Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Протокол обміну даними між пристроями Інтернету речей (IoT)
Назва теми

Рівень вищої освіти *перший (бакалаврський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Шифр КвРКІ 230199.23.01.03 ПЗ

Виконав здобувач III курсу, група КІ2с-23-1 Владислава БУРЯКОВСЬКА
Підпис Ініціали, прізвище

Керівник Дмитро МЕДЗАТИЙ
Науковий ступінь, учене звання Підпис Ініціали, прізвище

Нормоконтролер Сергій ЛИСЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КІІС Ольга ПАВЛОВА
«04» червня 2026 р. Підпис Ініціали, прізвище
дата

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Рівень вищої освіти перший (бакалаврський)

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
_____ Ольга ПАВЛОВА

" 10 " 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Буряковській Владиславі Вікторівні
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Протокол обміну даними між пристроями Інтернету речей (IoT)

Керівник проекту (роботи) Медзатий Дмитро Миколайович, д.т.н., проф.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Термін подання здобувачем роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

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

Аналіз предметної області та існуючих підходів до обміну даними в IoT

Проєктування протоколу обміну даними в IoT-системі

Програмна реалізація та перевірка роботи протоколу обміну даними

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Архітектура ПЗ проєкту

Алгоритм роботи системи

Апаратне забезпечення проєкту

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «_10_»_____01_____2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2026	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2026	виконано
3	Робота над розділом 1 – аналіз предметної області та постановка задачі	01.03.2026	виконано
4	Робота над розділом 2 – вибір компонентів для проєктування протоколу обміну даними в iot-системі	01.04.2026	виконано
5	Робота над розділом 3 – проєктування протоколу обміну даними	29.04.2026	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2026	виконано
7	Попередній захист ВКР	26.05.2026	виконано
8	Захист ВКР на засіданні ЕК	Червень 2026 року	

Здобувач_____ ___Владислава БУРЯКОВСЬКА__
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи_____ ____Дмитро МЕДЗАТИЙ___
Підпис Ім'я, ПРІЗВИЩЕ

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 230199.23.01.03 ПЗ	Пояснювальна записка	68		
			Графічні матеріали			
2		КвРКІ 230199.23.01.03 Е8	Алгоритм роботи системи	1		
3		КвРКІ 230199.23.01.03 Е8	Архітектура ПЗ для	1		
			кіберфізичної системи			
4		КвРКІ 230199.23.01.03 Е3	Апаратне забезпечення	1		
			проєкту			

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 230199.23.01.03 ВП		
Розробив		Буряковська		02.06	Відомість проекту	Літера	Аркуш	Аркушів
Перевір.		Медзатий		01.06		У	1	1
Н. контр.		Лисенко				ХНУ, КІ2с-23-1		
Затв.		Павлова		01.06				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Протокол обміну даними між пристроями Інтернету речей (IoT)».

Автор роботи: Владислава Буряковська.

Керівник роботи: Дмитро МЕДЗАТИЙ.

Пояснювальна записка: 68 с., 12 рис., 8 табл., 3 дод., 52 джерела.

Графічна частина: 3 креслення.

БІНАРНИЙ ПАКЕТ, ІНТЕРНЕТ РЕЧЕЙ, МІКРОКОНТРОЛЕР ESP32, ПРОТОКОЛ LDP-IOT, СЕРВЕРНА ЧАСТИНА, ТЕЛЕМЕТРИЧНІ ДАНІ.

Кваліфікаційна робота бакалавра присвячена розробці програмно-апаратного прототипу протоколу обміну даними між пристроями Інтернету речей. Актуальність теми зумовлена зростанням кількості IoT-пристроїв у локальних мережах, побутових, промислових і навчальних системах, де важливими є компактність повідомлень, стабільність передавання, контроль цілісності та можливість роботи пристроїв з обмеженими ресурсами.

Метою роботи є проєктування, програмна реалізація та перевірка протоколу обміну даними між IoT-вузлом і серверною частиною. Для досягнення поставленої мети проаналізовано особливості побудови IoT-систем і наявні протоколи обміну даними, сформовано архітектуру системи, розроблено структуру бінарного пакета, визначено службові поля, формат корисного навантаження TLV, механізми CRC32, ACK/NACK і повторної передачі.

У результаті роботи створено працездатний прототип, який забезпечує формування, передавання, перевірку та журналювання пакетів LDP-IoT.

Підпис здобувача

30.05.2026

Дата

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій спостерігається активне впровадження концепції Інтернету речей, яка передбачає об’єднання фізичних пристроїв у єдине інформаційне середовище. Такі системи вже використовуються у різних сферах, зокрема в промисловості, транспорті, енергетиці, медицині та побутовій автоматизації. Їх основною особливістю є здатність здійснювати безперервний збір даних, їх передачу та обробку з подальшим формуванням керуючих впливів або аналітичних висновків.

Одним із ключових елементів функціонування таких систем є організація обміну даними між пристроями. Саме від ефективності цього процесу залежить своєчасність отримання інформації, коректність її обробки та стабільність роботи всієї системи.

Існуючі протоколи обміну даними, такі як MQTT та CoAP, уже знайшли широке застосування у практичних системах. Вони забезпечують базові механізми передачі інформації, підтримують різні моделі взаємодії та дозволяють будувати масштабовані IoT-рішення. Проте проведений аналіз показує, що такі протоколи не завжди повністю враховують специфіку конкретних задач, особливо у випадках, коли пристрої мають обмежені ресурси або працюють у складних мережевих умовах.

У результаті цього виникає необхідність розроблення спеціалізованих підходів до організації обміну даними, які дозволяють забезпечити оптимальний баланс між продуктивністю, надійністю, енергоефективністю та складністю реалізації. Саме це визначає актуальність теми бакалаврської кваліфікаційної роботи, спрямованої на створення ефективного протоколу обміну даними для систем Інтернету речей.

Метою бакалаврської кваліфікаційної роботи визначено розроблення протоколу обміну даними між пристроями IoT, який забезпечує ефективну

передачу інформації в умовах обмежених ресурсів, нестабільних каналів зв’язку та підвищених вимог до енергоефективності.

Для досягнення поставленої мети сформульовано такі основні завдання: виконати аналіз предметної області та існуючих протоколів обміну даними; визначити їх переваги та обмеження; розробити архітектуру системи обміну даними; сформувати структуру протоколу та формат повідомлень; реалізувати алгоритм обміну даними; забезпечити базові механізми надійності та безпеки; виконати програмну реалізацію запропонованого рішення та оцінити його ефективність.

Об’єктом бакалаврської кваліфікаційної роботи є процес обміну даними в системах Інтернету речей. Предметом - методи та засоби організації протоколів передачі даних між IoT-пристроями.

Практичне значення отриманих результатів полягає у можливості використання розробленого протоколу в реальних IoT-системах, де необхідно забезпечити ефективний обмін даними при обмежених ресурсах і змінних умовах функціонування. Отримані рішення можуть бути використані як основа для подальшого вдосконалення систем обміну даними та інтеграції їх у більш складні інформаційні інфраструктури.

Структура бакалаврської кваліфікаційної роботи включає вступ, три розділи, висновки та список використаних джерел. У першому розділі виконано аналіз предметної області та існуючих підходів до організації обміну даними. У другому розділі розроблено архітектуру системи та запропоновано власний протокол обміну. У третьому розділі реалізовано програмну частину та проведено оцінку ефективності запропонованого рішення.

Обраний напрям дослідження дозволяє вирішити актуальні задачі підвищення ефективності обміну даними в IoT-системах та створити основу для подальшого розвитку інтелектуальних розподілених систем.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ ПІДХОДІВ ДО ОБМІНУ В ІОТ

1.1 Особливості побудови систем Інтернету речей та їх архітектура

Сучасний розвиток інформаційних технологій характеризується активним переходом до розподілених обчислювальних систем, у яких фізичні об’єкти інтегруються в єдине цифрове середовище. Одним із ключових напрямів такого розвитку визначено Інтернет речей, у межах якого пристрої здатні здійснювати збір, обробку та передавання даних у режимі реального часу. Це дозволяє реалізувати безперервний моніторинг процесів, автоматизацію прийняття рішень і підвищення ефективності функціонування систем різного призначення [13, 21].

Архітектура систем Інтернету речей формується як багаторівнева структура, що об’єднує різнорідні компоненти, кожен із яких виконує окремі функції. На нижньому рівні розташовано сенсорні вузли, які безпосередньо взаємодіють із фізичним середовищем. Ці пристрої оснащено датчиками, мікроконтролерами та модулями зв’язку, що забезпечують первинний збір і передачу даних. Водночас такі вузли характеризуються обмеженим обсягом оперативної пам’яті, низькою продуктивністю та високими вимогами до енергоефективності, що накладає суттєві обмеження на програмну реалізацію [24, 33].

Наступним елементом архітектури є комунікаційний рівень, який відповідає за передавання даних між вузлами системи. У межах цього рівня використовуються різні технології бездротового зв’язку, зокрема Wi-Fi, Bluetooth Low Energy, LoRa та інші. Вибір конкретної технології визначається умовами експлуатації, необхідною дальністю зв’язку, швидкістю передавання даних і рівнем енергоспоживання. При цьому характерною особливістю IoT-систем є нестабільність мережевого середовища, що призводить до втрат пакетів і потребує застосування спеціальних механізмів забезпечення надійності [21, 34].

У більш складних системах передбачено використання проміжного рівня у вигляді шлюзів або edge-пристроїв. Такі компоненти виконують функції агрегації, попередньої обробки та фільтрації даних, що дозволяє зменшити навантаження на центральні сервери. Крім того, edge-рівень забезпечує можливість локального прийняття рішень, що є критично важливим для систем із жорсткими вимогами до затримок [23, 36].

Верхній рівень архітектури представлено серверною або хмарною інфраструктурою, де здійснюється зберігання, аналіз та візуалізація отриманих даних. Саме на цьому рівні реалізуються алгоритми обробки великих обсягів інформації, інтеграція з іншими системами та формування керуючих впливів. Використання хмарних технологій забезпечує масштабованість і доступність сервісів, однак водночас підвищує вимоги до захисту інформації [25, 26].

У процесі побудови IoT-систем застосовуються різні моделі взаємодії між компонентами. Найбільш поширеним є клієнт-серверний підхід, у якому пристрої передають дані до центрального вузла обробки. Така модель є простою в реалізації та забезпечує централізоване управління. Альтернативою виступає модель із використанням брокера повідомлень, що реалізує принцип публікації-підписки, який дозволяє підвищити гнучкість системи та зменшити зв’язність між компонентами [29, 33].

Розподілені архітектури, у яких вузли взаємодіють безпосередньо один з одним, забезпечують підвищену відмовостійкість і масштабованість. Проте такі підходи ускладнюють реалізацію механізмів синхронізації, управління та безпеки, що обмежує їх широке застосування у практичних системах [24, 37].

У результаті аналізу особливостей побудови систем Інтернету речей можна зробити висновок, що їх архітектура формується під впливом обмежень апаратних ресурсів, вимог до надійності, енергоефективності та безпеки. Це призводить до необхідності використання спеціалізованих підходів до організації обміну даними, які враховують специфіку IoT-середовища та забезпечують ефективну взаємодію між компонентами системи.

1.2 Принципи організації обміну даними в IoT-системах

Організація обміну даними в системах Інтернету речей визначає ефективність функціонування всієї інфраструктури, оскільки саме через механізми передавання інформації забезпечується взаємодія між фізичними пристроями, обчислювальними вузлами та сервісними компонентами. У таких системах обмін даними розглядається не лише як технічний процес передачі повідомлень, а як комплексна взаємодія, що враховує обмеження апаратних ресурсів, умови середовища функціонування та вимоги до надійності, безпеки й енергоефективності [13, 33].

Однією з ключових особливостей IoT-систем є гетерогенність їх складових. У межах однієї системи можуть функціонувати пристрої з різною обчислювальною потужністю, різними типами сенсорів і різними каналами зв’язку. Це призводить до необхідності формування універсальних, але водночас адаптивних підходів до організації обміну даними, які дозволяють забезпечити сумісність між компонентами та стабільність роботи системи в цілому [24, 29].

Передавання даних у IoT здійснюється у межах багаторівневої моделі, де кожен рівень виконує окрему функцію. Фізичний рівень забезпечує передачу сигналів через середовище зв’язку, яке найчастіше є бездротовим. У таких умовах передавання даних супроводжується впливом зовнішніх факторів, зокрема електромагнітних завад, втрат сигналу та зміни якості каналу. Це призводить до появи помилок передачі та необхідності впровадження додаткових механізмів корекції та контролю [21, 34].

На транспортному рівні визначається спосіб доставки повідомлень між вузлами системи. Використання протоколів, що працюють поверх TCP, дозволяє забезпечити гарантовану доставку даних, контроль порядку повідомлень і перевірку їх цілісності. Проте така надійність досягається за рахунок збільшення службового навантаження, що є критичним для пристроїв із обмеженими ресурсами. У свою чергу застосування UDP дозволяє значно зменшити затримки

та обсяг переданих даних, однак перекладає відповідальність за надійність передачі на прикладний рівень [37, 39].

Прикладний рівень у системах Інтернету речей є визначальним, оскільки саме тут формується логіка обміну даними, структура повідомлень і правила взаємодії між пристроями. На цьому рівні визначається формат пакетів, який повинен бути достатньо компактним для ефективного використання ресурсів, але водночас достатньо інформативним для передачі необхідних даних. Використання легковагових форматів дозволяє зменшити навантаження на мережу та підвищити швидкість обробки інформації [33, 34].

Одним із базових принципів організації обміну є мінімізація службового трафіку. У системах із великою кількістю пристроїв навіть незначне збільшення обсягу службових даних може призвести до значного навантаження на мережу. Це особливо актуально для бездротових мереж із обмеженою пропускною здатністю. У зв’язку з цим розроблення протоколів обміну передбачає оптимізацію структури повідомлень і скорочення кількості службових полів [37].

Надійність передачі інформації забезпечується за рахунок використання механізмів підтвердження отримання повідомлень, повторної передачі у випадку втрат і контролю цілісності даних. У нестабільних мережах такі механізми є необхідними для гарантування коректності переданої інформації. Зазвичай це реалізується через використання унікальних ідентифікаторів повідомлень, тайм-аутів та алгоритмів повторної передачі, що дозволяє відслідковувати стан обміну та запобігати втратам даних [39].

У підсумку організація обміну даними в IoT-системах базується на поєднанні принципів мінімізації ресурсних витрат, забезпечення надійності передачі, підтримки безпеки та досягнення високої енергоефективності. Реалізація цих принципів потребує використання спеціалізованих підходів до формування протоколів обміну, що враховують особливості функціонування IoT-середовища.

1.3 Аналіз існуючих протоколів обміну даними в IoT

У сучасних системах Інтернету речей ефективність функціонування значною мірою визначається тим, який саме протокол використано для організації обміну даними між пристроями, шлюзами, серверними вузлами та хмарними сервісами. Саме протокол задає правила формування повідомлень, спосіб доставки даних, модель взаємодії між компонентами, а також рівень накладних витрат, затримок, надійності та безпеки. Через це вибір протоколу в IoT-середовищі не може розглядатися як другорядне технічне рішення, оскільки він безпосередньо впливає на масштабованість системи, тривалість автономної роботи вузлів, стабільність обміну та складність програмної реалізації [13, 33].

Актуальні IoT-рішення не обмежуються одним універсальним стандартом. Навпаки, у цій сфері сформувалася група прикладних протоколів, кожен із яких орієнтований на певний сценарій використання. Одні краще підходять для телеметрії та періодичної передачі невеликих повідомлень, інші - для керування пристроями, треті - для інтеграції з промисловими системами або хмарними платформами. Через це під час аналізу доцільно розглядати не лише функціональні можливості окремого протоколу, а і його відповідність реальним умовам роботи IoT-систем, де одночасно присутні обмеження щодо пам’яті, енергоспоживання, нестабільності каналу зв’язку, безпеки та вимог до швидкого відгуку [14, 37].

Найбільш відомим і поширеним рішенням у сфері IoT уже тривалий час залишається MQTT. Його популярність пояснюється тим, що цей протокол з самого початку орієнтовано на легкий обмін повідомленнями у мережах з обмеженою пропускною здатністю та нестабільним зв’язком. У специфікації MQTT версії 5.0 визначено модель взаємодії на основі брокера, де один пристрій публікує повідомлення у певну тему, а інші вузли, які підписалися на цю тему, отримують потрібні дані [6]. Такий підхід вже показав високу гнучкість у системах, де необхідно роз’єднати джерело даних і їх споживача, тобто зробити

так, щоб сенсорний вузол не “знав”, хто саме буде читати його телеметрію [6, 33].

Важливою перевагою MQTT, принцип якого зображено на рисунку 1.1 виступає невеликий розмір службового заголовка, що зменшує навантаження на мережу та дозволяє застосовувати цей протокол у середовищах із помірними ресурсними обмеженнями. Додатково MQTT підтримує три рівні якості обслуговування, що дає змогу обирати компроміс між швидкістю та надійністю доставки. [6, 35].

Разом із тим MQTT не є ідеальним універсальним рішенням. Його архітектура вимагає наявності брокера повідомлень, а це означає, що в систему вводиться додатковий проміжний компонент, від якого залежить уся логіка маршрутизації. У невеликих рішеннях це не створює критичних проблем, однак у великих системах саме брокер може перетворитися на вузол перевантаження або єдину точку відмови, якщо архітектуру не побудовано належним чином [29, 36]. Крім того, MQTT працює поверх TCP, що забезпечує високу надійність, але одночасно додає накладні витрати на встановлення з’єднання, підтримку сесії та службові підтвердження. Для частини надлегких пристроїв це вже не завжди є оптимальним [33, 37].

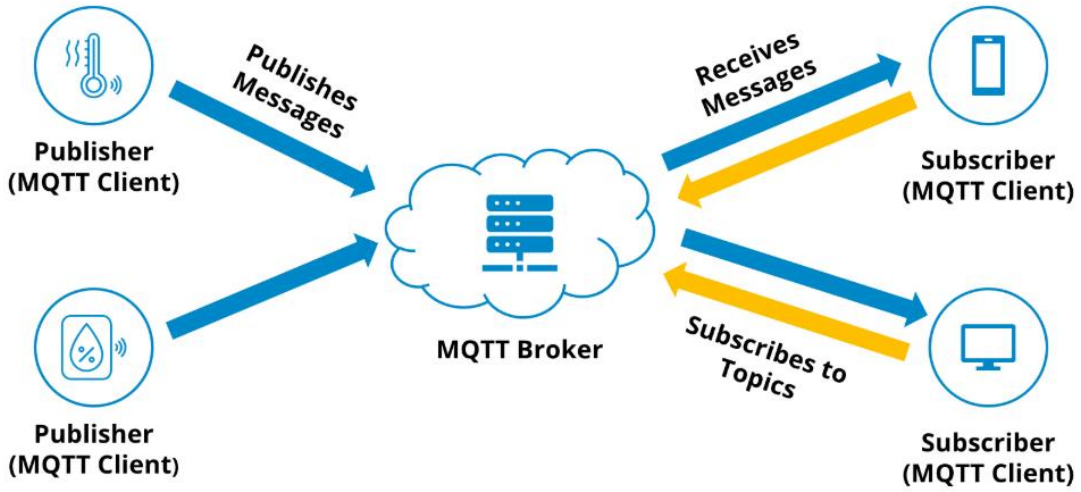


Рисунок 1.1 – Принцип роботи MQTT [50]

Окрему увагу в сучасних роботах приділено питанням безпеки MQTT. Базова специфікація не забезпечує повноцінного вбудованого шифрування на прикладному рівні, тому захист зазвичай реалізується через TLS або додаткові механізми автентифікації. У результаті цього виникає типова для IoT проблема: чим вищий рівень захисту, тим більшими стають обчислювальні витрати та затримки [35, 39]. Саме тому у спеціалізованих роботах уже виконано оцінювання MQTT з увімкненим TLS, а також запропоновано різні варіанти його посилення, орієнтовані на захист від підміни вузлів, несанкціонованого підключення й атак на брокер [35, 40].

Другим ключовим протоколом у сфері IoT вважається CoAP - Constrained Application Protocol. Його створено як легкий протокол для пристроїв з обмеженими ресурсами, причому він концептуально наближений до веб-моделі запит-відповідь. На відміну від MQTT, CoAP працює поверх UDP, а це дозволяє зменшити обсяг службових операцій і скоротити затримки передавання даних [13, 14]. Завдяки цьому CoAP добре підходить для сценаріїв, у яких важливими є компактність реалізації, швидкий обмін короткими повідомленнями та мінімальне енергоспоживання.

Водночас CoAP має і певні обмеження. Робота поверх UDP означає, що значна частина логіки надійності переноситься на прикладний рівень. Тобто якщо система працює у нестабільному каналі або передбачає підвищені вимоги до доставки повідомлень, то реалізація стає складнішою. У результаті цього розробнику потрібно самостійно продумувати повторні передачі, тайм-аути, контроль порядку повідомлень і частину захисних механізмів [13, 37]. Додатково CoAP зазвичай виглядає менш зручним у сценаріях, де потрібно реалізувати велику багатовузлову асинхронну взаємодію через центральний брокер, адже його логіка ближча до класичного клієнт-серверного обміну [14, 33].

З погляду енергоефективності CoAP часто демонструє кращі результати, ніж важчі стекові рішення. Це пов’язано як із роботою поверх UDP, так і з компактністю заголовків та повідомлень. Саме тому в аналітичних роботах CoAP

регулярно розглядається як один із найбільш доречних кандидатів для сценаріїв із критичною потребою у зниженні трафіку й витрат енергії [34, 37]. Водночас у системах, де пріоритетом виступає гнучка модель тематичної доставки повідомлень або інтеграція з брокерами та хмарними сервісами, MQTT часто виявляється більш практичним [33, 35].

Порівняння MQTT і CoAP уже стало класичним для IoT-тематики. MQTT є сильнішим у побудові асинхронних систем з розподілом повідомлень між численними підписниками, тоді як CoAP є легшим і компактнішим у прямому клієнт-серверному обміні. MQTT краще підходить для хмарної телеметрії, CoAP - для локальних або ресурсно-обмежених IP-сумісних вузлів. Якщо говорити простіше, MQTT зручніший як “транспорт подій”, а CoAP - як легкий “запит-відповідь” для сенсорного середовища [13, 14].

Принцип роботи Constrained Application Protocol подано на рисунку 1.2.

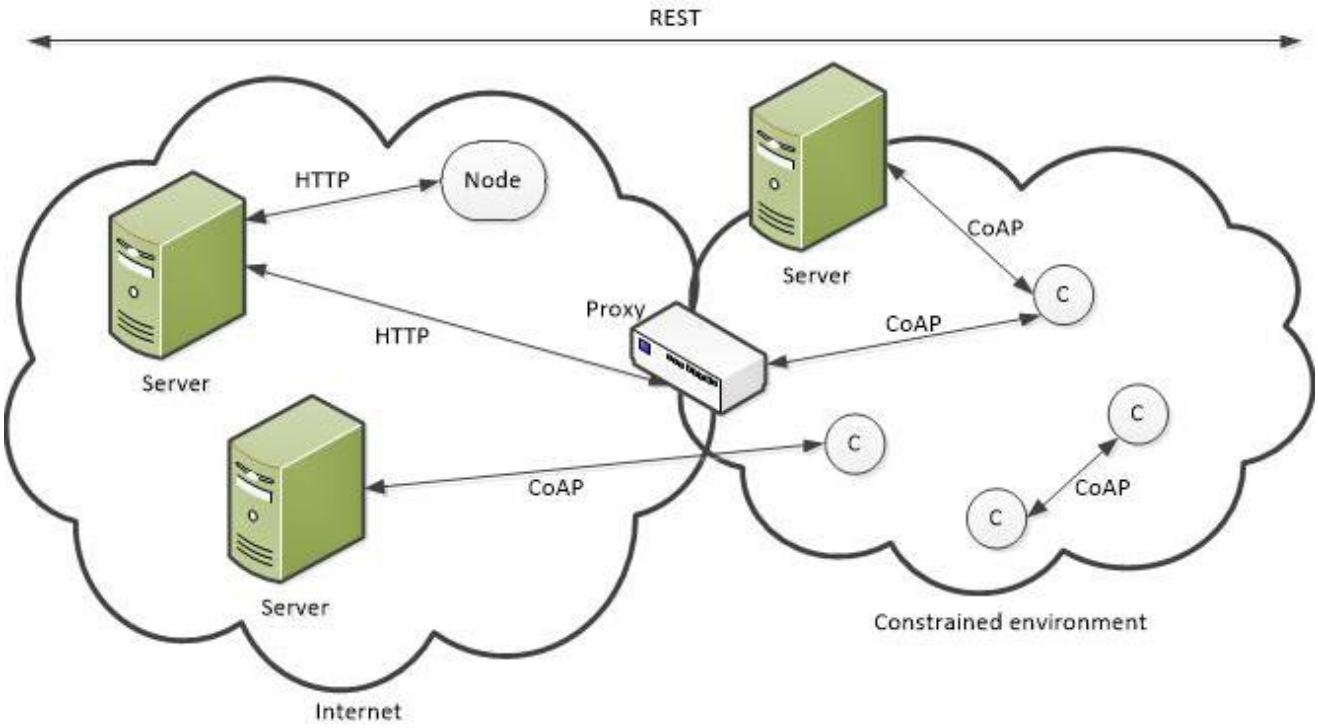


Рисунок 1.2 – Принцип роботи Constrained Application Protocol (CoAP) [51]

Ще одним важливим рішенням залишається HTTP, який історично не створювався спеціально для IoT, але все одно широко застосовується у практичних реалізаціях. Його головна перевага полягає в універсальності, зрозумілості та простоті інтеграції з веб-сервісами, REST API, серверними платформами та звичайними прикладними застосунками. Через це багато ранніх або спрощених IoT-рішень вже будувалися саме на HTTP, особливо коли важливішою була швидкість розробки, а не максимальна оптимізація мережевого обміну [13, 14].

Проте для класичних IoT-сценаріїв HTTP має низку суттєвих недоліків. Він супроводжується значним службовим навантаженням, більшими розмірами заголовків, вищими вимогами до ресурсів клієнта і додатковими витратами енергії. Це робить його менш придатним для мікроконтролерних платформ з обмеженим обсягом пам'яті та автономним живленням [33, 37]. HTTP добре працює там, де пристрій уже має достатній ресурс, а інфраструктура орієнтована на веб-сумісність, однак для легковагового протоколу обміну між вузлами IoT він часто є надмірним.

Принцип роботи HTTP подано на рисунку 1.3.

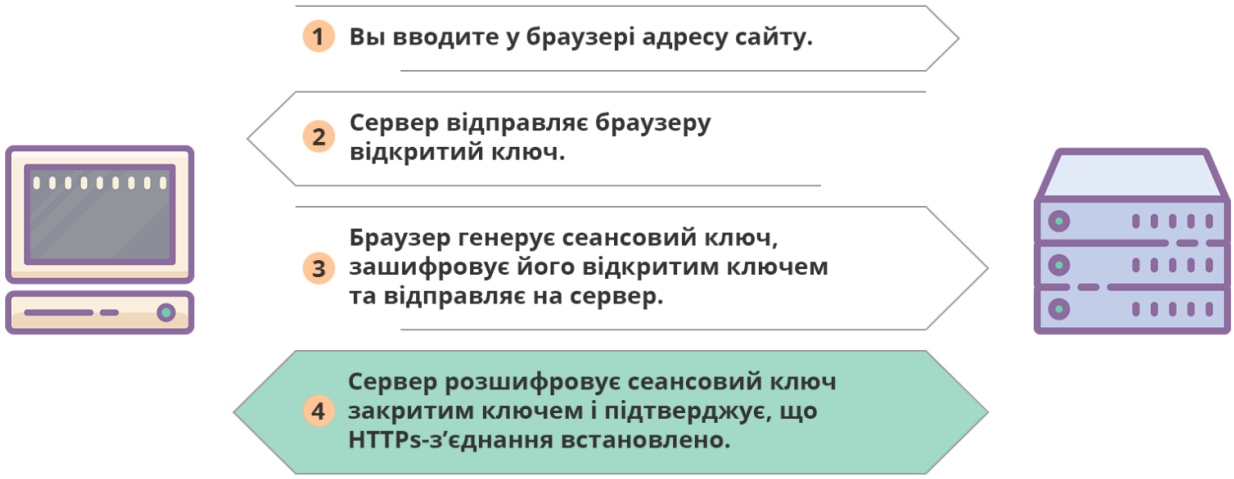


Рисунок 1.3 – Принцип роботи HTTP [52]

У деяких системах застосовується AMQP - Advanced Message Queuing Protocol. Це більш “важкий” протокол обміну повідомленнями, який орієнтовано на надійну доставку, черги, маршрутизацію і складні корпоративні сценарії. У порівнянні з MQTT його перевагою є потужніші механізми керування обміном, однак ці можливості досягаються ціною більшої складності реалізації та значно вищих накладних витрат [14, 33]. Через це AMQP частіше розглядається не як протокол для безпосереднього сенсорного вузла, а як рішення для серверної або шлюзової взаємодії у більш потужній інфраструктурі.

Близьким за ідеєю, але орієнтованим на специфічні задачі реального часу, є DDS - Data Distribution Service. Цей протокол уже показав сильні позиції у промислових, транспортних, робототехнічних та кіберфізичних системах, де важливими виступають низькі затримки, детальний контроль якості сервісу та висока швидкодія [14]. DDS є технологічно сильним рішенням, однак для типового малоресурсного IoT-вузла його використання часто виглядає надто складним. Саме тому у бакалаврській кваліфікаційній роботі його доцільніше розглядати як еталон потужного промислового обміну, а не як основу для легкого власного протоколу.

У багатьох оглядах також згадується XMPP, який спочатку активно використовувався у сфері обміну повідомленнями між користувачами, але згодом знайшов застосування і в IoT. Його сильна сторона полягає у розвиненій екосистемі, гнучкому форматі адресації та розширюваності. Проте XMPP використовує XML-подібну модель обміну, що робить його занадто важким для більшості мікроконтролерних пристроїв з обмеженим обсягом пам’яті [14, 33]. Через це у практиці сучасних IoT-систем XMPP уже не розглядається як пріоритетне рішення для телеметрії з малих вузлів.

У промисловому сегменті дедалі частіше згадується OPC UA, який уже розглядається як один із ключових стандартів інтеграції в індустріальних та автоматизованих середовищах. На відміну від MQTT чи CoAP, OPC UA орієнтовано не лише на передачу повідомлень, а на структурований, семантично

збагачений обмін даними між системами автоматизації, контролерами, промисловими серверами та аналітичними платформами [14, 33]. Це робить його особливо цінним у IIoT, однак для надлегкого сенсорного вузла він часто є занадто складним. Саме тому OPC UA доцільніше застосовувати на рівні шлюзів, серверів або промислових інтеграційних вузлів.

Другим критерієм є надійність доставки. Тут сильні позиції має MQTT завдяки роботі поверх TCP і підтримці рівнів QoS. CoAP теж дозволяє реалізовувати надійну доставку, але потребує для цього confirmable-повідомлень і додаткового контролю на прикладному рівні [6, 13]. HTTP також спирається на надійність TCP, однак його службові витрати вже роблять це рішення менш привабливим для малих вузлів.

Третім критерієм виступає затримка передавання. У сценаріях швидкого реагування зайві службові операції негативно впливають на час доставки. Через це CoAP або легкі UDP-орієнтовані рішення часто виявляються кращими у швидких коротких транзакціях, тоді як TCP-орієнтовані протоколи демонструють кращу стабільність, але вже з вищими затримками [34, 37].

Четвертим критерієм є масштабованість. MQTT завдяки брокерній моделі вже показав хороші можливості у великих системах, де багато джерел і багато споживачів даних [6, 29].

П’ятим критерієм слід вважати безпеку. Жоден із розглянутих протоколів не є абсолютно “самодостатнім” у цьому аспекті. MQTT найчастіше потребує TLS і зовнішніх механізмів керування доступом [35]. CoAP може використовувати DTLS або OSCORE, що вже краще адаптовано до обмежених середовищ [7, 11].

У результаті такого аналізу стає зрозуміло, чому у сфері IoT зберігається інтерес до створення власних прикладних протоколів або модифікованих полегшених рішень. У багатьох випадках системі потрібен не “великий універсальний стандарт”, а компактний механізм обміну, який враховує конкретні вимоги до формату пакета, підтвердження доставки, повторної

передачі, порядку повідомлень, базового захисту та мінімального енергоспоживання [33, 37].

Отже, аналіз існуючих протоколів уже показав, що сучасна IoT-екосистема містить велику кількість технічно зрілих рішень, серед яких найбільше практичне значення мають MQTT, CoAP, HTTP, AMQP, DDS, XMPP, LwM2M та OPC UA. Кожен із них уже сформував власну область доцільного використання, однак жоден не є абсолютно оптимальним для всіх сценаріїв. Саме ця невідповідність між універсальністю готових рішень і специфікою конкретних IoT-систем і створює підґрунтя для подальшого переходу до підрозділу, де будуть розглянуті проблеми та обмеження існуючих підходів [14, 39].

1.4 Проблеми та обмеження існуючих підходів до обміну даними

Аналіз сучасних протоколів обміну даними, що застосовуються в системах Інтернету речей, показує, що, незважаючи на значну кількість розроблених рішень, жоден із них не забезпечує універсального підходу, який однаково ефективно працював би в усіх умовах. Кожен протокол орієнтований на певний клас задач і враховує лише частину вимог, характерних для IoT-середовища. Це призводить до появи низки проблем і обмежень, які суттєво впливають на ефективність побудови систем обміну даними [14, 33].

Однією з ключових проблем є надлишковий службовий трафік, характерний для багатьох протоколів, що не були спочатку розроблені з урахуванням обмежених ресурсів пристроїв. Наприклад, використання HTTP або XML-орієнтованих рішень призводить до значного збільшення розміру повідомлень, що, у свою чергу, впливає на швидкість передачі даних і енергоспоживання вузлів.

Іншою важливою проблемою є залежність від архітектурних компонентів, зокрема брокерів повідомлень або центральних серверів. У протоколах, що

використовують централізовану модель, таких як MQTT, уся логіка маршрутизації та обміну концентрується в одному вузлі.

Це спрощує реалізацію, однак створює потенційну точку відмови, а також обмежує масштабованість системи при значному збільшенні кількості підключених пристроїв [29, 36]. У випадку перевантаження або відмови такого вузла вся система може втратити працездатність або перейти у деградований режим.

Ще одним обмеженням є складність забезпечення безпеки обміну даними. У сучасних IoT-системах необхідно враховувати загрози перехоплення, підміни та несанкціонованого доступу до даних. Реалізація криптографічних механізмів, таких як шифрування та автентифікація, потребує додаткових обчислювальних ресурсів, що може негативно впливати на продуктивність пристроїв і збільшувати енергоспоживання [25, 40]. У результаті виникає необхідність знаходження компромісу між рівнем захисту та ефективністю роботи системи.

Значним обмеженням є також недостатня гнучкість універсальних протоколів. Більшість існуючих рішень орієнтовані на широке коло застосувань і не враховують специфіку конкретних сценаріїв використання.

Це призводить до того, що розробники змушені адаптувати протоколи або створювати додаткові надбудови, що ускладнює систему та знижує її ефективність [14, 33].

Важливим фактором є також енергоспоживання пристроїв, яке безпосередньо залежить від характеристик обміну даними. Частота передачі, обсяг повідомлень і кількість службових операцій визначають рівень енергетичних витрат.

Використання складних протоколів або часті передачі даних можуть суттєво зменшити час автономної роботи пристрою, що є критичним для багатьох IoT-застосувань [34, 41].

У результаті аналізу можна відзначити, що існуючі підходи до організації обміну даними в IoT-системах мають ряд суттєвих обмежень, серед яких

надлишковий службовий трафік, залежність від централізованих компонентів, складність реалізації, проблеми забезпечення надійності та безпеки, а також обмеження масштабованості та енергоефективності.

Наявність цих проблем свідчить про необхідність пошуку альтернативних підходів, які дозволять більш ефективно враховувати специфіку IoT-середовища та забезпечувати оптимальну взаємодію між пристроями. Основні проблеми та обмеження існуючих протоколів обміну даними в IoT зображено на таблиці 1.2

Таблиця 1.2 – Основні проблеми та обмеження існуючих протоколів обміну даними в IoT

Проблема	Сутність проблеми	Вплив на систему
Надлишковий службовий трафік	Використання великих заголовків і текстових форматів повідомлень	Збільшення затримок, перевантаження мережі, підвищене енергоспоживання
Залежність від централізованих компонентів	Необхідність використання брокерів або серверів для маршрутизації	Наявність єдиної точки відмови, обмеження масштабованості
Складність реалізації	Використання складних мережевих стеків і протоколів	Високі вимоги до ресурсів мікроконтролерів
Проблеми надійності передачі	Втрати пакетів, порушення порядку доставки	Необхідність реалізації додаткових механізмів контролю
Складність забезпечення безпеки	Потреба у криптографічних механізмах	Збільшення навантаження на пристрої та енергоспоживання
Недостатня гнучкість	Орієнтація протоколів на універсальні сценарії	Складність адаптації під конкретні задачі

Кінець таблиці 1.2

Проблема	Сутність проблеми	Вплив на систему
Проблеми дублювання даних	Повторне надходження пакетів	Помилки обробки та некоректні результати
Обмеження масштабованості	Зростання навантаження при збільшенні кількості вузлів	Зниження продуктивності системи
Високе енергоспоживання	Часті передачі та великий обсяг даних	Скорочення часу автономної роботи

Внаслідок цього виникає доцільність розроблення спеціалізованого протоколу обміну даними, який буде орієнтований на роботу в умовах обмежених ресурсів, нестабільних мережевих каналів та підвищених вимог до енергоефективності й безпеки. Саме це стає логічним переходом до формулювання задачі бакалаврської кваліфікаційної роботи.

1.5 Висновки та постановка задачі

У попередніх підрозділах розглянуто особливості побудови систем Інтернету речей, принципи організації обміну даними та виконано аналіз існуючих протоколів і підходів. На основі опрацьованого матеріалу встановлено, що сучасні IoT-системи мають досить складну структуру, у якій одночасно поєднуються апаратні вузли, сенсорні модулі, бездротові канали передавання, серверні засоби обробки, сховища даних і користувацькі інтерфейси. Через це якість роботи такої системи залежить не лише від вибору мікроконтролера або мережевого модуля, а й від того, наскільки правильно організовано сам обмін даними між пристроями.

Для практичної реалізації обрано модель, у якій IoT-вузол на базі ESP32 формує пакет із телеметричними даними та передає його на сервер через Wi-Fi і UDP. Використання UDP є обґрунтованим, оскільки цей транспортний рівень не

забезпечує гарантовану доставку самостійно, а отже дає можливість реалізувати власні механізми ACK/NACK, тайм-аутів і повторних передач. Це важливо для змісту бакалаврської роботи, оскільки основним результатом має бути не просто використання готового протоколу, а створення власної логіки обміну.

Для наочності роботи системи необхідно реалізувати вебінтерфейс або панель контролю, у якій відображаються прийняті пакети, структура повідомлення, статистика обміну, кількість помилок, кількість повторних передач і останній прийнятий пакет. Такий інтерфейс не є головною частиною протоколу, але він значно полегшує перевірку результатів і демонструє роботу системи в зрозумілій формі. У межах бакалаврської роботи це дає можливість показати не лише програмний код, а й реальний результат обміну даними.

У результаті виконання поставленої задачі має бути отримано програмно реалізований прототип власного протоколу LDP-IoT. Він повинен показувати повний цикл роботи: від формування пакета на IoT-вузлі до приймання, перевірки, збереження та підтвердження на сервері. Важливо, щоб розроблене рішення не було обмежене одним конкретним параметром або датчиком. Структура TLV має забезпечити можливість подальшого розширення набору переданих даних без зміни загальної логіки протоколу.

У межах бакалаврської кваліфікаційної роботи поставлено задачу розробити архітектуру системи обміну даними, визначити структуру пакета протоколу LDP-IoT, сформувати алгоритм взаємодії між вузлом і сервером, реалізувати програмну частину ESP32, створити серверний модуль приймання та обробки пакетів, додати засоби журналювання й виконати тестування працездатності. Також потрібно оцінити ефективність, надійність і базовий рівень безпеки запропонованого рішення.

2 ПРОЄКТУВАННЯ ПРОТОКОЛУ ОБМІНУ ДАНИМИ В IOT-СИСТЕМІ

2.1 Загальна архітектура системи обміну даними

У другому розділі бакалаврської кваліфікаційної роботи сформовано проєктну основу системи обміну даними між пристроями Інтернету речей. На відміну від першого розділу, де основну увагу приділено аналізу наявних підходів, у цьому розділі розглянуто власне інженерне рішення, яке покладено в основу подальшого розроблення протоколу. Загальна архітектура системи визначає склад основних компонентів, порядок їх взаємодії, напрямки руху повідомлень, місця перевірки пакетів, збереження даних і формування відповідей.

Основою архітектури виступає клієнт-серверна модель взаємодії. У цій моделі IoT-вузол виконує роль клієнта, який формує та передає повідомлення, а сервер виступає центральним вузлом приймання, перевірки та обробки даних. Такий підхід є зручним для практичної реалізації, оскільки дозволяє чітко розділити функції між пристроями. Кінцевий вузол не перевантажується зайвими обчислювальними діями, а серверна частина бере на себе складніші операції, пов’язані з перевіркою структури пакета, журналюванням, збереженням інформації та формуванням відповіді.

У запропонованій архітектурі IoT-вузли можуть працювати як із реальними сенсорами, так і з тестовими даними, що імітують телеметрію. Це дає можливість перевірити роботу протоколу навіть без складної апаратної частини, зосередивши увагу на самому механізмі обміну. На рівні вузла передбачено формування пакета, заповнення службових полів, додавання корисного навантаження, розрахунок контрольного значення та передавання повідомлення до серверної частини. Після відправлення пристрій очікує підтвердження отримання. Якщо відповідь не надходить у межах заданого часу, активується повторна передача.

У ролі мережевого середовища для цієї системи обрано бездротову мережу Wi-Fi. Такий варіант добре підходить для бакалаврської роботи, оскільки підтримується поширеними мікроконтролерними платформами, зокрема ESP32, не потребує складного додаткового обладнання та дозволяє організувати демонстраційний обмін у лабораторних умовах. Через Wi-Fi IoT-вузол передає сформовані пакети на сервер, який може бути запущений на персональному комп'ютері або ноутбуці в межах тієї самої локальної мережі. Схему роботи системи обміну даними подано на рисунку 2.1.

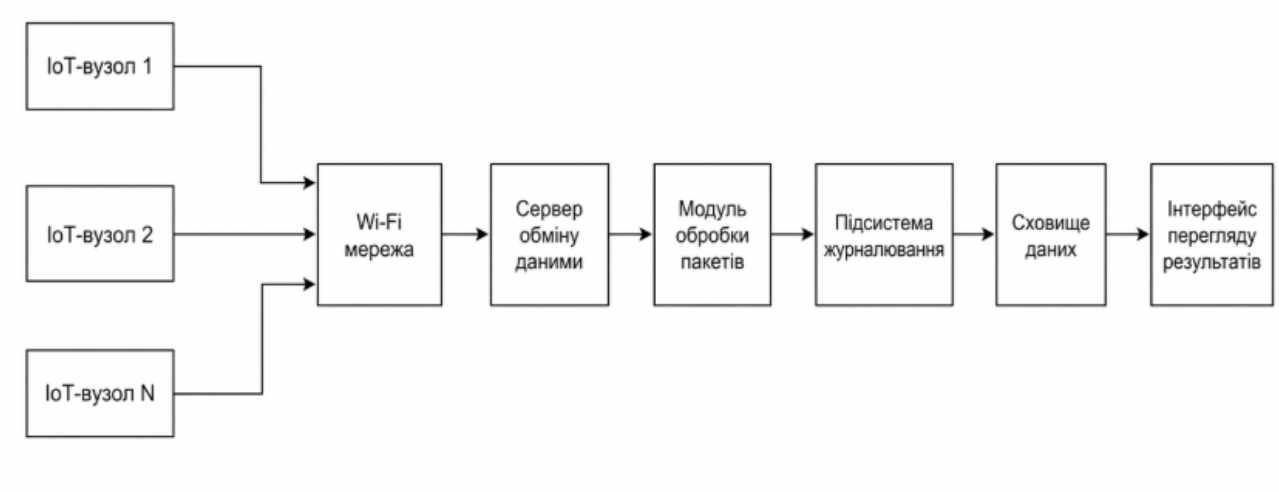


Рисунок 2.1 – Схема роботи системи обміну даними

Сервер обміну даними є центральним компонентом запропонованої архітектури. Він приймає повідомлення від IoT-вузлів, виконує первинну перевірку пакета, аналізує службові поля та передає отриману інформацію до модуля обробки. У межах серверної частини реалізовано логіку приймання, розпакування, перевірки цілісності, виявлення дублікатів і формування відповіді. За рахунок цього сервер виконує не лише роль пасивного приймача, а й роль активного учасника протоколу.

Окремо в архітектурі передбачено підсистему журналювання. Її використання є важливим для подальшої експериментальної частини, оскільки кожен отриманий пакет, час його приймання, результат перевірки, номер

повідомлення та відповідь сервера можуть фіксуватися в журналі. Це дозволяє надалі оцінити кількість успішно доставлених повідомлень, визначити повторні передачі, перевірити затримки та сформувати підсумкові показники ефективності протоколу.

У запропонованій системі передбачено двосторонній характер обміну. Це означає, що IoT-вузол не просто передає дані на сервер, а очікує відповідь про результат приймання. Такий підхід дає можливість реалізувати механізм підтвердження доставки. Якщо сервер отримав пакет і визнав його коректним, він повертає позитивне підтвердження. Якщо пакет має помилки або не відповідає структурі протоколу, сервер може повернути повідомлення про помилку. За відсутності відповіді IoT-вузол виконує повторну передачу, призначення компонентів подано в таблиці 2.1..

Таблиця 2.1 – Функціональне призначення основних компонентів системи обміну даними

Компонент системи	Основні функції	Роль у загальній архітектурі
IoT-вузол	Формування телеметрії, створення пакета, передавання повідомлення, очікування підтвердження	Джерело даних і ініціатор обміну
Мережеве середовище Wi-Fi	Передавання пакетів між пристроєм і сервером	Канал обміну даними
Сервер обміну даними	Приймання пакетів, формування відповідей, керування логікою обміну	Центральний вузол системи
Модуль обробки пакетів	Розбір структури повідомлення, перевірка службових полів, контроль коректності	Логічне ядро протоколу

Кінець таблиці 2.1

Компонент системи	Основні функції	Роль у загальній архітектурі
Модуль обробки пакетів	Розбір структури повідомлення, перевірка службових полів, контроль коректності	Логічне ядро протоколу
Підсистема журналювання	Фіксація подій обміну, часу приймання, статусів і помилок	Основа для аналізу роботи системи
Сховище даних	Збереження отриманої телеметрії та службової інформації	Накопичення результатів роботи
Інтерфейс перегляду результатів	Відображення отриманих пакетів, статусів і журналу подій	Засіб контролю роботи системи

Така логіка дозволяє перейти від простого одноразового надсилання даних до повноцінного керованого обміну. У практичному плані це означає, що протокол не обмежується передаванням телеметрії, а містить елементи контролю стану, перевірки доставки й обробки нестандартних ситуацій. Це є важливим для IoT-середовища, де канал зв’язку не завжди стабільний, а повідомлення можуть втрачатися або дублюватися. Узагальнений цикл обміну даними в системі зображено на рисунку 2.2

У межах архітектури передбачено використання ідентифікатора пристрою. Це дозволяє серверу розрізняти повідомлення від різних вузлів і створює основу для масштабування системи. Навіть якщо на етапі практичної реалізації використовується один пристрій, структура протоколу вже підтримує роботу з

кількома джерелами даних. У майбутньому це дозволяє підключити додаткові вузли без зміни загальної логіки обміну.

Рисунок 2.2 – Узагальнений цикл обміну даними в системі

Система також орієнтована на можливість подальшого розширення. За потреби між IoT-вузлами та сервером може бути додано проміжний шлюз, який виконуватиме функції агрегації, первинної фільтрації або перетворення форматів повідомлень. Проте в базовій архітектурі бакалаврської роботи окремий gateway не є обов’язковим компонентом, оскільки його наявність ускладнює реалізацію та не є критичною для перевірки власного протоколу.

Загальна логіка роботи системи починається з ініціалізації IoT-вузла. Після запуску пристрій підключається до мережі, формує службові параметри та готує перше повідомлення. Далі створюється пакет, у якому містяться службові поля та корисні дані. Пакет передається на сервер, після чого пристрій переходить у

стан очікування відповіді. Сервер приймає пакет, перевіряє його структуру, фіксує результат у журналі та надсилає відповідь. Після отримання підтвердження пристрій переходить до наступного циклу роботи.

2.2 Обґрунтування вибору технологій та середовища реалізації

Для реалізації системи обміну даними між пристроями Інтернету речей обрано такий набір технологій, який дозволяє поєднати простоту практичного впровадження, достатню технічну переконливість і відповідність темі бакалаврської кваліфікаційної роботи. Основне завдання цього підрозділу полягає не лише у переліченні використаних засобів, а й у поясненні, чому саме ці технології найкраще підходять для створення прототипу власного протоколу обміну даними.

Під час вибору технологій враховано, що розроблювана система повинна працювати з пристроями обмеженого класу, передавати невеликі пакети даних, підтримувати підтвердження доставки, повторну передачу, контроль цілісності та журналювання подій.

Через це не обрано готовий протокол типу MQTT або HTTP як основний механізм обміну, оскільки метою роботи є саме розроблення власного прикладного протоколу. Наявні технології використовуються як нижчі рівні інфраструктури: апаратна платформа, бездротовий канал, транспортний механізм, серверне середовище та засоби збереження результатів.

У ролі апаратної платформи для IoT-вузла обрано мікроконтролерну платформу ESP32. Такий вибір є доцільним, оскільки ESP32 має вбудований Wi-Fi-модуль, достатній обсяг оперативної та flash-пам’яті, підтримує роботу з TCP/UDP-з’єднаннями, має широку програмну екосистему та добре підходить для створення навчальних і прикладних IoT-прототипів. Для цієї бакалаврської кваліфікаційної роботи важливо, що ESP32 дозволяє реалізувати обмін даними

без додаткового зовнішнього мережевого модуля, а це спрощує апаратну частину й робить систему більш зрозумілою.

Вибір UDP дозволяє сформувати власний прикладний рівень, у якому визначено структуру пакета, службові поля, типи повідомлень, правила підтвердження та повторної передачі.

У такій моделі UDP виконує лише роль простого механізму доставки датаграм, а вся логіка керованого обміну реалізується в межах розроблюваного протоколу. Це добре відповідає темі роботи, оскільки протокол не просто використовує готову інфраструктуру, а має власну поведінку й власні правила взаємодії.

Для програмування ESP32 обрано середовище PlatformIO або Arduino IDE. Найбільш зручним варіантом для реалізації прототипу є PlatformIO у складі Visual Studio Code, оскільки воно забезпечує зручну структуру проєкту, контроль бібліотек, налаштування плати, компіляцію, завантаження прошивки та моніторинг серійного порту.

Arduino IDE також може використовуватися як спрощене середовище, однак для бакалаврської роботи PlatformIO виглядає більш охайним і технічно зрілим рішенням. Обґрунтування вибору транспортного рівня для системи обміну даними подано у таблиці 2.2.

Таблиця 2.2 – Обґрунтування вибору транспортного рівня для системи обміну даними

Критерій	TCP	UDP	Обране рішення
Надійність доставки	Забезпечується транспортним рівнем	Потребує реалізації на прикладному рівні	UDP, оскільки надійність реалізується власним протоколом
Службові витрати	Вищі через встановлення та підтримку з’єднання	Нижчі, оскільки з’єднання не встановлюється	UDP, оскільки важлива компактність обміну

Кінець таблиці 2.2

Критерій	TCP	UDP	Обране рішення
Контроль порядку повідомлень	Виконується автоматично	Реалізується окремо	UDP, оскільки порядок пакетів контролюється через sequence number
Придатність для власного протоколу	Частина логіки прихована в TCP	Більше можливостей для реалізації власних механізмів	UDP, оскільки краще демонструє роботу запропонованого протоколу

На стороні ESP32 доцільно використовувати бібліотеки WiFi.h і WiFiUdp.h. Перша відповідає за підключення до бездротової мережі, а друга забезпечує передавання UDP-пакетів. Формування власного пакета виконується без використання важких форматів на кшталт JSON, оскільки в роботі передбачено компактну структуру повідомлення. Для цього дані можуть записуватися в масив байтів, де кожне поле має визначене зміщення і розмір. Такий підхід краще відповідає ідеї легковагового протоколу.

У межах прототипу формат повідомлення може реалізовуватися як бінарний пакет. Це означає, що службові поля не передаються у вигляді тексту, а кодуються у компактному байтовому представленні. Наприклад, версія протоколу може займати 1 байт, тип повідомлення - 1 байт, ідентифікатор пристрою - 2 байти, номер повідомлення - 2 або 4 байти, довжина корисного навантаження - 1 або 2 байти. Така структура значно компактніша за JSON і краще підходить для мікроконтролерної системи.

Серверну частину обрано реалізувати мовою Python. Такий вибір пояснюється тим, що Python має прості засоби роботи з мережевими сокетами, файлами журналу, базами даних і часовими мітками. Для реалізації UDP-сервера достатньо стандартного модуля socket, який дозволяє приймати датаграми, аналізувати вміст пакета та надсилати відповідь назад на адресу пристрою. Це

суттєво спрощує створення робочого прототипу, оскільки не потребує складних фреймворків або додаткової серверної інфраструктури.

У серверній частині також можуть використовуватися стандартні модулі Python struct, time, logging, sqlite3, hashlib і hmac. Модуль struct дозволяє розбирати бінарні пакети відповідно до заданого формату, time забезпечує фіксацію моменту отримання повідомлення, logging використовується для журналювання подій, sqlite3 дає змогу зберігати результати в локальній базі даних, а hashlib і hmac можуть застосовуватися для перевірки цілісності або автентичності повідомлення. Такий набір засобів є достатнім для реалізації прототипу без перевантаження системи зайвими залежностями.

У ролі сховища даних обрано SQLite. Це локальна файлова база даних, яка не потребує окремого серверного процесу, легко підключається до Python і добре підходить для невеликого прототипу.

У межах бакалаврської роботи SQLite дозволяє зберігати отримані пакети, час їх приймання, ідентифікатор пристрою, тип повідомлення, номер пакета, значення корисного навантаження та результат перевірки. За рахунок цього створюється основа для подальшої експериментальної частини, де потрібно оцінювати кількість прийнятих повідомлень, повторні передачі, помилки та затримки.

На рівні прикладного протоколу обрано бінарне кодування повідомлень. Це рішення є більш придатним для IoT-вузлів, ніж текстові формати, оскільки дозволяє скоротити розмір пакета, зменшити обсяг переданих даних і спростити обробку на стороні мікроконтролера. JSON є зручним для налагодження та читання людиною, але кожне поле в ньому передається у вигляді тексту, що збільшує розмір повідомлення. Для системи, де передаються невеликі значення телеметрії, таке збільшення є небажаним. Обрані технології та їх призначення в системі показано в таблиці 2.3.

Таблиця 2.3 – Обрані технології та їх призначення в системі

Технологія або засіб	Призначення	Причина вибору
ESP32 DevKit	Реалізація IoT-вузла	Наявність Wi-Fi, достатні ресурси, доступність і простота програмування
Wi-Fi	Передавання даних у локальній мережі	Підтримується ESP32 і не потребує додаткових модулів
UDP	Транспортний рівень обміну	Низькі службові витрати та можливість реалізувати власну логіку надійності
Власний бінарний формат пакета	Прикладний рівень протоколу	Компактність, швидка обробка, контроль структури повідомлення
PlatformIO / Arduino IDE	Середовище розробки прошивки ESP32	Зручне програмування, завантаження коду та налагодження
Python	Реалізація серверної частини	Простота створення UDP-сервера та обробки пакетів
SQLite	Збереження прийнятих даних	Локальне сховище без потреби в окремому сервері бази даних
Журнал подій	Фіксація обміну та помилок	Основа для аналізу роботи протоколу
CRC або HMAC	Контроль цілісності пакета	Перевірка коректності отриманого повідомлення

Бінарний формат краще узгоджується з метою створення легковагового протоколу. У ньому кожне поле має чітко визначений розмір, а сервер може швидко розібрати пакет за допомогою наперед заданої структури. Наприклад, якщо пакет має фіксований заголовок і змінне поле корисного навантаження, сервер спочатку читає службову частину, визначає довжину даних і лише потім

обробляє payload. Це знижує ймовірність неоднозначного трактування повідомлення та полегшує перевірку коректності. Структуру програмно-технічного середовища реалізації подано на рисунку 2.3.

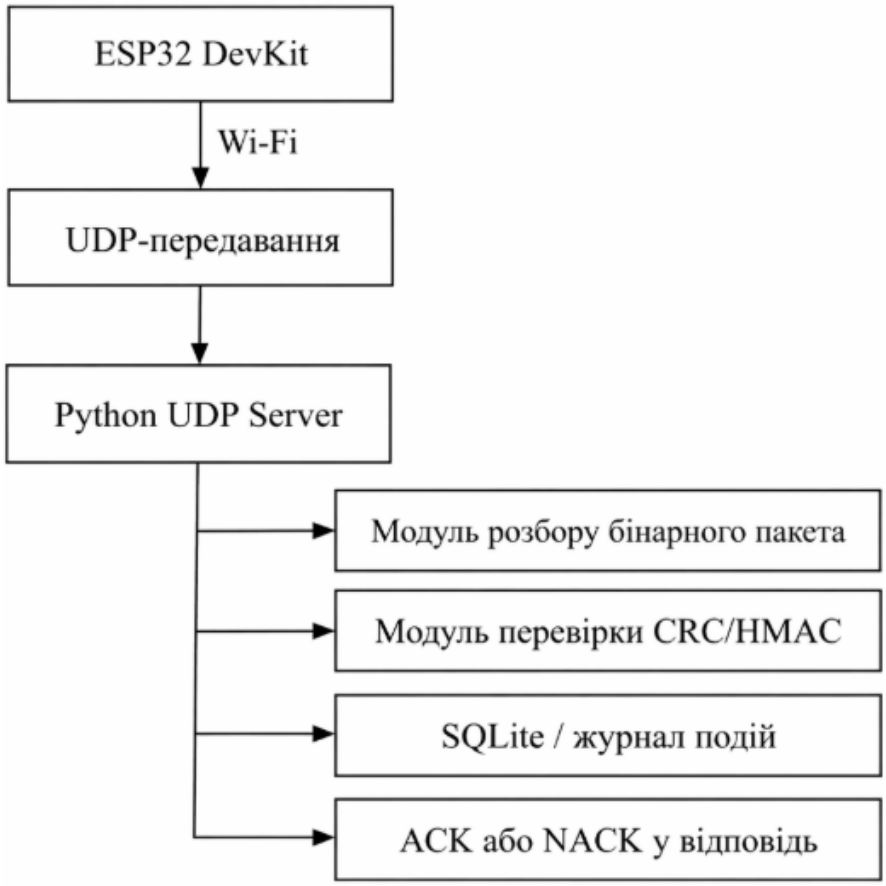


Рисунок 2.3 – Структура програмно-технічного середовища реалізації

Для контролю цілісності пакета на базовому рівні може застосовуватися CRC-16 або CRC-32. Такий механізм дозволяє виявляти випадкові помилки передачі, пошкодження даних або некоректне формування пакета. CRC не забезпечує криптографічного захисту, однак добре підходить для первинної перевірки цілісності в легковагових системах. У більш захищеному варіанті доцільно застосувати HMAC на основі секретного ключа, що дозволить не лише перевіряти цілісність, а й частково підтверджувати автентичність джерела повідомлення.

Для базової реалізації протоколу достатньо поєднання CRC і службових полів пакета. У такому варіанті сервер може перевірити, чи не пошкоджено

повідомлення під час передачі, чи відповідає довжина пакета заявленому значенню, чи не дублюється порядковий номер. Для розширеної реалізації може бути доданий HMAC, який розраховується на основі заголовка, корисного навантаження та секретного ключа. Це створює основу для захисту від підміни повідомлень.

Вибір середовища розробки також пов’язаний із потребою у зручному налагодженні. На стороні ESP32 важливо мати можливість переглядати службові повідомлення через серійний порт. Це дозволяє контролювати підключення до Wi-Fi, формування пакета, номер повідомлення, факт відправлення, очікування відповіді та результат отримання ACK або NACK. На стороні сервера зручно виводити журнал подій у консоль і паралельно записувати його у файл або базу даних.

Практична реалізація системи не потребує складної хмарної інфраструктури. Сервер може працювати локально на комп’ютері, що перебуває в одній мережі з ESP32. Такий підхід є достатнім для перевірки роботи протоколу, вимірювання затримок, аналізу кількості пакетів і відпрацювання повторної передачі. При цьому архітектура не закриває можливість майбутнього перенесення серверної частини на

Для вимірювання затримки обміну можна використовувати часові мітки на сервері та на пристрої. У найпростішому варіанті ESP32 фіксує момент відправлення пакета, очікує ACK і обчислює час між відправленням та отриманням відповіді. На сервері додатково фіксується час приймання повідомлення, що дозволяє сформувати журнал для подальшого аналізу. Це не потребує складних вимірювальних засобів, але дає достатню основу для експериментальної частини.

З погляду програмної структури реалізація поділяється на кілька модулів. На стороні ESP32 виділяються модуль підключення до Wi-Fi, модуль формування пакета, модуль передачі, модуль очікування відповіді та модуль повторної передачі. На стороні сервера виділяються модуль приймання UDP-

пакетів, модуль розбору структури, модуль перевірки цілісності, модуль журналювання, модуль збереження даних і модуль формування відповіді. Такий розподіл робить систему зрозумілою та зручною для опису в наступних підрозділах.

У підсумку для реалізації системи обміну даними обрано мікроконтролерну платформу ESP32, бездротове середовище Wi-Fi, транспорт UDP, власний бінарний формат повідомлень, серверну частину на Python і локальне сховище SQLite. Такий набір технологій забезпечує достатню простоту практичної реалізації, дозволяє продемонструвати власну логіку протоколу та створює основу для подальшого аналізу ефективності. Обрані засоби не перевантажують систему зайвими компонентами, але забезпечують усі необхідні можливості для формування, передавання, перевірки, збереження та підтвердження повідомлень.

2.3 Розроблення структури протоколу та формату повідомлень

Після визначення загальної архітектури системи та вибору технологічного середовища реалізації сформовано структуру прикладного протоколу обміну даними між IoT-вузлом і серверною частиною. Розроблюваний протокол призначено для передавання коротких телеметричних повідомлень у локальній мережі через UDP, але вся логіка керованого обміну реалізується не засобами транспортного рівня, а на рівні власного прикладного формату. Це дозволяє показати повний цикл роботи протоколу: формування пакета, перевірку службових полів, контроль цілісності, підтвердження доставки, повторне передавання та обробку помилок.

Початкове поле Start marker використовується як проста ознака початку пакета. У межах реалізації прийнято значення 0xA5, яке дозволяє швидко перевірити, чи отримане повідомлення справді відповідає формату LDP-IoT. Якщо перший байт не збігається з очікуваним значенням, сервер може одразу

відхилити пакет без подальшого розбору. Це зменшує кількість зайвих операцій у випадку отримання некоректних або випадкових даних.

Поле Protocol version визначає версію протоколу. У базовому варіанті використовується версія 0x01. Наявність цього поля є важливою, оскільки в майбутньому структура пакета може бути розширена або змінена. Завдяки версії сервер зможе розрізняти повідомлення різних форматів і коректно обробляти їх відповідно до підтримуваної реалізації.

Поле Message type визначає призначення конкретного повідомлення. Через нього сервер або IoT-вузол розуміє, чи є пакет службовим повідомленням ініціалізації, телеметричними даними, підтвердженням отримання, повідомленням про помилку або командою конфігурації. Такий підхід дозволяє не створювати окремі формати для кожного сценарію, а використовувати спільну структуру пакета з різними типами повідомлень.

Для кодування числових значень у пакеті використовується порядок байтів big-endian, тобто старший байт записується першим. Це рішення робить формат більш однозначним і зручним для обробки між різними платформами.

Усі поля мають фіксований розмір, тому під час формування пакета ESP32 записує їх у буфер послідовно, а сервер читає відповідні ділянки буфера згідно з наперед визначеною структурою.

Після заголовка розміщується корисне навантаження. Для нього обрано компактний формат TLV, де кожне значення описується трьома складовими: типом параметра, довжиною та самим значенням.

Такий підхід є зручним для IoT-систем, оскільки дозволяє передавати різні набори параметрів у межах одного пакета. Наприклад, один пакет може містити лише температуру, інший - температуру й вологість, третій - рівень заряду, стан вузла або службовий код.

Формат TLV дозволяє зберегти гнучкість без переходу до важчих текстових форматів, структура подана у таблиці 2.4. На відміну від JSON, тут не потрібно передавати назви полів у вигляді рядків. Кожен параметр має короткий

числовий тег, а сервер уже знає, як інтерпретувати відповідне значення. Це зменшує розмір повідомлення та спрощує його обробку на мікроконтролері.

Таблиця 2.4 – Структура TLV-поля корисного навантаження

Елемент TLV	Розмір	Призначення	Приклад
Type	1 байт	Код параметра	0x01
Length	1 байт	Довжина значення	0x02
Value	змінний	Значення параметра	0x09 0x2E

Для телеметричних повідомлень передбачено набір базових типів параметрів. Температура може передаватися як ціле число у сотих частках градуса, вологість - у сотих частках відсотка, напруга живлення - у мілівольтах, стан пристрою - одним байтом.

Повний пакет телеметрії складається з фіксованого заголовка, одного або кількох TLV-полів і контрольного значення, базові типи параметрів у корисному навантаженні зображено у таблиці 2.5. Якщо IoT-вузол передає температуру та вологість, корисне навантаження може містити два TLV-блоки. Перший блок описує температуру, другий - вологість. Заголовок при цьому залишається незмінним, а поле Payload length вказує сумарний розмір усіх TLV-блоків.

Таблиця 2.5 – Базові типи параметрів у корисному навантаженні

Код параметра	Назва параметра	Формат значення	Приклад інтерпретації
0x01	Температура	int16, значення × 100	2350 = 23,50 °C
0x02	Вологість	uint16, значення × 100	4820 = 48,20 %
0x03	Напруга живлення	uint16, мВ	3700 = 3,7 В
0x04	Рівень сигналу	int8, dBm	-62 dBm

Кінець таблиці 2.5

Код параметра	Назва параметра	Формат значення	Приклад інтерпретації
0x05	Стан пристрою	uint8	1 = активний
0x06	Код помилки	uint8	2 = помилка CRC
0x07	Лічильник циклів	uint32	150 циклів

У протоколі передбачено кілька типів повідомлень. Це потрібно для того, щоб система могла виконувати не лише передавання даних, а й службову взаємодію між пристроєм і сервером. Наприклад, перед початком регулярного обміну IoT-вузол може надіслати повідомлення HELLO, за допомогою якого повідомляє сервер про свою присутність. Далі основний обмін здійснюється через DATA-повідомлення. Після успішного приймання сервер повертає ACK, а у випадку помилки - NACK.

Повідомлення HELLO використовується на початку роботи пристрою. Воно дозволяє серверу зафіксувати факт появи вузла в мережі та зберегти його ідентифікатор. У корисному навантаженні такого повідомлення можуть передаватися тип пристрою, версія прошивки або службовий статус. У спрощеній реалізації HELLO може містити лише ідентифікатор вузла, оскільки основні поля вже розташовано в заголовку.

У розширеному режимі замість CRC або разом із ним може використовуватися HMAC. На відміну від CRC, HMAC дозволяє не лише перевірити цілісність, а й підтвердити, що повідомлення сформовано пристроєм, який має спільний секретний ключ. Для базового прототипу достатньо CRC, оскільки головним завданням підрозділу є опис структури протоколу. Проте в архітектурі вже закладено можливість переходу до захищеного режиму через прапор HMAC_USED.

Для базового пакета прийнято таку загальну формулу розміру: 16 байтів заголовка, змінна довжина корисного навантаження та 4 байти CRC-32. Якщо

payload становить 8 байтів, повний розмір пакета дорівнює 28 байтам. Це значно менше, ніж у разі використання текстового формату з назвами полів, лапками, розділювачами та іншими службовими символами. Саме компактність є однією з головних переваг запропонованої структури.

У наведеному прикладі корисне навантаження має 8 байтів, оскільки кожен TLV-блок складається з 1 байта типу, 1 байта довжини та 2 байтів значення. Температура 23,50 °C передається як число 2350, а вологість 48,20 % - як число 4820. Такий формат дозволяє уникнути рядкових значень і зменшити обсяг пакета. Призначення службових прапорів у полі Flags подано у таблиці 2.6.

Таблиця 2.6 – Призначення службових прапорів у полі Flags

Біт	Маска	Назва прапора	Призначення
0	0x01	ACK_REQUIRED	Пакет потребує підтвердження
1	0x02	RETRANSMIT	Пакет надіслано повторно
2	0x04	CRC_USED	У пакеті використано CRC
3	0x08	HMAC_USED	У пакеті використано HMAC
4	0x10	ENCRYPTED	Корисне навантаження зашифровано
5	0x20	CONFIG_ALLOWED	Пристрій приймає конфігураційні команди
6	0x40	RESERVED	Зарезервовано для розширення
7	0x80	RESERVED	Зарезервовано для розширення

Для повідомлень ACK і NACK використовується така сама базова структура заголовка, однак корисне навантаження має інше призначення. В ACK-повідомленні сервер передає номер пакета, який успішно прийнято. У NACK-повідомленні додатково передається код помилки. Це дозволяє IoT-вузлу розуміти, чи потрібно повторити повідомлення, змінити параметри обміну або перейти в режим помилки.

У випадку дублювання пакета сервер може діяти у двох режимах. Якщо дублікат уже успішно оброблено раніше, сервер повторно надсилає ACK, але не записує дані вдруге. Такий підхід запобігає появі повторних значень у сховищі й одночасно дозволяє IoT-вузлу завершити цикл передавання. Якщо ж дублікат містить пошкоджені дані або не збігається з попередньо прийнятим пакетом, сервер може повернути NACK із відповідним кодом помилки Послідовність формування та перевірки пакета LDP-IoT зображено на рисунку 2.4.

Структура протоколу також передбачає контроль максимального розміру пакета. Для базової реалізації доцільно встановити обмеження повного пакета на рівні 256 байтів.

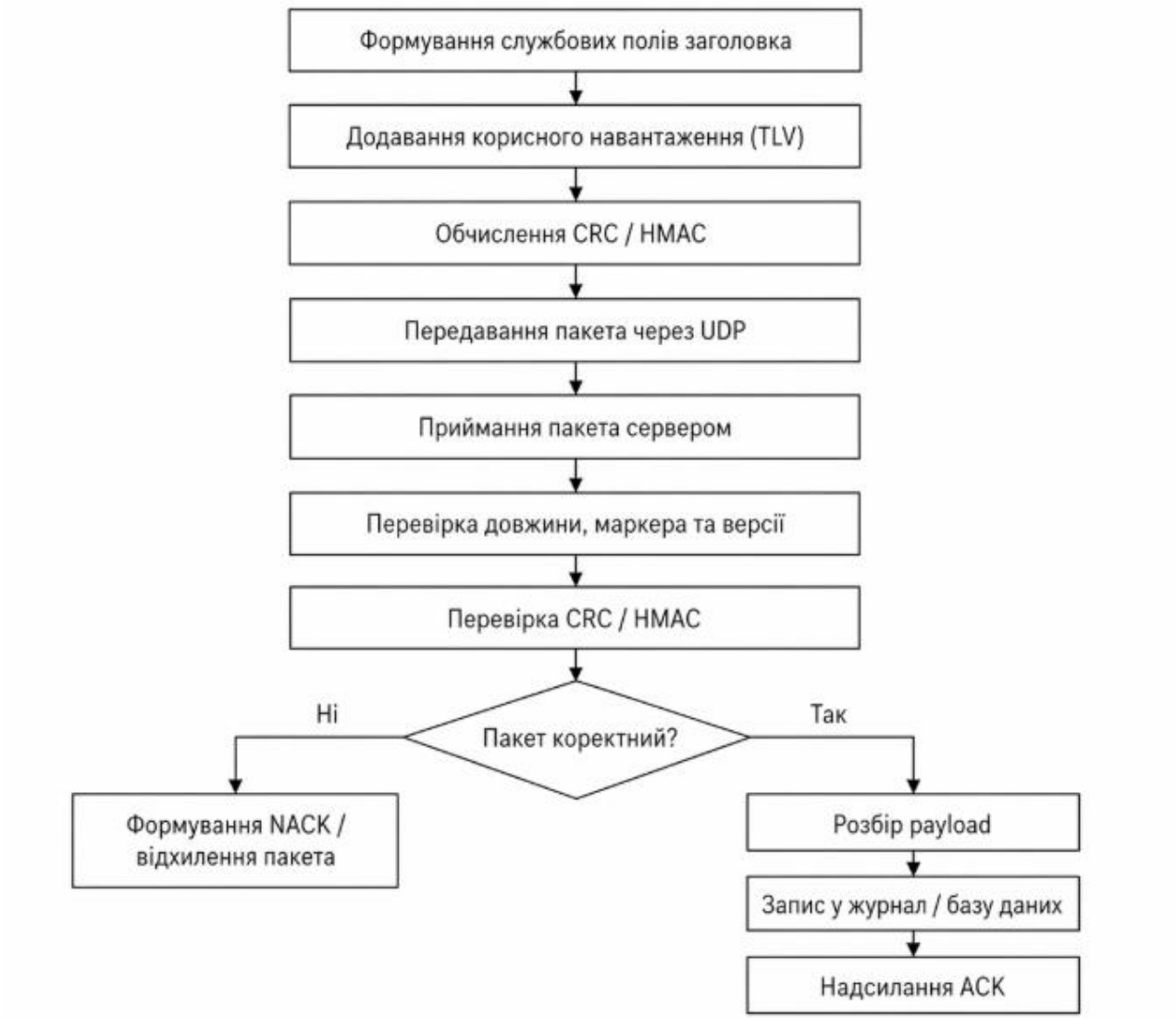


Рисунок 2.4 – Послідовність формування та перевірки пакета LDP-IoT

Протокол також підтримує розділення службової інформації та корисних даних. Заголовок завжди залишається відкритим для обробки сервером, оскільки саме він потрібен для маршрутизації та перевірки пакета. Корисне навантаження за потреби може бути зашифроване. У такому випадку прапор ENCRYPTED вказує серверу, що payload необхідно розшифрувати перед інтерпретацією TLV-полів. Це рішення дозволяє поєднати керованість протоколу з можливістю захисту переданих даних.

Загальна структура протоколу LDP-IoT сформована так, щоб забезпечити баланс між компактністю, простотою реалізації та функціональністю. Фіксований заголовок полегшує розбір пакета, TLV-формат робить payload гнучким, поле Sequence number забезпечує контроль порядку й повторів, Flags дозволяє керувати режимами обміну, а контрольна частина дає можливість виявляти пошкодження або підміну повідомлень.

У підсумку в межах підрозділу розроблено структуру прикладного протоколу LDP-IoT, визначено формат заголовка, корисного навантаження, контрольного поля, типи повідомлень, службові прапори та коди помилок. Запропонований формат є достатньо компактним для IoT-вузлів і водночас містить усі необхідні механізми для подальшої реалізації надійного обміну між пристроєм і сервером. Така структура створює основу для наступного підрозділу, у якому розглядається алгоритм роботи протоколу під час реального циклу передавання даних.

2.4 Розроблення алгоритму обміну даними

Після визначення архітектури системи, вибору технологій реалізації та формування структури протоколу LDP-IoT розроблено алгоритм обміну даними між IoT-вузлом і серверною частиною. Саме алгоритм визначає порядок дій пристрою та сервера під час передавання повідомлень, обробки відповідей, перевірки коректності пакетів і реагування на нестандартні ситуації. Якщо

структура пакета описує, з яких полів складається повідомлення, то алгоритм обміну пояснює, як ці повідомлення використовуються в реальному циклі роботи системи.

Розроблений алгоритм побудовано з урахуванням того, що транспортним рівнем обрано UDP. Це означає, що сам транспорт не гарантує доставку пакета, не забезпечує повторну передачу та не контролює порядок отримання повідомлень. Через це відповідні механізми реалізовано на прикладному рівні протоколу LDP-IoT. До таких механізмів належать порядкова нумерація пакетів, очікування підтвердження ACK, повторна передача після тайм-ауту, обробка NACK, перевірка контрольної суми та відсікання дублікатів.

Початковим етапом роботи є ініціалізація IoT-вузла. Після подачі живлення пристрій запускає програмну логіку, налаштовує внутрішні змінні, очищує буфери передавання та приймання, встановлює початкове значення лічильника повідомлень і виконує підключення до Wi-Fi-мережі. Якщо підключення до мережі не виконано, пристрій не переходить до передавання телеметрії, а повторює спроби з’єднання через заданий інтервал часу. Це дозволяє уникнути ситуації, коли повідомлення формується, але не може бути доставлене через відсутність мережевого доступу.

Після завершення службового етапу IoT-вузол переходить до основного циклу передавання даних. У межах цього циклу пристрій отримує значення від сенсора або формує тестові дані, після чого створює DATA-пакет. До заголовка пакета записуються службові поля, зокрема маркер початку, версія протоколу, тип повідомлення, прапори, ідентифікатор пристрою, порядковий номер, довжина корисного навантаження та часова мітка. Далі формується payload у форматі TLV, після чого обчислюється контрольна сума CRC-32 або інше контрольне значення.

На серверній стороні алгоритм починається з очікування вхідного UDP-пакета. Сервер постійно прослуховує заданий порт і після отримання повідомлення передає його до модуля розбору. Першою перевіркою є контроль

мінімальної довжини пакета. Якщо пакет коротший за мінімально допустимий розмір, він відхиляється, оскільки не може містити повний заголовок і контрольне поле. Далі сервер перевіряє маркер початку пакета, версію протоколу, тип повідомлення та заявлену довжину корисного навантаження.

Після первинної перевірки сервер обчислює контрольну суму для отриманих даних і порівнює її зі значенням, яке міститься в контрольній частині пакета. Якщо значення не збігаються, пакет вважається пошкодженим. У такому випадку сервер може надіслати NACK із відповідним кодом помилки або просто не підтверджувати приймання, залежно від обраного режиму реалізації. У межах цієї роботи більш доцільним є варіант із NACK, оскільки він дає пристрою змогу швидше зрозуміти причину відмови.

Окремо в алгоритмі передбачено обробку ситуації втрати відповіді. Така ситуація є типовою для UDP-обміну. Наприклад, сервер може успішно отримати DATA-пакет, перевірити його, записати до бази та надіслати ACK, але сам ACK може не дійти до пристрою. З погляду IoT-вузла це виглядає як невдала передача, тому після тайм-ауту він надсилає той самий пакет повторно. Сервер, отримавши дублікат, не дублює запис, а повторює ACK. Це дозволяє забезпечити узгодженість стану між пристроєм і сервером.

Важливою частиною алгоритму є контроль дублювання. Без такого механізму кожна повторна передача могла б створювати новий запис у базі даних, що призвело б до спотворення результатів. У запропонованому алгоритмі унікальність повідомлення визначається парою Device ID і Sequence number. Якщо така пара вже існує серед прийнятих повідомлень, пакет розглядається як повторний. Завдяки цьому повторне передавання не створює дублювання даних, але не блокує завершення циклу на стороні пристрою. Блок-схема алгоритму обміну даними за протоколом LDP-IoT зображено на рисунку 2.5.

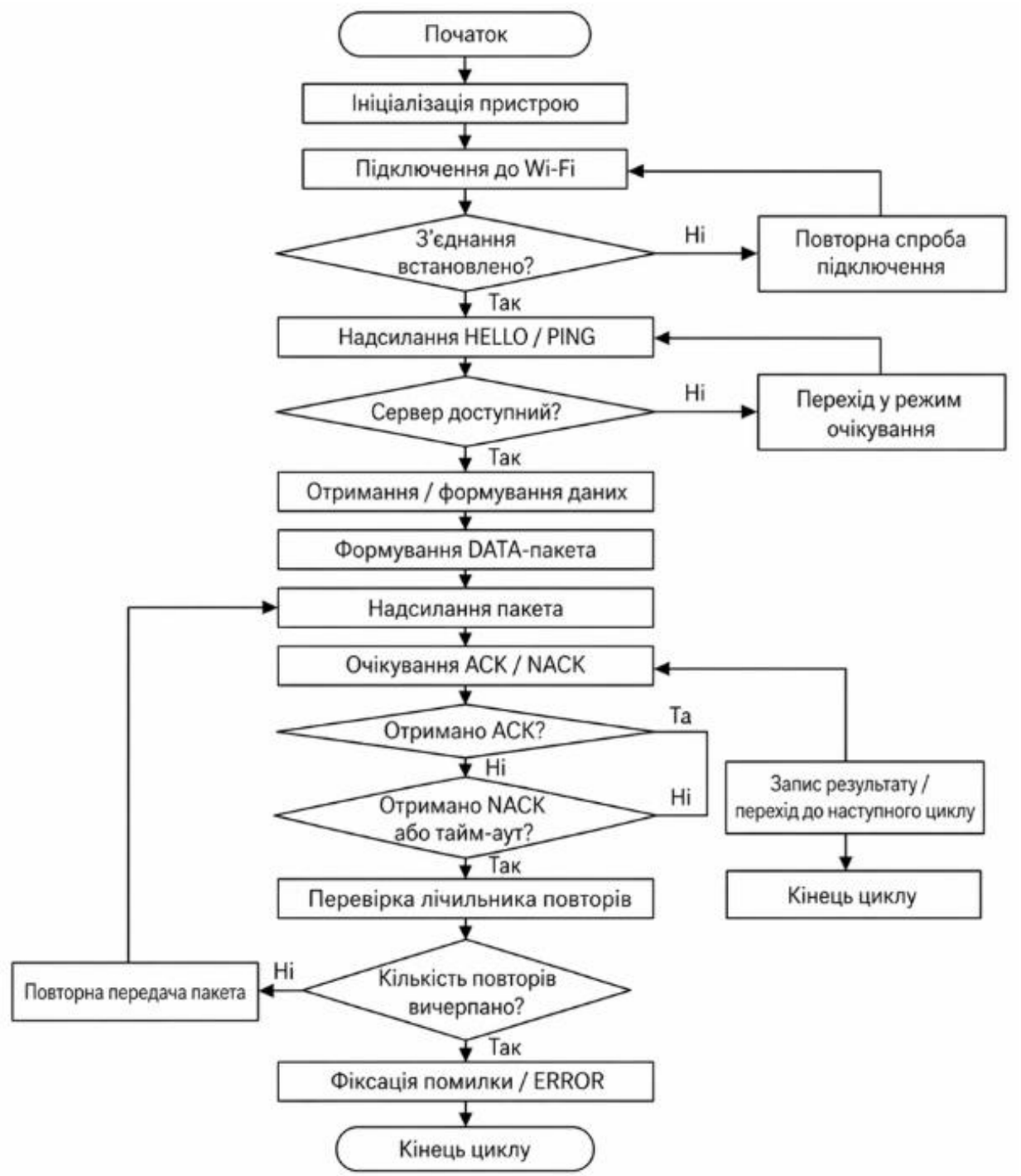


Рисунок 2.5 – Блок-схема алгоритму обміну даними за протоколом LDP-IoT

Ще одним елементом алгоритму є контроль порядку повідомлень. У базовому варіанті сервер не обов'язково відхиляє пакет, якщо його номер не є строго наступним, оскільки в UDP-мережах можлива зміна порядку доставки. Проте сервер фіксує таку ситуацію в журналі. Для спрощеної реалізації прийнято підхід, за якого всі коректні пакети з новими номерами приймаються, але

порушення очікуваної послідовності позначається як службова подія. Це дозволяє не втрачати дані, але водночас зберігати інформацію про якість обміну.

У процесі роботи алгоритму враховано також можливість тимчасової недоступності сервера. Якщо IoT-вузол не отримує відповіді після заданої кількості спроб, він не продовжує безперервне передавання. Замість цього пристрій може перейти в режим очікування перед наступною спробою. Такий підхід зменшує навантаження на мережу та дозволяє уникнути зайвого енергоспоживання. Після паузи пристрій повторно перевіряє доступність сервера через HELLO або PING.

У межах алгоритму передавання даних важливо зберегти баланс між надійністю та простотою. Надмірно складна логіка з великою кількістю станів, підтверджень і службових перевірок могла б перевантажити IoT-вузол. З іншого боку, повна відсутність контролю доставки зробила б систему ненадійною. Тому в запропонованому варіанті використано помірну кількість механізмів: один порядковий номер, один тайм-аут, обмежену кількість повторів, ACK/NACK і перевірку контрольної суми. Цього достатньо для базового прототипу та подальшого експериментального аналізу.

На стороні IoT-вузла основний цикл роботи має повторюваний характер. Після успішного передавання одного пакета пристрій очікує заданий інтервал, формує наступне повідомлення та повторює процедуру. Якщо пристрій працює від автономного живлення, між циклами він може переходити у режим зниженого енергоспоживання. У цьому підрозділі енергоощадні стани не розглядаються детально, оскільки модель станів пристрою винесено в окремий підрозділ, але алгоритм обміну вже враховує можливість пауз між активними циклами.

На серверній стороні алгоритм є безперервним. Сервер не завершує роботу після обробки одного пакета, а повертається до очікування наступного повідомлення. Це дозволяє приймати пакети від кількох IoT-вузлів. Якщо повідомлення надходять від різних пристроїв, сервер обробляє їх за спільними

правилами, але веде окремий облік Device ID і Sequence number. Така логіка робить протокол придатним для масштабування без зміни базового алгоритму.

2.5 Модель станів пристрою та оптимізація роботи

Для коректної роботи протоколу LDP-IoT важливим є не лише формат повідомлення та алгоритм обміну, а й поведінка самого IoT-вузла в різних умовах функціонування. Пристрій не може постійно перебувати в одному режимі, оскільки на практиці він проходить кілька етапів: запуск, підключення до мережі, підготовку пакета, передавання, очікування відповіді, обробку результату, повторну передачу або перехід у режим очікування. Через це для системи сформовано модель станів, яка описує логіку переходів пристрою між основними режимами роботи.

Модель станів IoT-вузла дозволяє зробити роботу пристрою більш передбачуваною. Кожен стан має чітке призначення, а перехід до наступного стану виконується лише після настання певної події. Наприклад, після запуску пристрій не може одразу передавати дані, доки не встановлено з’єднання з Wi-Fi-мережею. Після передавання пакета пристрій не повинен одразу формувати нове повідомлення, оскільки спочатку необхідно отримати ACK або визначити факт тайм-ауту. Такий підхід зменшує кількість випадкових помилок у програмній логіці та спрощує налагодження системи.

У межах розробленого протоколу поведінку IoT-вузла подано як скінченний автомат. Це означає, що пристрій у кожен момент часу перебуває лише в одному визначеному стані, а всі переходи між станами відбуваються за заздалегідь заданими правилами. Такий підхід добре підходить для мікроконтролерних систем, оскільки дозволяє уникнути хаотичного виконання дій і забезпечує просту, але надійну структуру керування.

Якщо відповідь від сервера не надходить у межах встановленого тайм-ауту, пристрій переходить до стану RETRY. У цьому стані перевіряється

кількість уже виконаних повторних передач. Якщо вона менша за допустиму межу, пристрій встановлює прапор RETRANSMIT і повторно надсилає той самий пакет із тим самим Sequence number. Це принципово важливо, оскільки повторна передача не повинна створювати нове повідомлення. Сервер за номером пакета зможе визначити, що це повтор, і не запише дані двічі. Якщо кількість спроб вичерпано, пристрій переходить до ERROR або SLEEP із позначенням невдалої передачі.

Стан SLEEP призначено для зниження енергоспоживання між циклами обміну. У цьому стані пристрій тимчасово припиняє активну мережеву взаємодію й очікує наступного моменту передавання. Для ESP32 цей стан може бути реалізований як програмна пауза, light sleep або deep sleep залежно від вимог до енергоефективності. У межах базового прототипу достатньо використати затримку між циклами, однак сама модель станів уже враховує можливість переходу до реальних енергоощадних режимів.

Стан ERROR використовується для обробки критичних або повторюваних помилок. До цього стану пристрій може перейти у випадку неможливості підключення до Wi-Fi, вичерпання кількості повторних передач, отримання некоректних службових відповідей або внутрішньої помилки формування пакета. У цьому стані може виконуватися запис повідомлення в серійний журнал, очищення буферів, збільшення інтервалу до наступної спроби або перезапуск окремих модулів. Після обробки помилки пристрій може повернутися до CONNECT, IDLE або SLEEP залежно від причини збою.

Розроблена модель станів дозволяє не тільки впорядкувати роботу пристрою, а й оптимізувати використання його ресурсів. Найбільше енергії IoT-вузол споживає під час активної роботи Wi-Fi-модуля, передавання пакета та очікування відповіді. Саме тому в алгоритмі передбачено, що пристрій не перебуває постійно в активному режимі, а виконує обмін короткими циклами. Після завершення передачі вузол переходить до очікування або сну, що зменшує загальне енергоспоживання.

Окремо враховано, що IoT-вузол може працювати в різних режимах інтенсивності. У звичайному режимі він періодично передає телеметрію з фіксованим інтервалом. У режимі підвищеної активності інтервал може скорочуватися, якщо значення параметрів швидко змінюються або виникає подія, яку потрібно передати негайно. У режимі економії інтервал передавання збільшується, а пристрій більшу частину часу перебуває у стані SLEEP. Такий підхід створює основу для адаптивної роботи протоколу.

Ще одним важливим аспектом є збереження простоти переходів між станами. У моделі не створено зайвих проміжних режимів, які могли б ускладнити програмну логіку. Наприклад, перевірка ACK виконується в межах WAIT_ACK, а рішення про повторну передачу - в межах RETRY. Помилки не розкидано по всьому коду, а зведено до стану ERROR. Така структура робить програму більш зрозумілою, а поведінку пристрою - більш контрольованою.

З позиції енергоефективності найбільш важливими є переходи між ACTIVE-станами та SLEEP. Активними вважаються MEASURE, PACKET_BUILD, TRANSMIT і WAIT_ACK, оскільки в них пристрій виконує обчислення, працює з мережею або очікує відповідь. Неактивним є SLEEP, у якому робота мікроконтролера та мережевого модуля може бути обмежена. Чим коротший активний цикл, тим менше енергії витрачає пристрій. Через це компактний пакет, швидке формування повідомлення та обмежене очікування ACK прямо впливають на оптимізацію роботи.

У практичному прототипі роботу моделі станів можна відобразити через серійний монітор. Під час запуску пристрій виводить повідомлення про INIT і CONNECT, після підключення - про перехід до IDLE, під час формування пакета - про PACKET_BUILD, після відправлення - про TRANSMIT, під час очікування відповіді - про WAIT_ACK. Якщо ACK отримано, фіксується успішне завершення циклу. Якщо відповідь відсутня, виводиться повідомлення про RETRY. Такий спосіб відображення є простим, але дуже корисним для демонстрації роботи системи.

2.6 Забезпечення надійності та безпеки обміну даними

Першим механізмом підвищення надійності є використання підтверджень ACK. Після надсилання DATA-пакета IoT-вузол не вважає передавання завершеним, доки не отримає від сервера позитивне підтвердження. У повідомленні ACK сервер повертає порядковий номер пакета, який успішно прийнято та перевірено. Це дає пристрою змогу точно визначити, що саме його поточне повідомлення доставлено коректно, а не сплутати відповідь із попереднім або наступним циклом обміну.

Другим механізмом є використання NACK, тобто негативного підтвердження. Якщо сервер отримує пакет, але виявляє помилку в його структурі, контрольній сумі, довжині корисного навантаження або типі повідомлення, він може повернути NACK із відповідним кодом помилки. Це дозволяє пристрою швидше реагувати на проблему, не очікуючи завершення тайм-ауту.

Важливою частиною забезпечення надійності є механізм тайм-аутів. Після відправлення пакета пристрій переходить у режим очікування відповіді. Якщо протягом заданого часу ACK або NACK не отримано, пристрій вважає передавання невдалим. У цьому випадку виконується повторна передача того самого пакета з тим самим порядковим номером. Такий підхід дозволяє врахувати ситуацію, коли втрачено сам DATA-пакет або відповідь сервера.

Кількість повторних передач обмежується. Це потрібно для того, щоб пристрій не витрачав енергію на нескінченні спроби передавання в умовах недоступності сервера або повної втрати мережевого з’єднання. У базовій реалізації доцільно встановити три спроби передавання одного пакета. Якщо після третьої спроби підтвердження не отримано, вузол фіксує помилку обміну й переходить у стан ERROR або SLEEP з подальшою повторною перевіркою доступності сервера.

Для перевірки цілісності повідомлення в базовій реалізації використовується CRC-32. Контрольне значення обчислюється на стороні IoT-вузла після формування заголовка та корисного навантаження. Отриманий результат додається в кінець пакета. Після приймання повідомлення сервер повторно обчислює CRC-32 для отриманих даних і порівнює його з переданим значенням. Якщо значення збігаються, пакет вважається неушкодженим. Якщо вони відрізняються, повідомлення відхиляється.

Механізм HMAC дає змогу перевіряти не лише цілісність, а й автентичність повідомлення. На відміну від CRC, HMAC розраховується із застосуванням секретного ключа, який відомий тільки пристрою та серверу. Якщо зловмисник змінить пакет або спробує сформувати власне повідомлення без знання ключа, сервер не зможе підтвердити коректність HMAC.

Доцільно передбачити два режими роботи протоколу: базовий і захищений. У базовому режимі використовується CRC-32, ACK/NACK, sequence number, тайм-аути та повторні передачі. Цього достатньо для демонстрації надійного обміну в лабораторних умовах. У захищеному режимі додається HMAC, який підвищує стійкість протоколу до підміни повідомлень. Такий поділ дозволяє зберегти простоту реалізації та водночас показати можливість розширення протоколу в напрямі безпеки. Механізми забезпечення надійності та безпеки в протоколі LDP-IoT подано на таблиці 2.7.

Таблиця 2.7 – Механізми забезпечення надійності та безпеки в протоколі LDP-IoT

Механізм	Призначення	Результат використання
ACK	Підтвердження успішного приймання пакета	Пристрій отримує підтвердження доставки
NACK	Повідомлення про помилку приймання	Пристрій швидше реагує на некоректний пакет

Кінець таблиці 2.7

Механізм	Призначення	Результат використання
Тайм-аут	Обмеження часу очікування відповіді	Виявлення втрати пакета або відповіді
Повторна передача	Повторне надсилання пакета після тайм-ауту	Підвищення ймовірності доставки
Sequence number	Нумерація повідомлень	Контроль порядку та виявлення дублікатів
Device ID	Ідентифікація джерела повідомлення	Розділення даних від різних вузлів
CRC-32	Перевірка технічної цілісності пакета	Виявлення пошкоджених повідомлень
HMAC	Перевірка цілісності та автентичності	Захист від підміни повідомлень
Обмеження повторів	Контроль кількості повторних спроб	Зменшення зайвого навантаження на мережу
Журналювання	Фіксація подій обміну	Можливість аналізу помилок і результатів

У запропонованому протоколі надійність і безпека не розглядаються як окремі ізольовані функції. Вони пов’язані між собою через загальну структуру пакета та алгоритм обміну. Наприклад, Sequence number одночасно використовується для підтвердження доставки, повторної передачі, відсікання дублікатів і захисту від повторного відтворення старих повідомлень.

Важливою вимогою є обмеження максимального розміру пакета. Якщо сервер отримує повідомлення, що перевищує допустимий розмір, воно відхиляється без подальшого розбору. Це дозволяє захистити систему від простих спроб перевантаження через надсилання надмірно великих UDP-пакетів. У базовій реалізації максимальний розмір пакета встановлено на рівні

256 байтів, що відповідає задачам передавання короткої телеметрії та спрощує роботу з буферами.

Окремо слід зазначити, що надійність у протоколі LDP-IoT не означає абсолютної гарантії доставки в будь-яких умовах. Якщо сервер недоступний або мережеве з’єднання повністю втрачено, пакет не може бути доставлений. Проте алгоритм дозволяє коректно виявити таку ситуацію, обмежити кількість спроб, зафіксувати помилку й не допустити неконтрольованого навантаження на пристрій. Це відповідає практичній логіці IoT-систем, де важливо не лише передати дані, а й коректно поводитися під час збоїв.

2.7 Висновки до другого розділу

У другому розділі бакалаврської кваліфікаційної роботи розроблено проєктну основу протоколу обміну даними між пристроями Інтернету речей. Сформовано загальну архітектуру системи, у якій IoT-вузол на базі ESP32 передає телеметричні дані через Wi-Fi та UDP на серверну частину, а сервер виконує приймання, перевірку, обробку й збереження пакетів. Обґрунтовано вибір технологій реалізації, зокрема ESP32, UDP, Python-сервера, SQLite та вебінтерфейсу контролю. Такий набір засобів дозволяє створити не надмірно складну, але повноцінну систему для перевірки власного прикладного протоколу.

Також у розділі розроблено структуру протоколу LDP-IoT, визначено формат бінарного пакета, службові поля заголовка, корисне навантаження у форматі TLV, контрольне поле CRC32, типи повідомлень і службові прапори. Окремо сформовано алгоритм обміну даними, модель станів IoT-вузла та механізми забезпечення надійності й базової безпеки. У результаті другий розділ створив теоретичну та проєктну основу для практичної реалізації системи, оскільки в ньому визначено не лише склад компонентів, а й логіку їхньої взаємодії.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ПЕРЕВІРКА РОБОТИ ПРОТОКОЛУ ОБМІНУ ДАНИМИ

3.1 Розроблення програмної частини IoT-вузла на базі ESP32

Програмну частину IoT-вузла реалізовано для мікроконтролерної платформи ESP32, яка в межах розробленої системи виконує роль кінцевого пристрою Інтернету речей. Цей вузол формує телеметричні дані, створює пакет відповідно до структури протоколу LDP-IoT, передає його на серверну частину через UDP та очікує підтвердження отримання. У прототипі ESP32 працює як окремий мережевий клієнт, який через Wi-Fi надсилає повідомлення на UDP endpoint сервера 127.0.0.1:9000 або на локальну IP-адресу комп'ютера в межах тієї самої мережі. Така реалізація показує не лише сам факт передавання даних, а повний цикл роботи власного протоколу з нумерацією повідомлень, службовими полями, контролем відповіді та повторною передачею у випадку помилки.

Для реалізації IoT-вузла обрано ESP32 DevKit, оскільки ця плата має вбудований Wi-Fi-модуль, достатню продуктивність для формування бінарних пакетів і підтримку UDP-обміну без використання додаткових мережевих модулів. У практичній реалізації використано робочу частоту мікроконтролера до 240 МГц, обсяг flash-пам'яті плати 4 МБ і оперативну пам'ять, достатню для роботи з невеликими буферами. Для протоколу LDP-IoT не потрібні великі обсяги пам'яті, оскільки один пакет обмежено буфером 256 байтів, а фактичний розмір робочого DATA-пакета з одним або кількома TLV-параметрами становить значно менше цього значення.

У програмі використано стандартні бібліотеки для підключення ESP32 до Wi-Fi-мережі та роботи з UDP-пакетами. На початку коду задано параметри мережі, адресу сервера, порт обміну, ідентифікатор пристрою та службові константи протоколу. Для тестового вузла прийнято DEVICE_ID = 101, що дозволяє серверу відрізняти цей пристрій від інших можливих IoT-вузлів. Версію

протоколу визначено як 0x01, маркер початку пакета - 0xA5, тип DATA-повідомлення - 0x03, ACK-повідомлення - 0x04, а NACK-повідомлення - 0x05. Такий набір значень забезпечує просту й однозначну ідентифікацію службових частин пакета.

Після запуску пристрій виконує підключення до Wi-Fi-мережі. Цей етап відповідає початковим станам INIT і CONNECT, описаним у попередньому розділі. До моменту успішного підключення вузол не формує DATA-пакети, оскільки передавання без активного мережевого з'єднання не має практичного сенсу. У програмі передбачено періодичну перевірку стану підключення з інтервалом 500 мс. Це дозволяє не перевантажувати мікроконтролер постійною перевіркою статусу, але водночас достатньо швидко визначати момент встановлення з'єднання.

Після підключення до мережі відкривається локальний UDP-порт. У прототипі для приймання відповіді від сервера може використовуватися локальний порт 4210, а серверна частина очікує пакети на порту 9000. Такий поділ дозволяє чітко розмежувати напрямки передавання: ESP32 надсилає DATA-пакети на серверний порт, а відповідь ACK або NACK повертається назад на адресу й порт відправника. Для налагодження використано серійний монітор зі швидкістю 115200 бод, у якому відображається стан підключення, номер пакета, результат передавання та факт отримання відповіді.

Тип 0x03 означає, що пакет містить телеметричні дані. Прапор 0x01 показує, що пристрій очікує підтвердження отримання. Це важливо для реалізації контрольованого обміну, оскільки після передавання такого пакета ESP32 переходить у режим очікування відповіді сервера. Якщо пакет передається повторно, у полі прапорів додатково встановлюється значення 0x02, яке позначає RETRANSMIT. У результаті сервер може розпізнати повторне повідомлення та не записувати його як новий вимір.

Передавання пакета здійснюється через UDP. ESP32 відкриває UDP-пакет, записує в нього лише фактичну довжину сформованого повідомлення та

завершує передавання. Через UDP не передається весь буфер розміром 256 байтів, а тільки використана частина, наприклад 28 або 32 байти. Це зменшує мережеве навантаження та відповідає ідеї легковагового протоколу. Максимальний розмір буфера залишено із запасом, щоб у майбутньому можна було додати нові TLV-параметри без зміни базової логіки.

Реалізована програмна частина відповідає моделі станів, сформованій у другому розділі. Після запуску пристрій проходить ініціалізацію, підключається до Wi-Fi, формує дані, створює пакет, передає його, очікує відповідь і в разі потреби виконує повторне надсилання. Якщо після трьох спроб ACK не отримано, пакет вважається недоставленим, а відповідна подія може бути виведена в серійний монітор. Така структура робить роботу вузла передбачуваною та зручною для подальшого тестування.

У програмі використано фіксований буфер розміром 256 байтів. Це дозволяє уникнути зайвого динамічного виділення пам’яті та зробити роботу стабільнішою. Навіть якщо фактичний пакет займає лише 28–32 байти, резерв буфера дає змогу додавати нові параметри. Наприклад, до температури та вологості можна додати рівень заряду, RSSI, код стану або службову помилку. При цьому загальна структура пакета не змінюється, оскільки payload уже побудовано у форматі TLV.

3.2 Розроблення серверної частини для приймання та обробки пакетів

Серверна частина розробленої системи виконує роль центрального програмного вузла, який приймає пакети від IoT-пристрою, перевіряє їхню структуру, розбирає корисне навантаження, зберігає отримані дані та формує відповідь для ESP32. Якщо програмна частина IoT-вузла відповідає за створення й передавання повідомлення, то сервер забезпечує приймання, перевірку, фіксацію результату та підтвердження доставки. Завдяки цьому система працює

не як простий канал передавання, а як повноцінний цикл обміну за протоколом LDP-IoT.

Серверну частину реалізовано мовою Python. Такий вибір пояснюється тим, що Python має зручні стандартні засоби для роботи з UDP-сокетами, бінарними структурами, часовими мітками та локальною базою даних SQLite. Для прототипу цього достатньо, оскільки не потрібно розгортати складний вебсервер або брокер повідомлень. Сервер може бути запущений на персональному комп'ютері або ноутбуці, який перебуває в одній локальній мережі з ESP32.

На початку роботи сервер створює UDP-сокет, прив'язується до визначеного порту та переходить у режим очікування вхідних пакетів. У такій схемі не встановлюється постійне з'єднання з пристроєм, оскільки UDP працює з окремими датаграмами. Це добре відповідає логіці розробленого протоколу, у

Після приймання пакета сервер виконує первинну перевірку. Спочатку контролюється мінімальна довжина повідомлення, оскільки пакет LDP-IoT повинен містити заголовок, корисне навантаження та контрольне поле. Далі розбирається службова частина: маркер початку, версія протоколу, тип повідомлення, прапори, ідентифікатор пристрою, порядковий номер і довжина payload. Якщо маркер або версія не збігаються з очікуваними значеннями, повідомлення не передається на подальшу обробку.

Цей фрагмент показує принцип розбору заголовка. Сервер читає перші 16 байтів пакета та перетворює їх у службові змінні. Завдяки цьому він може визначити, від якого пристрою надійшло повідомлення, який номер має пакет і яку довжину корисного навантаження заявлено.

Після перевірки заголовка сервер контролює цілісність повідомлення. Для цього з пакета відокремлюється контрольне поле, після чого CRC розраховується повторно для отриманих даних. Якщо обчислене значення не збігається з переданим, пакет вважається пошкодженим. У такому випадку сервер повертає NACK і не записує дані до сховища.

Перевірка CRC дозволяє виявити пошкодження пакета, помилки передавання або некоректне формування буфера на стороні ESP32. У базовій реалізації цього достатньо для контролю технічної цілісності повідомлення. Після успішної перевірки CRC сервер переходить до розбору payload.

Робота серверної частини побудована так, щоб кожен пакет проходив кілька послідовних етапів: приймання, перевірку заголовка, перевірку CRC, розбір payload, контроль дублювання, запис до сховища та формування відповіді. Така структура робить сервер не просто отримувачем UDP-повідомлень, а повноцінним учасником протоколу LDP-IoT.

Окреме значення має журналювання. Навіть якщо дані зберігаються в SQLite, у консоль також виводяться службові повідомлення про отримані пакети, помилки CRC, невідомі типи повідомлень або повторні передачі. Це полегшує налагодження й дозволяє під час перевірки бачити, як саме сервер реагує на повідомлення від ESP32.

Серверна частина не прив’язана лише до одного IoT-вузла. Завдяки полю Device ID вона може приймати пакети від кількох пристроїв. Для кожного вузла окремо відстежується порядковий номер повідомлення, що дозволяє контролювати дублікати та повторні передачі. Це створює основу для подальшого масштабування системи без зміни базової логіки протоколу.

У підсумку серверна частина забезпечує приймання та обробку пакетів за протоколом LDP-IoT, перевіряє службову структуру повідомлення, контролює CRC, розбирає корисне навантаження, запобігає дублюванню пакетів, зберігає дані в SQLite та повертає ACK або NACK. Така реалізація завершує базовий цикл обміну між ESP32 і сервером та створює основу для подальшої перевірки роботи протоколу в тестових умовах.

3.3 Реалізація формування, передавання та перевірки пакетів протоколу LDP-IoT

Після розроблення програмної частини IoT-вузла на базі ESP32 та серверного модуля приймання даних реалізовано повний цикл роботи протоколу LDP-IoT. На цьому етапі окремі програмні частини об’єднано в єдину логіку обміну, у якій IoT-вузол формує пакет, передає його через UDP, очікує відповідь від сервера, а сервер приймає повідомлення, перевіряє його структуру, контролює цілісність і повертає підтвердження. Така реалізація дозволяє показати, що запропонований протокол не є лише описовою моделлю, а може працювати як практичний механізм передавання телеметричних даних.

Основою реалізації виступає компактний бінарний пакет. На стороні ESP32 пакет формується послідовним записом байтів у заздалегідь підготовлений буфер. Такий підхід дає змогу точно контролювати розмір кожного поля та уникнути зайвого службового навантаження. На відміну від текстових форматів, бінарне подання не потребує назв полів, лапок, розділювачів або додаткового синтаксису, тому краще підходить для коротких повідомлень у системах Інтернету речей.

Формування пакета починається із запису службових полів заголовка. У ньому міститься маркер початку пакета, версія протоколу, тип повідомлення, службові прапори та ідентифікатор пристрою. Після цього додається корисне навантаження у форматі TLV. У базовому прикладі передається температура, закодована як ціле число у сотих частках градуса. Це дозволяє уникнути використання чисел із плаваючою комою та спростити подальшу обробку на сервері.

```
buffer[i++] = START_MARKER;
buffer[i++] = PROTOCOL_VERSION;
buffer[i++] = 0x03;        // DATA
buffer[i++] = 0x01;        // ACK_REQUIRED
buffer[i++] = highByte(DEVICE_ID);
buffer[i++] = lowByte(DEVICE_ID);
buffer[i++] = 0x01;        // температура
```

```
buffer[i++]   =   0x02;                          //   довжина   значення
buffer[i++] = highByte(temperature);
buffer[i++] = lowByte(temperature);
```

У наведеній частині коду показано загальний принцип формування пакета на стороні ESP32. Тип повідомлення 0x03 визначає DATA-пакет, тобто пакет із телеметричними даними. Прапор 0x01 означає, що для цього повідомлення потрібне підтвердження від сервера. Поле DEVICE_ID використовується для розпізнавання пристрою, а корисне навантаження містить параметр температури. У такому вигляді пакет залишається невеликим за розміром і зручним для обробки на мікроконтролері.

Після заповнення службових полів та корисного навантаження до пакета додається контрольне значення CRC. Воно розраховується для сформованої частини повідомлення та розміщується наприкінці пакета. Завдяки цьому сервер може перевірити, чи не змінився пакет під час передавання. Якщо контрольне значення, отримане від пристрою, не збігається з розрахованим на сервері, повідомлення вважається пошкодженим і не передається до подальшої обробки.

На серверній стороні реалізовано послідовну перевірку отриманого пакета. Спочатку сервер приймає UDP-датаграму, після чого виділяє перші байти заголовка та розбирає службові поля. Далі виконується перевірка маркера початку пакета, версії протоколу, типу повідомлення та довжини корисного навантаження. Після цього сервер відокремлює контрольне значення CRC і повторно обчислює його для отриманих даних.

```
marker, version, msg_type, flags, device_id, seq, payload_len, timestamp
= \
    struct.unpack("!BBBBHIHI", data[:16])
received_crc = struct.unpack("!I", data[-4:])[0]
calculated_crc = zlib.crc32(data[:-4]) & 0xFFFFFFFF
response = build_ack(seq) if calculated_crc == received_crc else
build_nack(seq)
```

У цій частині серверної програми показано основну перевірку пакета. Заголовок розбирається відповідно до встановленої структури, після чого контрольне значення порівнюється з повторно обчисленим CRC. Якщо перевірка є успішною, сервер формує ACK із номером прийнятого пакета. Якщо цілісність

порушена, формується NACK. Така логіка забезпечує базовий контроль коректності отриманих повідомлень.

Після успішної перевірки сервер переходить до розбору корисного навантаження. Оскільки payload має TLV-структуру, сервер послідовно читає тип параметра, довжину значення та саме значення. Якщо код параметра відповідає температурі, отримане двобайтове число перетворюється у звичний вигляд шляхом ділення на 100. Наприклад, значення 2435 інтерпретується як 24,35 °C. За таким самим принципом можуть оброблятися вологість, напруга живлення, рівень сигналу або службовий стан пристрою.

Окрему роль у реалізації відіграє порядковий номер пакета. Він потрібний для контролю повторних повідомлень і запобігання дублюванню даних у сховищі. Якщо ESP32 не отримує ACK, він повторно надсилає той самий пакет. Сервер, отримавши повідомлення з уже відомою парою ідентифікатора пристрою та порядкового номера, не записує дані повторно. Замість цього повторно надсилається ACK, щоб пристрій міг завершити цикл передавання. Такий механізм дозволяє коректно обробляти ситуацію, коли пакет на сервер дійшов, але відповідь сервера втрачена.

Після проходження перевірок отримані дані записуються до журналу або бази даних. У записі фіксується ідентифікатор пристрою, номер пакета, час приймання, значення переданого параметра та результат перевірки. Такий підхід дозволяє надалі переглядати історію обміну, аналізувати кількість успішних повідомлень, повторних передач і помилок. Це важливо для наступного етапу роботи, де оцінюється практична стабільність запропонованого протоколу.

Реалізований механізм формування та перевірки пакетів не прив’язаний до одного конкретного сенсора. У payload можуть передаватися різні параметри, якщо для них визначено код типу та формат значення. Через це структура протоколу залишається гнучкою. У базовій реалізації використано температуру як простий приклад телеметрії, але без зміни заголовка можна додати вологість, рівень заряду, напругу живлення або код стану пристрою.

Під час практичної перевірки роботу протоколу можна відстежити через повідомлення в консолі. На стороні ESP32 відображається факт формування пакета, його номер, передавання та результат очікування відповіді. На стороні сервера виводиться інформація про прийнятий пакет, ідентифікатор пристрою, порядковий номер, результат перевірки CRC і тип сформованої відповіді.

У підсумку реалізовано повний механізм формування, передавання та перевірки пакетів протоколу LDP-IoT. IoT-вузол створює компактне бінарне повідомлення, додає службові поля, корисне навантаження та контрольне значення, після чого передає пакет через UDP. Сервер приймає повідомлення, перевіряє його структуру, контролює CRC, обробляє payload, запобігає дублюванню даних і повертає ACK або NACK. Така реалізація підтверджує практичну працездатність запропонованого протоколу та створює основу для подальшого тестування роботи системи в контрольованих умовах.

3.4 Тестування роботи протоколу та аналіз результатів обміну даними

Тестування роботи протоколу LDP-IoT виконано після реалізації програмної частини IoT-вузла, серверного модуля приймання UDP-пакетів і вебінтерфейсу для контролю обміну. Основна мета перевірки полягала в тому, щоб підтвердити працездатність розробленого протоколу в умовах локального запуску, простежити проходження пакетів від вузла до сервера, перевірити коректність обробки CRC, роботу підтверджень ACK/NACK, фіксацію повторних передач і відображення результатів у панелі контролю.

У межах тестування використано локальне серверне середовище, у якому вебінтерфейс запущено за адресою 127.0.0.1:5000, а UDP-приймання пакетів організовано через endpoint 127.0.0.1:9000. Така конфігурація дала змогу перевірити роботу протоколу без підключення до віддаленого сервера, але зі збереженням повної логіки обміну. IoT-вузол або програмний симулятор формує

пакети LDP-IoT, надсилає їх на UDP-порт сервера, після чого сервер перевіряє структуру пакета, контрольну суму, тип повідомлення та формує відповідь.

На першому етапі перевірено загальний вигляд панелі контролю до початку активного обміну. У цьому стані інтерфейс відображає основні блоки системи: область останнього прийнятого пакета, графік активності обміну, структуру бінарного пакета, журнал прийнятих пакетів і події сервера. Таке подання дозволяє одразу побачити, що вебпанель підготовлена до приймання даних, а серверний модуль очікує пакети від IoT-вузла. На рисунку 3.1 зображено стан інтерфейсу, у якому пакети ще не отримано.

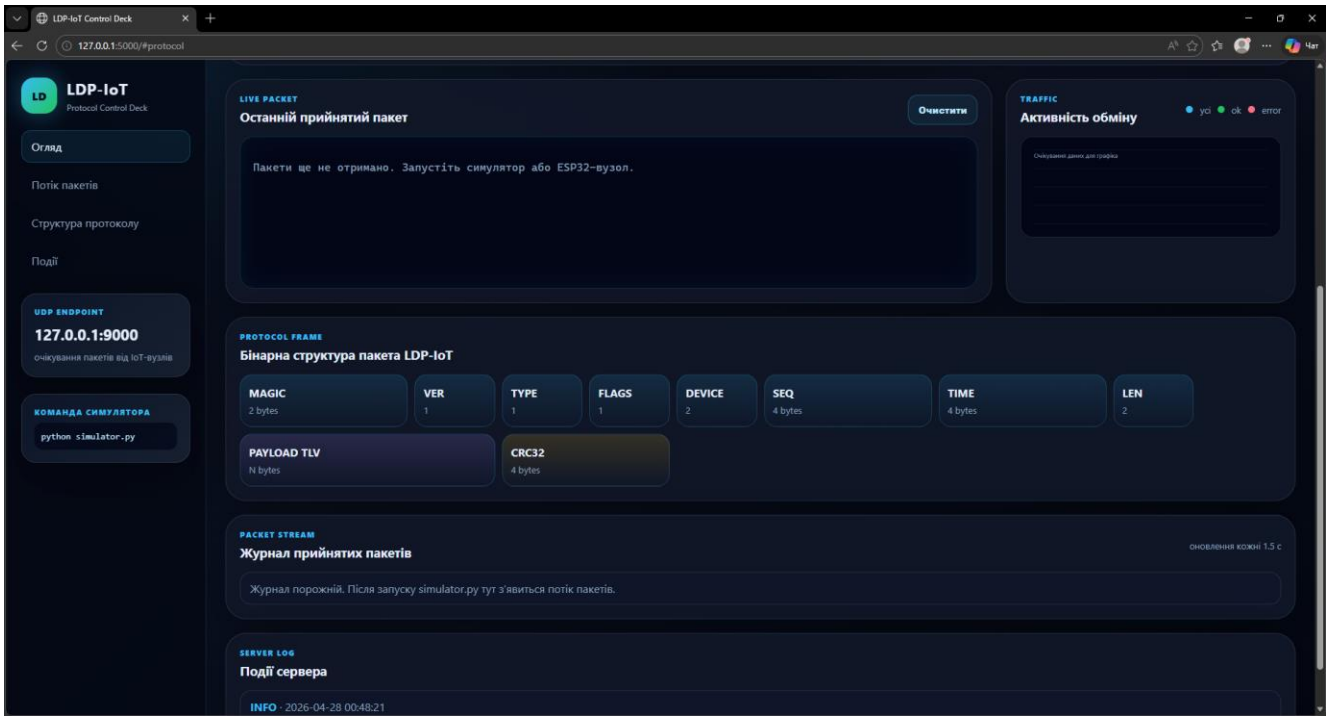


Рисунок 3.1 – Загальний вигляд панелі контролю протоколу LDP-IoT до початку обміну

У блоці останнього прийнятого пакета виведено повідомлення про очікування даних від симулятора або ESP32-вузла. У правій частині розміщено графік активності обміну, який також перебуває в початковому стані. Ліва панель містить навігаційні пункти, UDP endpoint і команду запуску симулятора python

simulator.py. Це підтверджує, що система має окремий інтерфейс не лише для перегляду результатів, а й для контролю стану серверного середовища.

Окремо в інтерфейсі відображено структуру пакета LDP-IoT. У цьому блоці показано склад бінарного кадру протоколу: MAGIC, VER, TYPE, FLAGS, DEVICE, SEQ, TIME, LEN, PAYLOAD TLV і CRC32. Такий блок має практичне значення, оскільки він наочно пов’язує програмну реалізацію з описаною у другому розділі структурою пакета. Завдяки цьому під час тестування можна не лише бачити отримані дані, а й розуміти, з яких службових частин складається повідомлення. На рисунку 3.2 показано вкладку, де розміщено структуру бінарного пакета та журнал прийнятих повідомлень

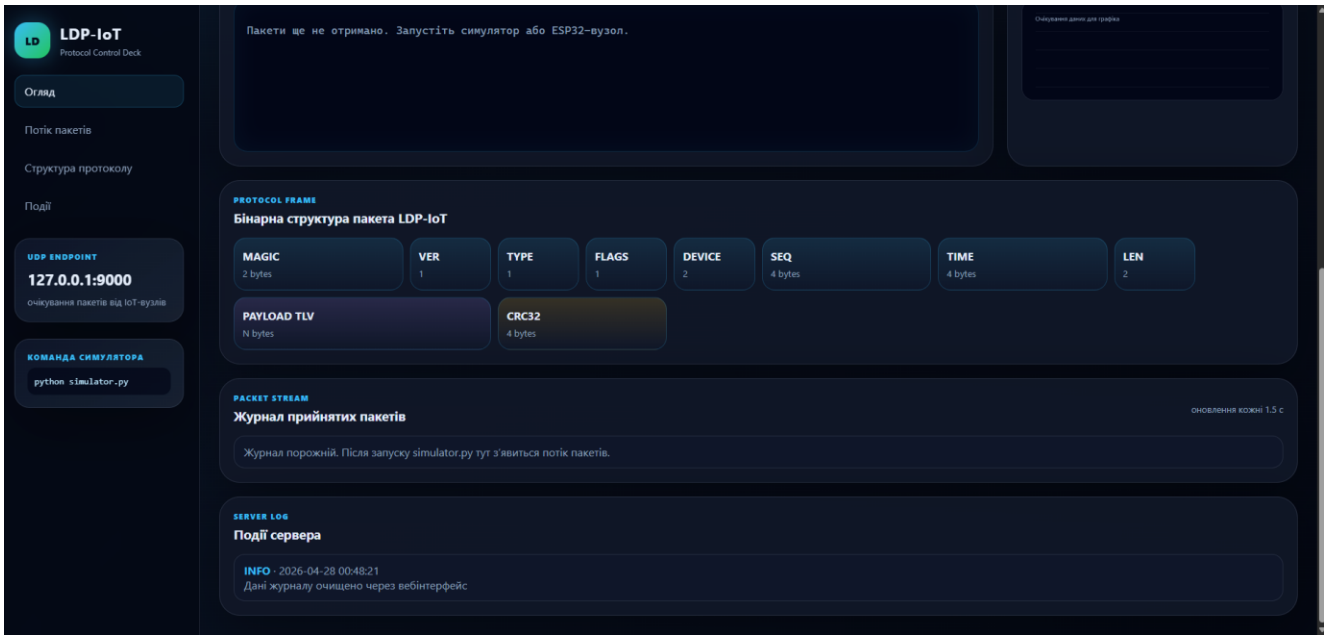


Рисунок 3.2 – Відображення структури пакета та журналу приймання у вебінтерфейсі

. До запуску симулятора журнал є порожнім, а система повідомляє, що потік пакетів з’явиться після запуску simulator.py. У нижній частині також відображено журнал подій сервера, де зафіксовано очищення даних через вебінтерфейс. Це демонструє, що серверна частина не тільки приймає пакети, а

й веде службовий облік подій, який надалі може використовуватися для аналізу роботи протоколу.

Після запуску симулятора розпочато передавання тестових пакетів. Симулятор виконує роль IoT-вузла, який формує DATA-повідомлення з телеметрією та надсилає їх на UDP endpoint. Сервер приймає ці пакети, перевіряє CRC, розбирає payload, фіксує дані в журналі та оновлює вебпанель. Такий підхід дозволяє перевірити роботу протоколу навіть без фізичної плати ESP32, оскільки симулятор повторює логіку формування пакетів і передачі даних.

Під час активного тестування інтерфейс показав, що сервер приймає пакети коректно. У статистичних блоках зафіксовано 9 отриманих пакетів, з яких усі 9 прийнято успішно. Кількість помилок становить 0, що свідчить про коректне формування структури пакета, правильний розрахунок CRC і стабільну роботу модуля приймання. Окремо зафіксовано 1 повторну передачу, що підтверджує роботу механізму retransmit. Середня затримка обміну під час тесту становила 587,67 мс. На рисунку 3.3 зображено активний стан панелі після приймання тестових пакетів.

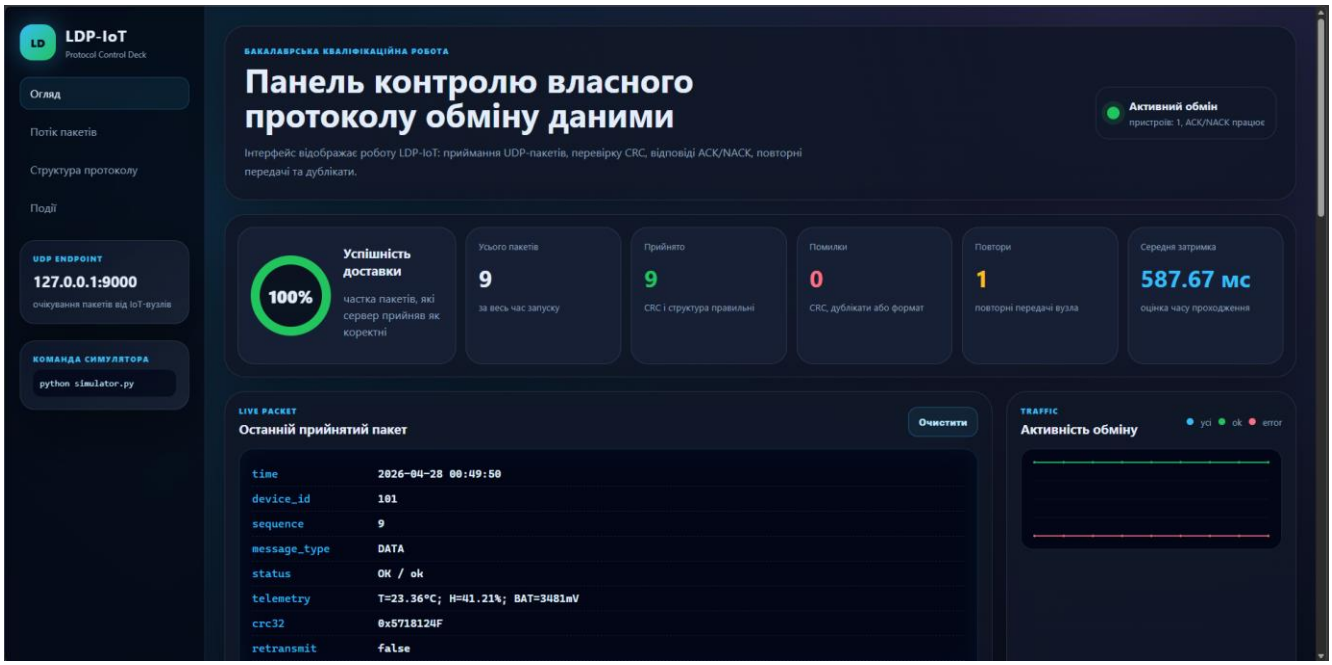


Рисунок 3.3 – Результати тестового обміну пакетами в панелі контролю LDP-IoT

У верхньому блоці відображено статус “Активний обмін”, що підтверджує роботу серверної частини та наявність зв’язку з вузлом. У центральній частині показано показник успішності доставки - 100 %. Це означає, що всі пакети, які сервер прийняв у межах тестового запуску, пройшли перевірку структури та CRC. Показники Усього пакетів – 9, Прийнято – 9, Помилки – 0, Повтори – 1 дають змогу оцінити не лише факт передачі, а й поведінку протоколу при повторному надсиланні.

У блоці останнього прийнятого пакета виведено службові та телеметричні дані останнього повідомлення. Для пакета з послідовним номером 9 зафіксовано device_id = 101, тип повідомлення DATA, статус OK / ok, телеметрію T=23.36°C; H=41.21%; BAT=3481mV, контрольне значення crc32 = 0x5718124F і ознаку retransmit = false. Це підтверджує, що сервер не тільки приймає сам факт надходження пакета, а й правильно розбирає його корисне навантаження у форматі TLV.

Під час перевірки важливим є те, що інтерфейс відображає не лише підсумкову статистику, а й потік окремих пакетів. У журналі прийнятих пакетів можна переглянути кожне повідомлення за його порядковим номером, ідентифікатором пристрою, payload, CRC і причиною обробки. Це дозволяє простежити стабільність обміну не за одним окремим повідомленням, а за послідовністю пакетів.

На рисунку 3.4 показано журнал прийнятих пакетів, де відображено послідовні повідомлення з номерами від #24 до #30. Для кожного пакета вказано пристрій 101, значення корисного навантаження, CRC і статус ok. Наприклад, для пакета #30 зафіксовано payload 22.76°C · 47.96% · 3606mV, CRC 0xA6EC3E96 і результат обробки ok. Для інших пакетів також показано зміну температури, вологості та напруги живлення, що імітує реальний потік телеметричних даних від IoT-вузла.

Під час тестування не зафіксовано помилок CRC або некоректних пакетів. Показник помилок дорівнює нулю, що свідчить про правильне узгодження

формату пакета між симулятором та сервером. У практичному сенсі це означає, що службові поля, payload і CRC формуються та перевіряються однаково на обох сторонах обміну. Журнал прийнятих пакетів під час тестування протоколу LDP-IoT зображено на рисунку 3.4.

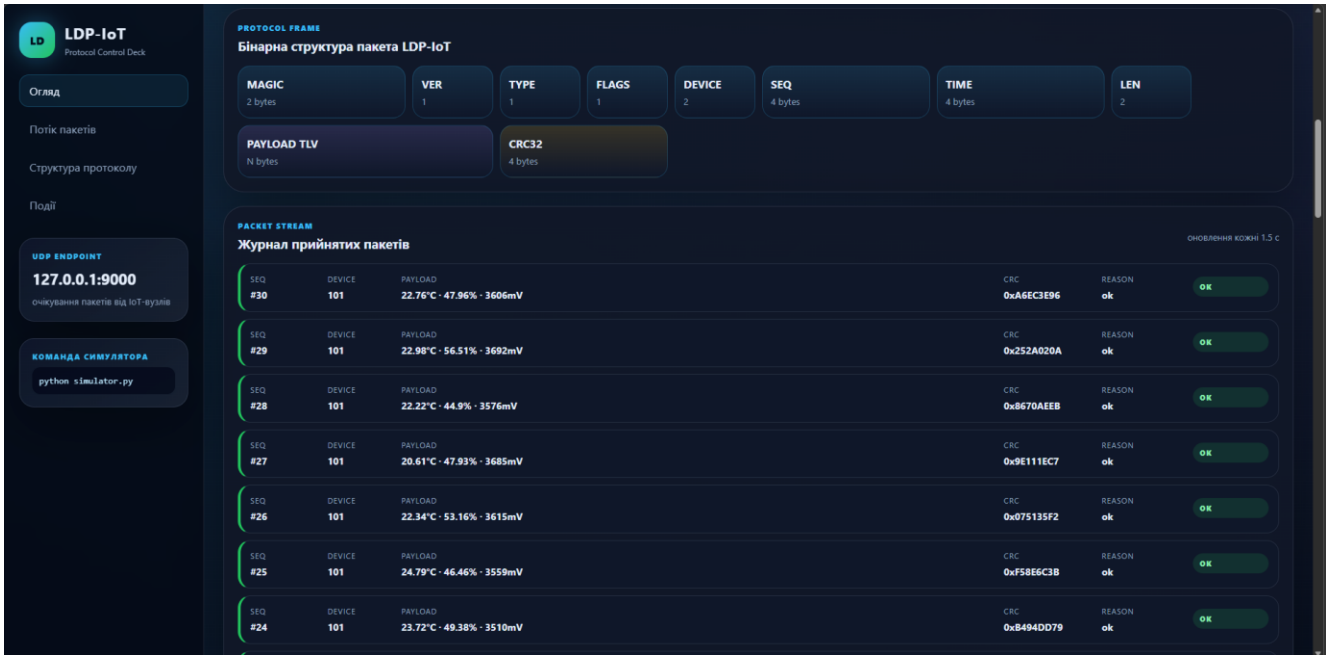


Рисунок 3.4 – Журнал прийнятих пакетів під час тестування протоколу LDP-IoT

У підсумку тестування підтвердило працездатність основних механізмів протоколу LDP-IoT: формування DATA-пакетів, передавання через UDP, перевірку CRC, розбір TLV-навантаження, формування ACK/NACK, фіксацію повторних передач і відображення результатів у вебпанелі. Отримані результати показують, що реалізована система коректно працює в локальному середовищі та може бути використана як прототип власного протоколу обміну даними між IoT-пристроями.

3.5 Оцінка ефективності, надійності та безпеки запропонованого рішення

Після реалізації та тестування протоколу LDP-IoT виконано оцінку його ефективності, надійності та базового рівня безпеки. Основну увагу приділено

тому, наскільки запропоноване рішення відповідає поставленій задачі: забезпечити компактний обмін даними між IoT-вузлом і сервером, перевірити коректність приймання пакетів, підтвердити роботу механізмів ACK/NACK, проконтролювати повторні передачі та показати результат у вебінтерфейсі.

Оцінювання виконано на основі роботи програмного прототипу, у якому серверна частина приймає UDP-пакети, перевіряє їхню структуру, розраховує CRC, обробляє payload у форматі TLV і повертає відповідь пристрою або симулятору. Для зручності контролю створено вебпанель LDP-IoT Control Deck, де відображаються загальні показники обміну, останній прийнятий пакет, активність трафіку, структура бінарного кадру, журнал прийнятих пакетів і події сервера. Це дозволяє оцінювати роботу протоколу не лише за консольними повідомленнями, а й через наочний інтерфейс.

З погляду ефективності запропонований протокол показав перевагу за рахунок компактної структури повідомлення. У пакеті не використовуються текстові поля, JSON-рядки або надлишкові службові конструкції. Основна інформація передається у вигляді бінарного кадру, де кожне поле має чітко визначений розмір. Заголовок містить службові параметри, payload передається у форматі TLV, а в кінці пакета розміщується CRC32. Такий підхід зменшує розмір повідомлення та робить його зручним для передавання короткої телеметрії від IoT-вузлів.

Практична перевірка показала, що сервер коректно приймає сформовані пакети та відображає їх у журналі. У тестовому запуску зафіксовано 9 отриманих пакетів, з яких усі 9 прийнято успішно. Кількість помилок становить 0, а успішність доставки відображена на рівні 100 %. Це свідчить про правильне узгодження формату пакета між вузлом і серверною частиною. Сервер правильно розпізнає службові поля, перевіряє CRC, обробляє телеметрію та формує відповідь.

Надійність протоколу забезпечено через поєднання кількох механізмів. Першим із них є підтвердження отримання пакета. Після передавання DATA-

повідомлення IoT-вузол очікує ACK від сервера. Якщо підтвердження отримано, пакет вважається доставленим, а вузол переходить до наступного циклу. Якщо відповідь не надходить або сервер повертає NACK, активується логіка повторної передачі. Це дозволяє не втрачати дані при короткочасних збоях або затримках у мережі.

Під час тестування зафіксовано 1 повторну передачу. При цьому загальна кількість помилок залишилася нульовою, а всі прийняті пакети мали статус OK. Це означає, що повторне передавання не порушило логіку роботи протоколу. Сервер зміг обробити ситуацію коректно, а вебінтерфейс зафіксував сам факт повтору. Такий результат підтверджує, що механізм retransmit працює не формально, а реально бере участь у контролі доставки повідомлень.

Важливим елементом надійності є порядковий номер пакета. Він дозволяє серверу відстежувати послідовність повідомлень і відрізняти нові пакети від повторних. Якщо пристрій повторно надсилає пакет із тим самим номером, сервер може не записувати його як нові дані, а лише повторно сформувати ACK. Це захищає журнал і базу даних від дублювання результатів. Такий підхід є особливо важливим для IoT-систем, де однакові повторні значення можуть спотворити підсумковий аналіз.

Ще одним важливим аспектом є захист від повторного відтворення повідомлень. Оскільки кожен пакет має порядковий номер, сервер може визначити, чи не надходило таке повідомлення раніше. Це зменшує ризик повторної обробки старих даних. У поєднанні з часовою міткою та контролем Device ID такий механізм може стати основою для більш стійкої перевірки актуальності пакетів.

Під час оцінювання також враховано зручність практичного контролю роботи системи. Вебінтерфейс відображає не лише кінцевий результат, а й проміжні службові показники: кількість пакетів, кількість прийнятих повідомлень, помилки, повтори, середню затримку, CRC і payload. Завдяки цьому процес обміну стає прозорим.

Узагальнюючи результати, можна зазначити, що запропонований протокол LDP-IoT виконав основні завдання, визначені в бакалаврській роботі. Він забезпечив компактне формування пакета, передавання через UDP, перевірку цілісності, підтвердження доставки, повторну передачу та відображення результатів у вебінтерфейсі.

У процесі оцінювання також враховано, що розроблений протокол має модульну структуру, тому окремі його частини можуть змінюватися або доповнюватися без повної переробки всієї системи. Наприклад, у разі потреби до наявної структури пакета можна додати нові TLV-параметри, розширити журнал подій, змінити інтервал передавання або посилити перевірку повідомлень за рахунок криптографічних механізмів. Це свідчить про те, що запропоноване рішення не обмежується лише демонстраційним обміном тестовими пакетами, а має основу для подальшого розвитку.

У підсумку розроблене рішення можна вважати працездатним прототипом власного протоколу обміну даними між IoT-пристроями. Його сильними сторонами є простота реалізації, компактний формат пакета, зрозуміла логіка ACK/NACK, контроль цілісності та зручна вебпанель для перегляду результатів. Обмеженням залишається базовий рівень безпеки, оскільки повноцінне криптографічне підтвердження автентичності повідомлень у поточній реалізації не є основним механізмом. Водночас структура протоколу вже передбачає можливість додавання HMAC, шифрування payload і розширеного контролю пристроїв, що робить розроблене рішення придатним для подальшого вдосконалення.

3.6 Висновки до третього розділу

У третьому розділі бакалаврської кваліфікаційної роботи виконано програмну реалізацію основних компонентів системи обміну даними за протоколом LDP-IoT. На стороні ESP32 реалізовано формування бінарного

пакета, додавання TLV-навантаження, обчислення CRC32, передавання через UDP та очікування ACK/NACK. На серверній стороні реалізовано приймання пакетів, розбір заголовка, перевірку контрольного значення, обробку корисного навантаження, запобігання дублюванню повідомлень і формування відповіді. Для візуального контролю роботи створено вебінтерфейс LDP-IoT Control Deck, у якому відображаються статистика, журнал пакетів, останнє повідомлення та події сервера.

Проведене тестування підтвердило працездатність запропонованого рішення в локальному середовищі. Сервер коректно приймав пакети, перевіряв CRC32, розпізнавав Device ID, порядковий номер і телеметричні дані, а також фіксував результати у вебпанелі. Під час перевірки отримано 100 % успішності доставки для тестового набору пакетів, зафіксовано відсутність помилок CRC та роботу механізму повторної передачі. Це дозволяє вважати реалізований прототип функціональним і придатним для подальшого розвитку, зокрема додавання HMAC, шифрування payload і підтримки більшої кількості IoT-вузлів.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі розглянуто питання розроблення протоколу обміну даними між пристроями Інтернету речей. У процесі виконання роботи встановлено, що для IoT-систем важливими є не лише самі пристрої та канали зв’язку, а й правила організації обміну, які визначають структуру повідомлень, спосіб підтвердження доставки, контроль цілісності, обробку помилок і можливість подальшого масштабування системи. Саме тому основну увагу зосереджено на створенні власного легковагового протоколу, який може працювати поверх UDP і забезпечувати передачу коротких телеметричних повідомлень між IoT-вузлом і серверною частиною.

У першому розділі проаналізовано особливості побудови систем Інтернету речей, принципи організації обміну даними та поширені протоколи, що використовуються в IoT-середовищі. Розглянуто MQTT, CoAP, HTTP, AMQP, DDS, LwM2M, OPC UA та інші рішення. На основі виконаного аналізу визначено, що готові протоколи мають власні переваги, однак не завжди забезпечують оптимальний баланс між компактністю, простотою реалізації, енергоефективністю та контролем обміну. Це дозволило обґрунтувати необхідність розроблення власного прикладного протоколу, орієнтованого на роботу з невеликими пакетами даних і пристроями з обмеженими ресурсами.

У другому розділі розроблено загальну архітектуру системи обміну даними, у якій IoT-вузол формує повідомлення, передає його через Wi-Fi та UDP на сервер, а сервер виконує приймання, перевірку, обробку й збереження отриманих даних. Для запропонованого рішення сформовано структуру протоколу LDP-IoT, яка включає службовий заголовок, корисне навантаження у форматі TLV та контрольне поле CRC32.

У третьому розділі виконано програмну реалізацію основних компонентів системи. На стороні IoT-вузла реалізовано логіку формування бінарного пакета, додавання телеметричних даних, розрахунку контрольного значення,

передавання пакета через UDP та очікування відповіді від сервера. На серверній стороні реалізовано приймання UDP-повідомлень, розбір службових полів, перевірку CRC, обробку payload, формування ACK/NACK і збереження результатів. Для контролю роботи системи створено вебінтерфейс, у якому відображаються структура пакета, журнал прийнятих повідомлень, останній пакет, статистика успішності, повтори, помилки та середня затримка.

Тестування показало працездатність запропонованого протоколу в локальному середовищі. Під час перевірки сервер коректно приймав DATA-пакети, розпізнавав ідентифікатор пристрою, порядковий номер, корисне навантаження та контрольну суму. У тестовому запуску зафіксовано успішне приймання пакетів, відсутність помилок CRC, роботу механізму повторної передачі та коректне відображення результатів у вебпанелі.

Оцінка отриманих результатів показала, що запропоноване рішення має низку практичних переваг. Протокол LDP-IoT використовує компактний бінарний формат, не потребує складного брокера або важкого прикладного рівня, підтримує підтвердження доставки, повторну передачу, контроль цілісності та базове журналювання подій. Водночас визначено й обмеження поточної реалізації. Зокрема, CRC32 забезпечує перевірку технічної цілісності, але не є повноцінним криптографічним захистом. Через це в подальшому протокол може бути розширений механізмом HMAC, шифруванням корисного навантаження та автентифікацією пристроїв за секретними ключами.

У підсумку мету бакалаврської кваліфікаційної роботи досягнуто. Розроблено архітектуру системи, сформовано структуру власного протоколу LDP-IoT, реалізовано програмні компоненти IoT-вузла та сервера, створено вебпанель контролю й виконано перевірку роботи обміну даними. Отримане рішення може використовуватися як навчальний і практичний прототип легковагового протоколу для IoT-систем, а також як основа для подальшого вдосконалення у напрямі підвищення безпеки, масштабованості та енергоефективності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Малохвій Є., Молчанов Г. Дослідження протоколів передачі даних в умовах Інтернету речей. *Системи управління, навігації та зв'язку*. 2022. № 1(67). С. 66–70. URL: https://doi.org/10.26906/SUNZ.2022.1.066 (date of access: 27.04.2026)

2. Комісаров О. С. Оптимізація протоколу MQTT для підвищення продуктивності IoT-систем. *Вісник Херсонського національного технічного університету*. 2024. № 4. С. 291–296. URL: https://doi.org/10.35546/kntu2078-4481.2024.4.38 (date of access: 27.04.2026)

3. Рубан С., Харламенко В., Леонов Д., Лазарєв В. Створення веб-орієнтованого людино-машинного інтерфейсу для моделі робота SCARA. *Journal of Kryvyi Rih National University*. 2025. Т. 23, № 1. С. 21–31. URL: https://doi.org/10.31721/2306-5451-2025-1-23-21-31 (date of access: 27.04.2026)

4. Горин І. С. Аналіз безпеки протоколу MQTT у IoT-комунікаціях. Тернопіль. 2025. 101 с. URL: https://elartu.tntu.edu.ua/handle/lib/49985 (date of access: 27.04.2026)

5. Чертанов Ю. Підхід до вибору протоколу та архітектури мережі в IoT-системах. *Системи управління, навігації та зв'язку*. 2026. № 2(84). С. 21–31. URL: https://doi.org/10.26906/SUNZ.2026.2 (date of access: 27.04.2026)

6. OASIS. MQTT Version 5.0. 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (date of access: 28.04.2026)

7. Selander G., Mattsson J., Palombini F., Seitz L. Object Security for Constrained RESTful Environments. 2019. URL: https://www.rfc-editor.org/rfc/rfc8613 (date of access: 28.04.2026)

8. Fagan M., Megas K., Scarfone K., Smith M. IoT Device Cybersecurity Capability Core Baseline. 2020. URL: https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8259A.pdf (date of access: 28.04.2026)

9. ETSI. Cyber Security for Consumer Internet of Things: Baseline Requirements. 2020. URL: https://www.etsi.org/deliver/etsi_en/303600_303699/303645 (date of access: 28.04.2026)

10. Iyengar J., Thomson M. QUIC: A UDP-Based Multiplexed and Secure Transport. 2021. URL: https://www.rfc-editor.org/rfc/rfc9000 (date of access: 28.04.2026)

11. Rescorla E., Tschofenig H., Fossati T., Nir Y. Datagram Transport Layer Security Protocol Version 1.3. 2022. URL: https://www.rfc-editor.org/rfc/rfc9147 (date of access: 28.04.2026)

12. Fagan M., Megas K., Watrobski P. Profile of the IoT Core Baseline for Consumer IoT Products. 2022. URL: https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8425.pdf (date of access: 28.04.2026)

13. Gerodimos A., Siabi S., Moumgiakmas A., Kalogeras A. IoT communication protocols and security threats. *Internet of Things and Cyber-Physical Systems*. 2023. Vol. 1, P. 1—12. URL: https://doi.org/10.1016/j.iotcps.2022.12.003 (date of access: 29.04.2026)

14. Quincozes V., Quincozes S., Kazienko J. Survey on IoT application layer protocols. *International Journal of Information Security*. 2024. Vol. 23, P. 1975–2002. URL: https://doi.org/10.1007/s10207-024-00828-w (date of access: 29.04.2026)

15. Ferraris D., Fernández Gago C., Roman R., López J. IoT trust model frameworks. *The Journal of Supercomputing*. 2024. Vol. 80, P. 8259–8296. URL: https://doi.org/10.1007/s11227-023-05765-4 (date of access: 29.04.2026)

16. Hossain M., Kayas G., Hasan R., Skjellum A. IoT security analysis. *Future Internet*. 2024. Vol. 12, no. 2. P. 40. URL: https://doi.org/10.3390/fi16020040 (date of access: 29.04.2026)

17. Soto-Cruz J. Lightweight cryptography for IoT systems. *Technologies*. 2024. Vol. 13, no. 1. P. 3. URL: https://doi.org/10.3390/technologies13010003 (date of access: 29.04.2026)

18. Radhakrishnan I., Shanmugam P. Lightweight cryptographic algorithms. *Sensors*. 2024. Vol. 24, no. 12. P. 4008. URL: https://doi.org/10.3390/s24124008 (date of access: 30.04.2026)

19. Gaina L., Stangaciu C. Secure IoT communications. *Sensors*. 2024. Vol. 24, no. 23. P. 7528. URL: https://doi.org/10.3390/s24237528 (date of access: 30.04.2026)

20. Laghari A., Li H. IoT security challenges. *Discover Internet of Things*. 2024. Vol. 4, no. 36. P. 1-18. URL: https://doi.org/10.1007/s43926-024-00090-5 (date of access: 30.04.2026)

21. Hofer-Schmitz K., Stojanović B.. Verification of IoT protocols. *Computer Networks*. 2020. Vol. 174, no. 107233. P. 1. URL: https://doi.org/10.1016/j.comnet.2020.107233 (date of access: 30.04.2026)

22. Canavese D., Mannella L. IoT security at the edge. *Sensors*. 2024. Vol. 24, no. 2. P. 590. URL: https://doi.org/10.3390/s24020590 (date of access: 30.04.2026)

23. Dauda A., Flauzac O. IoT architectures. *Sensors*. 2024. Vol. 24, no. 16. P. 5320. URL: https://doi.org/10.3390/s24165320 (date of access: 30.04.2026)

24. Pirbhulal S., Chockalingam S. IoT cybersecurity review. *International Journal of Information Security*. 2024. Vol. 23, no. 1. P. 2827–2879. URL: https://doi.org/10.1007/s10207-024-00865-5 (date of access: 30.04.2026)

25. Sahu S., Mazumdar K. Exploring security threats and solutions Techniques for Internet of Things (IoT): from vulnerabilities to vigilance. *Frontiers in Artificial Intelligence*. 2024. Vol. 7, no. 1. P. 1. URL: https://doi.org/10.1007/s10207-024-00865-5 (date of access: 30.04.2026)

26. Magara T., Zhou Y. Internet of Things (IoT) of Smart Homes: Privacy and Security. *Journal of Electrical and Computer Engineering*. 2024. Vol. 1, no. 1. P. 1. URL: https://doi.org/10.3389/frai.2024.1397480 (date of access: 30.04.2026)

27. Rana M., Mamun Q. Enhancing IoT Security: An Innovative Key Management System for Lightweight Block Ciphers. *Sensors*. 2023. Vol. 23, no. 18. P. 7678. URL: https://doi.org/10.3390/s23187678 (date of access: 30.04.2026)

28. Gil S., Zapata-Madrigal G. IoT protocol integration. *Journal of Electrical Systems and Information Technology*. 2022. Vol. 9, no. 1. P. 1. URL: https://doi.org/10.1186/s43067-022-00043-4 (date of access: 30.04.2026)

29. Ansari S., Ali S. Lightweight cryptography for IoT. *Discover Computing*. 2025. Vol. 28, no. 266. P. 1. URL: https://doi.org/10.1007/s10791-025-09755-3 (date of access: 30.04.2026)

30. Alotaibi A., Aldawghan H. IoT authentication. *Sensors*. 2025. Vol. 25, no. 6. P. 1. URL: https://doi.org/10.3390/s25061649 (date of access: 30.04.2026)

31. Sebestyen H., Popescu D. IoT security review. *Computers*. 2025. Vol. 14, no. 2. P. 61. URL: https://doi.org/10.3390/computers14020061 (date of access: 30.04.2026)

32. Petrescu I., Niculae E. IoT protocols review. *Technologies*. 2025. Vol. 13, no. 12. P. 583. URL: https://doi.org/10.3390/technologies13120583 (date of access: 30.04.2026)

33. Krawiec J. IoT communication efficiency. *Sensors*. 2025. Vol. 25, no. 19. P. 6042. URL: https://doi.org/10.3390/s25196042 (date of access: 30.04.2026)

34. Gavriilidis N., Halkidis S. TLS-enhanced MQTT. *Applied Sciences*. 2025. Vol. 15, no. 15. P. 8398. URL: https://doi.org/10.3390/app15158398 (date of access: 30.04.2026)

35. Abdelwahed S., Hefny I. IoT gateways. *Internet of Things*. 2025. Vol. 33, no. 1. P. 1. URL: https://doi.org/10.1016/j.iot.2025.101703 (date of access: 30.04.2026)

36. Kreković D., Krivić P. IoT communication overhead. *Internet of Things*. 2025. Vol. 31 no. 1. P. 1. URL: https://doi.org/10.1016/j.iot.2025.101553 (date of access: 30.04.2026)

37. Gușiță B., Anton A. IoT edge security. *International Journal of Information Security*. 2025. Vol. 24 no. 149. P. 1. URL: https://doi.org/10.1007/s10207-025-01071-7 (date of access: 30.04.2026)

38. Mishra R., Mishra A. IoT security protocols. *Computers & Security*. 2025. Vol. 151 no. 1. P. 1. URL: https://doi.org/10.1016/j.cose.2024.104310 (date of access: 30.04.2026)

39. Szymoniak S., Piątkowski J. IoT defense mechanisms. *Applied Sciences*. 2025. Vol. 15 no. 2. P. 499. URL: https://doi.org/10.3390/app15020499 (date of access: 30.04.2026)

40. Serepas F., Papias I. IoT gateway on ESP32. *Computers*. 2025. Vol. 14 no. 9. P. 391. URL: https://doi.org/10.3390/computers14090391 (date of access: 30.04.2026)

41. Sefati S., Arasteh B. IoT cybersecurity techniques. *Cluster Computing*. 2025. Vol. 26 no. 792. P. 1. URL: https://doi.org/10.1007/s10586-025-05449-z (date of access: 01.05.2026)

42. Jamshidi S., Nikanjam A. IoT security patterns. *International Journal of Information Security*. 2025. Vol. 24 no. 91. P. 1. URL: https://doi.org/10.1007/s10207-025-01011-5 (date of access: 01.05.2026)

43. Żatuchin D. IoT protocol selection. *Future Internet*. 2025. Vol. 12 no. 4. P. 125. URL: https://doi.org/10.3390/informatics12040125 (date of access: 01.05.2026)

44. Chai T., Kim D. IoT communication systems. *Sensors*. 2025. Vol. 25 no. 8. P. 2509. URL: https://doi.org/10.3390/s25082509 (date of access: 01.05.2026)

45. Sorescu T., Chiriac V. IoT cryptography. *Sensors*. 2025. Vol. 25 no. 18. P. 5887. URL: https://doi.org/10.3390/s25185887 (date of access: 01.05.2026)

46. Silva C., Tenório N. Encryption algorithms for IoT. *Computers*. 2025. Vol. 14 no. 12. P. 505. URL: https://doi.org/10.3390/computers14120505 (date of access: 01.05.2026)

47. Zhong J., He S. IoT authentication frameworks. *Sensors*. 2025. Vol. 25 no. 17. P. 5594. URL: https://doi.org/10.3390/s25175594 (date of access: 01.05.2026)

48. Kalamaras S., Tsitsimpikou M. IoT system on ESP32. *Applied Sciences*. 2025. Vol. 15 no. 1. P. 34. URL: https://doi.org/10.3390/app15010034 (date of access: 01.05.2026)

49. Han Y., Wang P. IoT protocol vulnerability detection. *Heliyon*. 2024. Vol. 10 no. 12. P. 1. URL: https://doi.org/10.1016/j.heliyon.2024.e31846 (date of access: 01.05.2026)

50. Принцип роботи MQTT. URL: https://www.manubes.com/what-is-mqtt/ (date of access: 01.05.2026)

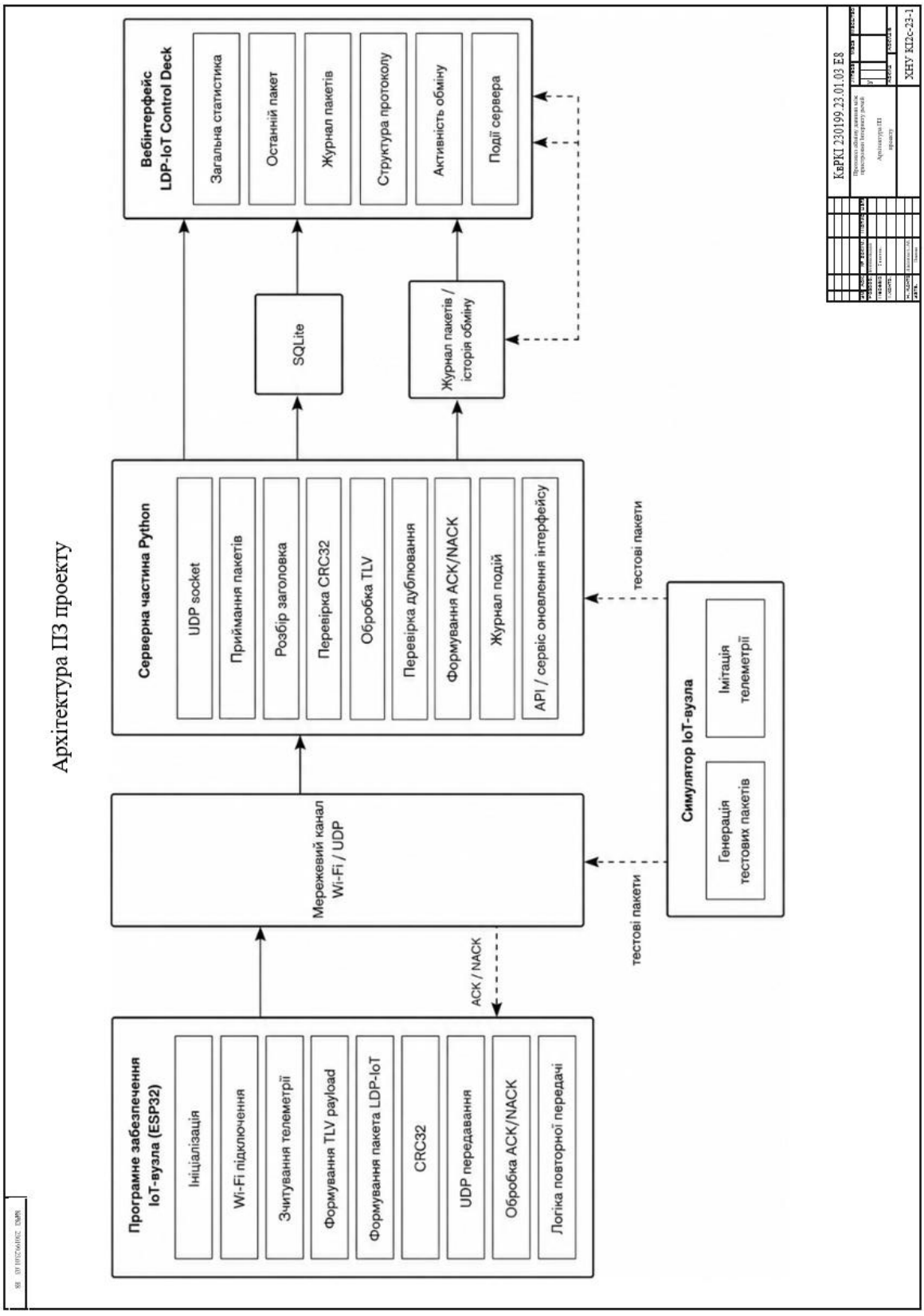
51. Принцип роботи Constrained Application Protocol (CoAP). URL: https://pervasivecomputinginfo.blogspot.com/2017/02/constrained-application-protocol-coap.html (date of access: 01.05.2026)

52. Принцип роботи HTTP. URL: https://hostiq.ua/wiki/ukr/http-https/ (date of access: 01.05.2026)

ДОДАТОК А

(обов’язковий)

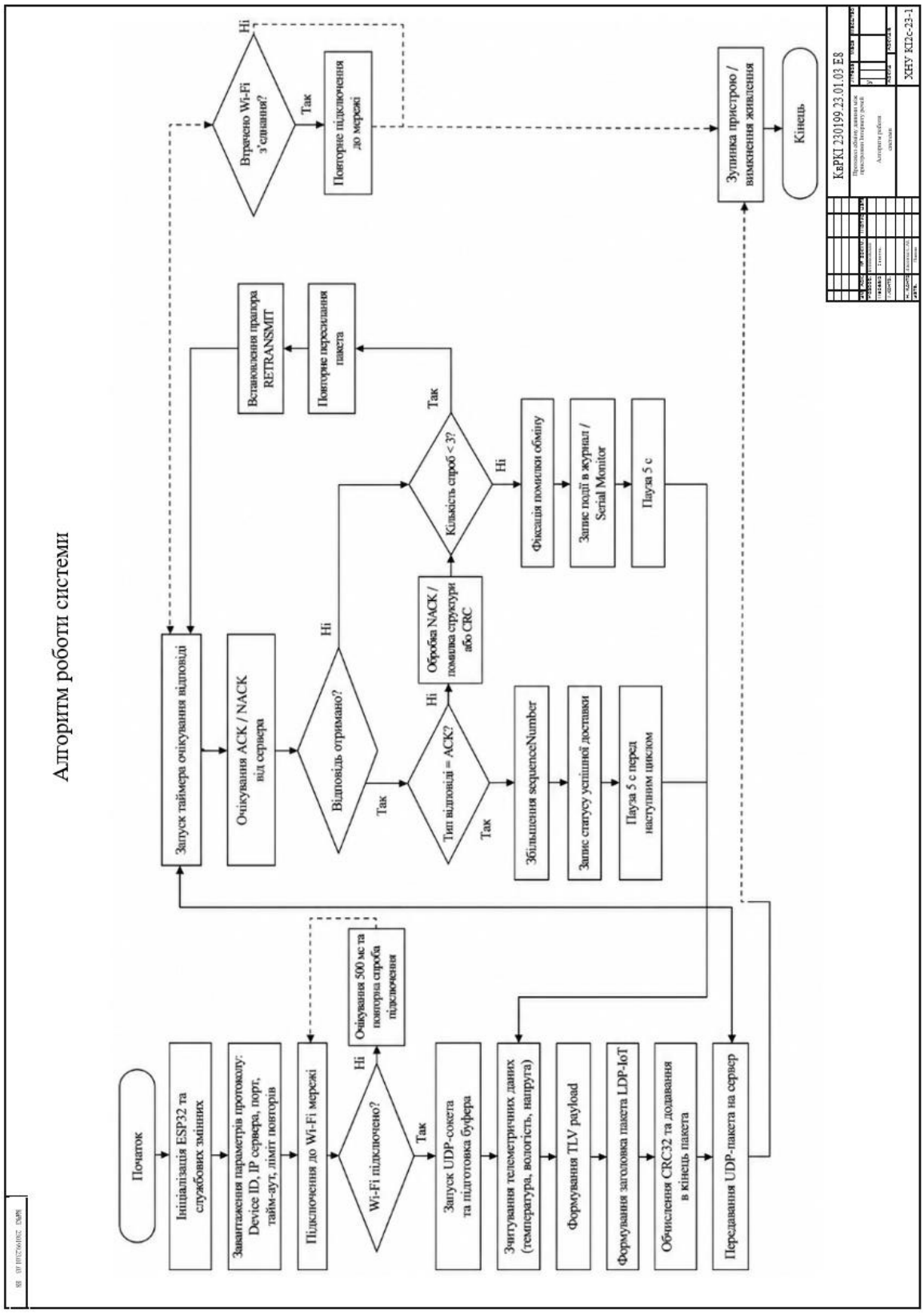
Копія креслення «Архітектура ПЗ проєкту»



ДОДАТОК Б

(обов’язковий)

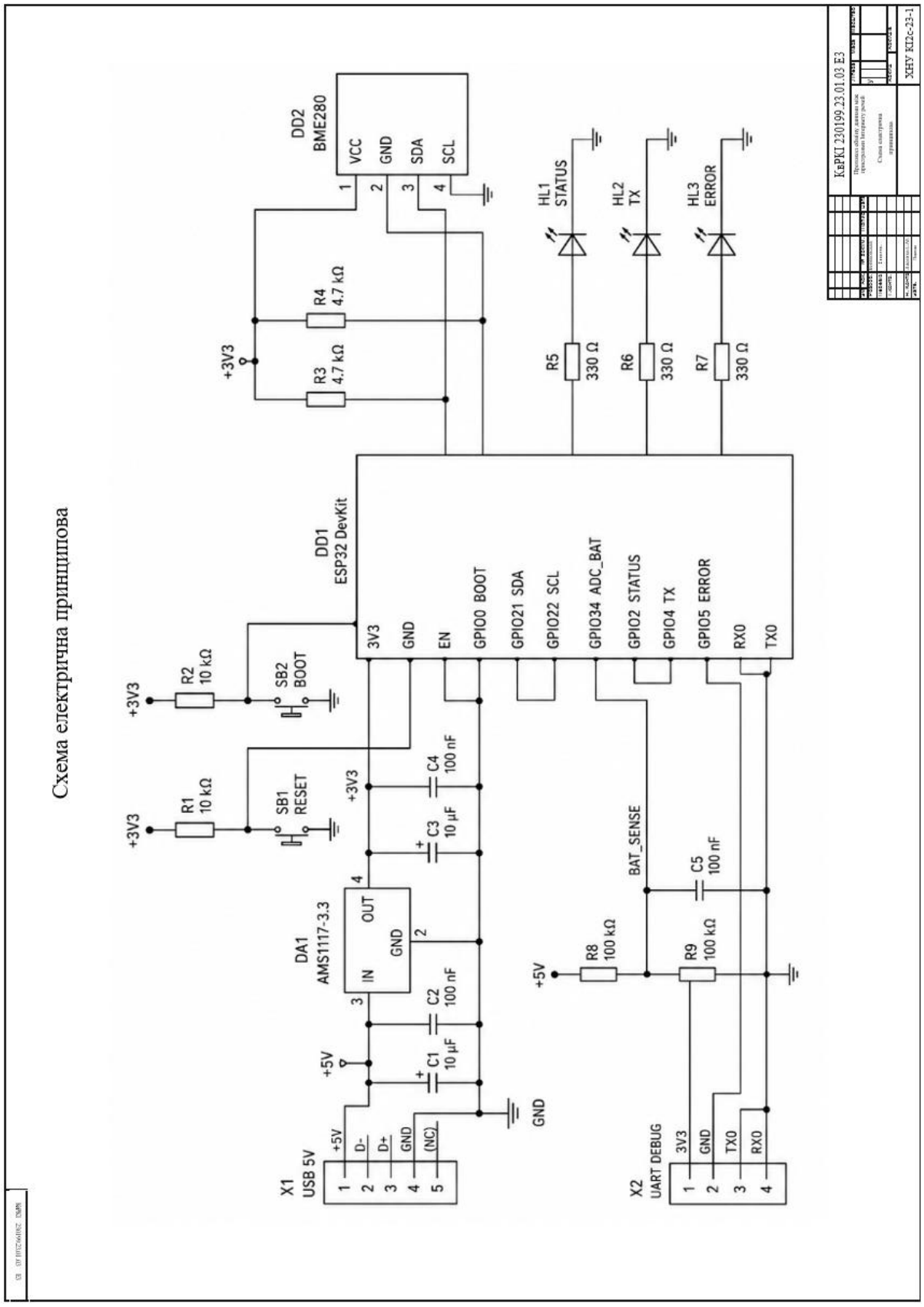
Копія креслення «Алгоритм роботи системи»



ДОДАТОК В

(обов'язковий)

Копія креслення «Апаратне забезпечення проєкту»



Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Владислава БУРЯКОВСЬКА

Співавтор:

Назва: Протокол обміну даними між пристроями Інтернету речей (IoT)

Експерт: Дмитро МЕДЗАТИЙ

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:1.59%

Коефіцієнт подібності 2:0.23%

Мікропробіли: 3

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-05-27 20:55:16.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2026-05-28 Доцент Андрій Нічепорук

Дата експерт

Thu May 28 08:49:09 EEST 2026, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 7%**

	Документ		Сумарний збіг по Базі Даних	
ID: 272539 Назва: БКР Протокол обміну даними між пристроями Інтернету речей (IoT) Додано в БД: 2026-05-28 Автора: Владислава БУРЯКОВСЬКА Керівники: Дмитро МЕДЗАТИЙ Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	116564	985	1318 (1%)	18 (2%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Буряковська Владислава Вікторівна

Тема: Протокол обміну даними між пристроями Інтернету речей (IoT)

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___68___

1. Короткий зміст роботи та прийнятих рішень: Метою роботи є проєктування, програмна реалізація та перевірка протоколу обміну даними між IoT-вузлом і серверною частиною.

2. Висновок про відповідність роботи дипломному завданню:

Робота в цілому відповідає завданню, однак забезпечення безпеки та оцінювання ефективності виконані лише частково.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: Перший розділ містить аналіз існуючих протоколів IoT (MQTT, CoAP, HTTP, AMQP та інших). Проте аналіз носить переважно описовий характер і не супроводжується кількісним порівнянням протоколів за ключовими показниками.

Другий розділ присвячено проєктуванню протоколу. Структура пакета описана задовільно, однак обґрунтування окремих архітектурних рішень є поверховим.

Третій розділ містить програмну реалізацію та тестування. Тестування проведено лише в умовах стабільної локальної мережі без моделювання втрат пакетів і нестабільності з'єднання, що є характерними умовами для IoT-середовища.

4. Позитивні сторони роботи: Розроблено власний прикладний протокол із бінарним форматом пакетів, що є актуальним для IoT-систем з обмеженими ресурсами. Реалізовано базовий цикл обміну даними між ESP32 і сервером із механізмами ACK/NACK та повторної передачі. Використання TLV-формату корисного навантаження надає протоколу певну гнучкість.

5. Негативні сторони роботи: Відсутнє порівняльне тестування протоколу LDP-IoT із наявними аналогами (MQTT, CoAP) за показниками затримки, обсягу трафіку та енергоспоживання, що не дозволяє оцінити реальні переваги розробленого рішення.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена в цілому згідно з вимогами, однак містить окремі недоліки в структурі викладу матеріалу. Графічний матеріал представлено трьома кресленнями. Деякі рисунки та таблиці недостатньо пов'язані з текстом і не супроводжуються аналізом.

7. Відгук про роботу в цілому: Робота виконана на достатньому технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: Задовільно (D / 70)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Яшина О.М., доцент кафедри УТЗ

_____

"___"_____2026 р. _____(підпис)

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Владислава БУРЯКОВСЬКА

ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи КІ2с-23-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

1 травня 2026 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Протокол обміну даними між пристроями Інтернету речей (IoT)
Автор Владислава БУРЯКОВСЬКА
Освітня програма Комп'ютерна інженерія та програмування
Рівень вищої освіти перший (бакалаврський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: д.т.н., проф. Дмитро МЕДЗАТИЙ

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;

3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 1,59%; та системою Anti-Plagiarism складає 1%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

01.06.2026

Завідувач кафедри _____ Ольга ПАВЛОВА
Підпис Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми _____ Андрій НІЧЕПОРУК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Дмитро МЕДЗАТИЙ
Підпис Ім'я, ПРІЗВИЩЕ