [11].

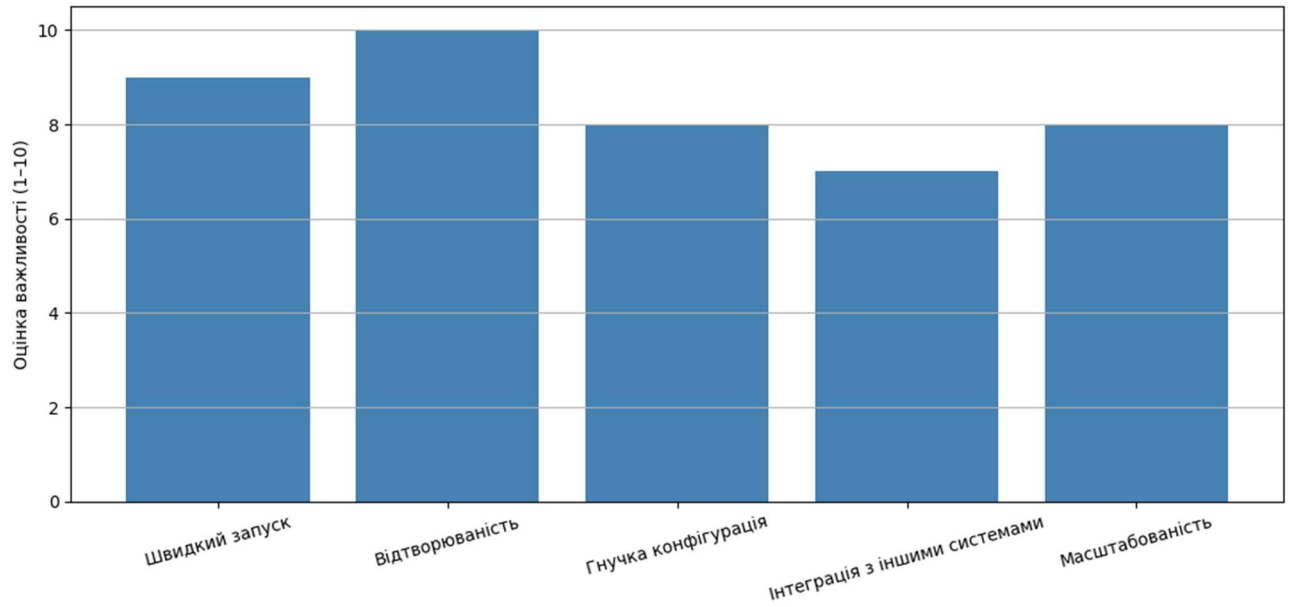


Рисунок 1.8 – Переваги використання Docker для мережевих експериментів

Найпершою перевагою є надзвичайна швидкість запуску контейнерів і їх легка вага: Docker-контейнери стартують за лічені секунди, не потребуючи повноцінного завантаження гостьової операційної системи, що дозволяє швидко формувати складні віртуальні мережі з десятками або сотнями вузлів без істотного навантаження на хостову систему. Крім того, забезпечується абсолютна відтворюваність експериментів, оскільки контейнерний образ містить усю потрібну конфігурацію, бібліотеки, залежності та навіть ядро ОС, що дозволяє іншим дослідникам запустити ідентичне середовище незалежно від їх локального налаштування. Традиційні віртуальні машини або робота на базі хостової системи не забезпечують такої консистентності, адже можуть бути впливи від різниці в оточенні, що унеможливлює точне повторення мережевих експериментів [11].

Наступною перевагою є гнучкість налаштування мережевих сценаріїв: Docker підтримує створення ізольованих мережевих просторів і бриджів, а також інтеграцію з утилітами типу tc для трафік-шейпінгу, симуляції затримок, втрат пакетів, зміни пропускної здатності, маршрутизації тощо, що дозволяє будувати реалістичні умови тестування. Можливість інтеграції Docker із зовнішніми інструментами робить його ще привабливішим для мережевих досліджень —

зокрема, підтримка роботи з NS-3, Mininet, GNS3 дозволяє поєднувати можливості програмних симуляторів і реальних протоколів або додатків усередині контейнерів, що створює середовище, в якому можна одночасно моделювати мережеву топологію й тестувати реальну взаємодію в мережі [11].

Окремо варто виділити можливість масштабування таких середовищ за допомогою інструментів оркестрації, таких як Docker Compose або Kubernetes. Вони дозволяють централізовано описувати всю конфігурацію розподіленої системи в YAML-файлах і легко розгортати її на будь-якому іншому хості або хмарному сервері, що відкриває додаткові можливості для побудови великих емуляційних лабораторій. До цього додається простота збереження та обміну — контейнери можна завантажити в публічні або приватні реєстри, як-от Docker Hub або GitHub Container Registry, завдяки чому дослідники, викладачі або студенти можуть легко ділитися готовими конфігураціями, експериментами або шаблонами для навчання.

Приклади інструментів, які підтримують запуск у Docker-контейнерах. Серед прикладів інструментів, які безпосередньо підтримують запуск у Docker-контейнерах і дозволяють реалізовувати мережеві дослідження різного масштабу, одним із найпоширеніших є симулятор NS-3, що офіційно підтримує Docker-образи з попередньо налаштованим середовищем, яке містить всі необхідні залежності, що спрощує роботу, уникаючи потреби ручної компіляції або встановлення. GNS3, починаючи з версії 2.2, також забезпечує підтримку Docker-контейнерів як елементів мережевої топології — контейнери можна використовувати як сервери, маршрутизатори, клієнти, що спрощує побудову гібридних емуляційних систем. Ще одним потужним інструментом є Mininet, який можна запускати усередині Docker для побудови топологій програмно-керованих мереж (SDN), що зручно при моделюванні OpenFlow-середовищ [12].

Окрім них, існують утиліти для моніторингу, аналізу й візуалізації трафіку, як-от sFlow-RT або Grafana в поєднанні з Wireshark, які можуть бути легко розгорнуті як окремі сервіси в контейнерах для побудови повноцінної мережевої лабораторії. Розширення типу Containernet, що ґрунтується на Mininet, дозволяє

запускати реальні сервіси в Docker усередині топологій SDN, поєднуючи гнучкість програмної симуляції з реальними компонентами. Завдяки цьому стає можливим тестування повноцінних мережевих стеків — від рівня протоколів (DNS, HTTP, FTP, MQTT) до IoT-додатків, з особливим акцентом на дослідження в галузі бездротового зв'язку, інтернету речей, автономних систем тощо (див. рис. 1.9) [12].

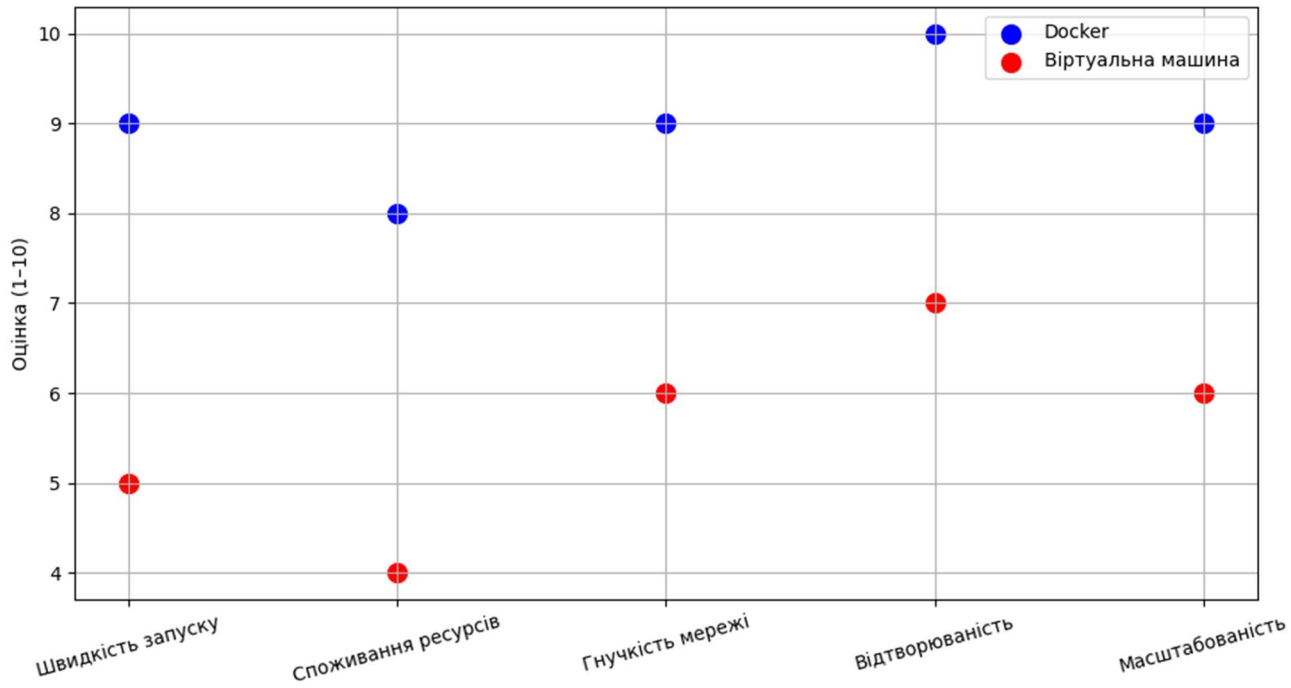


Рисунок 1.9 – Порівняння Docker і віртуальних машин за критеріями

У підсумку, використання Docker у контексті мережевих досліджень не просто зручне — воно відкриває широкі перспективи як для швидкого створення емуляційних сценаріїв, так і для їх масштабування, відтворення та інтеграції з реальними програмними стеками, що робить його незамінним інструментом в академічному й дослідницькому середовищі.

Масштабування та побудова складних топологій у Docker. Масштабування та побудова складних топологій у середовищі Docker не є унікальною або виключною функцією лише цієї платформи, однак саме в контексті мережевої емуляції Docker надає набір інструментів, які дозволяють розгортати масштабовані інфраструктури з десятками або навіть сотнями вузлів, що є необхідним для дослідження протоколів бездротових мереж, тестування взаємодії сервісів та

моделювання складних сценаріїв роботи IoT або SDN. Усе це реалізується за рахунок декількох ключових підходів, що використовуються як у Docker, так і в інших системах віртуалізації, але саме тут вони дають змогу гнучко будувати розгалужені мережеві структури для тестування. Одним із таких інструментів є Docker Compose — засіб, який дозволяє описувати багатоконтейнерні інфраструктури у форматі YAML, що в свою чергу відкриває можливості для організації мережевих вузлів з індивідуальними змінними середовища, залежностями між службами та ізольованими мережами. Зокрема, контейнери можуть бути пов’язані через іменовані мережі, запускатись у визначеній послідовності, наприклад, сервер до клієнта, і це дозволяє точно повторювати цільову архітектуру протоколу чи сервісу [13].

Крім Docker Compose, важливу роль відіграє використання мережевих просторів імен (Network namespaces) і віртуальних Ethernet-пар (veth-пар), які завдяки вбудованій мережевій підсистемі Linux дозволяють створювати ізольовані мережі з довільною топологією, що забезпечує можливість ручного з’єднання контейнерів у формі зірки, кільця, дерева або сітки — залежно від того, яка схема потрібна для дослідження. До того ж, завдяки утилітам контролю пропускної здатності, затримок, втрат пакетів або інших параметрів, емуляція може наблизитись до умов реального середовища, що є особливо важливим для дослідження мобільних вузлів, де перепідключення до різних точок доступу можна реалізувати динамічно, імітуючи рух клієнта в мережі, що дозволяє не лише оцінювати продуктивність сервісів, а й досліджувати вплив нестабільності каналів на якість зв’язку [13].

Для великих проєктів Docker також підтримує масштабування за допомогою таких рішень, як Docker Swarm або Kubernetes, що дозволяє розгортати інфраструктуру на декількох фізичних або віртуальних хостах з автоматичним балансуванням навантаження, резервуванням та оновленням компонентів. Завдяки цьому з’являється можливість працювати не з поодинокими контейнерами, а з повноцінними розподіленими мережами, що охоплюють сотні вузлів, які можуть симулювати, наприклад, трафік від тисяч пристроїв у мережі 5G, відмовостійкість

критичних служб або сценарії перенесення навантаження під час аварій чи перевантаження окремих сегментів інфраструктури [13].

У контексті симуляції або емуляції бездротових мереж Docker не є прямим конкурентом спеціалізованим інструментам, однак легко інтегрується з ними. Наприклад, одним із найпопулярніших мережевих симуляторів є NS-3, тоді як до емуляторів можна віднести GNS3 або Mininet. Всі вони можуть використовуватись паралельно з Docker для побудови гібридного середовища, де частина системи працює в симуляторі, а частина — в реальних контейнерах, що дозволяє створити середовище з реальними службами, які взаємодіють з віртуальними пристроями, що особливо корисно при тестуванні нових мережевих протоколів або застосувань, коли потрібно, щоб, наприклад, MQTT-брокер або сервер CoAP працював у Docker, обслуговуючи клієнтів, що імітуються через NS-3, і завдяки цьому така архітектура дає змогу поєднати абстрактне моделювання з реальними прикладами трафіку [13].

Підсумовуючи, використання Docker для моделювання, масштабування та інтеграції з симуляторами дозволяє побудувати ізольоване, відтворюване і кероване середовище, яке може адаптуватись під різні сценарії дослідження та розробки у сфері бездротових технологій. Завдяки своїй гнучкості, відкритості та підтримці численних інструментів, Docker виступає платформою, яка не лише дозволяє пришвидшити експериментування, а й може слугувати мостом між теоретичними симуляціями та реальними мережевими рішеннями, відкриваючи можливості як для дослідників, так і для практичних інженерів, що працюють з IoT, SDN або мобільними мережами нового покоління.

1.4 Архітектура та модулі NS-3

Розглядаючи наявні рішення для створення емуляції мережевих протоколів, зокрема у сфері бездротового зв’язку, стає очевидно, що на перший погляд можна було б зупинитися на будь-якому з емуляторів, адже концепція дослідження

передбачає моделювання саме таких протоколів. Проте порівняння симуляторів і емуляторів дозволяє зробити висновок, що найбільш оптимальним за співвідношенням витрат ресурсів і функціональності є саме NS-3. Хоча він формально є симулятором, на практиці він здатен реалізовувати сценарії емулювання мережевих протоколів і відтворення роботи бездротових мереж у досить реалістичному вигляді, що особливо важливо для даного дослідження. Завдяки підтримці масштабування до великої кількості вузлів NS-3 дозволяє будувати повноцінні топології з великою деталізацією, а його використання в поєднанні з контейнеризацією, зокрема Docker, значно спрощує розгортання та інтеграцію з іншими компонентами [14-17].

Загальна структура NS-3 – ядро, модулі протоколів, інтеграція з Python. У загальній архітектурі NS-3, або Network Simulator 3, закладена гнучка та модульна структура, яка робить цей інструмент потужним середовищем для дослідження, розробки та верифікації мережевих протоколів у різних сценаріях, зокрема в бездротовому середовищі. Цей симулятор, який є наступником NS-2, розроблений повністю з нуля на основі мови C++ із підтримкою інтеграції з Python, що забезпечує гнучке конфігурування та можливість використання сучасних підходів, таких як машинне навчання. Архітектура NS-3 базується на ядрі симулятора — Core NS-3, яке відповідає за обробку подій у симульованому часі за принципом подієво-орієнтованої симуляції Discrete Event Simulation, створення та керування вузлами, управління мережевим трафіком, пакетами, сокетами, а також забезпечує взаємодію між модулями. Уся система працює на основі детермінованого планувальника подій, що дозволяє досягти повторюваних результатів симуляції шляхом зміни лише параметрів запуску, без зміни внутрішньої логіки роботи моделі (див. рис. 1.10) [14].

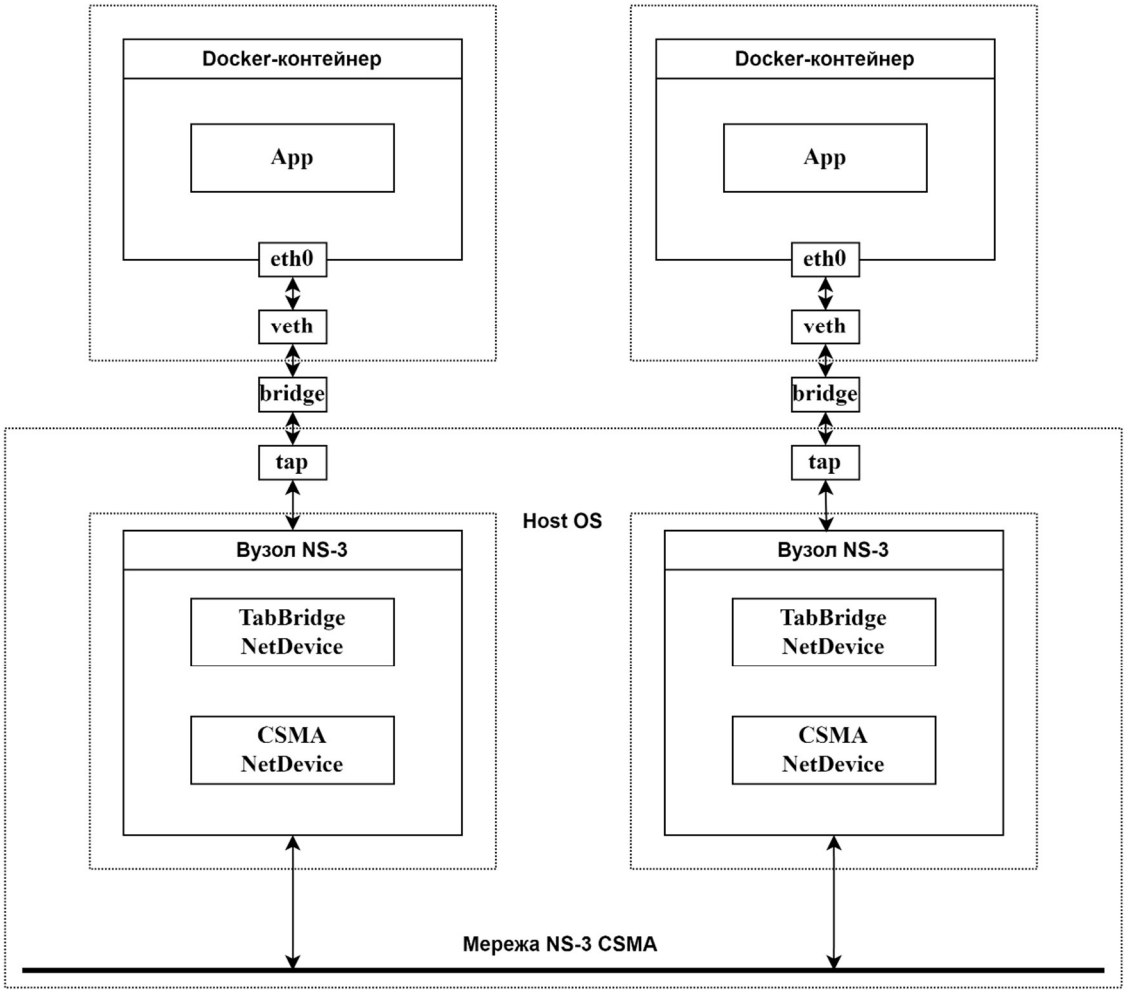


Рисунок 1.10 – Загальна структура симулятора NS-3

Модульність NS-3 полягає у тому, що функціональність реалізована у вигляді окремих бібліотек, кожна з яких відповідає за певну частину стеку протоколів. Наприклад, реалізовані модулі для TCP/IP, включаючи підтримку IPv4, IPv6, TCP, UDP; є модулі маршрутизації, доступу до середовища — Point-to-Point, CSMA, Wi-Fi, LTE, LoRa. Для реалізації додатків у симуляціях існують прикладні модулі, такі як Echo, OnOff, VoIP, VideoStreaming, які дозволяють емулювати поведінку кінцевих пристроїв або сервісів у мережі. Мобільність реалізується через окремий модуль Mobility, що дозволяє задавати різні сценарії руху вузлів, у той час як модуль Energy дозволяє відтворювати споживання енергії пристроями, що особливо актуально при моделюванні IoT-сценаріїв або автономних сенсорних вузлів (див. табл. 1.3) [14].

Таблиця 1.3 – Основні модулі NS-3 та їх призначення

Модуль	**Призначення**
wifi	Моделювання бездротових мереж за стандартом IEEE 802.11
mobility	Задання сценаріїв руху вузлів у просторі
internet	Підтримка стеку IP-протоколів
applications	Створення та моделювання трафіку
config-store	Збереження та завантаження параметрів симуляції
netanim	Візуалізація симуляцій у графічному інтерфейсі

Особливу увагу привертає інтеграція з Python, яка реалізована через систему автоматичного обгортання C++ функцій за допомогою PyBindGen. Такий підхід забезпечує можливість створення, запуску та керування симуляціями за допомогою Python, що є зручним як у випадку побудови складних сценаріїв, так і при інтеграції з зовнішніми інструментами для обробки результатів, алгоритмами оптимізації або машинного навчання. Модулі, такі як ns.network, ns.internet, ns.wifi доступні для імпорту в Python, що відкриває широкі можливості для динамічного управління параметрами симуляції, створення адаптивних моделей та реалізації автоматичного збору та обробки результатів експериментів. Хоча за максимальної продуктивності рекомендовано використання C++, гнучкість, яку надає Python, робить його незамінним інструментом у дослідницькій роботі, що вимагає багаторазового повторення експериментів із варіативними параметрами [14].

Підтримка протоколів бездротового зв'язку (Wi-Fi, LoRaWAN, LTE). Однією з основних причин вибору платформи NS-3 для дослідження мережевих протоколів стало те, що вона забезпечує повноцінну підтримку двосторонніх моделей найпоширеніших бездротових протоколів, що дозволяє точно моделювати як фізичний, так і канальний рівні, відкриваючи простір для детального аналізу, оптимізації та подальших покращень мережевої архітектури (див. рис. 1.11) [15].

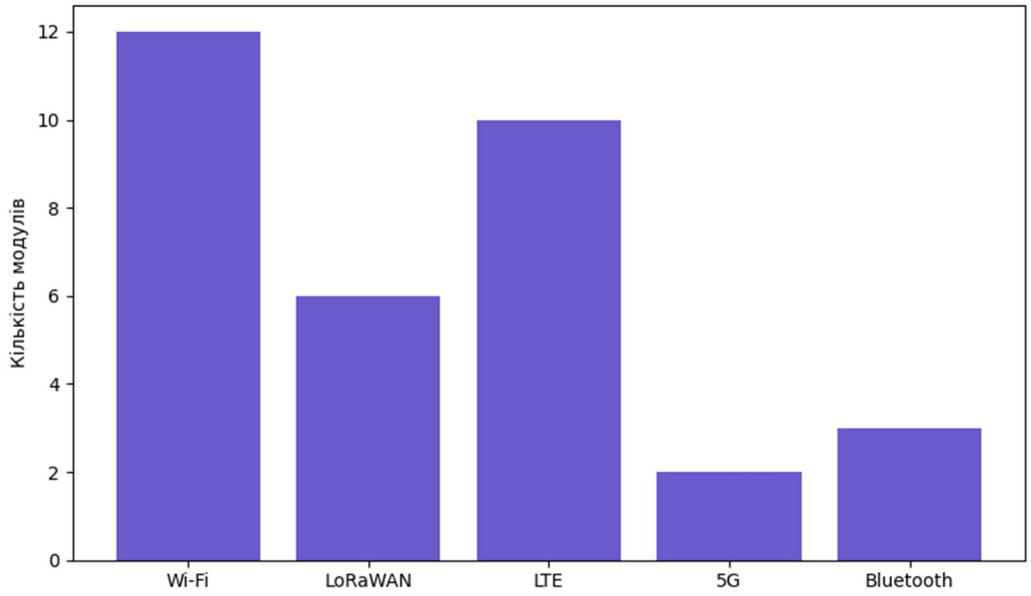


Рисунок 1.11 – Підтримка бездротових протоколів у NS-3

Якщо розглядати модуль Wi-Fi, він охоплює стандарти 802.11a/b/g/n/ac з можливістю конфігурації режимів доступу, таких як DCF чи EDCA, схем модуляції, включаючи QAM або OFDM, а також параметрів енергетики на кшталт потужності передавача, рівня RSSI чи вибору типу топології, будь то інфраструктурна, Ad-hoc чи Mesh. У цьому ж модулі реалізовано точне моделювання затримок, втрат пакетів та перешкод, що особливо важливо при аналізі ефективності алгоритмів рівня MAC або поведінки багатоканальних мереж [15].

У випадку протоколу LoRaWAN реалізація в NS-3 ґрунтується на відкритому проекті, що забезпечує можливість моделювання повноцінних сценаріїв з вузлами, шлюзами та сервером, включаючи гнучке налаштування фізичних параметрів таких як spreading factor, ширина смуги, клас пристрою (A, B або C), затримки прийому, вікна прийому, а також можливі конфлікти в середовищі передачі. Завдяки цьому LoRaWAN-модуль дозволяє проводити глибокий аналіз сценаріїв з обмеженим енергоспоживанням та великою щільністю сенсорів, що працюють на великих відстанях, характерних для IoT-мереж з протоколом LoRa [15].

Коли йдеться про мобільні мережі, NS-3 надає потужний модуль LTE під назвою LENA, у якому реалізовано підтримку фізичного рівня, рівнів MAC та RLC,

з гнучким управлінням радіоресурсами через Scheduler. Крім того, реалізовано механізми мобільності, зокрема handover між eNodeB, з можливістю підключення до EPC, що дозволяє відтворювати повноцінний стека мобільного зв'язку. Доступна також емуляція таких сервісів як VoIP, відеотрафік або Kiosk-mode. Доповнюючи це, модуль NR (New Radio) розширює можливості до підтримки 5G з урахуванням динамічного розподілу спектру, сценаріїв масового IoT (mMTC) або наднизької затримки (URLLC), що відкриває нові горизонти у дослідженні сучасних бездротових технологій [15].

Режими запуску (командний рядок, Python API, візуалізація). Не менш значущим є спосіб запуску симуляцій у NS-3, оскільки саме механізм запуску визначає зручність взаємодії з інструментом, масштабованість і гнучкість під час моделювання різних сценаріїв. Найбільш базовим є запуск через командний рядок, коли C++-сценарій компілюється і виконується з передачею параметрів через аргументи або файли конфігурації. Такий підхід забезпечує високу продуктивність та передбачуваність симуляції. Для більш динамічного управління параметрами використовується запуск через Python API, завдяки чому можна швидко моделювати змінні топології, генерувати трафік у режимі реального часу, інтегрувати з іншими бібліотеками або навіть реалізувати машинне навчання, що особливо корисно у випадках, коли потрібно протестувати велику кількість варіантів або автоматизувати збір результатів [16].

Щодо візуалізації, то NS-3 має кілька інструментів, які дозволяють не лише аналізувати числові показники, а й побачити поведінку мережі наочно. NetAnim є одним із таких інструментів, що надає візуальну топологію в реальному часі з анімацією переміщення вузлів і передачею пакетів. PyViz забезпечує інтерактивну візуалізацію безпосередньо під час виконання симуляції, відкриваючи графічний інтерфейс для спостереження за мережею. FlowMonitor використовується для збору статистики щодо TCP та UDP, таких як середня затримка, втрати, джиттер, пропускна здатність, що дає змогу порівнювати протоколи на рівні продуктивності. Крім цього, результати можна виводити у форматах, сумісних з Gnuplot або Matplotlib, що дозволяє будувати графіки та діаграми з логів симуляції, надаючи

додаткову візуальну інформацію про поведінку мережі під час тестування, що буде практично продемонстровано у рамках цього дослідження [16].

Порівняння з іншими симуляторами за точністю та функціональністю. Серед численних засобів моделювання мережевих середовищ NS-3 вирізняється як один із найбільш потужних і гнучких інструментів, здатних забезпечити високу точність на фізичному рівні, повноцінну підтримку сучасних бездротових технологій, таких як Wi-Fi, LTE і LoRa, а також масштабованість до понад 1000 вузлів. Хоча сам по собі NS-3 є симулятором подій, його функціональні можливості часто виходять за межі класичного симулювання завдяки можливості елементарної емуляції, інтеграції з Python і підтримці складних сценаріїв мобільності. У порівнянні з OMNeT++, іншим симулятором, NS-3 демонструє вищу точність на фізичному рівні та більш органічну інтеграцію з мовами програмування, що дає змогу реалізовувати динамічні та реалістичні моделі мережевої поведінки. OMNeT++ має хорошу масштабованість і підтримку доповнень, однак поступається NS-3 у гнучкості реалізації нових протоколів і в повноті підтримки бездротових технологій без необхідності додаткових налаштувань [17].

З іншого боку, платформи, що належать до категорії емуляторів, такі як Mininet або GNS3 / EVE-NG, мають зовсім інші підходи до реалізації мережевих досліджень. Mininet працює із реальними TCP/IP стеком і може створювати середовища, наближені до практичного використання, однак має низьку точність на фізичному рівні, не підтримує бездротові технології, не має сценаріїв мобільності, обмежену інтеграцію з Python і не може масштабуватись понад кілька десятків вузлів без значного падіння продуктивності. GNS3 або його розширена версія EVE-NG в деяких аспектах більш потужні завдяки можливості використання віртуальних машин, Docker-контейнерів і збереженню реального вигляду мережевої інфраструктури, однак їхня ефективність безпосередньо залежить від якості використаних образів, обмежена підтримка Wi-Fi і бездротових стандартів, повна відсутність сценаріїв мобільності та інтеграції з Python не дозволяють їм вийти за межі демонстраційних або лабораторних цілей [17].

NS-3, попри складність засвоєння, яка вимагає базового володіння C++ або Python та розуміння принципів подієво-орієнтованої симуляції, залишається універсальним засобом дослідження сучасних комп'ютерних мереж з високим рівнем деталізації. Його модульна архітектура дозволяє реалізовувати сценарії з різними топологіями, рівнями навантаження, типами трафіку, а також експериментувати з новими мережевими протоколами та середовищами, що робить NS-3 надзвичайно привабливим у контексті економічних, наукових і прикладних досліджень, де потрібна висока точність, масштабованість і підтримка інноваційних технологій. Завдяки повноцінній підтримці візуалізації, численним допоміжним бібліотекам та активній спільноті розробників, NS-3 забезпечує виняткову гнучкість у побудові дослідницьких сценаріїв, що охоплюють як класичні дротові, так і новітні бездротові середовища. У підсумку саме ця багатофункціональність і детальність моделювання ставить NS-3 на перше місце серед усіх проаналізованих рішень [17].

1.5 Можливості моделювання бездротових мереж в NS-3

Можливості моделювання бездротових мереж у середовищі NS-3 охоплюють надзвичайно широкий спектр сценаріїв, які дозволяють проводити як базове тестування, так і складне дослідження поведінки мережевих протоколів у динамічному середовищі. З огляду на гнучкість платформи, NS-3 може бути використаний не лише для симуляції, а й для часткової емуляції реальних протоколів бездротового зв'язку, що забезпечує глибоке занурення в процес розробки та аналізу систем зв'язку нового покоління. У цьому контексті особливе значення має моделювання бездротових мереж, оскільки без нього неможливо повноцінно відтворити умови роботи таких технологій, як IoT, ad-hoc або мобільні мережі [18-21].

Приклади сценаріїв (adhoc-мережі, IoT-сценарії, мобільні вузли). Платформа NS-3 забезпечує моделювання adhoc-мереж, у яких відсутня фіксована інфраструктура, а кожен вузол може виконувати роль як передавача, так і ретранслятора даних, створюючи динамічну топологію, що реагує на зміни положення або стану вузлів. За допомогою інструментів WifiHelper з відповідною конфігурацією MAC-рівня можна задати режим роботи мережі як ad-hoc, додатково поєднуючи з MobilityHelper для задання довільних моделей руху, що особливо корисно для дослідження систем тактичного зв'язку в умовах військових операцій або екстрених ситуацій, де розгортання інфраструктури неможливе чи недоцільне. Вузли в такій мережі можуть переміщуватись відповідно до моделей руху, що задаються вручну або з використанням готових шаблонів — від випадкових блуждань до цільового переміщення з фіксованою швидкістю. У поєднанні з підтримкою повторного підключення, зниженням якості сигналу та інших динамічних параметрів зв'язку це дозволяє емулювати реалістичні сценарії з високою точністю (див. рис. 1.12) [18].

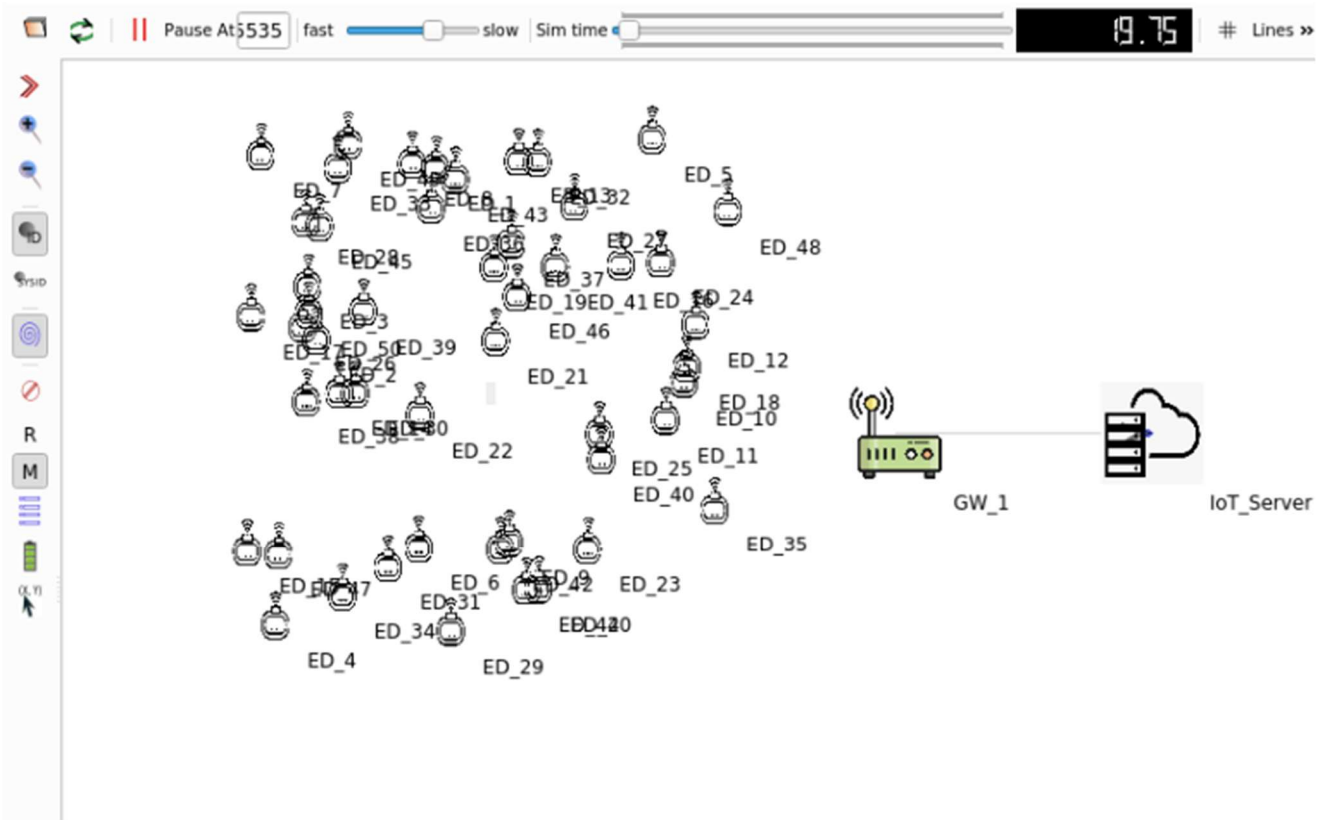


Рисунок 1.12 – Приклад adhoc-мережи IoT в NS-3

Особливо перспективним напрямом є підтримка IoT-сценаріїв, де NS-3 дозволяє будувати мережі з великою кількістю сенсорних вузлів із різними характеристиками енергоспоживання, рівнем взаємодії та фізичними інтерфейсами. Використання таких технологій як LoRa, LTE-M, NB-IoT або IEEE 802.15.4 забезпечується за рахунок відповідних модулів, які імітують специфіку передачі даних у реальних умовах з урахуванням обмежень по живленню, затримках та втраті пакунків. Можна створити ієрархічні мережі, де сенсорні вузли взаємодіють із центральними агрегаторами, періодично надсилаючи дані, з урахуванням моделей енергоспоживання, що дозволяє об’єктивно оцінити ефективність мереж у різних умовах. Завдяки можливості задавати профілі генерації трафіку, часові інтервали, сценарії відмов і повторних запусків NS-3 стає ідеальним інструментом для моделювання міських IoT-інфраструктур, контролю навколишнього середовища, промислових систем або систем розумного дому [18].

Додатково важливою функцією є підтримка мобільних вузлів, що передбачає можливість зміни розташування пристроїв у реальному часі з урахуванням фізичних моделей руху та впливу цих змін на параметри зв’язку. Різні типи моделей мобільності, такі як Random Walk, Constant Velocity або Gauss-Markov, дозволяють дослідити вплив швидкості та напрямку руху на якість зв’язку, стабільність маршрутизації та ефективність механізмів передачі. Поєднання сценаріїв мобільності з моделями втрати сигналу та handover дозволяє вивчати ситуації, характерні для рухомих об’єктів — пішоходів, автомобілів або дронів — у міських і сільських умовах, що забезпечує реалістичну емуляцію поведінки користувачів у сучасних Wi-Fi або LTE/5G мережах [18].

Налаштування параметрів (відстань, потужність, затримка). Однією з найсильніших сторін платформи NS-3 є її виняткова гнучкість у налаштуванні параметрів фізичного та канального рівнів, що дозволяє максимально точно відтворити умови реального мережевого середовища для моделювання різних типів бездротових мереж, що охоплює налаштування просторового розташування вузлів, потужності передавання, затримок сигналу, загасання, втрат і перешкод. Відстань між вузлами задається через координати, використовуючи модуль

MobilityHelper із такими параметрами, як GridPositionAllocator, що дозволяє формувати топології типу решітки, симетричні або випадкові. Визначення потужності сигналу, наприклад, для Wi-Fi або LTE, здійснюється через параметри на кшталт wifiPhy.Set("TxPowerStart", DoubleValue(16.0));, що прямо впливає на дальність передачі, якість з'єднання та енергоспоживання пристроїв (див. рис. 1.13) [19].

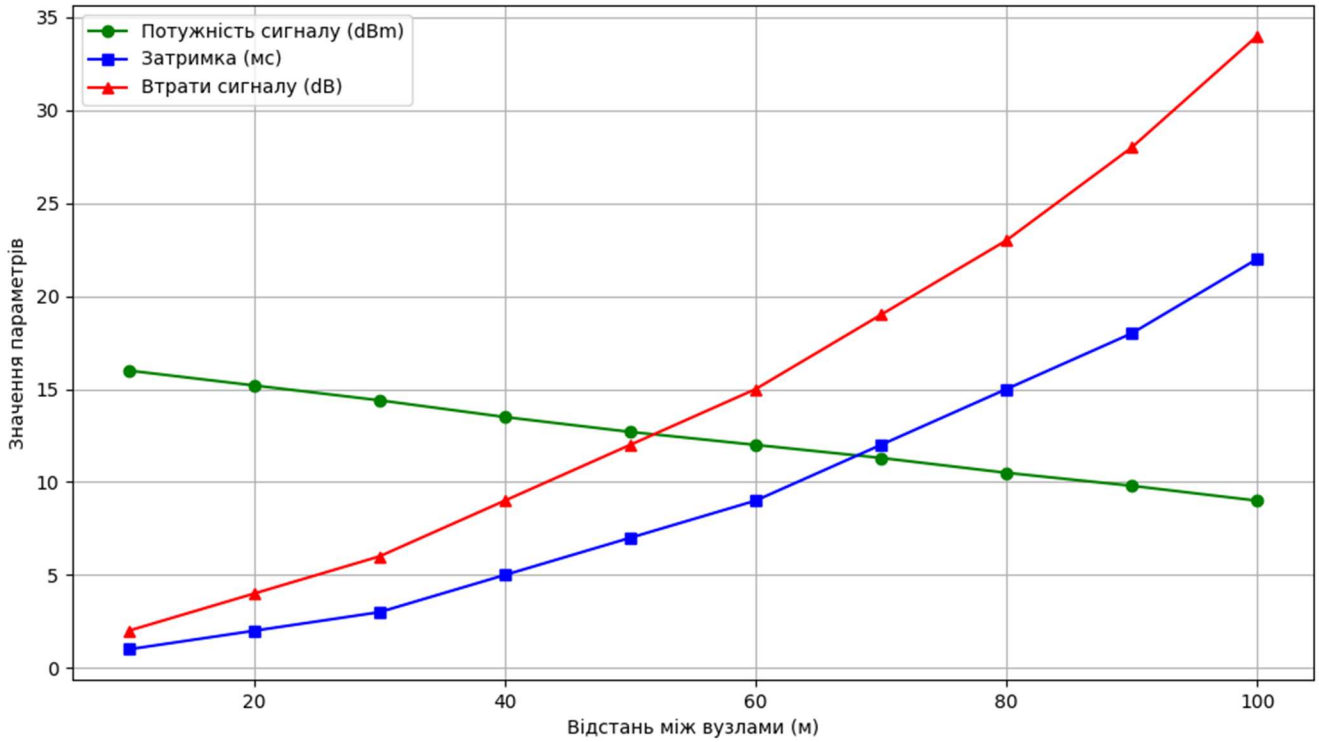


Рисунок 1.13 – Вплив параметрів моделі: потужність, затримка, відстань

Крім того, можна застосовувати моделі затримки (PropagationDelayModel) та втрат (PropagationLossModel), що дозволяє імітувати ефекти реального середовища — затухання, колізії, перешкоди в ефірі, включно з впливом об'єктів чи зміни положення вузлів [19].

Обробка результатів (логування, графіки, трасування). Разом із детальним налаштуванням фізичних умов, NS-3 надає широкі можливості для обробки результатів моделювання, що дозволяє аналізувати ефективність протоколів, сценаріїв або окремих параметрів мережі. Система логування дозволяє виводити детальну інформацію про стан модулів, проходження пакетів, помилки та

маршрутизацію через інструкції на кшталт LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);, що забезпечує повний контроль над тим, що відбувається в системі. Для глибшого аналізу передбачено генерацію трасувальних файлів: pcap для перегляду трафіку у Wireshark або ascii для перегляду подій у текстовому форматі. Ці механізми дозволяють визначити час затримки, втрати пакетів, повторні передачі та інші критично важливі параметри. Інтеграція модуля FlowMonitor автоматизує збір статистики по TCP/UDP потоках, у тому числі метрики пропускної здатності, середньої затримки, джитеру чи кількості втрат, з можливістю збереження результатів у форматах XML чи CSV для подальшої обробки в Gnuplot, Excel або Python-бібліотеках на кшталт Pandas і Matplotlib (див. рис. 1.14) [20].

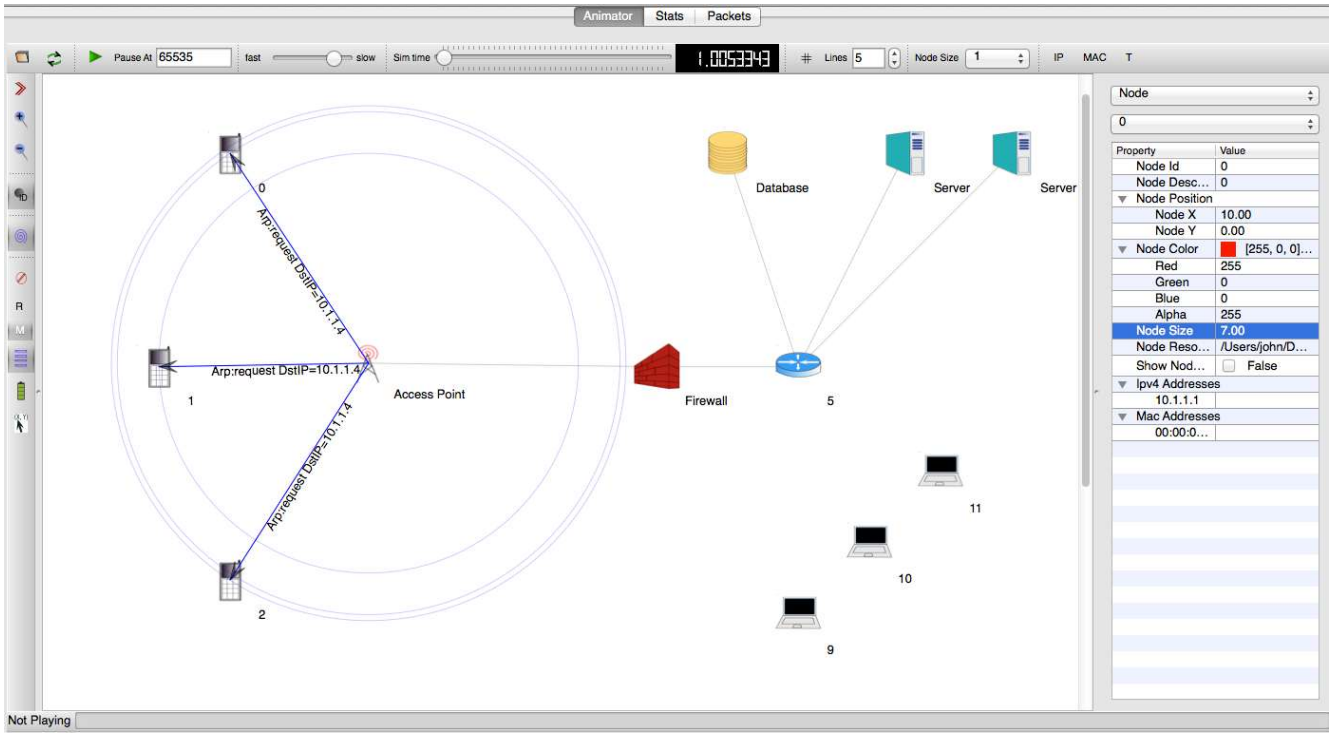


Рисунок 1.14 – Візуалізація логів та графіків з симуляцій NS-3

Незважаючи на текстовий характер виведення інформації в NS-3, графічна інтерпретація даних відіграє критичну роль у розумінні поведінки мережі. Візуалізація здійснюється через сторонні інструменти. Наприклад, NetAnim дозволяє бачити переміщення вузлів та маршрути проходження пакетів у

реальному часі у вигляді анімації. Побудова графіків і діаграм на основі логів — ще один метод для інтерпретації результатів, що виконується через Gnuplot або засоби візуалізації на Python, що дозволяє виявити залежності між змінами параметрів і результатами моделювання, візуалізувати навантаження на мережу, ефекти обривів зв'язку, зміни топології або продуктивність при зміні потужності та відстані. У підсумку така комбінація текстової, числової та візуальної інформації забезпечує повне та гнучке уявлення про досліджувану бездротову систему, дозволяючи моделювати, аналізувати й адаптувати складні сценарії з високою точністю [20].

Підтримка кастомних конфігурацій та експериментів. NS-3 розроблений як платформа для гнучкого налаштування, масштабування та розширення, що дозволяє створювати експерименти будь-якої складності без обмежень, характерних для більшості простих симуляторів, які не здатні забезпечити підтримку великих топологій або складної поведінки. Завдяки своїй архітектурі він дозволяє здійснювати повноцінну параметризацію сценаріїв, що відкриває можливість запуску серій симуляцій з варіативними параметрами, наприклад у batch mode, де автоматичне виконання та порівняння результатів значно полегшує дослідницький процес. При цьому йдеться не лише про використання готових моделей, а й про глибоку підтримку кастомізації, включаючи створення власних протоколів, модулів, моделей транспортного рівня або алгоритмів маршрутизації, які можна реалізовувати на основі існуючих класів або повністю з нуля відповідно до потреб конкретного дослідження чи експерименту.

Таке середовище ідеально підходить для дослідження нових підходів до організації мереж, поведінки користувачів або управління енергоспоживанням, з можливістю розширення функціоналу шляхом інтеграції з зовнішніми інструментами, зокрема модулями машинного навчання. Підтримка мови Python дозволяє поєднувати NS-3 з такими бібліотеками як TensorFlow або PyTorch для реалізації адаптивного управління мережею, або з Pandas та Scikit-learn для поглибленого аналізу результатів симуляцій, що особливо корисно при роботі з великими обсягами даних. Більш того, розгортання симуляцій у хмарних

середовищах або CI/CD-платформах можливе завдяки сумісності з Docker, що дає змогу виконувати моделювання у віддалених або автоматизованих інфраструктурах без потреби у ручному налаштуванні [21].

Підсумовуючи, можливості NS-3 у моделюванні як існуючих, так і гіпотетичних бездротових мереж охоплюють повний цикл – від проектування та налаштування середовища до виконання і збору детальної статистики з подальшим аналізом. Підтримка різноманітних технологій зв’язку, можливість глибокої конфігурації середовища, вузлів і протоколів, а також адаптивна інтеграція з інструментами аналізу та автоматизації роблять цю платформу універсальним інструментом для досліджень, прототипування та перевірки ефективності мережевих рішень. У сучасному контексті розвитку бездротових технологій NS-3 виступає як незамінна основа для побудови інноваційних моделей комунікації, управління ресурсами і тестування складних концепцій, що потребують достовірної емуляції та гнучкого середовища для їх верифікації [21].

РОЗДІЛ 2 АРХІТЕКТУРА ТА РЕАЛІЗАЦІЯ ДОСЛІДНИЦЬКОЇ СИСТЕМИ

2.1 Загальна архітектура дослідницької системи

Для побудови дослідницької системи, призначеної для мережевої емуляції протоколів бездротового зв'язку, необхідно створити гнучку архітектуру, здатну забезпечити універсальність, масштабованість, швидке розгортання, повторюваність експериментів і підтримку широкого спектру середовищ. Така архітектура повинна охоплювати весь цикл — від конфігурації мережі й запуску симуляцій до збору результатів і аналізу. Головною вимогою до системи стає можливість моделювання протоколів, як вже існуючих у промислових стандартах, так і тих, що ще перебувають на стадії теоретичної розробки, з можливістю вільного налаштування параметрів середовища, вузлів, рівнів взаємодії та симульованих подій. Саме через потребу у швидкому масштабуванні та повторюваності було обрано підхід на основі контейнеризації — а саме платформу Docker Desktop, яка дозволяє створювати цілісні, незалежні образи системи, автоматично розгортати середовище за допомогою конфігураційного файлу та запускати дослідження без додаткових складностей із налаштуванням [22-25].

Використання Docker як базової платформи. У сучасних умовах розробки та тестування мережевих рішень контейнеризація, і зокрема використання Docker, стає оптимальним способом реалізації таких систем. На відміну від традиційної віртуалізації, Docker забезпечує ізоляцію процесів на рівні операційної системи, що дозволяє уникнути накладних витрат на запуск повноцінних віртуальних машин, зберігаючи при цьому високу гнучкість і контроль. Docker було обрано як основну платформу для створення середовища мережевого симулятора на базі NS-3 через його здатність автоматизовано розгортати систему з усіма залежностями, бібліотеками та сценаріями моделювання. За допомогою Dockerfile можливо створити середовище з повною кастомізацією — вибором версій компіляторів,

бібліотек, конфігурацій симуляції та методів обробки вихідних даних. Крім того, контейнеризація дозволяє жорстко фіксувати версії компонентів, що гарантує ідентичність результатів навіть при повторному виконанні симуляцій через тривалий час або на інших пристроях, незалежно від апаратного чи програмного середовища [22].

Контейнерна архітектура забезпечує масштабованість, що дозволяє запускати паралельно кілька експериментів в окремих контейнерах, легко змінювати їхню конфігурацію або відтворювати задані сценарії без необхідності повторного ручного налаштування. Завдяки цьому архітектура легко адаптується до потреб CI/CD-процесів, хмарних середовищ або локального запуску на персональному комп'ютері. Одні й ті самі контейнери працюють на Linux, Windows, macOS, що спрощує розгортання як на робочій машині дослідника, так і в продуктивному середовищі лабораторії. Docker не лише виконує функції платформи для запуску NS-3, а й виступає як інструмент для керування повноцінним середовищем дослідження, уможливлюючи обмін повністю готовими системами між розробниками, дослідниками або членами команди (див. рис. 2.1) [22].

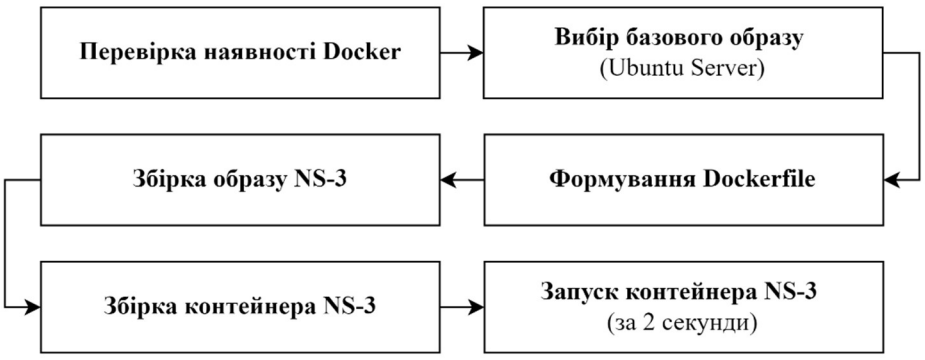


Рисунок 2.1 – Принцип реалізації системи емуляції бездротових мереж на базі Docker (NS-3)

Розгортання через Docker дає змогу мінімізувати часові витрати на підготовку та налаштування середовища, забезпечуючи готовність до досліджень практично одразу після створення образу. Всі експерименти, моделі, скрипти та

інструменти аналізу включаються до контейнера, що дозволяє миттєво відтворити повноцінну інфраструктуру дослідження у будь-якому середовищі. Завдяки цьому навіть складні симуляційні сценарії з великою кількістю параметрів і умов можна легко переносити, масштабувати або інтегрувати в більші системи. Архітектура на основі NS-3 і Docker дозволяє не лише реалізовувати дослідження з точністю до найдрібніших деталей, а й створювати універсальне середовище, придатне для спільного використання, перевірки гіпотез, тестування нових підходів і розробки експериментальних протоколів з високим рівнем достовірності результатів.

Структура контейнеризованого середовища. Контейнеризоване середовище, побудоване в межах даної дослідницької системи, реалізовано як спрощена, але повноцінна інфраструктура для моделювання мережевих сценаріїв з акцентом на дослідження бездротового зв’язку. Така структура складається з одного або кількох контейнерів, які забезпечують необхідну функціональність, а саме — запуск емуляцій, збирання результатів, збереження сценаріїв та параметризацію експериментів. Хоча конфігурація може змінюватися відповідно до завдань, реалізований варіант орієнтований на максимально швидке розгортання з можливістю подальшого масштабування та інтеграції більш складних рішень. Основу структури становить базовий образ, що містить платформу NS-3, обраний на основі дистрибутива Debian, хоча альтернативно міг бути використаний і Ubuntu. Всередині образу встановлено ключові залежності — компілятор g++, інтерпретатор Python 3, утиліти make, GTK, GSL, pkg-config тощо. У нашому випадку використовується повністю зібрана версія NS-3.41, яка зберігається у контейнері та забезпечує симуляційну основу [23].

Поруч із цим компонентом створено каталог simulation, у якому зберігаються вихідні .cc-файли сценаріїв, а також конфігураційні скрипти з можливістю запуску за допомогою Waf з аргументами командного рядка, що дозволяє виконувати сценарії динамічно, наприклад, через команду docker run --rm ns3-image ./waf --run "wifi-example". Поряд із цим існує каталог results, куди виводяться всі отримані в процесі моделювання результати, які зберігаються у форматах pcap, xml та log. Враховуючи обмеження доступу до внутрішньої файлової системи контейнера, для

організації зручного доступу використовується мапінг директорій, наприклад, через аргумент -v $(pwd)/results:/simulations/results. Конфігураційні файли, представлені у форматах JSON або YAML, дозволяють задавати параметри симуляцій як змінні середовища, що дає змогу адаптувати поведінку сценаріїв до конкретного завдання або експерименту, забезпечуючи кастомізацію без потреби зміни коду [23].

Взаємодія між компонентами (Docker, NS-3, інтерфейси). Функціональна взаємодія між усіма компонентами реалізована через кілька рівнів — від запуску контейнера до обробки результатів. Першим елементом цієї взаємодії є програмне забезпечення Docker, зокрема його версія Docker Desktop, яка забезпечує середовище виконання у Windows 10. Дане рішення автоматично використовує підсистему Windows для Linux (WSL), що дозволяє запускати контейнер на базі Linux-дистрибутива, ізольовано від основної ОС. Завдяки цьому симуляції виконуються в уніфікованому середовищі, незалежно від платформи користувача, що підвищує повторюваність експериментів. Сам Docker виконує роль інструменту управління контейнерами, забезпечує ізоляцію процесів, контроль версій образів, завантаження з Docker Hub або локального репозиторію, а також підтримує одночасну роботу з кількома ізольованими екземплярами (див. рис. 2.2).

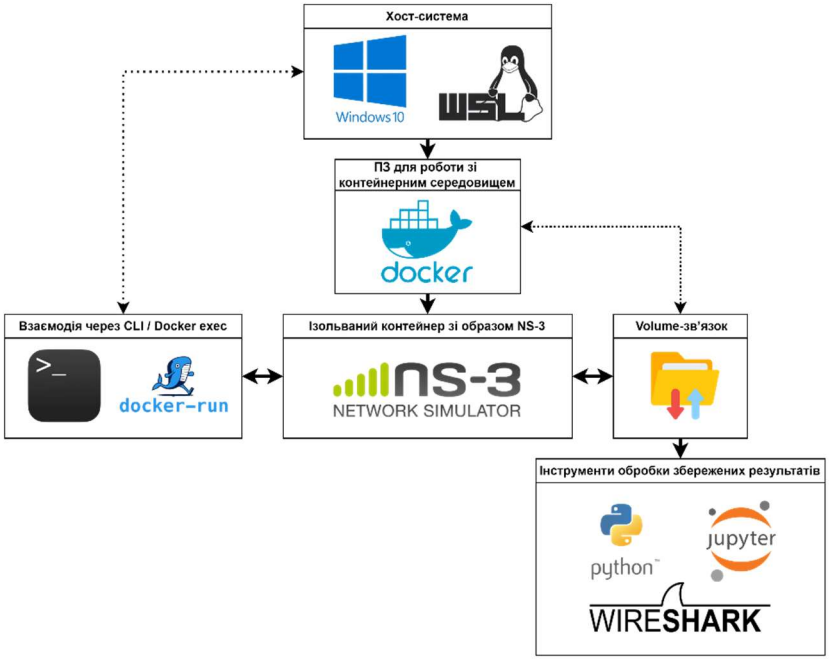


Рисунок 2.2 – Взаємодія між компонентами системи (Docker, NS-3, CLI)

Наступним елементом є контейнер із NS-3, який є логічним ядром дослідницької частини. Усередині цього контейнера виконується компіляція сценаріїв, їх параметризація, запуск, а також генерація результатів у вигляді звітів, графіків чи логів. Сам NS-3 запускається, наприклад, за допомогою команди docker exec ns3-container ./waf --run "mobility-sim --speed=10 --nodes=20", де параметри можуть бути змінені у командному рядку. Ця гнучкість дозволяє керувати процесом симуляції зовні, без потреби входити в контейнер, що особливо корисно в умовах масштабованих експериментів. При наявності кількох контейнерів можна емулювати паралельні мережеві процеси, не змінюючи базову архітектуру [24].

Останній елемент — інтерфейси обробки результатів, які охоплюють як файлову систему хост-машини, так і спеціалізовані аналітичні інструменти. Результати симуляцій мапуються у директорії хоста, що забезпечує простий доступ до вихідних файлів. Для обробки даних можуть використовуватись Python-скрипти або середовища Jupyter, які здатні запускатись або всередині контейнера, або зовнішньо, на хості. За допомогою таких інструментів виконується аналіз PCAP-трейсів, побудова графіків, обчислення затримок та візуалізація проходження пакетів. Оскільки вся система побудована на контейнерах, реалізується також підтримка взаємодії між кількома симуляційними екземплярами. Наприклад, при використанні docker-compose up --scale, можна запускати кілька паралельних контейнерів, які емулюють різні конфігурації мереж або порівнювані протоколи зв’язку [24].

Таблиця 2.1 — Компоненти дослідницької архітектури та їх функції

Компонент	**Функція**
Docker Engine	Платформа контейнеризації, що забезпечує ізоляцію процесів, автоматизацію розгортання середовища та кросплатформенність (Windows/macOS/Linux)
Контейнер NS-3	Віртуалізоване середовище для запуску емулятора, що містить усі необхідні бібліотеки, компілятор, інструменти та версію NS-3.41 для моделювання мереж

Продовження таблиці 2.1 — Компоненти дослідницької архітектури та їх функції

Компонент	**Функція**
Хост-система	Фізична або віртуальна машина, що запускає Docker та виконує роль середовища розробки або експлуатації (дослідницький ПК, CI/CD, лабораторія)
Тестові сценарії	Файли .cc, скрипти Waf, конфігурації YAML/JSON, які описують мережеву топологію, протоколи, параметри мобільності тощо
Конфігураційні файли	Визначають параметри емуляцій через змінні середовища або CLI-аргументи, забезпечують кастомізацію без модифікації коду сценаріїв
Каталог simulation	Директорія для зберігання сценаріїв моделювання, які виконуються всередині контейнера за допомогою waf
Каталог results	Зовнішньо змонтована директорія для виводу результатів (.pcap, .log, .xml), яка забезпечує доступ з хост-системи
Лог-система / Аналіз	Інструменти (Python, Jupyter, Wireshark), що аналізують результати, будують графіки, оцінюють параметри (затримки, втрати, RTT, тощо)
Dockerfile	Шаблон створення образу контейнера з усіма залежностями, версіями та інструментами, забезпечує відтворюваність експериментів
Docker Compose	Засіб оркестрації для масштабування симуляцій за рахунок запуску кількох контейнерів з різними сценаріями або параметрами
Інтерфейси взаємодії	CLI-інструменти Docker, обмін файлами через volume mapping, взаємодія з контейнерами без входу в них (docker exec, docker run, --env)

Така структура (див. табл. 2.1) дозволяє ізолювати, автоматизувати та масштабувати мережеві експерименти, зберігаючи високу ступінь керованості та повторюваності, що робить її придатною як для дослідницької роботи, так і для навчальних демонстрацій чи віддаленого тестування нових протоколів без потреби у фізичному мережевому обладнанні [24].

Гнучкість та портативність підготовленої архітектури. Система, побудована на контейнеризованій архітектурі, зокрема в реалізації дослідницької мережевої емуляційної платформи, забезпечує високий рівень гнучкості, масштабованості та портативності, що є особливо важливим у контексті досліджень, тестування гіпотез, створення навчальних демонстрацій або розгортання повторюваних симуляційних середовищ. Завдяки такій архітектурі конфігурації можна змінювати без необхідності перекомпіляції образів — це досягається шляхом редагування змінних середовища або використання монтування зовнішніх томів, що значно прискорює підготовку і адаптацію середовища до нових умов. До аспектів гнучкості також належить підтримка кількох конфігурацій одночасно, що дозволяє швидко перемикатися між різними мережевими топологіями, протоколами чи сценаріями моделювання, а також поступово розширювати можливості системи, додаючи нові модулі — наприклад, засоби логування через Kafka, обробку результатів симуляцій за допомогою скриптів на Python або інтеграцію з аналітичними панелями [25].

Окрім гнучкості, реалізація забезпечує повну портативність, оскільки один контейнер може містити повноцінну платформу, яку легко поширювати у вигляді Docker-образу чи Dockerfile, що дає змогу швидко перенести готове середовище на інший пристрій або іншу операційну систему без необхідності повторної інсталяції залежностей — єдина умова полягає в наявності Docker-оточення, зокрема Docker Desktop або його альтернатив. Система також може бути інтегрована у будь-який сучасний CI/CD пайплайн, наприклад GitHub Actions, GitLab CI або Jenkins, що дозволяє автоматизувати запуск симуляцій, збір результатів і навіть підготовку звітів. Розгортання в таких середовищах стає можливим завдяки стандартизованій

структурі контейнерів, яка дозволяє точно відтворити середовище незалежно від платформи, де відбувається виконання [25].

Масштабованість досягається завдяки підтримці мультиконтейнерних середовищ, які можуть бути створені за допомогою Docker Compose, що особливо корисно для одночасного запуску групових експериментів. В рамках цієї архітектури моделювання та емуляція можуть бути розподілені на кластери або розгорнуті у хмарному середовищі з використанням Docker Swarm або Kubernetes. Такий підхід дає змогу запускати паралельні симуляції або емулювати великі мережеві сценарії, зберігаючи контроль над ресурсами та конфігурацією. Розширення системи охоплює і можливість глибокої інтеграції з іншими сервісами — наприклад, запуск NS-3 симуляцій у середовищі, що включає Jupyter Notebook, обробку вхідних даних з баз даних, формування звітів у форматі PDF або LaTeX, що автоматично генеруються після кожного експерименту з використанням бібліотек на кшталт Pandas [25].

Таким чином, архітектура дослідницького мережевого емулятора, побудованого на контейнеризації, забезпечує не лише високий рівень адаптивності до потреб користувача, але і стабільність, повторюваність результатів та легкість розгортання на будь-якому апаратному чи програмному забезпеченні, що дозволяє проводити повноцінні дослідження, моделювати та розробляти бездротові протоколи зв’язку у повністю контрольованому середовищі без необхідності занурення у складні залежності системи. Поєднання симуляційного ядра NS-3 з гнучкістю контейнерної інфраструктури створює універсальну платформу, яка однаково добре підходить як для локального використання, так і для віддалених кластерних обчислень, у тому числі з автоматизованим збиранням результатів і формуванням статистики в режимі реального часу.

2.2 Налаштування та розгортання середовища NS-3 у Docker

Після опрацювання архітектури системи, побудованої на базі контейнеризації, наступним кроком стає налаштування та розгортання середовища емуляції NS-3 у Docker. Йдеться про реалізацію повноцінного ізольованого середовища, здатного виконувати дослідження протоколів бездротового зв’язку на основі симулятора NS-3. Реалізація здійснюється у Docker Desktop з використанням WSL2, що автоматично встановлюється разом із основною платформою, оскільки сама робота з Docker передбачає наявність Linux-середовища навіть на системах із Windows. У контексті поточної реалізації середовища йдеться про систему на базі Windows 10, яка виступає як хост для всієї конфігурації. Початковим етапом розгортання є встановлення Docker Desktop та увімкнення підсистеми Linux для Windows, після чого забезпечується повна готовність до використання терміналу для роботи з контейнеризованими середовищами [26-29].

Клонування офіційного NS-3-докер-репозиторію. Далі відбувається підготовка самого симуляційного середовища шляхом отримання готового образу або, як у даному випадку, повноцінного репозиторію з усім необхідним функціоналом. Проєкт NS3D у Docker — це офіційний спільнотний репозиторій, підтримуваний безпосередньо розробниками NS-3, який містить весь набір файлів, потрібних для запуску, конфігурування, збирання та оновлення симуляційного середовища. Репозиторій включає в себе Dockerfile з базовими інструкціями щодо формування образу, а також bash-скрипти для спрощення збирання і запуску, такі як build.sh та run.sh, орієнтовані на використання в Linux або через термінал WSL. Щоб отримати доступ до вмісту цього середовища, достатньо виконати послідовні команди git clone https://gitlab.com/nsnam/ns-3-dev-docker.git та cd ns-3-dev-docker, після чого відкривається повна структура з усім необхідним (див. рис. 2.3) [26].



Рисунок 2.3 – Клонування репозиторію NS-3 Docker

Усередині репозиторію знаходиться не лише опис Dockerfile для створення середовища, а й додаткові скрипти, інструменти, підкаталоги для взаємодії з мережею, трасування пакетів, маршрутизації та інших допоміжних функцій, які забезпечують не тільки запуск, а й комфортну інтеграцію з зовнішніми джерелами або сервісами. Завдяки такій структурі контейнер може бути легко адаптований для виконання різноманітних задач, починаючи від запуску окремих моделей і завершуючи побудовою складних симуляційних сценаріїв з інтеграцією в більш масштабні CI/CD пайплайни. Репозиторій підтримується в актуальному стані, постійно оновлюється відповідно до нових релізів NS-3, а також містить посилання на документацію й приклади використання, що дозволяє швидко зануритися у роботу з платформою навіть без глибокого попереднього досвіду. Усе це забезпечує ефективне, автоматизоване та контрольоване розгортання симуляційного середовища для подальшого дослідження мережевих протоколів у відтворюваному середовищі [26].

Команди для створення та запуску контейнера. Після завершення процесу клонування офіційного репозиторію NS-3 у Docker середовищі наступним кроком стає створення власного образу та запуск контейнера, який забезпечить ізольоване середовище для виконання емуляцій мережевих протоколів (див. табл. 2.2).

Таблиця 2.2 — Основні команди для роботи з NS-3 у Docker-контейнері

Команда	**Опис дії**	**Приклад використання**
docker build	Створення образу Docker з Dockerfile	docker build -t ns3 .
docker run	Запуск контейнера на основі образу	docker run -it ns3 bash
./waf configure	Конфігурація перед компіляцією у NS-3	./waf configure
./waf build	Компіляція симулятора	./waf build
./waf --run <назва_сценарію>	Запуск конкретного сценарію	./waf --run wifi-simple-adhoc

В умовах використання Docker Desktop на операційній системі Windows 10 потрібно перейти в директорію з клонованим репозиторієм і вже з цього місця виконати команду docker build -t ns3-docker ., яка ініціює побудову Docker-образу з назвою ns3-docker на базі дистрибутива Ubuntu 22.04, автоматично додаючи в нього основні інструменти для розробки, зокрема компілятор g++, систему збирання cmake, інтерпретатор python3, а також бібліотеки для графічного представлення результатів моделювання, такі як GTK, GSL та NetAnim. Цей образ також включає набір залежностей, потрібних для компіляції та симуляції моделей бездротового зв’язку, зокрема протоколів LoRa, Wi-Fi, LTE, 5G тощо, що уможливлює гнучке моделювання мережевих сценаріїв для досліджень [27].

Після створення образу, він зберігається в локальному сховищі Docker як шаблон, проте сам контейнер на цьому етапі ще не існує. Для запуску контейнера необхідно використати команду docker run -it --name ns3-sim ns3-docker, яка ініціює запуск в інтерактивному режимі з повним доступом до внутрішнього термінала середовища /home/ns3/ns-3-dev. Такий підхід дозволяє безпосередньо запускати сценарії емуляції одразу після входу в контейнер. Щоб забезпечити збереження результатів виконання симуляцій безпосередньо на локальній файловій системі

хост-машини, доцільно змонтувати відповідну директорію за допомогою команди docker run -it --name ns3-sim -v $(pwd)/results:/home/ns3/ns-3-dev/results ns3-docker, завдяки чому всі отримані логи, файли формату .pcap, графіки та інші результати аналізу будуть зберігатися у локальній директорії results, що спрощує доступ до них після завершення симуляції [27].

Організація директорій для запуску тестів. Щодо внутрішньої структури директорій у середовищі NS-3 всередині контейнера, вона має чітко визначену ієрархію, яка полегшує навігацію та організацію тестів. Основним коренем є директорія /home/ns3/ns-3-dev/, у якій розміщено кілька ключових підкаталогів. Директорія scratch/ використовується для збереження кастомних симуляційних сценаріїв, де кожен окремий файл відповідає окремому експерименту. examples/ містить вбудовані приклади для різноманітних технологій, серед яких Wi-Fi, TCP, LoRa, та інші, що дозволяє швидко ознайомитися з базовими можливостями емулятора. У підкаталозі src/ зосереджено вихідний код симуляційних модулів, включаючи реалізації мережевих протоколів та стеків, а results/ відповідає за зберігання вихідних файлів симуляцій і є змонтованим ресурсом, який забезпечує зручний обмін файлами з хостовою системою. Директорія build/ виконує функцію буфера для проміжних об’єктів, згенерованих під час компіляції, тоді як файл waf використовується як основний інтерфейс для побудови та запуску сценаріїв [28].

Запуск певного симуляційного сценарію може бути реалізований через інтерфейс waf, шляхом виконання команди ./waf --run "scratch/….", що дозволяє відтворити заданий експеримент безпосередньо з директорії scratch. (див. рис. 2.4).

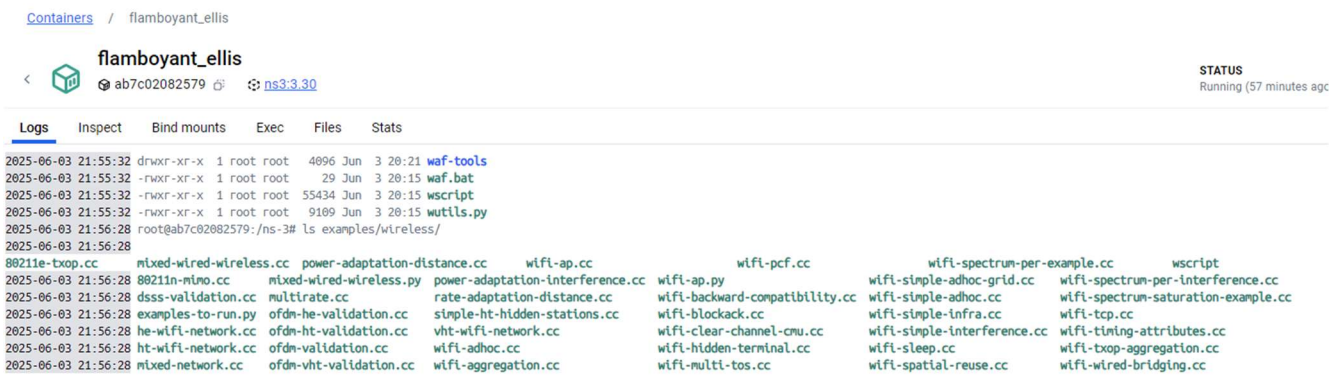


Рисунок 2.4 – Структура директорій для тестових сценаріїв NS-3

Такий підхід відкриває можливості для автоматизації серій експериментів шляхом зміни конфігурації за допомогою Bash- або Python-скриптів, що є корисним під час проведення досліджень із варіативними умовами моделювання [28].

Налаштування середовища для підтримки Wi-Fi/LoRaWAN симуляцій. Після реалізації попередніх етапів для забезпечення повноцінної підтримки Wi-Fi та LoRaWAN симуляцій у середовищі NS-3 необхідно виконати додаткове налаштування, що полягає у перевірці активності відповідних модулів. Контейнер ns3-docker за замовчуванням містить увімкнені модулі Wi-Fi, LoRaWAN, LTE та інші основні компоненти, однак у випадку зміни конфігурації або ручного створення образу доцільно переконатися в їх активності за допомогою команди ./waf list, яка дозволяє отримати повний перелік зібраних модулів. У разі їх відсутності або необхідності включення нових, варто застосувати команду ./waf configure --enable-modules=lorawan,wifi із подальшою збіркою проєкту через ./waf build, що дозволить інтегрувати відповідні функціональні можливості безпосередньо у середовище емулювання [29].

Модуль Wi-Fi у складі NS-3 підтримує моделювання протоколів IEEE 802.11 усіх основних специфікацій, включаючи a/b/g/n/ac/ax, а також містить реалізацію моделей мобільності, пропускної здатності, інтерференції та підтримку режимів STA/AP/ad-hoc, що дозволяє детально симулювати поведінку бездротових мереж у складних умовах. Для LoRaWAN, починаючи з версії NS-3.33, офіційно доступний відповідний модуль, який охоплює фізичний (PHY) та канальний (MAC) рівні, дозволяючи симулювати передачу даних у діапазонах 868 та 915 МГц, а також моделювати роботу енергоефективних вузлів з великими інтервалами передачі [29].

Запуск симуляцій здійснюється стандартним механізмом waf, зокрема для LoRaWAN прикладом є команда ./waf --run "wifi-simple-adhoc", яка активує базовий сценарій з підтримкою типових параметрів.

У випадку необхідності розширення функціональності підтримкою специфічного протоколу, наприклад NB-IoT або будь-якого власного варіанту, можливо створити нову директорію всередині каталогу src/, розмістити там

відповідні файли та оновити wscript або CMake залежно від конфігурації для інтеграції нових модулів у середовище [29].

Таблиця 2.3 — Кроки виконання симуляції та результати

Крок	**Дія**	**Результат**
1	Запуск Docker-контейнера з образом NS-3	Симуляційне середовище готове до роботи
2	Виконання сценарію через docker run або docker exec з параметрами	Запуск емуляції з заданими умовами
3	Автоматичне збереження результатів у змонтовану директорію results/	Отримання логів, PCAP-файлів, XML-звітів
4	Перевірка результатів через інструменти аналізу (наприклад, Python-скрипти або Jupyter)	Виведення графіків, статистик, трас аналізу трафіку
5	Повторення або масштабування запусків із новими параметрами через Docker Compose	Паралельне моделювання та порівняння декількох кейсів
6	Перевірка Web UI або лог-файлів у разі додаткової інтеграції з веб-інтерфейсом (не обов’язково)	Візуальне представлення даних

Загалом налаштування симуляційного середовища у межах контейнеризованого рішення на базі Docker (див. табл. 2.3) вирізняється прямолінійністю, однак вимагає чіткого розуміння структури NS-3, механізмів взаємодії модулів, а також принципів роботи контейнерів. Завдяки такому підходу досягається висока повторюваність експериментів, ізольованість середовища від системи хоста та гнучкість у конфігурації протоколів, що критично для цілей тестування, освітніх курсів, наукових досліджень і впровадження у CI/CD середовища. Підготовлене середовище забезпечує змогу повноцінно запускати сценарії Wi-Fi, LoRaWAN та LTE без потреби додаткового встановлення бібліотек чи утиліт поза межами

контейнера, що мінімізує вплив зовнішніх чинників і полегшує обмін сценаріями між учасниками команди або спільнотою.

2.3 Алгоритми проведення емуляцій протоколів Wi-Fi/LoRaWAN

Після завершення налаштування середовища та розміщення всіх необхідних компонентів у Docker-контейнері з підтримкою модулів Wi-Fi та LoRaWAN симулятор NS-3 стає повністю готовим до запуску симуляцій бездротових протоколів. Під час реалізації таких сценаріїв важливо не лише правильно організувати вихідні файли та залежності, а й врахувати послідовність виклику необхідних команд та модулів, які будуть задіяні у процесі. Симуляції бездротових протоколів дозволяють відтворити реальні умови передачі даних, оцінити якість каналу, затримки, вплив перешкод і взаємодію між вузлами у різних топологіях, тому доцільно звертатися до прикладів з офіційної документації, де наведені репрезентативні базові конфігурації, які можна адаптувати під власні потреби. Таким чином, підхід до симуляції зосереджується не лише на виконанні коду, але й на аналітичному підході до побудови сценаріїв, оцінювання результатів та порівняння ефективності різних реалізацій.

Опис тестових сценаріїв (wifi-simple-adhoc). У випадку з протоколом Wi-Fi одним із найтиповіших сценаріїв, що демонструє базовий принцип роботи бездротового з’єднання між вузлами без використання точки доступу, є wifi-simple-adhoc, розташований у каталозі scratch, який реалізується як C++-скрипт. Сценарій передбачає створення ad-hoc мережі з двома вузлами, що взаємодіють один з одним напряму. Реалізація починається з ініціалізації об’єкта NodeContainer nodes; та виклику nodes.Create(2); для створення двох вузлів, після чого налаштовується фізичний рівень за допомогою YansWifiChannelHelper channel = YansWifiChannelHelper::Default();, YansWifiPhyHelper phy = YansWifiPhyHelper::Default(); та phy.SetChannel(channel.Create());. Далі

макрорівень налаштовується через WifiMacHelper mac; та mac.SetType("ns3::AdhocWifiMac");, що дозволяє вузлам функціонувати у режимі прямої взаємодії без центральної точки керування. Після цього задаються IP-адреси через Ipv4AddressHelper address; address.SetBase("10.1.1.0", "255.255.255.0");, а для передачі даних запускається додаток UdpEchoServerHelper echoServer(9); з наступною ініціалізацією клієнта, визначенням часу завершення симуляції через Simulator::Stop(Seconds(10.0));, запуском симулятора Simulator::Run(); та очищенням ресурсів Simulator::Destroy();. Такий базовий шаблон можна масштабувати або змінювати під будь-який інший сценарій, модифікуючи кількість вузлів, їхній розподіл, тип протоколу або рівень навантаження (див. рис. 2.5).



Рисунок 2.5 – Алгоритм роботи сценарію wifi-simple-adhoc

Для LoRaWAN прикладом типової симуляції є сценарій lorawan-end-device, що демонструє передачу даних від вузла до шлюзу через LoRa PHY. У даному сценарії можливо налаштовувати ключові параметри фізичного рівня, зокрема частоту (868 чи 915 МГц), силу сигналу, коефіцієнт розпилення (Spreading Factor), клас пристрою та інтервали передачі. Ці параметри визначають характер зв'язку між малопотужними пристроями, які функціонують з мінімальним енергоспоживанням та довготривалими паузами між пакетами. Подібно до прикладу з Wi-Fi, сценарій включає створення вузлів, налаштування фізичного і MAC рівнів, конфігурацію адресації, ініціалізацію додатків та запуск симуляції.

У разі необхідності підтримки специфічного протоколу, як-от NB-IoT, можна розширити функціональність симулятора шляхом додавання нового модуля до каталогу src, після чого внести зміни у відповідні wscript-файли для інтеграції нових компонентів у систему збірки. Завдяки такому підходу забезпечується можливість кастомізації під специфічні дослідницькі чи прикладні потреби. Важливо, що запуск усіх згаданих сценаріїв здійснюється безпосередньо в межах Docker-контейнера, що гарантує повторюваність експериментів, ізольованість середовища та незалежність від системи розробника, а також дає змогу інтегрувати ці тести у CI/CD-пайплайни, наукові дослідження чи навчальні лабораторії без додаткового встановлення зовнішніх бібліотек.

Можливість зміни параметрів симуляції (частоти, затримки). Після завершення реалізації системи, коли її архітектура повністю сформована, середовище налаштоване, а функціональні компоненти розміщено у відповідних контейнерах, постає необхідність у проведенні серії емуляцій або симуляцій, які дозволяють оцінити роботу протоколів бездротового зв'язку, зокрема Wi-Fi та LoRaWAN. Ці експерименти демонструють, наскільки коректно функціонує програмне середовище, наскільки реалістично передається мережева динаміка, і надають можливість зафіксувати параметри, що критично впливають на продуктивність. У цьому контексті доцільно звернутися до вже підготовлених сценаріїв, що доступні у складі офіційного репозиторію NS-3, деякі з яких прямо

згадані у супровідній документації, що підтверджує їхню достовірність і придатність до тестування базових мережевих моделей.

Одним із базових прикладів для Wi-Fi є сценарій wifi-simple-adhoc, що знаходиться у каталозі scratch і моделює побудову adhoc-мережі з двома вузлами, які взаємодіють без використання точки доступу. Створення сценарію відбувається у файлі на мові C++ із використанням базових класів NS-3, де спочатку створюються вузли через NodeContainer nodes; nodes.Create(2);, після чого задається фізичний рівень Wi-Fi через YansWifiChannelHelper і YansWifiPhyHelper із відповідним прив’язуванням каналу, далі конфігурується MAC-рівень у режимі adhoc через WifiMacHelper, і присвоюються IP-адреси за допомогою Ipv4AddressHelper. Після мережевої конфігурації відбувається запуск серверного додатку UdpEchoServerHelper, і відповідно запуск клієнта, що дозволяє протестувати базовий зв’язок. Час моделювання задається через Simulator::Stop(Seconds(10.0));, а виконання завершується командами Simulator::Run(); Simulator::Destroy();, що підводить до завершення симуляції. Зміну типу протоколу або кількості вузлів, варіацію додатку або інтерфейсу надзвичайно легко реалізувати, завдяки модульності скрипту. У випадку LoRaWAN, типовим прикладом є сценарій lora-one-end-device, у якому емулюється передача від одного LoRa-пристрою до шлюзу, з можливістю конфігурації частоти передачі, потужності сигналу, spreading factor та затримок, які суттєво впливають на характер зв’язку.

Крім модифікації структури сценарію, також можлива гнучка зміна параметрів симуляції, які визначають фізичні, логічні й часові характеристики експерименту. Серед них найбільш поширеними є частота каналу, потужність передавача, затримки пересилання, мобільність вузлів тощо. Для Wi-Fi, наприклад, зміна частоти здійснюється через команду phy.Set("ChannelNumber", UintegerValue(11));, яка встановлює передачу на частоті 2.462 ГГц, тоді як зміна потужності може бути реалізована через phy.Set("TxPowerStart", DoubleValue(16.0)); і phy.Set("TxPowerEnd", DoubleValue(16.0));, що дозволяє задати сталу вихідну потужність у dBm. Аналогічно, для LoRaWAN можна

змінювати параметри топології або мобільності через mobility.SetPositionAllocator(...), де задається схема розташування вузлів, інтервали між ними та тип сітки (див. рис. 2.6).

```
20 #include "ns3/point-to-point-module.h"
21 #include "ns3/applications-module.h"
22
23 // Default Network Topology
24 //
25 //       10.1.1.0
26 // n0 -------------- n1
27 //    point-to-point
28 //
29
30 using namespace ns3;
31
32 NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
33
34 int
35 main (int argc, char *argv[])
36 {
37   CommandLine cmd (__FILE__);
38   cmd.Parse (argc, argv);
39
40   Time::SetResolution (Time::NS);                    Thread 1: breakpoint 1.1 (1)
41   LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
42   LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
43
44   NodeContainer nodes;
```

Рисунок 2.6 – Приклад реалізації змін параметрів емуляції

Всі ці зміни можуть задаватися як безпосередньо в коді, так і через командний рядок під час запуску моделі, наприклад за допомогою ./waf --run "scratch/wifi-simple-adhoc --TxPower=18.0 --SimulationTime=20", що дозволяє ефективно автоматизувати серії експериментів, варіюючи одні й ті самі моделі з різними параметрами, отримуючи при цьому статистично релевантні результати.

Обробка та збереження логів результатів. Важливим аспектом після запуску симуляційного сценарію є збір, обробка та збереження результатів, що дозволяє не лише порівняти конфігурації, а й формалізувати результати у вигляді логів і графіків. Для цього NS-3 передбачає кілька механізмів, зокрема створення ASCII-логів, PCAP-знімків, івент-трейсингу та виведення додаткової статистики. Наприклад, генерація текстового лог-файлу виконується через AsciiTraceHelper ascii; phy.EnableAsciiAll(ascii.CreateFileStream("wifi-adhoc.tr"));, що зберігає вміст у форматі, придатному для подальшого аналізу. PCAP-файл можна створити за

допомогою phy.EnablePcapAll("wifi-adhoc");, що дозволяє відкривати його у Wireshark для низькорівневої перевірки. Для збору даних про отримання чи передачу пакетів використовується механізм підключення зворотного виклику, наприклад через Config::Connect("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/MacRx", MakeCallback(&RxCallback));, де всі події прийому передаються у відповідну функцію обробки. Якщо ж необхідно отримати підсумкову статистику, вона доступна на рівні застосунків, як у випадку з udpEchoClientHelper.SetAttribute("MaxPackets", UintegerValue(5)); (див. рис. 2.7).

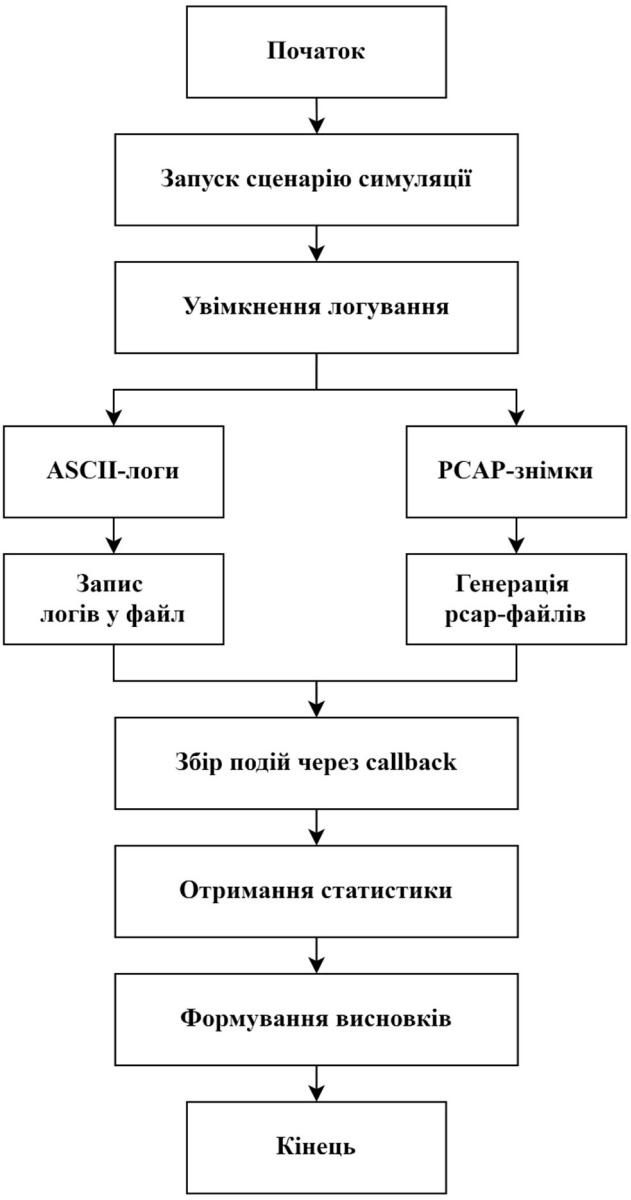


Рисунок 2.7 – Алгоритм обробки логів та результатів

Усі логи за замовчуванням зберігаються у директоріях проєкту, однак при розгортанні в контейнеризованому середовищі, наприклад у Docker, ці директорії монтовано на хост-машину через змінні типу volume, завдяки чому дані зберігаються навіть після знищення контейнера, гарантуючи сталість результатів та можливість повторного аналізу.

Візуалізація результатів за допомогою графіків або трас-файлів. Для досягнення більш глибокого розуміння процесів, які відбуваються під час симуляції, критичною є можливість візуалізації, оскільки вивід результатів у терміналі або у вигляді числових логів не завжди дозволяє швидко інтерпретувати суть подій у мережі. Незважаючи на відсутність повноцінного графічного інтерфейсу в NS-3, існує низка засобів візуального представлення результатів, що дозволяють суттєво підвищити зручність аналізу. Одним із основних інструментів є NetAnim, який реалізує перегляд топології, маршрутизації, руху вузлів та подій рівня MAC або PHY. Для того щоб скористатися цією можливістю, необхідно експортувати результати симуляції у форматі XML через параметр AnimationInterface anim("adhoc.xml"), після чого файл відкривається у NetAnim, де графічно показуються переміщення, передача пакетів та інші мережеві події.

Крім цього, іншим способом побудови візуалізацій є генерація графіків за допомогою інструментів на основі gnuplot, коли NS-3 створює вихідні файли у форматі .plt або .dat, які можуть бути автоматично збережені у форматах зображень, зокрема PNG, через такі виклики, як Gnuplot gnuplot("throughput.png"); Gnuplot2dDataset dataset, що дозволяє зручно виводити графіки пропускної здатності, затримок або втрат пакетів у статичному вигляді, що є ефективним при підготовці звітів чи презентацій.

Також доступна можливість глибокого аналізу на рівні окремих кадрів за допомогою PCAP-файлів, які можуть бути відкриті в зовнішніх програмах типу Wireshark. Після створення файлу через phy.EnablePcapAll("wifi-adhoc") його можна дослідити командою wireshark wifi-adhoc-0-0.pcap, що дозволяє візуалізувати структуру кадрів, заголовки, поля протоколів, рівень сигналу та інші

параметри. Такий спосіб особливо корисний для протоколів Wi-Fi або LoRaWAN з глибокою ієрархією протоколів передачі.

Ще одним підходом до побудови графіків є використання сторонніх інструментів, зокрема бібліотек мови Python, таких як matplotlib, які надають ще більшу гнучкість в обробці та представленні результатів. Якщо результати логуються у форматах CSV або збережені у вигляді структурованих логів, їх можна імпортувати та обробити командами типу import matplotlib.pyplot as plt; plt.plot(time, throughput), що дозволяє створювати графіки різних параметрів симуляції. У разі використання контейнерного середовища можна паралельно запускати кілька симуляцій з різними параметрами, зберігати їхні результати у відповідних директоріях, а потім агрегувати ці дані для побудови порівняльних графіків або проведення статистичного аналізу, наприклад розрахунку середніх значень, медіани чи дисперсії, що дозволяє отримати систематизовані узагальнення (див. рис. 2.8).

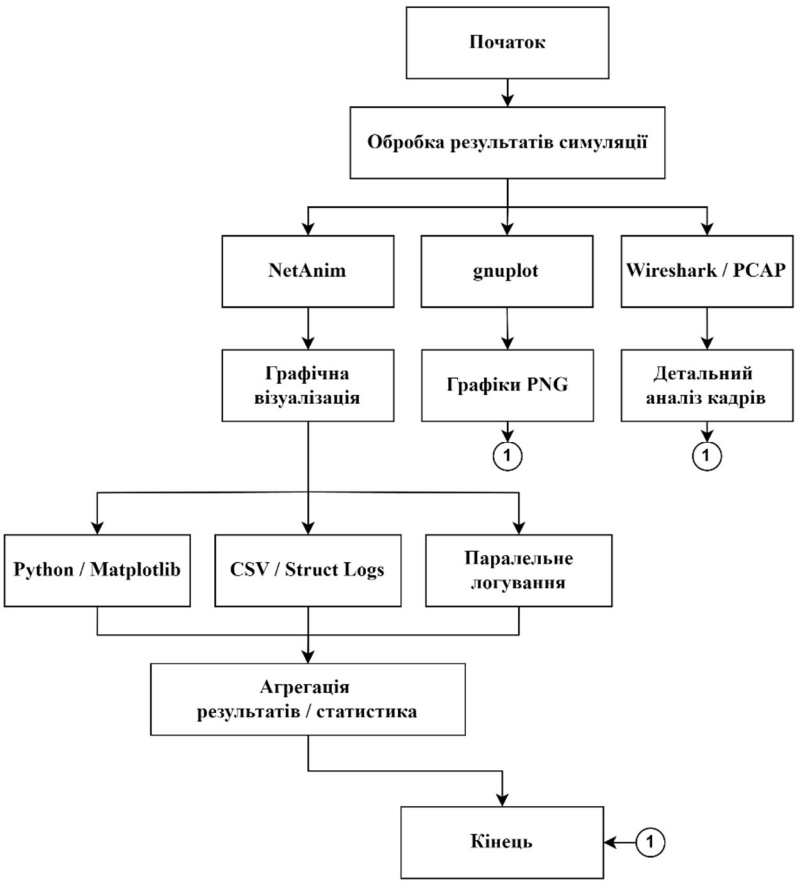


Рисунок 2.8 – Алгоритм побудови графіків або трас-файлів для візуалізації

Таким чином, при моделюванні мережевих сценаріїв з використанням NS-3 візуалізація, побудова графіків та трасування значно полегшують аналіз результатів. Інтеграція з такими інструментами, як NetAnim, Gnuplot, Wireshark або matplotlib, дозволяє реалізувати повноцінне відображення параметрів симульованих мереж і взаємодій між їхніми елементами. У поєднанні з збереженням результатів у зручних форматах та можливістю відтворення симуляцій у контейнерному середовищі створюються умови для достовірного та гнучкого аналізу великої кількості мережевих сценаріїв, які можуть бути адаптовані під специфіку різних протоколів бездротового зв'язку або типів мобільного середовища.

2.4 Реалізація дослідного прикладу та демонстрація

Реалізація дослідного прикладу починається з підготовки середовища всередині вже зібраного контейнера, де розміщено вихідні коди NS-3 ; далі виконується перехід у кореневу директорію симулятора, наприклад cd /workspace/ns-3-dev, після чого запускається процес збирання за допомогою ./waf build, що компілює всі необхідні модулі й приклади.

Запуск базової симуляції Wi-Fi-сценарію в NS-3. Коли збірка завершується, настає черга безпосереднього запуску базового Wi-Fi-сценарію — файла scratch/wifi-simple-adhoc.cc, який змодельовано як мінімальний adhoc-зв'язок між двома вузлами; команда ./waf --run "scratch/wifi-simple-adhoc" ініціює симуляцію, завдяки чому створюється in-memory топологія з двох нод, конфігурується PHY-рівень через YansWifiPhyHelper, MAC-рівень у режимі Ad-hoc через WifiMacHelper, а далі IP-адреси видаються об'єктом Ipv4AddressHelper. Протягом десяти секунд симульованого часу клієнт надсилає UDP-пакети на сервер, реєструються часові мітки передачі й приймання, ведеться підрахунок доставлених та втрачених кадрів, а також логуються значення RSSI, швидкість бітового потоку

і затримка, що генеруються в процесі роботи канальної та фізичної моделей (див. рис. 2.9).

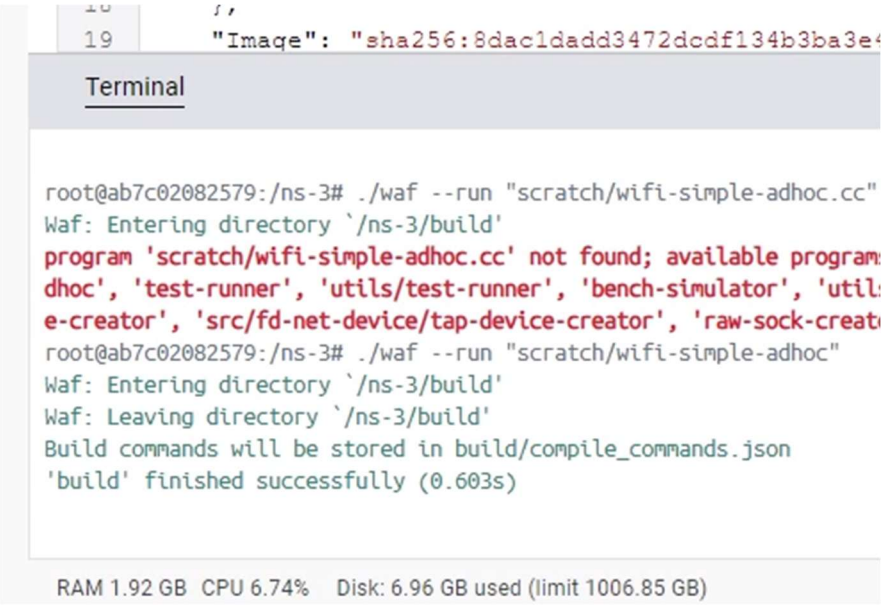


Рисунок 2.9 – Запуск базового Wi-Fi-сценарію

Після завершення виконання сценарію у змонтовану на хості директорію results потрапляють два основні набори вихідних даних — ASCII-трасування з розширенням .tr, яке містить покадровий опис усіх подій, і PCAP-файл, сформований функцією phy.EnablePcapAll("wifi-adhoc"); обидва типи файлів у подальшому аналізуються окремо. ASCII-логи відкриваються у текстовому редакторі або парсяться скриптами Python, що робить доступною швидку агрегацію статистики, обчислення середнього RTT, джитеру чи пропускної здатності, тоді як PCAP-трасу зручно завантажити у Wireshark, де відображаються заголовки 802.11, поля MAC-кадрів і час між окремими фреймами. Розраховані метрики зберігаються у CSV-таблицях, а візуальне представлення формується через matplotlib — після імпорту даних виконується побудова лінійних графіків пропускної здатності та затримки у часі, що дає змогу одним поглядом оцінити вплив конфігурації на якість каналу.

Для демонстрації гнучкості архітектури топологію зазвичай змінюють: кількість вузлів у NodeContainer збільшується до 10, відстань між ними

регулюється блоком MobilityHelper шляхом уставлення ConstantPositionMobilityModel із кроком 50 м, а час симуляції продовжується до 30 секунд, що здійснюється рядком Simulator::Stop(Seconds(30.0));. Такий варіант повторно компілюється за долі секунди й запускається тим самим механізмом waf; підсумкові файли автоматично з’являються у вже знайомому томі results, де нові назви формуються за схемою, що додає до базового префікса номер симуляції або хеш конфігурації. Наступний крок полягає у порівнянні двох серій експериментів — початкової і розширеної — шляхом накладання графіків у спільній системі координат, що одразу виявляє зміну затримки з ростом відстані, варіацію рівня сигналу та збільшення джитеру через більшу кількість одночасних передач.

Демонстрація логів та обробки результатів. Для демонстрації результатів роботи дослідного прикладу моделювання Wi-Fi-сценарію в середовищі NS-3 після запуску базової симуляції необхідно виконати низку дій, які дозволяють зібрати, проаналізувати та інтерпретувати отримані логи і мережеві дані. У процесі виконання симуляції генеруються кілька типів логів та трасувальних файлів, що фіксують події в мережі, зокрема, одним із базових форматів є ASCII-трасування, яке активується через виклик AsciiTraceHelper ascii; phy.EnableAsciiAll(ascii.CreateFileStream("adhoc.tr")); результатом чого стає збереження текстового лог-файлу з подіями фізичного рівня у файлі "adhoc.tr", де фіксуються зміни стану передавача, приймача, затримки, час надсилання кадрів тощо. Іншим способом фіксації є генерація pcap-файлів, які зберігають повний трафік на рівні кадрів IEEE 802.11 і можуть бути відкриті у WireShark для глибокого аналізу, активуються за допомогою phy.EnablePcapAll("adhoc"); що генерує файл типу "adhoc-0-0.pcap", який дозволяє відстежити, наприклад, MAC-адреси, типи кадрів, дублікати, втрати, час надсилання та отримання, ідентифікатори фреймів тощо (див. рис. 2.10).

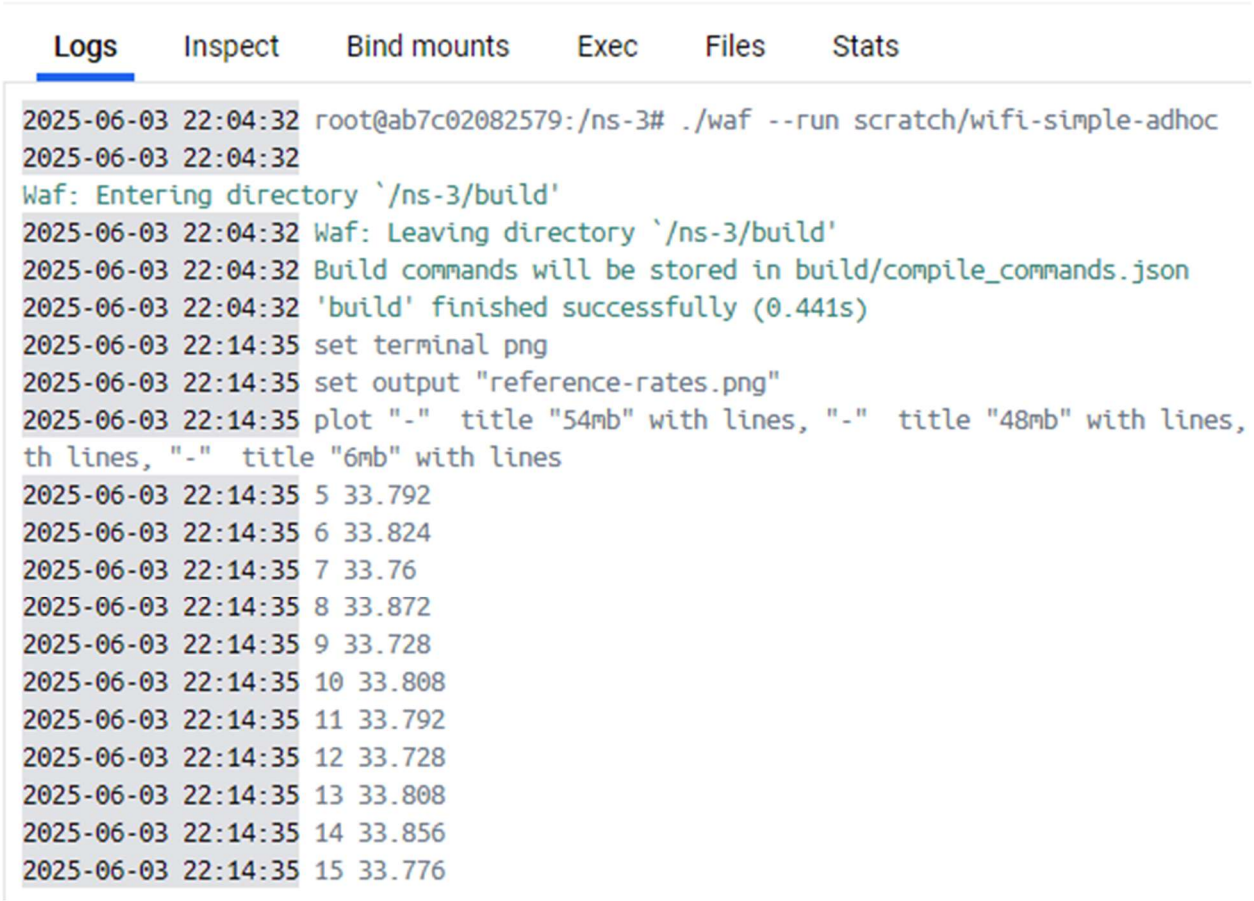


Рисунок 2.10 – Приклад логу результатів

Аналіз цих файлів дозволяє обчислити різні метрики ефективності симульованої мережі, включаючи кількість переданих пакетів, середній час доставки, кількість повторних передач або втрачених кадрів, що особливо корисно при порівнянні змінених топологій або умов передачі. Наприклад, затримка між двома вузлами може бути розрахована через обробку часу надсилання та прийому пакетів, який доступний у логах, або ж через додавання власної функції обробки подій за допомогою виклику типу Config::Connect. Далі, використовуючи отримані логи, можливо згенерувати графіки візуалізації продуктивності за допомогою Python, зокрема з бібліотекою matplotlib, імпортуючи її через import matplotlib.pyplot as plt і формуючи графік передачі як plt.plot(time, throughput), що дає змогу візуально оцінити продуктивність проведеної симуляції залежно від часу, топології або змінених параметрів.

Модифікація сценарію (варіанти топології, кількість вузлів). Щоб продемонструвати масштабованість побудованої архітектури, базовий сценарій із

двома вузлами було модифіковано до складнішої конфігурації з шістьма вузлами, розміщеними у формі прямокутної сітки, яка дозволяє створити реалістичніші умови багатовузлової передачі даних. Для цього було змінено кількість вузлів у сценарії, після чого визначено їх розміщення та встановлено топологічну структуру, у якій вузли 1, 2 та 3 виконують роль клієнтів, кожен із яких запускає UDP Echo Client для передачі даних на вузол 0, що виконує роль сервера. Така зміна дозволяє змоделювати односпрямований трафік від кількох джерел до одного приймача, що дозволяє спостерігати ефект накопичення трафіку та впливу конкуренції за доступ до середовища передавання.

У випадку багатовузлової топології виникає потреба або у ручній маршрутизації, або в активації протоколу маршрутизації типу OLSR, що дозволяє автоматично формувати маршрути між віддаленими вузлами у динамічному середовищі бездротової сітки. Після запуску такого сценарію та збору логів з усіх вузлів можна отримати статистику передачі, включаючи кількість отриманих пакетів кожним вузлом, середню затримку доставки, втрати, а також порівняти їх між собою. У підсумку формується повна картина функціонування моделі Wi-Fi-сценарію в умовах навантаженої сітчастої топології, що дозволяє в подальшому не лише виявити вузькі місця чи слабкі ділянки, а й підготувати умови для подальшої оптимізації параметрів або переходу до інших протоколів, якщо існує така потреба.

Знімки екранів/графіки результатів проведеної емуляції. У результаті виконання емуляції з використанням мережевого симулятора NS-3 можна отримати досить вичерпну інформацію про поведінку мережі, однак для кращої наочності ці результати доцільно представити у вигляді графіків і знімків екрана, що візуалізують ключові параметри симуляції. На основі зібраних даних, наприклад, будується графік пропускної здатності між вузлами залежно від часу, де вісь X відповідає часу симуляції в секундах, а вісь Y — пропускній здатності в кілобайтах за секунду, що дозволяє простежити зміну навантаження впродовж усього періоду тестування. Крім графіка throughput, можливе створення графіка затримки доставки пакетів delay для кожного вузла, що бере участь у передачі, ідентифікованого за номером на осі X, з відображенням середнього часу затримки

на осі Y у мілісекундах, що дає змогу провести аналіз затримок у передачі даних між конкретними вузлами.

Під час емуляції також здійснювався аналіз логів мережевого трафіку через файли pcap, відкриті у програмному забезпеченні Wireshark, у якому візуалізовано фрейми стандарту Wi-Fi 802.11, включаючи типи кадрів, параметри затримки між передаваннями (delta time), кількість спроб повторної передачі та інші параметри нижнього рівня. Подібне представлення даних дозволяє глибше зрозуміти динаміку трафіку та виявити аномалії або неефективності в маршрутизації. Окрім аналізу через Wireshark, додатково здійснювалася обробка XML-файлу формату NetAnim, створеного для візуалізації топології мережі у вигляді анімації, що відображає взаємодію вузлів у реальному часі, при цьому кожен вузол зображено у формі круга, а передавання трафіку відображається анімованими лініями, що значно спрощує сприйняття та дозволяє відстежити активність усередині мережі. Тривалість такої симуляції склала близько 10 секунд, що було достатньо для демонстрації основних характеристик мережевого трафіку в межах тестової топології (див. рис. 2.11).

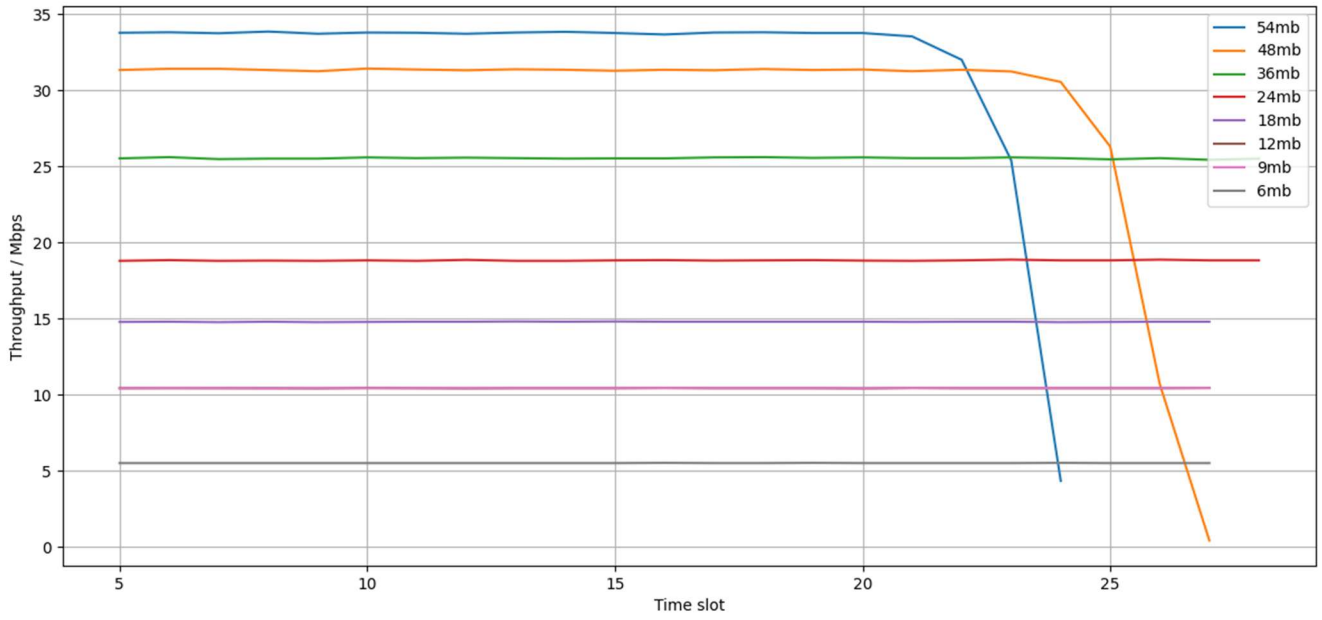


Рисунок 2.11 – Візуалізація результатів емуляції базового Wi-Fi-сценарію (вивід у gnuplot)

Загалом безперервне виконання емуляцій у середовищі Docker, інтегрованому з NS-3, дозволяє реалізовувати репліковані дослідження бездротових протоколів зв’язку, забезпечуючи стабільне, ізольоване та повністю контрольоване середовище для тестування. Базовий сценарій хоч і є доволі простим з архітектурної точки зору, однак його розгортання дозволяє з високою швидкістю отримати доступ до ключових параметрів поведінки мережі, а подальша модифікація кількості вузлів, типів додатків, топологій або протоколів маршрутизації дає змогу масштабувати дослідження до складніших варіантів, дозволяючи відстежити вплив змін на продуктивність. На основі логів, таких як ascii або pcap, можна побудувати графіки в середовищі Python з використанням бібліотеки matplotlib, зобразивши зміну пропускної здатності, затримки чи кількості доставлених пакетів, що перетворює симуляцію на потужний інструмент аналітики. Завдяки таким можливостям результат можна без труднощів відтворити на будь-якій платформі, гарантуючи ідентичні умови дослідження в межах контейнеризованого середовища, і, відповідно, забезпечити повторюваність результатів, які у свою чергу становлять важливу складову для оцінки ефективності обраного підходу до побудови мережевої системи в різних умовах використання.

РОЗДІЛ 3 АНАЛІЗ, ПОРІВНЯННЯ ТА ПЕРСПЕКТИВИ РОЗВИТКУ

3.1 Аналіз результатів та методи покращення моделі

Після здійснення успішної реалізації базових та розширених сценаріїв у рішенні, що базується на емуляторі NS-3 у середовищі Docker, виникає необхідність у глибшому аналізі ефективності отриманих результатів, а також виявленні обмежень, що впливають на точність, швидкодію та загальну відтворюваність моделей. Важливо зосередити увагу на основних недоліках поточного підходу, дослідити потенціал використання кластерних конфігурацій, можливості автоматизації серійних емуляцій та впровадження додаткових зовнішніх засобів аналізу, що дозволить сформувати більш цілісне уявлення про можливості вдосконалення моделі у найближчому майбутньому.

Слабкі місця емуляції. З погляду обмежень, що стосуються точності, швидкості та способів виводу результатів, актуальними залишаються три ключові слабкі місця (див. табл. 3.1).

Таблиця 3.1 — Типові проблеми при запуску емуляції в Docker та шляхи їх вирішення

Проблема	**Причина**	**Рішення**
Відсутність доступу до інтерфейсів	Docker за замовчуванням ізолює мережу	Використання режиму host або bridge
Неможливість виводу графіки	Відсутній доступ до X-сервера	Проксі-доступ через X11, налаштування xhost +
Витік ресурсів	Некоректне завершення контейнера	Обов’язкове використання --rm або моніторинг процесів
Проблеми з доступом до файлів	Відсутність монтованих томів	Використання параметра -v при запуску контейнера

Першим є обмеження точності, які проявляються навіть за високої деталізації подій, оскільки обробка відбувається не в реальному часі, а за допомогою механізму подієвої черги, що в умовах високого навантаження призводить до відставань між розрахунком події та фактичною обробкою відповідних даних. Окрім цього, модель середовища часто ідеалізується, зокрема в базових сценаріях Wi-Fi, де не враховуються або надмірно спрощуються ефекти багатошляхового розповсюдження, перешкод, відбиття сигналу, що знижує ступінь реалізму навіть при наявності точного сценарію. Така ситуація не унікальна для NS-3, однак все ще обмежує можливість повного перенесення результатів у фізичну площину.

Другою суттєвою проблемою є обмеження швидкодії, оскільки NS-3 за своєю архітектурою є однопотоковим симулятором, тобто обробка всієї моделі виконується в одному потоці, що унеможливлює ефективне масштабування при зростанні кількості вузлів або трафіку. Навіть при запуску в контейнеризованому середовищі, як-от Docker, доступ до апаратних ресурсів, особливо для прискорення або паралельної обробки, не є тривіальним і потребує додаткового налаштування, хоча технічно цю проблему можливо вирішити шляхом специфічної конфігурації середовища виконання або перенесення емуляції в розподілені системи (див. рис. 3.1).

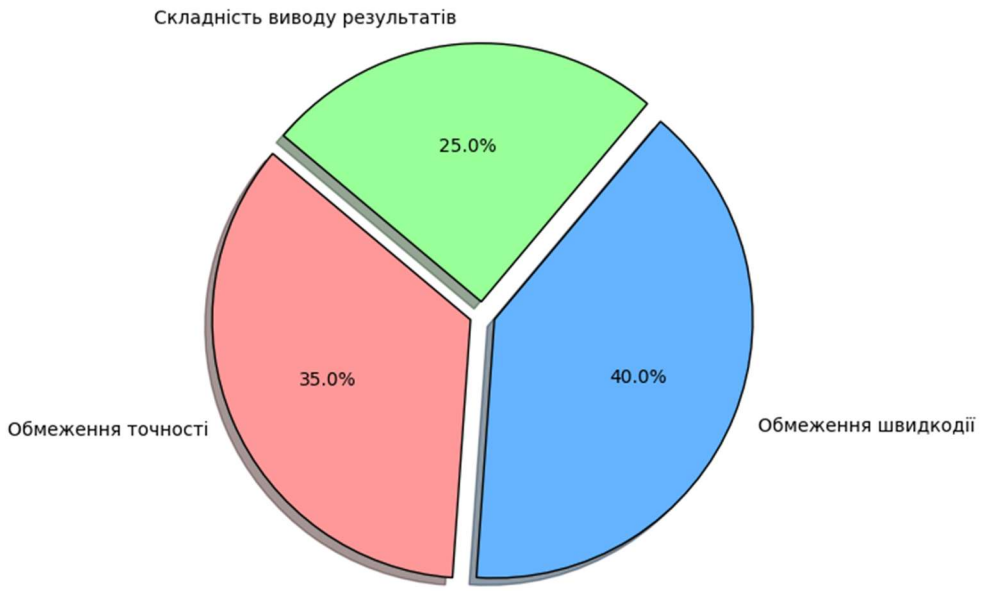


Рисунок 3.1 – Виявлені слабкі місця поточної моделі емуляції

Окремо варто звернути увагу на складнощі з виведенням результатів, оскільки генерація великих обсягів логів, зокрема у форматі .pcap, може значно уповільнювати сам процес емуляції, особливо коли трасування активне на десятках вузлів одночасно. При цьому власних засобів для зручного виводу статистичних показників або агрегованих даних у NS-3 фактично немає, що створює залежність від сторонніх інструментів аналізу, таких як Python з бібліотеками matplotlib або pandas, а також від власноруч розроблених скриптів для фільтрації, візуалізації та обробки інформації, що ускладнює початкову роботу з результатами та потребує додаткових знань у суміжних інструментах.

Можливість використання Docker Compose або кластеризації. Для забезпечення більш масштабованих експериментів та ефективної організації паралельних емуляцій доцільним є впровадження не лише базової контейнеризації, а й комплексних рішень, зокрема використання Docker Compose або побудови кластерної архітектури з допомогою таких інструментів як Kubernetes, GNU Parallel чи Docker Swarm. Впровадження Docker Compose дозволяє створювати сценарії з розподіленою логікою, де можна організувати кілька окремих контейнерів для різних ролей, наприклад, окремий контейнер для запуску емуляційного сценарію, інший для збору та обробки логів, а також окремий контейнер для візуалізації результатів у вигляді графіків. У такій структурі кожен тестовий запуск виконується ізольовано, що знижує навантаження на основну систему та дозволяє точніше контролювати обчислювальні ресурси і отримані метрики, що особливо актуально при великій кількості варіацій параметрів або сценаріїв (див. рис. 3.2).

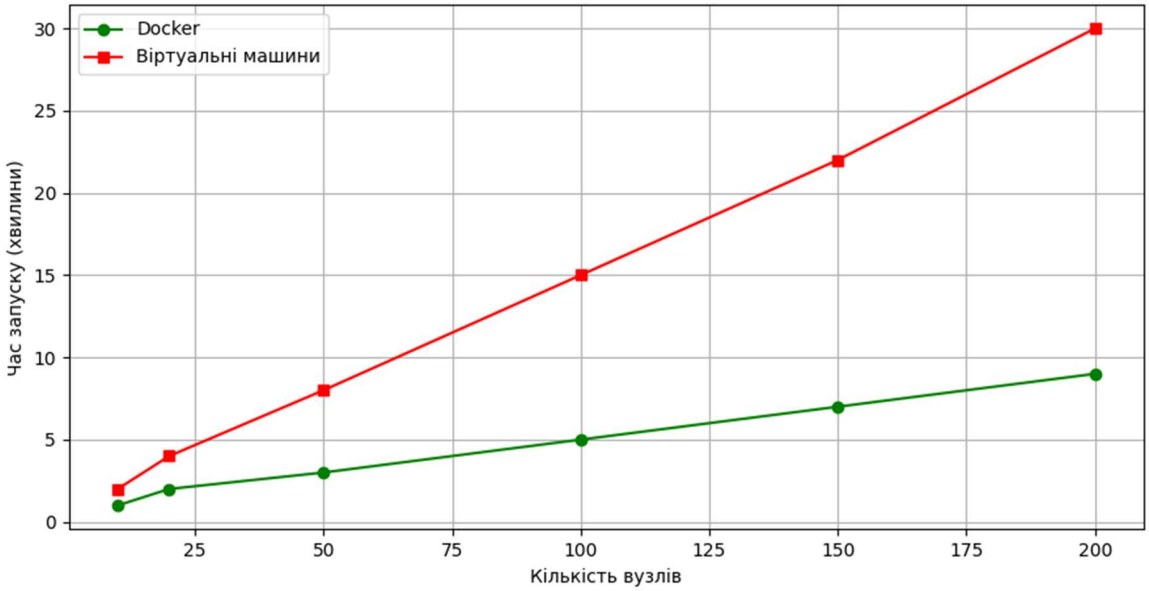


Рисунок 3.2 – Масштабування мережевого експерименту з Docker Compose

Водночас кластеризація може забезпечити ще вищий рівень ефективності, особливо при потребі запуску численних паралельних симуляцій з різними параметрами. Інструменти на кшталт GNU Parallel, Kubernetes або спеціалізованих систем на зразок Job Scheduler дають змогу розподіляти емуляційні навантаження між кількома вузлами, дозволяючи одночасне моделювання десятків варіантів конфігурацій із різними затримками, інтервалами, частотами або кількістю вузлів, що дозволяє значно знизити навантаження на окрему машину, підвищити загальну продуктивність тестування і досягти більшої точності в моделюванні реальних мережевих умов за рахунок кращого розподілу ресурсів і мінімізації локальних обмежень.

Автоматизація запусків сценаріїв для серійних тестів. Окрім структурної організації, важливо забезпечити автоматизацію запусків сценаріїв для серійних тестів, оскільки ручне виконання кожного експерименту уповільнює робочий процес і створює додаткові ризики людського фактора. Автоматизація таких запусків може бути реалізована через прості shell-цикли або більш гнучкі Python-скрипти, які дозволяють створювати великі серії тестів із варіативними параметрами, наприклад, відстанню між вузлами, потужністю передавача, частотою або розміром пакетів. При цьому результати кожного запуску можуть

зберігатися в окремі файли з відповідною назвою, що спрощує подальший аналіз та систематизацію отриманих даних. Наприклад, типовий shell-скрипт для серії запусків з різною відстанню може виглядати як цикл, у якому після кожного запуску результат переміщується до окремої директорії з відповідною міткою, що дозволяє не тільки автоматизувати процес, а й створити зручну структуру для подальшої обробки (див. рис. 3.3).

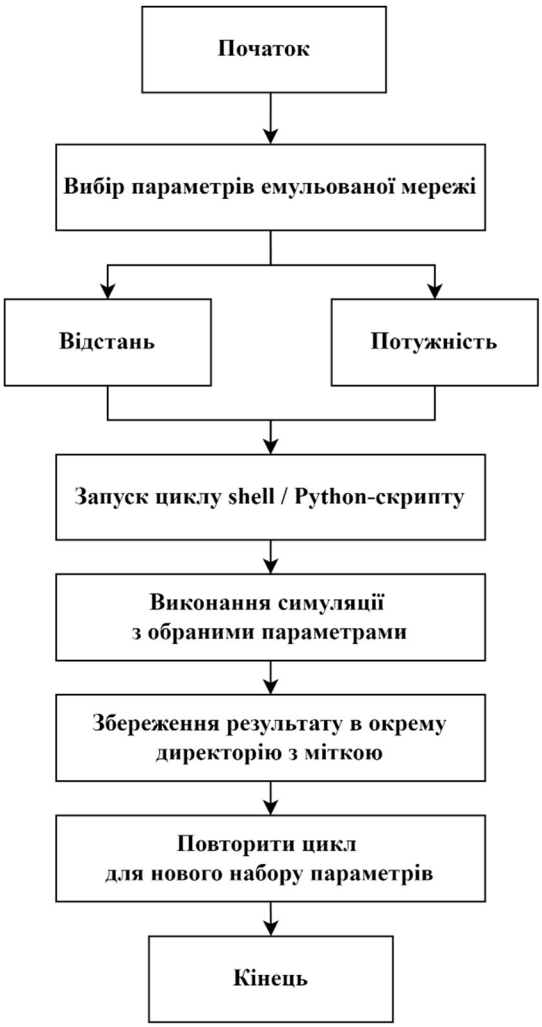


Рисунок 3.3 – Алгоритм автоматизованого запуску емуляції зі параметрами

Додатково можливе впровадження CI/CD-підходів, зокрема через інтеграцію з GitHub Actions або подібними сервісами, що дозволяє автоматично запускати тестові сценарії при кожній зміні конфігурації або коду. У такому випадку можна автоматично аналізувати результати, будувати графіки або генерувати звіти у

форматі PDF чи HTML без необхідності втручання з боку користувача. Такий підхід не тільки пришвидшує розробку, а й забезпечує стабільне збереження результатів у структурованій формі, наприклад, у вигляді CSV-файлів із ключовими метриками, як-от середня затримка, джиттер, втрати пакетів тощо. Застосування інструментів аналізу, таких як Pandas або Gnuplot, у поєднанні з такими CSV-логами дозволяє швидко агрегувати та порівнювати результати, роблячи можливим проведення глибокого аналізу безпосередньо після завершення серій симуляцій і забезпечуючи високу продуктивність і достовірність отриманих висновків.

Включення додаткових інструментів аналізу. Важливою складовою подальшого вдосконалення процесу дослідження симуляцій є включення додаткових інструментів аналізу, зокрема тих, що стосуються обробки PS App файлів та аналізу передачі кадрів на канальному та фізичному рівнях. При розгляданні такого інструменту, як Wireshark, стає можливим детальне відстеження таймінгів між кадрами, виявлення дублікатів, затримок і повторень, що особливо корисно при роботі з великим обсягом симуляцій. Завдяки підтримці цього інструменту в офіційній docker-версії NS-3, яка вже містить попередньо скомпільований образ з відповідними версіями утиліт, зокрема CLI-версією tshark, забезпечується зручність інтеграції у середовище автоматизованих запусків. Інструмент PacketViewer, вбудований у NS-3, також дозволяє візуалізувати топологію мережі, дії вузлів і обмін пакетами в реальному часі, що надає можливість не лише контролювати правильність виконання сценарію, а й швидко виявляти аномальні стани — відсутність маршрутів, неочікувані затримки чи збої в обміні даними (див. рис. 3.4).

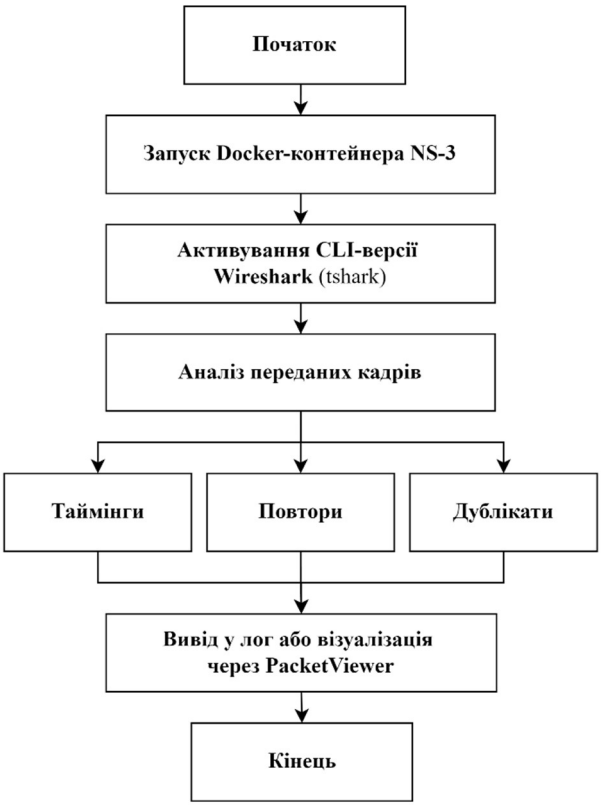


Рисунок 3.4 – Включення аналізаторів: Wireshark у Docker-контейнері

Поряд з цим корисним може бути розширення можливостей аналізу за рахунок зовнішніх фреймворків, таких як Prometheus і Grafana, з використанням експортерів метрик, що дозволяє перетворити отримані результати симуляції в числові показники з подальшою візуалізацією. У цьому випадку інтеграція з такими бібліотеками, як ns3gym або ns3-ai, забезпечує взаємодію з середовищами підкріпленого навчання (reinforcement learning), відкриваючи перспективу автоматичного налаштування параметрів моделі з урахуванням статистики минулих запусків. Такий підхід дозволяє не просто запускати сценарії, а здійснювати адаптивну оптимізацію мережевої поведінки відповідно до динамічно змінюваних умов.

Загальний підсумок усіх цих можливостей вказує на високу ефективність симулятора та емулятора NS-3 як потужного дослідницького інструмента у сфері вивчення протоколів бездротового зв’язку. Попри виявлені обмеження — відсутність підтримки паралельного виконання, спрощена модель фізичного середовища та обмежена гнучкість в автоматизації — запропоновані шляхи

вдосконалення, зокрема використання Docker Compose для створення сервісної архітектури, кластеризація за допомогою GNU Parallel чи Docker Swarm, CI/CD-методики для серійного запуску, глибока інтеграція з Wireshark і візуалізаційними фреймворками, дають змогу значно покращити масштабованість, точність і відтворюваність експериментів. У результаті створюється не лише інструмент для академічних досліджень, а повноцінна практична основа для проєктування, моделювання та верифікації бездротових мереж будь-якої складності та розміру, що дозволяє впроваджувати розроблені підходи у реальні задачі мережевої інженерії.

3.2 Порівняння з іншими інструментами дослідження

Окрім NS-3 як мережевого емулятора, у сфері моделювання та тестування комп'ютерних мереж існує велика кількість інших інструментів, кожен з яких має свою функціональну спрямованість, сильні сторони та недоліки. Серед них доцільно виділити ті, що найбільш близькі за призначенням і структурою до використаного під час дослідження рішення, зокрема такі платформи, як OMNeT++, Mininet-WiFi та GNS3, що є поширеними варіантами для моделювання, аналізу та перевірки протоколів, включно з бездротовими технологіями. Порівняння з цими інструментами доцільно проводити не лише на рівні архітектури чи реалізації, а й у контексті наявних функціональних можливостей, практичного досвіду використання, гнучкості сценаріїв, точності відтворення мережевої поведінки та можливості масштабування. Основу такого аналізу становлять базові характеристики кожної платформи, які дозволяють зрозуміти її доцільність у певних випадках і визначити переваги обраного інструмента у загальному контексті дослідження [34].

Загальна характеристика інструментів. Розглядаючи NS-3 як базове рішення, варто зазначити, що його архітектура побудована за принципами

дискретно-подієвого симулятора з акцентом на моделювання поведінки мережі на рівні фізичного і канального (MAC) рівнів. Його основна сила полягає у низькорівневій деталізації, яка дозволяє моделювати бездротові протоколи, включно з Wi-Fi, LTE, 5G, LoRaWAN, а також стек TCP/IP і механізми маршрутизації. Його застосування передбачає текстове описання сценаріїв та виконання на основі часової осі подій, що забезпечує високу точність і контроль над кожним процесом у симульованому середовищі.

Для порівняння, OMNeT++ є модульним симулятором, розробленим на C++, який має графічний інтерфейс та орієнтований на візуальне проектування мережевих структур. Його розширюваність реалізується через використання фреймворків, таких як INET, Veins та Castalia, що дозволяє адаптувати його під моделювання бездротових сенсорних мереж, VANET або IoT-систем. Хоча OMNeT++ є гнучким у створенні складних архітектур і надає зручний інтерфейс для візуалізації, йому бракує точності у фізичному шарі, а сценарії часто реалізуються у спрощеному вигляді без глибокого втручання в симульовані події. Водночас OMNeT++ зручний у навчальному процесі або для побудови демонстраційних моделей, де важливішим є інтерфейс взаємодії, ніж повна достовірність результатів.

Mininet-WiFi як інструмент більше тяжіє до емуляторів у реальному часі і побудований на базі віртуалізації мережевих просторів у середовищі Linux. Його особливість полягає в тому, що він дозволяє створювати топології з вузлів, які запускаються як окремі процеси із віртуальними інтерфейсами, що надає змогу емулювати SDN-мережі та досліджувати протоколи маршрутизації або взаємодію контролерів з реальним трафіком. Його обмеженням є прив’язка до часової лінії реального часу, що робить його менш придатним для високоточних симуляцій середовища або вивчення впливу затримок та інтерференції в бездротових мережах. У свою чергу він може ефективно використовуватися для тестування поведінки програмного забезпечення та контролерів SDN, коли важливо працювати з умовно реальними стеками та живими системами.

GNS3 пропонує візуальне моделювання мережевої інфраструктури, з можливістю запуску повноцінних операційних систем для маршрутизаторів, таких як Cisco IOS, Juniper або інші віртуальні мережеві ОС. Його основна перевага полягає в тому, що він здатен запускати реальні образи програмного забезпечення мережевих пристроїв, що робить його ефективним для створення лабораторій із конфігурування протоколів, тестування безпеки або навчання інженерів мережевих систем. Проте у контексті моделювання фізичного середовища або детального симулювання бездротової поведінки GNS3 не є доцільним вибором, адже зосереджений переважно на візуалізації взаємодії між пристроями та перевірці конфігурацій, а не на створенні математичних моделей поведінки середовища.

Підсумовуючи це порівняння, можна стверджувати, що кожен з інструментів має своє специфічне призначення, залежно від цілей дослідження. Якщо завдання полягає у глибокому аналізі протоколів нижнього рівня, точній емуляції бездротового середовища, контролі над параметрами фізичного шару та розробці нових архітектур передачі даних, NS-3 має найбільшу відповідність до таких вимог. У випадках, коли важливою є візуальна побудова топологій, простота розгортання або емуляція реальних мережевих пристроїв, ефективніше застосовувати OMNeT++, Mininet-WiFi чи GNS3. Проте саме NS-3 дозволяє створити повноцінне експериментальне середовище для дослідження протоколів будь-якої складності із максимальною точністю і контролем.

Порівняння за критеріями. Порівнюючи інструменти моделювання та емуляції мереж за критеріями, що стосуються підтримки протоколів, гнучкості, інтерфейсів та сценаріїв використання, можна відзначити, що NS-3 має вбудовану підтримку широкого спектра бездротових технологій, зокрема Wi-Fi за стандартами 802.11 a/b/g/n/ac, а також LoRaWAN, LTE, стек TCP/IP та моделі маршрутизації. Його структура орієнтована на деталізоване моделювання на рівні фізичного та канального шарів, що дозволяє отримувати високоточні результати в дослідницьких експериментах. У випадку OMNeT++, підтримка протоколів реалізується переважно через зовнішні фреймворки на кшталт INET, що включає TCP/IP, кастомізовані протоколи та можливості моделювання VSN-сценаріїв, при

цьому підтримка LoRaWAN є менш повною. MiniNet-WiFi базується на технології Linux namespaces, що дозволяє використовувати всі протоколи, реалізовані на рівні ядра, включно з Wi-Fi, але з деякими обмеженнями в точності симуляції фізичного середовища через реальний масштаб часу. У свою чергу, GNS3 забезпечує підтримку протоколів на основі прошивок самих пристроїв, таких як Cisco IOS, Mikrotik RouterOS та інших, що забезпечує точність конфігурацій у мережевих лабораторіях, однак не дозволяє гнучко моделювати фізичний шар [34].

Якщо оцінювати ці інструменти за способом запуску та управління, NS-3 потребує попередньої компіляції і не має графічного інтерфейсу, його керування відбувається через CLI або за допомогою Docker-контейнерів, що ускладнює вхід для початківців. OMNeT++ має повноцінне графічне середовище з редактором топологій, запуском через інтегроване середовище розробки та візуалізацією мережевих подій, що робить його зручнішим для користувача. MiniNet-WiFi не вимагає компіляції, запускається за допомогою простого CLI, має мінімальні вимоги до ресурсів і добре підходить для швидкого прототипування сценаріїв з SDN або Wi-Fi. GNS3 натомість використовує drag-and-drop середовище з можливістю запуску реальних образів мережевого обладнання, але потребує додаткового налаштування образів і ресурсів для емуляції, що ускладнює інтеграцію в дослідницькі середовища (див. рис. 3.5) [34].

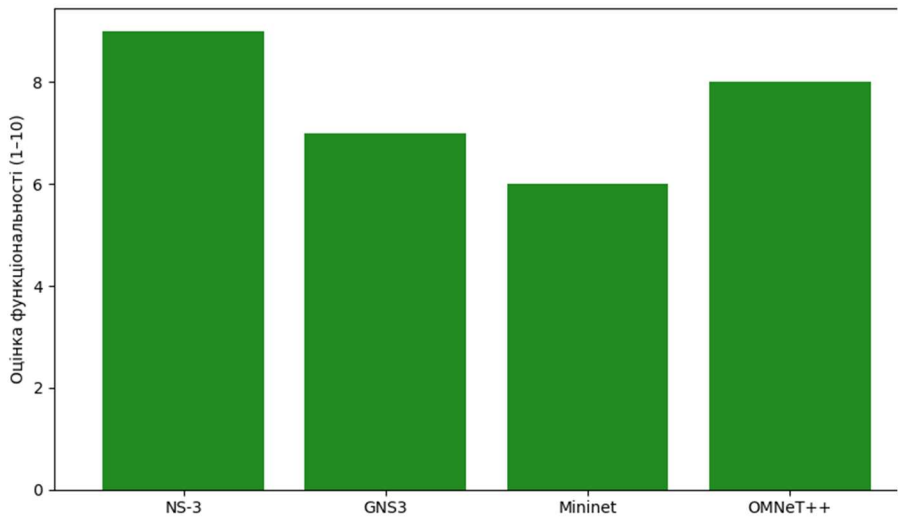


Рисунок 3.5 – Порівняння дослідницьких інструментів за функціональністю

З огляду на розширюваність, NS-3 є відкритим і розширюється за рахунок додавання нових модулів на мові C++ або сценаріїв на Python, але має високий поріг входу через складність архітектури та потребу у знаннях програмування. OMNeT++ забезпечує модульне розширення на основі фреймворків INET або Veins, з глибокою інтеграцією з графічними редакторами та сценаріями. MiniNet-WiFi легко конфігурується за допомогою Python, дає змогу створювати кастомні топології та використовує стандартні Linux-інструменти для емуляції. GNS3 дозволяє додавати нові пристрої через Docker, KVM або QEMU, однак не забезпечує точного моделювання фізичних властивостей середовища, орієнтуючись на відтворення поведінки реально існуючого мережевого обладнання [34].

Приклади сценаріїв для альтернативних інструментів. Щодо прикладів сценаріїв, OMNeT++ дозволяє реалізовувати моделі мобільних мереж з перешкодами, наприклад, симуляцію мережі з 20 вузлами, що переміщуються згідно заданих траєкторій. Завдяки графічному інтерфейсу користувач може зручно налаштовувати події, візуалізувати рух об’єктів та змінювати умови середовища. У випадку NS-3, який використовувався у впровадженому рішенні, доцільно реалізовувати сценарії дослідження нових бездротових протоколів або тестування поведінки мереж в умовах змінних навантажень, при цьому можливе інтегрування реального TCP/IP-стека операційної системи, що підходить для розробки прототипів. MiniNet-WiFi дозволяє створити сценарії з контролем SDN-мереж, зокрема за допомогою контролерів типу POX або Ryu, що забезпечує гнучкість у тестуванні маршрутизації та перемикання потоків. GNS3 найбільше підходить для сценаріїв із тестуванням традиційних протоколів маршрутизації, таких як OSPF або BGP, на прикладі реальних пристроїв типу Cisco чи MikroTik, що дозволяє відтворити лабораторії корпоративного класу з високим рівнем відповідності промисловим умовам (див. рис. 3.6).

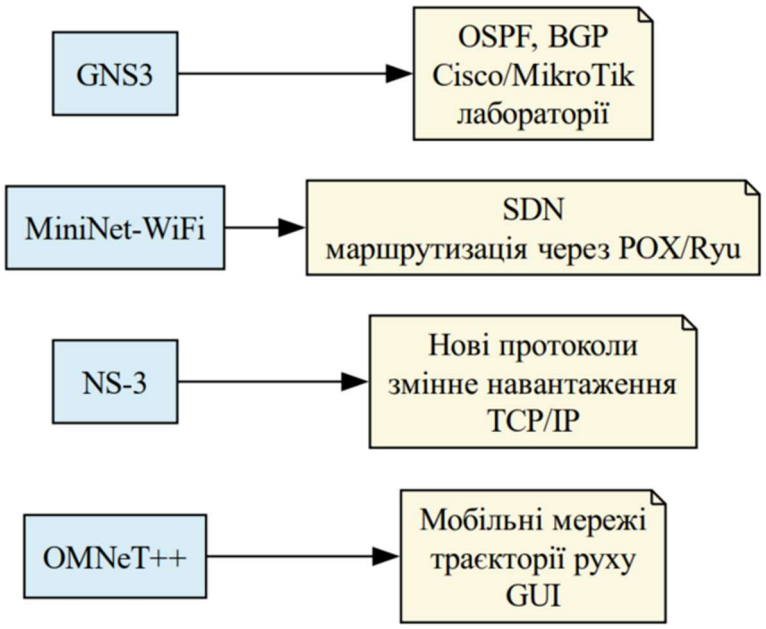


Рисунок 3.6 – Сценарії, що реалізуються в альтернативних симуляторах

З урахуванням проведеного порівняння (див. табл. 3.2) можна зробити висновок, що NS-3 є потужним інструментом для дослідницьких задач, пов'язаних з детальним симулюванням фізичного та канального рівнів мережі, а також розробкою та тестуванням нових мережевих протоколів у масштабованих умовах. OMNeT++, MiniNet-WiFi та GNS3 більше підходять для прикладних завдань, навчання, створення візуальних моделей або тестування конфігурацій обладнання.

Таблиця 3.2 — Порівняння NS-3 з альтернативними інструментами симуляції

Критерій / Інструмент	**NS-3**	**OMNeT++**	**GNS3**	**Mininet-WiFi**
Мова програмування	C++, Python	C++, NED	Python, інше	Python
Точність моделювання	Висока, подрібнена фізична модель	Висока, модульна модель	Середня, емуляція реальних образів	Низька–середня (імітація Wi-Fi)

Продовження таблиці 3.2 — Порівняння NS-3 з альтернативними інструментами симуляції

Критерій / Інструмент	**NS-3**	**OMNeT++**	**GNS3**	**Mininet-WiFi**
Підтримка Wi-Fi	Так (IEEE 802.11 a/b/g/n/ac/ax)	Так (через INET framework)	Обмежена (через VirtualBox/QEMU)	Так (імітація з wmediumd)
Підтримка LoRaWAN	Так (через зовнішній модуль)	Частково (моделі користувача)	Ні	Ні
Підтримка LTE / 5G	Так (lte, nr модулі)	Обмежена	Ні	Ні
Візуалізація	Так (NetAnim, PyViz)	Вбудована GUI	Так (реальна топологія)	Так (GUI + matplotlib)
Сценарії IoT/AdHoc	Так	Так	Ні	Так (імітація IoT)
Контейнеризація (Docker)	Так	Через налаштування	Так	Так
Інтеграція з Python	Повна (Python API)	Часткова (виклики з NED)	Повна	Повна
Масштабованість	Висока	Висока	Обмежена	Середня

Продовження таблиці 3.2 — Порівняння NS-3 з альтернативними інструментами симуляції

Критерій / Інструмент	**NS-3**	**OMNeT++**	**GNS3**	**Mininet-WiFi**
Реалістичність трафіку	Висока (можна задавати вручну)	Висока (через модулі)	Висока (емуляція реального ПЗ)	Низька–середня
Підтримка розподілених сценаріїв	Частково (з MPI)	Частково (з INET-MPI)	Ні	Ні
Активність спільноти	Висока (оновлення, патчі)	Висока (академічна спільнота)	Середня	Середня
Навчальна документація	Так (офіційна + приклади)	Так (дуже докладна)	Помірна	Обмежена
Тип моделювання	Симуляція (event-driven)	Симуляція (discrete event)	Емуляція (реальні пакети)	Імітація/емуляція

Вибір інструменту безпосередньо залежить від мети дослідження: для точного аналізу бездротових технологій на фізичному рівні безальтернативним є NS-3, у випадках, коли необхідно швидко реалізувати та перевірити роботу мережевих конфігурацій з візуалізацією або інтеграцією з реальним обладнанням, перевагу варто надати іншим платформам.

3.3 Можливі напрями розвитку та удосконалення

З урахуванням реалізованого та проаналізованого рішення на базі NS-3, яке демонструє гнучкість та можливість глибокого налаштування як мережевого емулятора для дослідження бездротових технологій, можна визначити кілька напрямів розвитку, що суттєво підвищують зручність, масштабованість і адаптивність даного інструмента під потреби ширшої аудиторії. Першочерговим залишається питання зниження технічного бар'єру для користувачів, які не мають досвіду роботи з інтерфейсами командного рядка чи середовищами Linux, тому впровадження більш інтуїтивних інструментів для запуску симуляцій та управління ними є природним вектором еволюції. У цьому контексті перспективним виглядає створення веб-інтерфейсу для керування симуляціями, що надасть можливість налаштовувати та запускати експерименти без необхідності прямого редагування вихідного коду чи ручної компіляції, що особливо актуально для навчальних цілей, інтерактивних демонстрацій та швидкого тестування прототипів [30-33].

В умовах, коли стандартний підхід до роботи з NS-3 полягає в ручному написанні конфігураційних скриптів мовою C++ або Python і запуску через CLI, такий процес обмежує доступність інструмента для студентів, викладачів або розробників, які не мають досвіду з подібними середовищами. Наявність зручного веб-інтерфейсу, реалізованого, наприклад, на основі Flask, Django чи Node.js, дозволить користувачу обрати попередньо визначений шаблон симуляції — типу Wi-Fi, LTE, MANET тощо — задати параметри експерименту (кількість вузлів, характер трафіку, відстані між вузлами, тип протоколів) у формі інтерактивних елементів, після чого ініціювати запуск симуляції у фоновому режимі з відображенням результатів у браузері (див. табл. 3.3) [35].

Таблиця 3.3 — Можливості розширення дослідницької системи

Напрям розвитку	**Опис можливості**	**Потенційна перевага**
Веб-інтерфейс	Розгортання UI для запуску симуляцій через браузер	Зручність для не технічних користувачів
Хмарна інфраструктура	Запуск контейнерів у середовищі AWS, GCP або Azure	Масштабованість та розподілене моделювання
CI/CD для тестів	Автоматичне тестування сценаріїв після змін у коді	Надійність і безперервна перевірка результатів
Шаблони сценаріїв	Готові конфігурації для різних топологій і випадків	Швидкий старт та багаторазове використання

Додавання веб-інтерфейсу для запуску симуляцій. Технічна реалізація подібного веб-інтерфейсу потребує створення двошарової архітектури: на рівні бекенду виконується зв’язок з симулятором через скрипти оболонки або прямий виклик python-класів NS-3, які відповідають за ініціалізацію та керування запуском моделі; на рівні фронтенду створюється інтерфейс користувача з використанням React або інших JavaScript-фреймворків для реалізації інтерактивних форм, таблиць з параметрами та блоків виводу результатів. Сам сервер може бути запущений у середовищі Docker з монтуванням локальних томів або вбудованими шаблонами, що дозволить уникнути прив’язки до певної операційної системи і забезпечить гнучке оновлення системи. Очікуваними перевагами впровадження подібної моделі стає перш за все доступність: для користувача достатньо лише браузера, і не потрібно налаштовувати середовище чи компілювати вихідний код вручну. Такий підхід особливо доцільний у контексті дистанційного навчання або під час проведення лабораторних робіт у межах віртуальних освітніх середовищ, таких як LMS Moodle, Google Classroom або інші хмарні платформи, де кожен студент може отримати індивідуальний доступ до симулятора.

Крім освітнього аспекту, веб-інтерфейс дозволить масштабувати NS-3 як сервіс для командної роботи, наприклад у проєктах з дослідження нових протоколів

або побудови цифрових двійників мережевої інфраструктури, де кілька учасників мають паралельний доступ до середовища. Підключення до CI/CD-процесів та автоматичний запуск тестів для нових моделей протоколів дозволить зробити NS-3 частиною сучасного конвеєра розробки, де моделювання є одним з етапів перевірки функціональності. У цьому контексті потенційним напрямом удосконалення є також створення API для віддаленого запуску сценаріїв через REST-запити або WebSocket-зв'язок, що відкриває можливість інтеграції з іншими системами [33].

Використання хмарних платформ для масштабного моделювання. У контексті розглянутого рішення, що базується на використанні емулятора NS-3 як універсального інструменту для моделювання бездротових мереж, особливу увагу варто приділити можливим напрямам його вдосконалення та розвитку, зважаючи на гнучкість архітектури, відкритий код та активну спільноту розробників, що дозволяє адаптувати даний інструмент під потреби конкретних проєктів. Серед найперспективніших напрямів варто виділити реалізацію веб-інтерфейсу, інтеграцію з хмарними платформами, створення шаблонів для типових досліджень, а також удосконалення механізмів автоматизації з можливістю гнучкого конфігурування параметрів та аналізу результатів у зручному форматі. Такий підхід дозволяє суттєво знизити бар'єр входу для новачків, оптимізувати обчислювальні ресурси при моделюванні складних топологій та пришвидшити підготовку повторюваних досліджень [32].

З огляду на те, що на поточному етапі взаємодія з NS-3 переважно відбувається через термінал або безпосереднє редагування вихідного коду, виникає потреба у створенні веб-інтерфейсу, що здатен абстрагувати низькорівневу складність і зробити роботу з симулятором інтуїтивно зрозумілою навіть для тих, хто не має досвіду роботи з Linux-середовищем або командною оболонкою. Такий інтерфейс може бути реалізований з використанням Flask, Django або Node.js у зв'язці з фронтендом на базі HTML5, React або Vue, що забезпечить інтерактивне налаштування параметрів, вибір шаблонів (наприклад, Wi-Fi, LTE, MANET), зазначення кількості вузлів, характеристик трафіку, топологій або відстаней між

елементами. Запуск таких сценаріїв може бути реалізований у фоновому режимі з подальшим виведенням результатів у браузер, включаючи графіки, журнали та log-файли. Технічна реалізація передбачає поділ на декілька логічних шарів: бекенд-обгортка для взаємодії з NS-3 через Python або shell-скрипти, графічний інтерфейс для користувача та середовище розгортання, яке може базуватись на Docker-контейнерах із використанням BIND mount. Завдяки такій архітектурі стає можливим розгортання сервісу в локальній мережі або як частини LMS-платформи, що значно розширює можливості для навчального або демонстраційного використання без необхідності встановлення та налаштування NS-3 на стороні користувача [32].

Коли йдеться про масштабні симуляції, які включають тисячі вузлів, складні топології, моделі мобільності, перешкод або фізичних ефектів, обчислювальні потреби зростають до такого рівня, що локальні машини не здатні забезпечити ані швидкодію, ані стабільність при обробці таких сценаріїв, що особливо актуально у випадках, коли необхідно виконати серії варіативних тестів, батч-моделювання або аналіз на кшталт Монте-Карло. У такому разі доцільним рішенням є перенесення обчислень у хмарне середовище, що відкриває можливість масштабованого запуску симуляцій паралельно. Наприклад, за допомогою Google Cloud Platform, Amazon Web Services або Azure можна реалізувати короткотривалі сценарії на базі спот-інстанцій, які знижують витрати, а при необхідності масового оброблення — розгортати NS-3 в контейнеризованому середовищі з використанням Kubernetes. Також можлива інтеграція NS-3 з JupyterHub, Google Colab або іншими web-ноутбуками, що дозволяє дослідникам запускати симуляції безпосередньо в браузері з можливістю доступу до GPU або розподілених ресурсів (див. рис. 3.7).



Рисунок 3.7 – Модель використання хмарного середовища
для розподілених симуляцій

Такий підхід дозволяє реалізовувати варіативні сценарії, змінювати параметри трафіку, інтервали передачі або протоколи на окремих контейнерах, що особливо корисно для проведення автоматизованого тестування. Додатково, результатами таких досліджень можна одразу інтегруватися з системами аналітики на кшталт BigQuery, Grafana або Prometheus для побудови графіків, heatmap або логічного аналізу симуляцій у контексті реальних кейсів [32].

Створення шаблонів симуляцій для наукових досліджень. Ще одним шляхом оптимізації є створення готових шаблонів (template) для симуляцій, які особливо корисні у науковому або освітньому середовищі, де запуск кожного нового експерименту потребує попереднього налаштування мережі, топології, параметрів трафіку та фільтрації результатів. Використання шаблонів дозволяє пришвидшити старт дослідження та забезпечити уніфікацію підходів. Такі шаблони можуть включати коментовані приклади коду для найтиповіших сценаріїв — наприклад, просту Wi-Fi симуляцію з двома точками доступу та варіацією клієнтів, експерименти з налаштуванням затримок, втрат або пропускної здатності для аналізу Quality of Service, або ж порівняння TCP-алгоритмів, таких як CUBIC, Reno та BBR під навантаженням. Для зручності шаблони можуть базуватись на конфігураційних файлах JSON або YAML, які задають параметри без зміни коду, а покрокові інструкції у вигляді ноутбуків з поясненням функцій кожного блоку

полегшують розуміння, адаптацію та перевикористання сценаріїв. Також важливим аспектом є стандартизація вихідних форматів — CSV, PCAP, графіки, що дозволяє використовувати інструменти статистичного аналізу, зводити дані до таблиць, будувати звіти або експортувати їх у зовнішні аналітичні системи для подальшої обробки.

Інтеграція з CI/CD для тестування змін у симуляціях. У разі розробки нових модулів, протоколів або оптимізації в рамках мережевого емулятора NS-3 виникає потреба у забезпеченні стабільності коду та оперативній перевірці внесених змін, що в умовах ручного тестування є надзвичайно складним та ненадійним процесом. Інтеграція із системами безперервної інтеграції та розгортання, зокрема CI/CD-пайплайнами на базі GitHub Actions, GitLab CI чи інших інструментів, дозволяє автоматизовано перевіряти працездатність емуляційних сценаріїв при кожному оновленні репозиторію, тим самим знижуючи ризик появи помилок або регресій. Наприклад, за умови модифікації конкретного модуля, як-от реалізації нового алгоритму в стеку Wi-Fi, автоматизована система збирає середовище, компілює NS-3, запускає наперед задані сценарії з еталонними параметрами, аналізує отримані результати, включно з логами, затримками, пропускною здатністю та втратами пакетів, та порівнює їх із очікуваними значеннями. У разі відхилення зафіксовані зміни одразу маркуються як потенційно нестабільні, що дає змогу вчасно внести корективи (див. рис. 3.8) [30-31].



Рисунок 3.8 – Інтеграція NS-3 у CI/CD-процеси

Крім автоматизованої перевірки на стабільність та продуктивність, такий підхід передбачає також генерацію звітів у зручних форматах, таких як Markdown або HTML, що значно полегшує аналіз результатів і дозволяє відслідковувати історію змін із належним рівнем документації. Створення шаблонів тестування для різних протоколів і сценаріїв у межах CI/CD забезпечує не тільки повторюваність досліджень, а й спрощує колективну роботу над кодом, оскільки вся команда має однаковий доступ до результатів кожної ітерації змін без потреби у ручному втручанні чи ручному запуску сценаріїв. Повністю автоматизоване середовище перевірки коду дозволяє сфокусуватися на логіці моделювання замість витрачання часу на рутинні технічні операції, що особливо актуально у великих дослідницьких проєктах або при розробці універсальних бібліотек розширень до симулятора [30-31].

Підкреслюючи сукупність зазначених переваг, подальший розвиток таких рішень, як NS-3, можливо розглядати не лише з позиції технічного оновлення, а і в контексті розширення можливостей для інтеграції з сучасними практиками DevOps, адаптації до хмарних середовищ і формування дружніх інтерфейсів взаємодії, які відкривають шлях до реалізації масштабованих і відтворюваних досліджень. Таке поєднання дає змогу не лише задовольняти потреби академічної спільноти, а й адаптувати інструмент для застосування в інженерних процесах, що потребують високої надійності та гнучкості, а заплановані напрями розвитку при цьому залишаються не просто досяжними, а логічними в умовах сучасних вимог до комп’ютерної інженерії.

ВИСНОВКИ

У процесі реалізації дослідного прикладу було успішно налаштоване середовище мережевого симулятора NS-3 у контейнері Docker, що забезпечило ізольованість, стабільність і простоту розгортання. Проведено запуск базового сценарію Wi-Fi-зв’язку з використанням стандартних модулів NS-3, що дозволило візуалізувати базову топологію, отримати лог-файли з параметрами передачі даних, затримками та втратою пакетів. Виведення результатів у текстовому форматі дало змогу виконати базовий аналіз продуктивності зв’язку між вузлами.

На наступному етапі були змінені параметри сценарію, зокрема кількість вузлів, типи моделей руху та конфігурації трафіку, що дало змогу проаналізувати вплив змін у топології на поведінку мережі. Також була продемонстрована гнучкість сценаріїв, які можна адаптувати під різні потреби досліджень. Для кожної модифікації отримано графіки з результатами передачі даних та динамікою затримок, що дозволило візуально оцінити ефективність конфігурацій.

Значну увагу було приділено обробці результатів симуляції, а саме збереженню логів у структурованому форматі, експорту та подальшому аналізу за допомогою сторонніх інструментів, що надало змогу проводити серійні тестування зі збереженням однакових умов запуску. Візуалізація даних, зокрема графіки зміни швидкості передавання, дала змогу краще зрозуміти вплив параметрів сценарію на загальну продуктивність моделі.

На завершальному етапі були проаналізовані способи оптимізації та подальшого розвитку моделі. Зокрема, розглянуто автоматизацію запуску симуляцій, інтеграцію з системами CI/CD та можливості хмарного масштабування для великомасштабних експериментів. Запропоновано напрям розробки веб-інтерфейсу для спрощення роботи з NS-3, що може бути корисним як у навчальному процесі, так і для дослідників, які працюють із протоколами бездротового зв’язку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wireless Protocols Explained. Network World. URL: https://www.networkworld.com/article/3602272/wireless-networking-protocols-explained.html (дата звернення: 27.02.2025)
2. LoRa vs Zigbee vs Wi-Fi: Differences Explained. IoT Design Pro. URL: https://iotdesignpro.com/articles/lora-vs-zigbee-vs-wifi (дата звернення: 02.03.2025)
3. Wireless Data Transmission Techniques. Electronics Hub. URL: https://www.electronicshub.org/data-transmission-wireless/ (дата звернення: 06.03.2025)
4. OSI Model and Protocol Layers. Cloudflare. URL: https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/ (дата звернення: 10.03.2025)
5. Wireless Network Testing Challenges. Spirent. URL: https://www.spirent.com/blogs/testing-wireless-networks-challenges (дата звернення: 14.03.2025)
6. Simulation vs Emulation: Key Differences. TechTarget. URL: https://www.techtarget.com/searchnetworking/definition/simulation-and-emulation (дата звернення: 18.03.2025)
7. Network Simulators in Education. ResearchGate. URL: https://www.researchgate.net/publication/332017236_Use_of_Network_Simulators_in_Education (дата звернення: 22.03.2025)
8. Pros and Cons of Software Emulation. Toptal. URL: https://www.toptal.com/software/software-emulation-vs-hardware (дата звернення: 26.03.2025)
9. Top Tools for Network Simulation and Emulation. GeeksforGeeks. URL: https://www.geeksforgeeks.org/network-simulation-tools/ (дата звернення: 30.03.2025)
10. What is Container Networking? Docker Docs. URL: https://docs.docker.com/network/ (дата звернення: 03.04.2025)
11. Benefits of Docker in Research Environments. IBM Developer. URL: https://developer.ibm.com/articles/docker-benefits/ (дата звернення: 07.04.2025)
12. Network Simulation Tools in Docker. Medium. URL: https://medium.com/@netresearch/docker-based-network-simulation-tools (дата звернення: 11.04.2025)
13. Scalable Network Topologies with Docker. Dev.to. URL: https://dev.to/simonmar/docker-network-topology-experiments-2ce0 (дата звернення: 15.04.2025)
14. NS-3 Documentation Overview. NS-3 Project. URL: https://www.nsnam.org/documentation/ (дата звернення: 19.04.2025)
15. Wireless Modules in NS-3. GitHub Wiki. URL: https://github.com/nsnam/ns-3-dev/wiki/Wireless-Modules (дата звернення: 23.04.2025)

16. NS-3 Simulation Execution Modes. Tutorialspoint. URL: https://www.tutorialspoint.com/ns3/ns3_simulation_modes.htm (дата звернення: 27.04.2025)
17. NS-3 vs Other Simulators. Towards Data Science. URL: https://towardsdatascience.com/ns3-vs-other-network-simulators-834fd147c65e (дата звернення: 01.05.2025)
18. Example Scenarios in NS-3. NS-3 Docs. URL: https://www.nsnam.org/docs/models/html/examples.html (дата звернення: 05.05.2025)
19. NS-3 Simulation Parameters. ResearchGate. URL: https://www.researchgate.net/publication/330189500_Parameter_Tuning_in_NS3 (дата звернення: 25.02.2025)
20. Data Logging in NS-3. GitHub Gist. URL: https://gist.github.com/ns3-user/logging-examples (дата звернення: 01.03.2025)
21. Custom Topologies in NS-3. Stack Overflow. URL: https://stackoverflow.com/questions/50422185/custom-topology-in-ns3 (дата звернення: 05.03.2025)
22. NS-3 Docker Image. Docker Hub. URL: https://hub.docker.com/r/nsnam/ns-3-dev (дата звернення: 09.03.2025)
23. Containerized Simulation Design. Medium. URL: https://medium.com/@devnetwork/containerized-ns3-architecture (дата звернення: 13.03.2025)
24. Communication Between Docker & NS3. NS-3 Wiki. URL: https://www.nsnam.org/wiki/Docker_NS3_Integration (дата звернення: 17.03.2025)
25. Portability of Network Research Systems. Arxiv.org. URL: https://arxiv.org/abs/1904.10431 (дата звернення: 21.03.2025)
26. NS-3 Docker Repository Usage. GitHub. URL: https://github.com/nsnam/ns-3-docker (дата звернення: 25.03.2025)
27. Commands to Build NS-3 Container. Stack Overflow. URL: https://stackoverflow.com/questions/54969346/building-ns3-in-docker (дата звернення: 29.03.2025)
28. Organizing Test Directories. GitLab Docs. URL: https://docs.gitlab.com/ee/ci/variables/#file-structure (дата звернення: 02.04.2025)
29. Enabling Wireless Support in NS-3 Docker. Medium. URL: https://medium.com/@netresearch/ns3-docker-wifi-setup-guide (дата звернення: 06.04.2025)
30. Continuous Integration for Network Simulations. GitHub Actions Docs. URL: https://docs.github.com/en/actions (дата звернення: 30.04.2025)
31. Markdown and HTML Reporting Automation. GitLab Docs. URL: https://docs.gitlab.com/ee/ci/yaml/#artifactsreports (дата звернення: 18.04.2025)
32. Cloud Platforms for Network Modeling. AWS Research Blog. URL: https://aws.amazon.com/blogs/publicsector/simulating-networks-cloud-scale/ (дата звернення: 22.04.2025)
33. Using Web Interfaces in Simulators. Hackaday. URL: https://hackaday.com/2021/09/14/web-based-network-simulation-tools/ (дата звернення: 26.04.2025)

34. Comparison of NS-3 and GNS3. TechRadar. URL: https://www.techradar.com/comparison/ns3-vs-gns3 (дата звернення: 17.04.2025)
35. Research Templates for NS-3. Academia.edu. URL: https://www.academia.edu/42415793/NS3_Research_Templates (дата звернення: 13.04.2025)

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра комп'ютерної інженерії



КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

НА ТЕМУ: «МЕРЕЖЕВИЙ ЕМУЛЯТОР ДЛЯ ДОСЛІДЖЕННЯ ПРОТОКОЛІВ БЕЗДРОТОВОГО ЗВ'ЯЗКУ»

Виконав студент групи КІД-42
Татусько Ростислав Ігорович
Керівник дипломної роботи:
Волохін Віталій Васильович,
канд. техн. наук, доц.

Київ – 2025

Мета, об’єкт, предмет дослідження

Мета роботи – спрощення дослідження протоколів бездротового зв’язку за рахунок використання моделі контейнеризованого середовища емуляції на основі мережевого симулятора NS-3

Об’єкт дослідження – процес моделювання бездротової взаємодії вузлів у мережевому середовищі

Предмет дослідження – програмні засоби емуляції протоколів бездротового зв’язку, що реалізовані в інструментарії NS-3 у поєднанні з контейнерними технологіями

Актуальність роботи

У роботі розглянуто архітектуру та функціональні можливості симулятора NS-3 для дослідження протоколів бездротового зв’язку. Описано процеси розгортання середовища емуляції у контейнерах Docker, налаштування базового Wi-Fi-сценарію, модифікації параметрів топології та збору логів. Проведено запуск симуляції, збереження результатів, їх попередній аналіз та візуалізацію. Особливу увагу приділено забезпеченню відтворюваності експериментів шляхом ізоляції середовища запуску та автоматизації серійних тестів.

Здійснено порівняння NS-3 з іншими інструментами дослідження мереж — OMNeT++, Mininet-WiFi, GNS3 — за критеріями підтримки протоколів, гнучкості сценаріїв, простоти використання. Окреслено потенційні напрямки вдосконалення симуляційного середовища: реалізація веб-інтерфейсу для користувача, запуск на хмарних платформах, створення шаблонів для освітніх задач і включення в CI/CD-процеси.

Проблематика у способах та засобах дослідження мереж

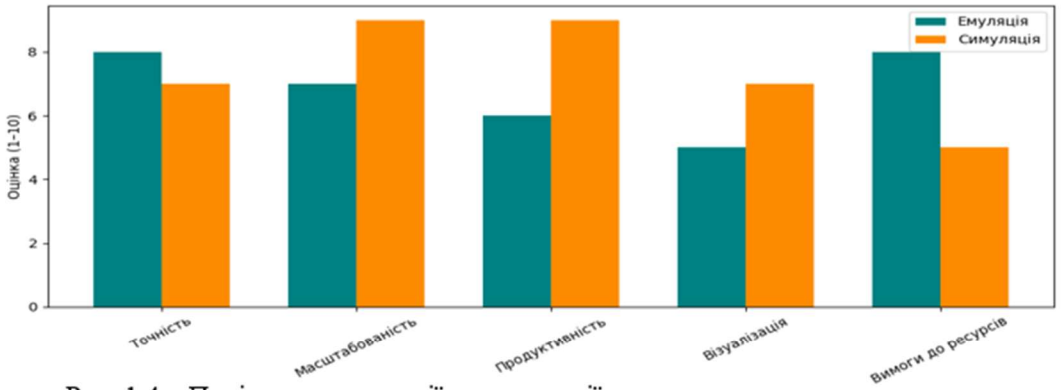


Рис. 1.4 – Порівняння емуляції та симуляції за ключовими характеристиками

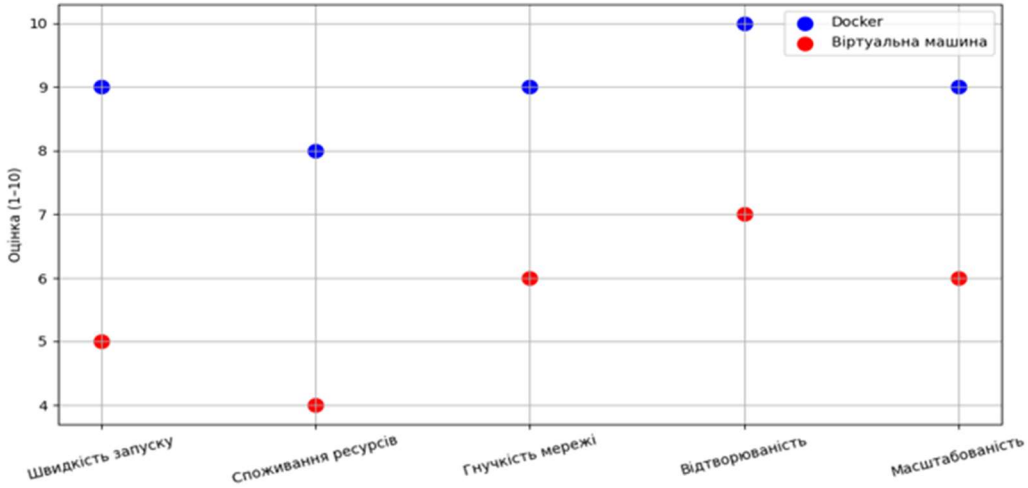


Рис. 1.9 – Порівняння Docker і віртуальних машин за критеріями

Загальна архітектура запропонованої системи

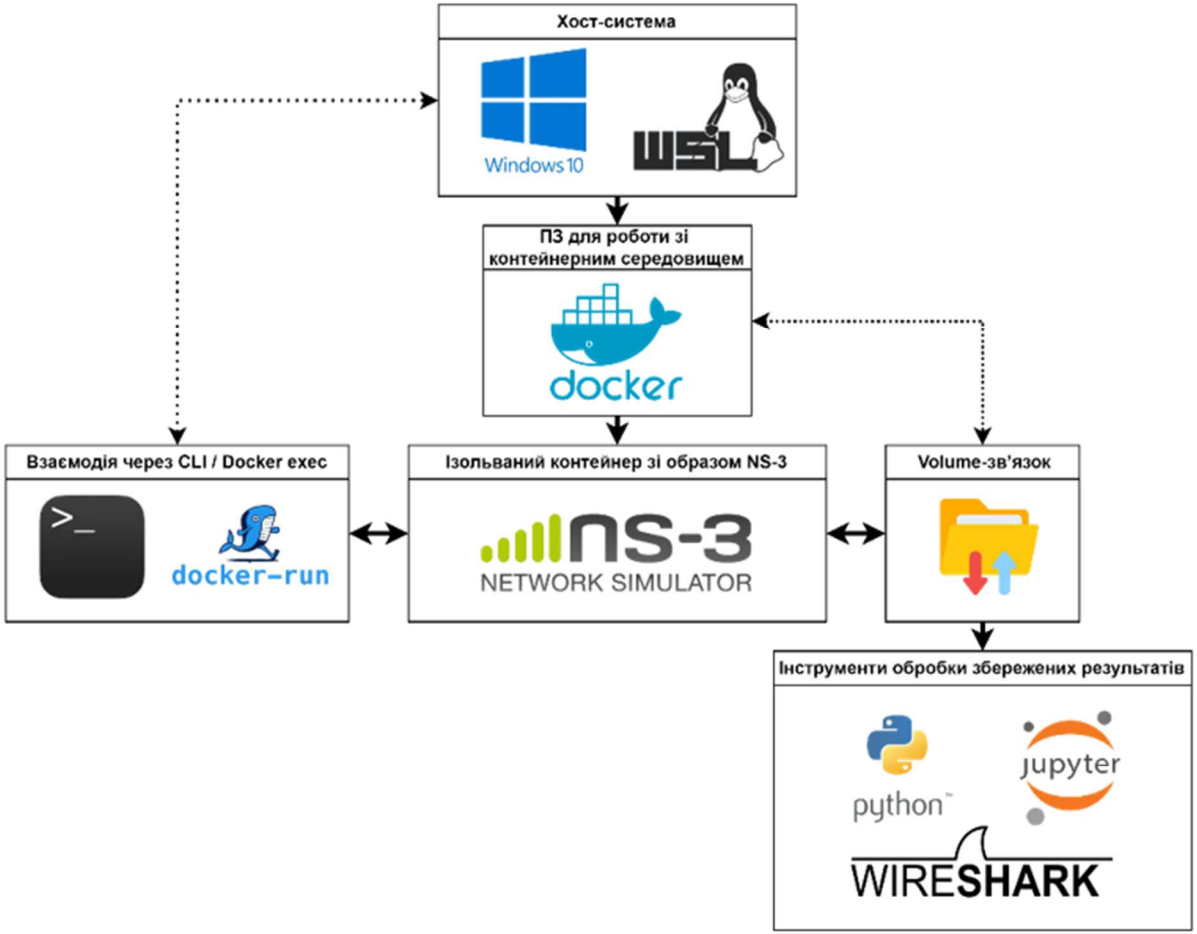


Рис. 2.2 – Взаємодія між компонентами системи (Docker, NS-3, CLI)

Розгортання віртуального ізольованого середовища



Рис. 2.3 – Клонування репозиторію NS-3 Docker

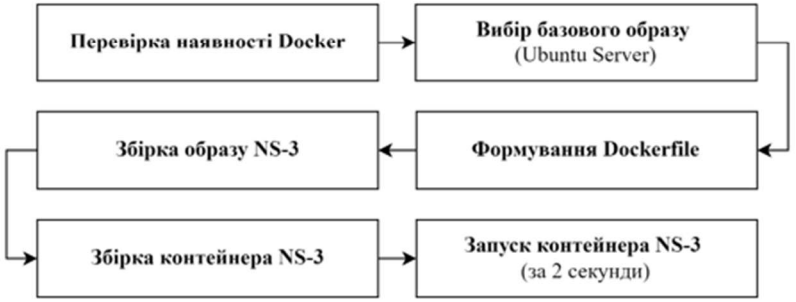


Рис. 2.1 – Принцип реалізації системи емуляції бездротових мереж на базі Docker (NS-3)

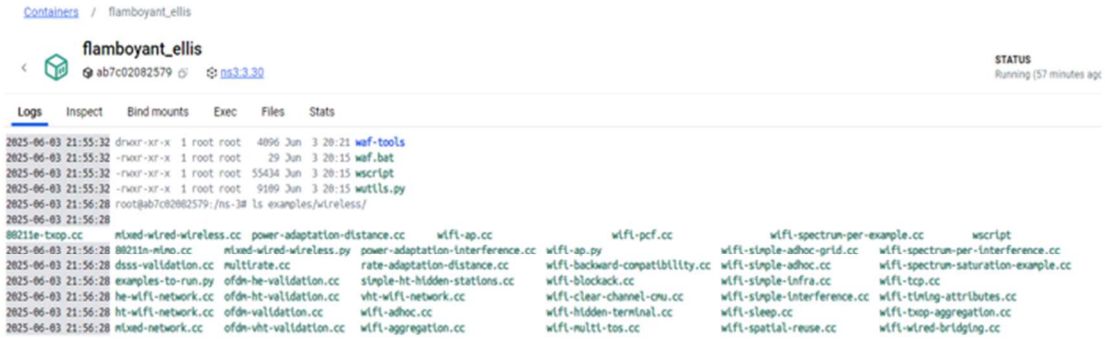


Рис. 2.4 – Структура директорій для тестових сценаріїв NS-3

Демонстрація функціонування системи (CLI)



Рис. 2.5 – Алгоритм роботи сценарію wifi-simple-adhoc

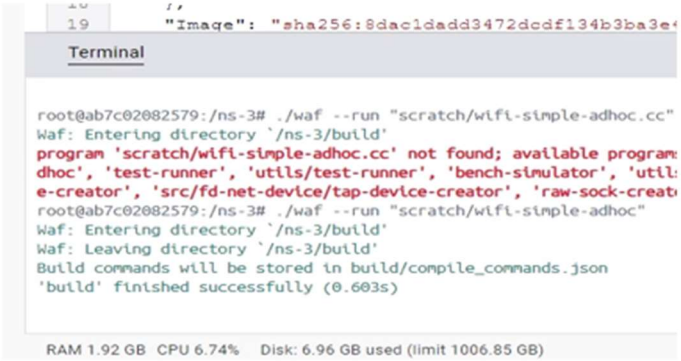


Рис. 2.9 – Запуск базового Wi-Fi-сценарію

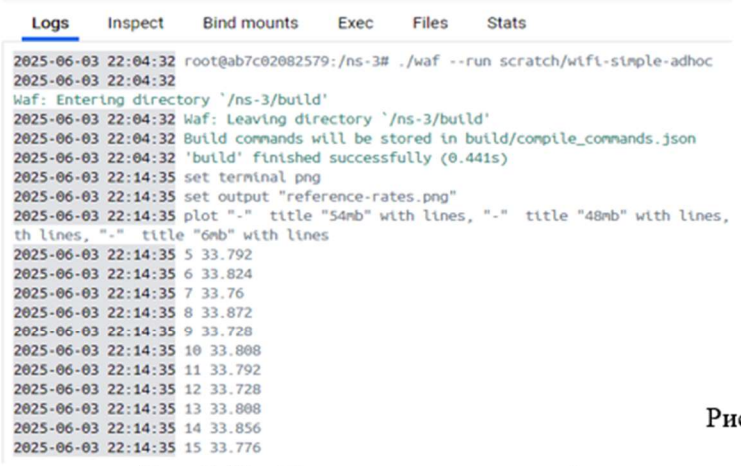


Рис. 2.10 – Приклад логу результатів



Рис. 2.7 – Алгоритм обробки логів та результатів

Демонстрація функціонування системи (gnuplot)

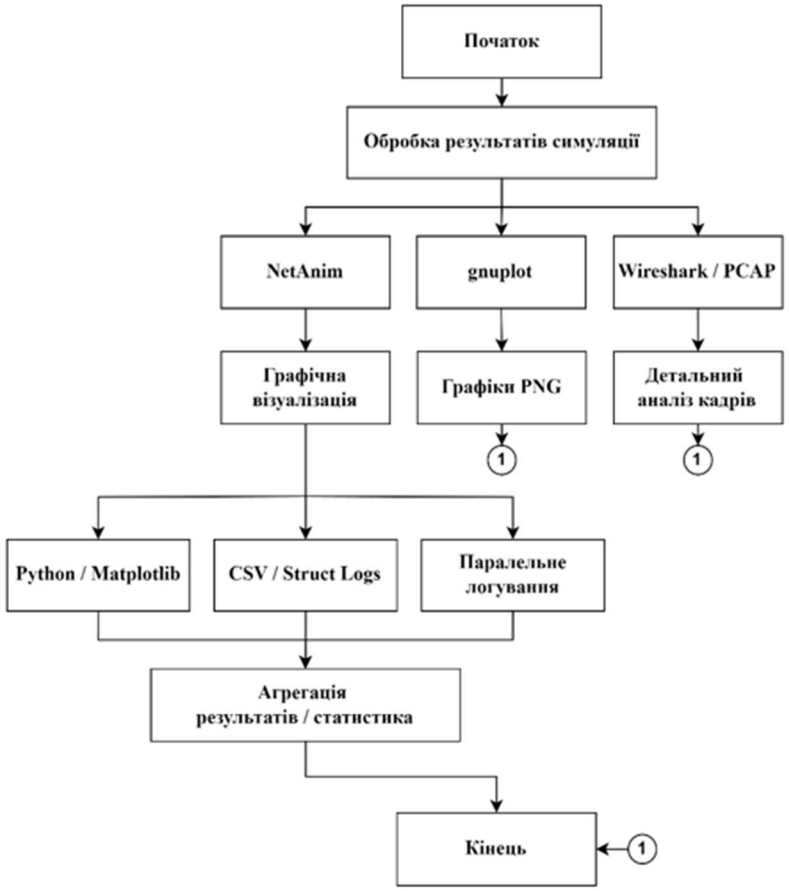


Рис. 2.8 – Алгоритм побудови графіків або трас-файлів для візуалізації

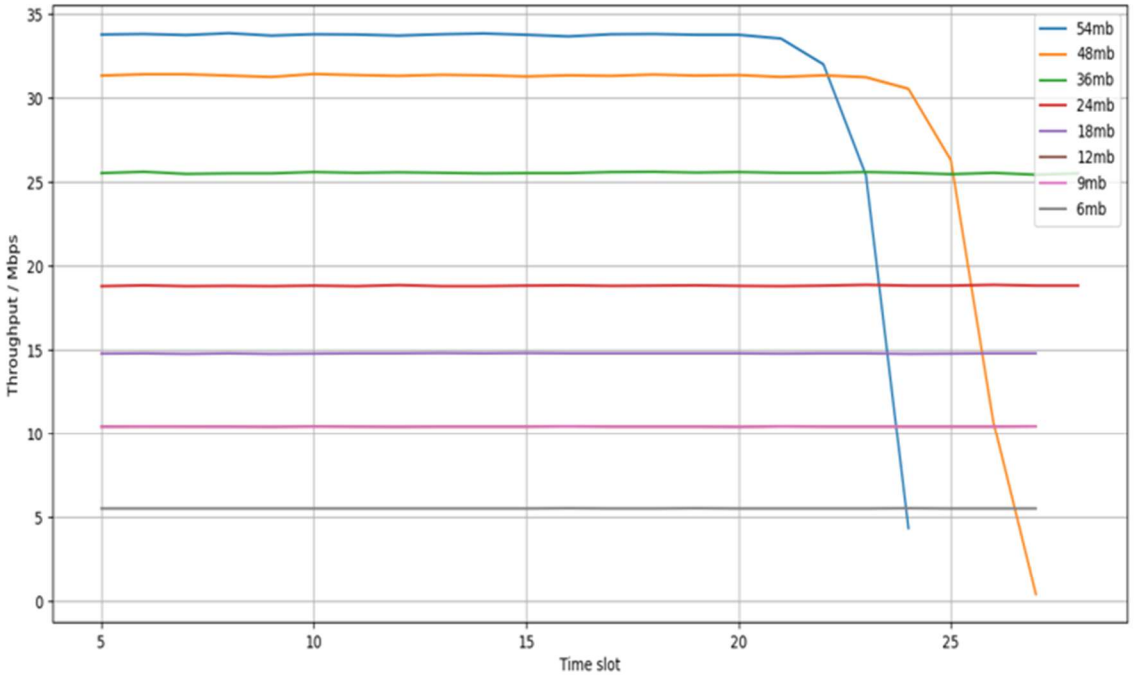


Рис. 2.11 – Візуалізація результатів емуляції базового Wi-Fi-сценарію (вивід у gnuplot)

Виявленні недоліки під час тестування

Табл. 3.1 – Типові проблеми при запуску емуляції в Docker та шляхи їх вирішення

Проблема	Причина	Рішення
Відсутність доступу до інтерфейсів	Docker за замовчуванням ізолює мережу	**Використання режиму host або bridge**
Неможливість виводу графіки	Відсутній доступ до X-сервера	**Проксі-доступ через X11, налаштування xhost +**
Витік ресурсів	Некоректне завершення контейнера	**Обов'язкове використання --rm або моніторинг процесів**
Проблеми з доступом до файлів	Відсутність монтованих томів	**Використання параметра -v при запуску контейнера**

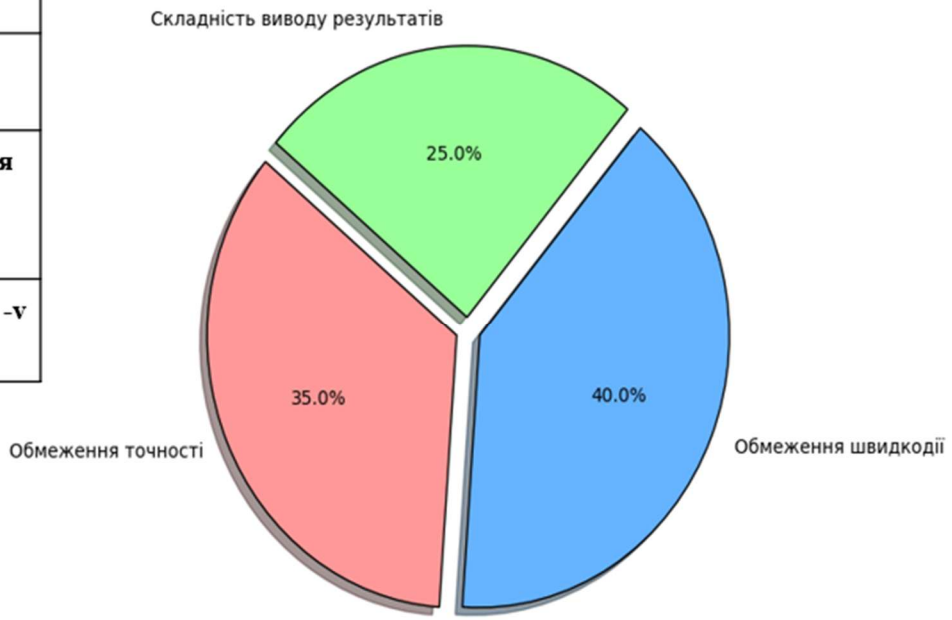


Рис. 3.1 – Виявлені слабкі місця поточної моделі емуляції

Методи покращення існуючої системи

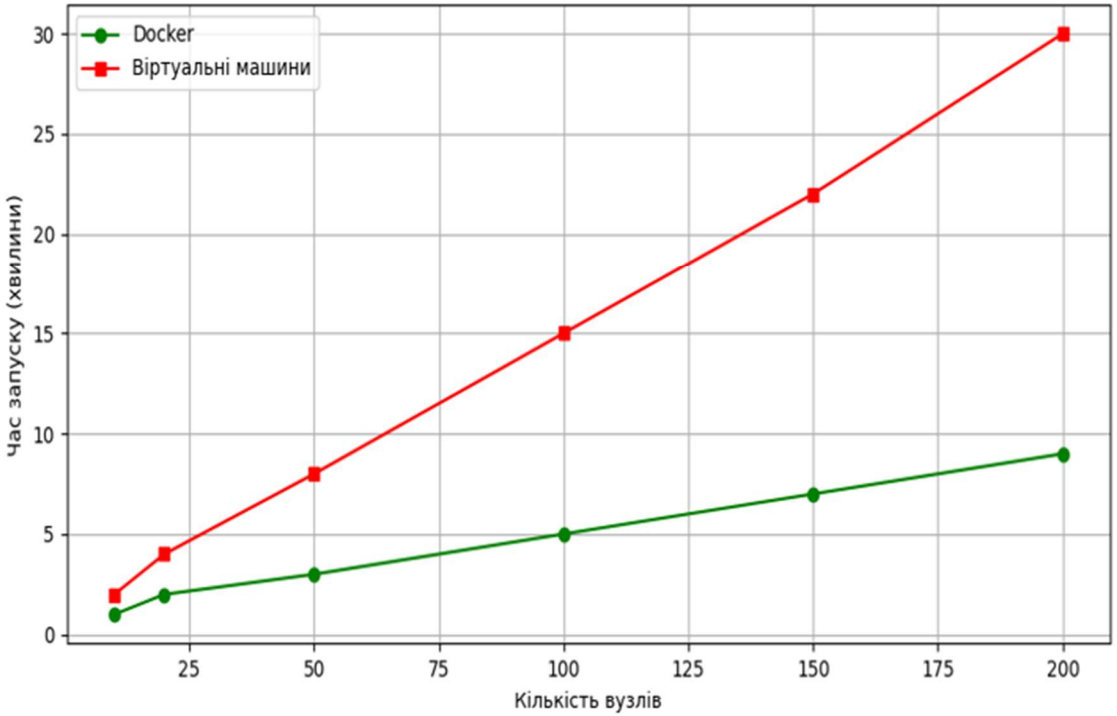


Рис. 3.2 – Масштабування мережевого експерименту з Docker Compose



Рис. 3.3 – Алгоритм автоматизованого запуску емуляції зі параметрами

Порівняння запропонованої системи зі альтернативами

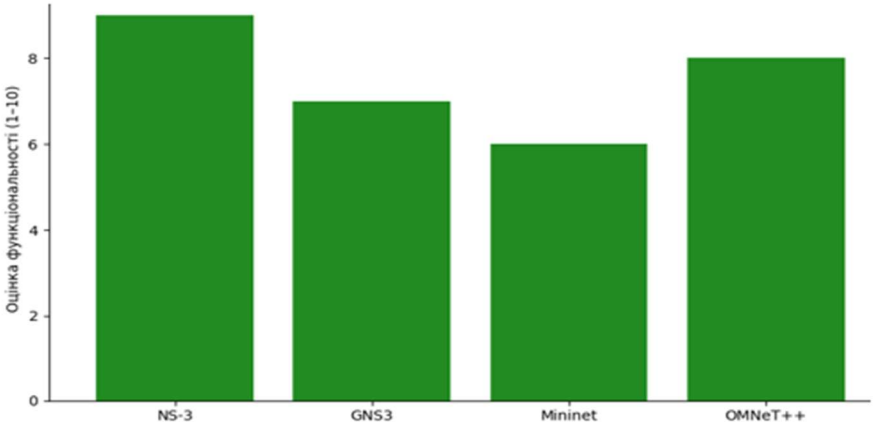


Рис. 3.5 – Порівняння дослідницьких інструментів за функціональністю

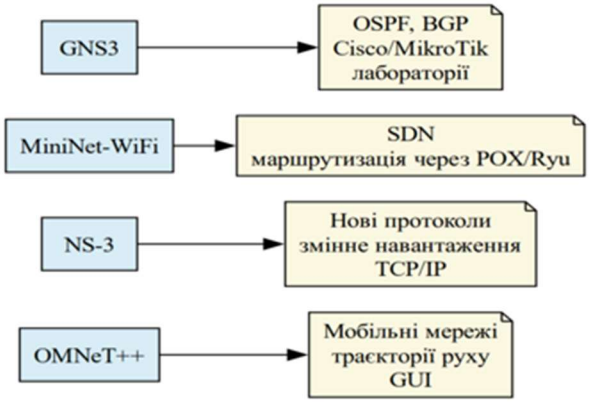


Рис. 3.6 – Сценарії, що реалізуються в альтернативних симуляторах

Табл. 3.1 – Порівняння NS-3 з альтернативними інструментами симуляції

Критерій / Інструмент	**NS-3**	**OMNeT++**	**GNS3**	**Mininet-WiFi**
Мова програмування	**C++, Python**	C++, NED	Python, інше	Python
Точність моделювання	**Висока, подрібнена фізична модель**	Висока, модульна модель	Середня, емуляція реальних образів	Низька–середня (імітація Wi-Fi)
Підтримка Wi-Fi	**Так (IEEE 802.11 a/b/g/n/ac/ax)**	Так (через INET framework)	Обмежена (через VirtualBox/QEMU)	Так (імітація з wmediumd)
Підтримка LoRaWAN	**Так (через зовнішній модуль)**	Частково (моделі користувача)	Ні	Ні
Підтримка LTE / 5G	**Так (lte, nr модулі)**	Обмежена	Ні	Ні
Візуалізація	**Так (NetAnim, PyViz)**	Вбудована GUI	Так (реальна топологія)	Так (GUI + matplotlib)
Сценарії IoT/AdHoc	**Так**	Так	Ні	Так (імітація IoT)
Контейнеризація (Docker)	**Так**	Через налаштування	Так	Так
Інтеграція з Python	**Повна (Python API)**	Часткова (виклики з NED)	Повна	Повна
Масштабованість	**Висока**	Висока	Обмежена	Середня
Реалістичність трафіку	**Висока (можна задавати вручну)**	Висока (через модулі)	Висока (емуляція реального ПЗ)	Низька–середня
Підтримка розподілених сценаріїв	**Частково (з MPI)**	Частково (з INET-MPI)	Ні	Ні
Активність спільноти	**Висока (оновлення, патчі)**	Висока (академічна спільнота)	Середня	Середня
Навчальна документація	**Так (офіційна + приклади)**	Так (дуже докладна)	Помірна	Обмежена
Тип моделювання	**Симуляція (event-driven)**	Симуляція (discrete event)	Емуляція (реальні пакети)	Імітація/емуляція

Висновки

У процесі реалізації роботи було успішно налаштовано середовище симулятора NS-3 у контейнері Docker, що забезпечило ізольованість, повторюваність та гнучкість запуску мережевих сценаріїв. Базовий сценарій Wi-Fi-зв'язку дозволив візуалізувати топологію, проаналізувати параметри передачі даних, затримки й втрати пакетів. Проведено серію експериментів зі зміною кількості вузлів, конфігурацій руху та параметрів трафіку, що дало змогу оцінити вплив змін на мережеву продуктивність та адаптивність сценаріїв.

Особливу увагу приділено обробці результатів: лог-файли експортувалися для подальшого аналізу за допомогою сторонніх інструментів та побудови графіків, що надало змогу оцінити динаміку мережевого трафіку в різних конфігураціях. У рамках подальшого розвитку проаналізовано можливості автоматизації запуску симуляцій, інтеграції з CI/CD та хмарного масштабування, а також створення веб-інтерфейсу для спрощення взаємодії з NS-3.