Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та управління
Кафедра інформаційної та кібернетичної безпеки
імені професора Володимира Бурячка

«Допущено до захисту»
Завідувач кафедри інформаційної та
кібернетичної безпеки імені професора
Володимира Бурячка
кандидат технічних наук,
П.М. Складанний
_____
«____» _____ 2025 р.

АТЕСТАЦІЙНА РОБОТА
на здобуття освітнього ступеня «магістр»
спеціальність 125 «Кібербезпека»
Тема роботи:

ІМІТАЦІЙНЕ МОДЕЛЮВАННЯ КОНФЛІКТНОЇ ВЗАЄМОДІЇ ІНФОРМАТИВНИХ І ЗАВАДОВИХ РАДІОСИГНАЛІВ

Виконав
Студент групи БІКСм-1-24-1.4д
Новицький Антон Анатолійович
_____

Науковий керівник
д.т.н. Крючкова Лариса Петрівна
_____

КИЇВ – 2025

Міністерство освіти і науки України
Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра інформаційної та кібернетичної безпеки
імені професора Володимира Бурячка

Освітньо-кваліфікаційних рівень – магістр
Спеціальність 125 Кібербезпека та захист інформації
Освітня програма 125.00.01 Безпека інформаційних і комунікаційних систем

«Затверджую»
Завідувач кафедри інформаційної та
кібернетичної безпеки імені
професора Володимира Бурячка
кандидат технічних наук, доцент
Складанний П.М.

_____
(підпис)
« ___ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Новицькому Антону Анатолійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи: Імітаційне моделювання конфліктної взаємодії інформативних і завадових радіосигналів;
керівник Крючкова Лариса Петрівна, д.т.н., поофесор, затверджені наказом ректора від «___»_____ 20___ року №__.
2. Термін подання студентом роботи «___»_____ 20___ р.
3. Вихідні дані до роботи:
   3.1 науково-технічна та нормативна література з теми дослідження.
   3.2 методи: аналітичний, статистичний, моделювання, програмування.
   3.3 технології: Bluetooth Low Energy, BLE, імітаційне моделювання, конфліктна взаємодія сигналів.
4. Зміст текстової частини роботи (перелік питань, які потрібно розробити):
   4.1 дослідити принцип роботи технології BLE і методи роботи з нею;
   4.2 проаналізувати структуру BLE-пакетів та можливості стандартного BLE-стека Android;
   4.3 дослідити залежність RSSI та похибки оцінки відстані до пристрою на основі експериментів з ESP32-S3 та Ubertooth One;
   4.4 модифікувати програмне забезпечення ubertooth-BLE для високоточної часової прив’язки пакетів;
   4.5 розробити Android-застосунок для сканування BLE-пакетів та візуалізації параметрів радіоефіру;
   4.6 провести експериментальні дослідження поведінки BLE-пакетів у сценарії з завадами і без завад засобами створеного Android-застосунку з зовнішньою

антеною і Python-оркестратором, створенного для керування BLE-маячками реклами з налаштовувальними параметрами

5. Перелік графічного матеріалу:

5.1 Презентація доповіді, виконана в Microsoft PowerPoint.

6. Дата видачі завдання «___»_____ 20___ р.

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів підготовки роботи	Термін виконання	Примітка
1.	Уточнення постановки завдання		
2.	Аналіз літератури		
3.	Обґрунтування вибору рішення		
4.	Збір даних		
5.	Виконання та оформлення розділу 1.		
6.	Виконання та оформлення розділу 2.		
7.	Виконання та оформлення розділу 3.		
8.	Вступ, висновки, реферат		
9.	Апробація роботи на науково-методичному семінарі та/або науково-технічній конференції		
10.	Оформлення та друк текстової частини роботи		
11.	Оформлення презентацій		
12.	Отримання рецензій		
13.	Попередній захист роботи		
14.	Захист в ЕК		

Студент _____ Новицький Антон Анатолійович
(підпис) (прізвище, ім’я, по батькові)

Науковий керівник _____ Крючкова Лариса Петрівна
(підпис) (прізвище, ім’я, по батькові)

РЕФЕРАТ

Магістерська атестаційна робота присвячена розробці та експериментальному дослідженню методу моделювання Bluetooth з низьким енергоспоживанням ‘Bluetooth Low Energy’ (BLE) пакетів і конфліктної взаємодії інформативних та завадових радіосигналів у діапазоні 2,4 ГГц. У роботі досліджено структуру реальних BLE-рекламних кадрів, побудовано програмно-апаратний стенд на основі Android-застосунку, зовнішніх BLE-антен (ESP32-S3, nRF52840) та програмно-кероване радіо ‘Software-Defined Radio’ (SDR) платформи HackRF One, а також проведено серію експериментів із впливу BLE-генератора завад на якість приймання пакетів. Робота складається зі вступу, чотирьох розділів, що містять 24 рисунки та 7 таблиць, висновків та списку використаних джерел, що містить 25 найменувань. Загальний обсяг роботи становить 133 аркушів, з яких 54 аркушів займають додатки на окремих аркушах, перелік умовних скорочень та список використаних джерел.

Об’єкт дослідження – безпроводовий BLE-радіоканал під впливом електромагнітних завад.

Предмет дослідження – показники якості BLE-зв’язку, ймовірність втрати пакетів, відношення сигнал-шум, похибка оцінки відстані, завантаженість каналу при різних параметрах передачі.

Мета роботи – підвищення інформативності та достовірності аналізу стану BLE-радіоканалу в умовах завад шляхом розроблення та експериментальної перевірки методу моделювання BLE-пакетів і конфліктної взаємодії інформативних та завадових радіосигналів на основі спеціалізованого програмно-апаратного стенду.

Для досягнення поставленої мети, у роботі: досліджено принцип роботи технології BLE і методи роботи з нею, проаналізовано структуру BLE-пакетів та можливості стандартного BLE-стека Android, досліджено залежність RSSI (Received Signal Strength Indicator) та похибки оцінки відстані до пристрою на основі експериментів з ESP32-S3 та Ubertooth One, модифіковано програмне забезпечення ubertooth-BLE для високоточної часової прив’язки пакетів,

розроблено Android-застосунок для сканування BLE-пакетів та візуалізації параметрів радіоефіру, проведено експериментальні дослідження поведінки BLE-пакетів у сценарії з завадами і без завад засобами створеного Android-застосунку з зовнішньою антеною і Python-оркестратором, створенного для керування BLE-маячками реклами з налаштовувальними параметрами.

Наукова новизна одержаних результатів полягає в розробці програмно-апаратного методу моделювання BLE-пакетів, який поєднує Android-застосунок, зовнішні BLE-антени на базі ESP32-S3 і nRF52840, а також SDR HackRF One. Запропоновано підхід до формування налаштовувальних BLE-пакетів в яких є можливість задавати: канал, інтервал, довжину, потужність та вміст корисного навантаження та експериментально встановлено особливості впливу BLE-генератора завад на втрати пакетів та ефективну пропускну спроможність каналу в умовах багатопередавацького середовища.

Галузь застосування. Запропонований метод може бути використаний для побудови військових і цивільних систем на базі технології BLE, а також оцінки стійкості IoT-мереж до радіозавад.

Ключові слова: Bluetooth Low Energy, BLE, імітаційне моделювання, конфліктна взаємодія сигналів, BLE-реклама, BLE-антена, ESP32-S3, nRF52840, HackRF One, генератор шуму, завадостійкість радіоканалу.

СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДОСЛІДЖЕННЯ

1. **Новицький, А.** (2025). Імітаційне моделювання конфліктної взаємодії інформативних і завадових радіосигналів. In XII Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології», Київ, 302–304.

2. **Novytskyi, A.**, Sokolov, V., Kriuchkova, L., & Skladannyi P. (2025). Determining the Error Distribution of BLE Beacons at Antenna Near and Far Fields. In Proceedings of the 4th Int. Conf. on Cyber Hygiene & Conflict Management in Global Information Networks (CH&CMiGIN'2025), Kyiv, vol. 4024, 133–143. (Scopus).

3. Соколов, В., **Новицький, А.**, & Бодненко, Д. (2025). Імітаційне моделювання конфліктної взаємодії BLE-пакетів. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 2(30), 662–681. https://doi.org/10.28925/2663-4023.2025.30.997 (Категорія Б).

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface ‘інтерфейс прикладного програмування’

BLE – Bluetooth Low Energy ‘Bluetooth з низьким енергоспоживанням’

CRC – Cyclic redundancy check ‘циклічний надлишковий код’

MTU – Maximum Transmission Unit ‘максимальний розмір корисного навантаження’

RSSI – Received Signal Strength Indicator ‘показник потужності прийнятого сигналу’

SDR – Software-Defined Radio ‘програмно-кероване радіо’

ВЧ – Високочастотний

ОС – операційна система

ПК – персональний комп’ютер

ПЗ – програмне забезпечення

ВСТУП

В умовах сьогодення стрімкий розвиток технологій сричинив потребу у безпроводових технологіях, що призводить до широкого впровадження малопотужних радіотехнологій короткого радіусу дії. Однією з найрозповсюдженіших і найдоступніших технологій є BLE, який спроєктовано для енергоефективного обміну даними між датчиками, мобільними та периферійними пристроями, які працюють в діапазоні 2,4 ГГц. Ця технологія використовується у побутовій електроніці, медичних та фітнес-пристроях, промислових сенсорних мережах, системах позиціонування та доступу, що робить його стандартом у використанні для великої кількості IoT-рішень.

Однак, технологія BLE працює в режимі перенасиченого радіочастотного серидовища, оскільки в одному і тому самому діапазоні працюють Wi-Fi, класичний Bluetooth, засоби радіоелектронної боротьби, тощо. Все це призводить до конфліктної взаємодії між ними, зростання рівня інтерференції – в результаті чого втрачаються пакети та падає якість зв’язку.

З точки зору інформаційної безпеки, ця технологія має найбільший пріорітет в дослідженні через свою розповсюдженність: використання як у системах медичного моніторингу, розумного дому, системах контролю доступу. Можливі вразливості протоколу, помилки реалізації, цілеспрямоване створення радіозавад можуть призвести до порушення конфіденційності, цілісності та доступності сервісів, які користуються даною технологією.

Актуальність роботи полягає в стрімкому розповсюдженні і розвитку технології BLE IoT-системах, функціонування яких відбувається в перенасиченому неліцензованому дапазоні частот 2.4 ГГц.

Метою даної роботи є підвищення інформативності та достовірності аналізу стану BLE-радіоканалу в умовах завад шляхом розроблення та експериментальної перевірки методу моделювання BLE-пакетів і конфліктної взаємодії інформативних та завадових радіосигналів на основі спеціалізованого програмно-апаратного стенду.

Для досягнення поставленої мети були поставлені та вирішенні такі **завдання**:

- досліджено принцип роботи технології BLE і методи роботи з нею;
- проаналізовано структуру BLE-пакетів та можливості стандартного BLE-стека Android;
- досліджено залежність RSSI та похибки оцінки відстані до пристрою на основі експериментів з ESP32-S3 та Ubertooth One; модифіковано програмне забезпечення ubertooth-BLE для високоточної часової прив’язки пакетів;
- розроблено Android-застосунок для сканування BLE-пакетів та візуалізації параметрів радіоефіру;
- проведено експериментальні дослідження поведінки BLE-пакетів у сценарії з завадами і без завад засобами створеного Android-застосунку з зовнішньою антеною і Python-оркестратором, створенного для керування BLE-маячками реклами з налаштовувальними параметрами.

Об’єктом дослідження безпроводовий BLE-радіоканал під впливом електромагнітних завад.

Предметом дослідження є показники якості BLE-зв’язку, ймовірність втрати пакетів, відношення сигнал-шум, похибка оцінки відстані, завантаженість каналу при різних параметрах передачі.

Розділ 1
АНАЛІЗ АРХІТЕКТУРИ, ХАРАКТЕРИСТИК ТА СУЧАСНИХ ДОСЛІДЖЕНЬ BLE ТЕХНОЛОГІЇ

1.1. Архітектура BLE технології

1.1.1. Принципи роботи BLE

BLE – безпроводова технологія передачі даних малого радіусу дії, спеціально розроблена для мінімального енергоспоживання [1, 2]. Вперше BLE було представлено у 2010 році в специфікації Bluetooth 4.0, що стало значним кроком у розвитку Bluetooth-технології [2]. На відміну від класичного Bluetooth (BR/EDR), який орієнтований на безперервну передачу даних великого обсягу (наприклад, аудіо або файли), BLE призначений для періодичної передачі невеликих обсягів даних з мінімальним навантаженням на джерело живлення пристрою [1–3].

В основі дизайну BLE лежить асиметричний розподіл ролей та енерговитрат між пристроями: малопотужні периферійні пристрої (наприклад, датчики на батарейках) виконують мінімальні обчислення і періодично передають дані, тоді як більш потужні центральні пристрої (наприклад, смартфони) беруть на себе основне навантаження з обробки та з’єднання. Такий підхід дозволяє пристроям на основі BLE працювати місяцями або навіть роками від малих батарей, що відкриває можливості для нового класу пристроїв з використанням безпроводового зв’язку [1, 2].

BLE функціонує в неліцензованому діапазоні частот 2,4 ГГц (ISM-діапазон) та використовує 40 каналів шириною 2 МГц кожний [1–3]. З них 3 канали зарезервовані для широкомовного рекламування, а решта для даних при встановленому з’єднанні [2–5]. Пристрій, що бажає бути виявленим, періодично надсилає короткі рекламні пакети на рекламних каналах. Інший пристрій, який виконує сканування, прослуховує ці канали і при виявленні потрібного пристрою може ініціювати з’єднання [3, 5].

У BLE визначено дві основні ролі: периферійна (peripheral) – це пристрій, що рекламується і може приймати вхідні з’єднання, та центральна (central) – це пристрій, що сканує і ініціює з’єднання з периферійним. Така модель зв’язку є точка-точка. Також BLE підтримує чисто широкомовний режим без встановлення з’єднання, таким чином дані передаються у рекламних пакетах на всіх, хто слухає і починаючи з Bluetooth 5.0 має можливість побудови mesh-мережі поверх BLE для покриття більших площ [2].

Протокол стеку BLE складається з контролера та хосту [1–3]. На фізичному рівні використовується модуляція GFSK, яка забезпечує енергоефективну та достатньо надійну передачу в умовах шумів. Рівень каналу зв’язку відповідає за процеси рекламування, сканування, встановлення з’єднання, а також реалізує частотне перестрибування. Це означає, що під час активного з’єднання BLE-пристрої динамічно змінюють канал передачі за певною псевдовипадковою послідовністю, що дозволяє зменшити вплив перешкод і забезпечити більш надійний зв’язок у перенасиченому ефірі[1, 3, 6].

1.1.2. Переваги та недоліки BLE

Основна перевага BLE над Wi-Fi – це значно менше енергоспоживання. Пристрої BLE можуть працювати від батареї набагато довше, оскільки більшу частину часу знаходяться в сплячому режимі і виходять на зв’язок лише для короткої передачі даних в моменти, коли це необхідно. Натомість Wi-Fi витрачає на порядок більше енергії, оскільки підтримує високошвидкісне підключення та постійно активним прийма. BLE-пристрій може передавати дані з середньою швидкістю сотні кілобіт на секунду, при цьому споживаючи мікроампери в режимі очікування, тоді як Wi-Fi забезпечує десятки чи сотні мегабіт на секунду, але вимагає активного живлення радіомодуля і типово розрахований на живлення від мережі або ємної батареї [1, 3, 4].

Також, BLE має просту комунікацію типу точка-точка, яка працює без необхідності налаштовування і запуску окремої інфраструктури: для взаємодії двох пристроїв не потрібний спільний пристрій, як у випадку Wi-Fi (BLE-пристрої

з’єднуються напряму). З іншого боку, недоліком BLE є менша дальність та пропускна здатність порівняно з Wi-Fi. Стандартний радіус дії BLE складає біля двадцяти метрів у приміщенні, тоді як сучасні Wi-Fi мережі можуть покривати від пятидесяти до ста метрів і більше при вищій потужності передавача [1–3].

Пропускна спроможність BLE обмежена. Максимальна теоретична швидкість складає приблизно 1 Мбіт/с, у новіших версіях Bluetooth 5 введено режим 2 Мбіт/с, але навіть це на кілька порядків менше, ніж у Wi-Fi, де реальна швидкість вимірюється десятками і сотнями [3, 4]. Таким чином, BLE не підходить для передавання великих обсягів даних. Водночас BLE виграє у сценаріях, де потрібні короткі обміни даними та енергоефективність, як наприклад періодичні вимірювання сенсору температури, в той час як Wi-Fi краще підходить для високошвидкісного доступу до мережі і живлення від постійного джерела [1, 3].

1.1.3. Обмеження BLE

Хоча технологія BLE оптимізована для низького енергоспоживання, вона має певні обмеження та компроміси, які необхідно враховувати при проєктуванні систем:

- в енергоспоживанні, BLE значно економніший за інші безпроводові протоколи - це досягається за умови правильного налаштування режимів роботи: більшість часу повинен перебувати у сплячому режимі, передача та прийом виконуються короткими імпульсами і при спробі організувати безперервну передачу з високою частотою обміну переваги в енергоспоживанні зникають; [1, 3];
- радіус дії BLE зазвичай обмежується десятками метрів. У специфікації Bluetooth 4.x типова дальність ~50 м на відкритому просторі при сприятливих умовах і ~10 м в умовах типового офісу чи квартири [1, 2]. У версії Bluetooth 5.0 було впроваджено так званий кодований PHY (Coded PHY, 125 кбіт/с або 500 кбіт/с) для збільшення дальності; як заявлено до чотирьох разів більший радіус дії порівняно зі стандартним

режимом [2]: в ідеальних умовах понад 100 метрів, але при цьому знижується швидкість передачі [2, 3];

– BLE обмежений у швидкості обміну даними. Теоретична пікова швидкість у режимі один Мбіт/с на фізичному рівні дає близько 0,27 Мбіт/с користувацьких даних через накладні витрати протоколу, але для завдань передавання аудіо, зображень або файлів BLE 4.0 не зможе працювати з прийнятною швидкістю. З появою Bluetooth 5 з'явився режим два Мбіт/с, який подвоїв швидкість, з'явилися пакети довжиною до 251 байту корисного навантаження, що підвищує ефективність передачі великих блоків даних [3, 4]. Також з'явилось BLE Audio з кодеком LC3, що дозволило передавати аудіо по BLE, хоча й з меншим енергоспоживанням та якістю, але оптимізованою для слухових апаратів та безпроводових навушників [2];
– BLE працює у 2,4 ГГц ISM-діапазоні, який перенасичений різноманітними пристроями, такі як: Wi-Fi, мікрохвильові печі, безпроводові телефони, деякі види радіомодулів для дронів, гральні консолі, що може спричинити певного роду завади. BLE використовує адаптивне перестрибування по каналах, щоб зменшити вплив завад - протокол може уникати їх і передавати на менш завантажених каналах [1, 3, 6];
– BLE спроєктований для відносно простої схеми, де один центральний пристрій підключається до кількох периферійних. На практиці смартфон або інший хаб може одночасно підтримувати десяток і більше BLE-з'єднань, але кожне активне з'єднання займає часові слоти і ресурси контролера. Велика кількість одночасних підключень може призвести до затримок опитування пристроїв, пропуску реклами тощо. Тому при великій кількості пристроїв доцільно використовувати BLE Mesh [2, 3].

Підсумовуючи можна сказати, що BLE досягає малої витрати енергії та достатньої швидкодії ціною обмеженої дальності та стійкості до перевантаженого ефіру [1–3, 6].

1.2. Особливості проєктування BLE-застосунків на Android

Операційна система Android забезпечує підтримку BLE починаючи з версії Android 4.3. Розробникам надається стандартний стек Bluetooth через Android Bluetooth API, який дозволяє додаткам працювати як у центральній, так і в периферійній. Архітектура BLE в Android складається зі служби Bluetooth OS (частина системи, що взаємодіє з апаратним модулем Bluetooth) та прикладного рівня API. Основні класи API включають BluetoothAdapter, BluetoothDevice, BluetoothGatt та BluetoothGattCallback, BluetoothLeScanner. На Android передбачено використання BLE переважно у центральній ролі (телефон як сканер або клієнт) і саме цей сценарій найчастіше зустрічається у застосунках, зокрема в мобільних медичних та моніторингових рішеннях [2, 5, 7].

Підтримка периферійного режиму (рекламування) з’явилась з Android 5.0, але з певними обмеженнями: зокрема, Android-пристрій може рекламувати лише один BLE-сервіс одночасно і з досить повільною частотою реклами [5].

1.2.1. Обмеження сканування та підключення

З точки зору розробника, одне з ключових обмежень – це контроль з боку ОС за тим, як часто і в якому режимі додаток може сканувати ефір BLE. Починаючи з Android 7.0 (Nougat), в систему було введено обмеження, через яке додаток не може запускати BLE-сканування занадто часто. Якщо програма викликає startScan() більше ніж п’ять разів за 30 секунд, то подальші спроби тихо блокуються системою. Це зроблено для економії батареї і запобігання зловживанню безперервним скануванням. Розробникам рекомендується не стартувати або зупиняти сканер надмірно часто, а запускати одне сканування з потрібними фільтрами [5].

Крім того, Android 8.0 запровадив обмеження для фонових режимів. Якщо додаток працює у фоні і не має активного видимого екрану, частота його доступу до BLE теж обмежується. Зокрема, без спеціальних заходів, фонове сканування на Android 8.0 може виконуватись приблизно раз на 15 хвилин. Це створює проблему для додатків, які повинні постійно слухати маячки BLE. Такі додатки вимушено

переводять у тип foreground-service, за якого система вважає, що додаток активно працює і дозволяє частіше сканувати [5, 7].

Таким чином, в Android існують суттєві обмеження частоти сканування: програмі потрібно ретельно спланувати свою роботу з BLE, щоб не бути обмеженою системою. Це відрізняє мобільну платформу від простих вбудованих систем, де розробник може сканувати як завгодно часто. На Android необхідно дотримуватися стратегій, які ощадливо використовують заряд батареї [5].

1.2.2. Особливості GATT-взаємодії та багатопоточності

Android BLE-стек не підтримує одночасне виконання кількох операцій; більше того, якщо викликати BLE-операції занадто швидко одна за одною, можуть виникати збої (неочікувані помилки статусу, від'єднання тощо) [5]. Вбудованої черги запитів BLE не має і за її реалізацію і розробку відповідає розробник застосунку.

Таким чином, з точки зору розробки, багатопоточність у BLE на Android потребує створення власної черги операцій або використання готових бібліотек, що це реалізують. Оскільки всі BLE-виклики в Android зрештою виконуються через один системний потік до апаратного контролера, спроби пришвидшення роботи призводять лише до колізій.

1.2.3. Додаткові обмеження та вимоги

Розробляючи BLE-додаток під Android, слід пам'ятати про низку системних вимог.

По-перше, починаючи з Android 6.0, для сканування BLE потрібен дозвіл на геолокацію 'ACCESS_FINE_LOCATION', а в Android 12+ введено окремі дозволи 'BLUETOOTH_SCAN' і 'BLUETOOTH_CONNECT', що розділяють права на сканування і на підключення до пристроїв.

По-друге, для роботи BLE у фоні додаток має оголосити, що використовує foreground-service з типом bluetoothScanning, інакше система може взагалі зупинити сканування через кілька хвилин після переходу в фон.

По-третє, Android може знищувати або відключати BLE-активність додатка під час режимів економії енергії [5, 7].

При розробці застосунків з використанням BLE під Android необхідно враховувати політики ОС щодо енергозбереження: платформа намагається мінімізувати вплив постійно працюючих BLE-додатків на час роботи телефону від батареї. Це продемонстровано мобільними системами моніторингу стану здоров'я, де BLE-датчики взаємодіють з Android-додатком, а останній повинен одночасно бути енергоефективним і достатньо чутливим до подій [7].

1.3. Огляд сучасних наукових досліджень

BLE-технологія продовжує розвиватися, і за останні роки з'явився значний пласт наукових досліджень, присвячених як удосконаленню самої технології, так і аналізу її вразливостей та ефективності в різних сценаріях [6, 8–10]. Розглянемо кілька актуальних напрямків досліджень, що отримали особливу увагу: безпека BLE, вплив зовнішніх завад і питання коексистенції, надійність зв'язку в мережах BLE, а також підвищення ефективності передачі даних.

1.3.1. Безпека та конфіденційність BLE

З огляду на широке розповсюдження BLE-пристроїв (від медичних сенсорів до замків і трекерів), питання захищеності з'єднань є надзвичайно важливим [2, 8–10]. У BLE передбачені механізми безпеки – по-перше, шифрування трафіку (128-бітовий ключ AES-CCM) після процедури парування, по-друге, можливість автентифікації при паруванні і використання динамічних приватних адрес, що змінюються і ускладнюють відстеження пристрою [2, 8].

Однак дослідники виявили низку уразливостей та провели аналіз стійкості BLE до атак. Зокрема, ранні версії BLE (Bluetooth 4.0/4.1) мали спрощену схему парування Just Works, яка не забезпечувала захист від атак типу Man-in-the-Middle (через відсутність справжньої взаємної автентифікації). Навіть після впровадження Secure Connections (Bluetooth 4.2, з використанням ECDH для обміну ключами)

низка атак залишилася можливою на етапі повторного з’єднання та збережених зв’язків [8–10].

Наприклад, у 2020 р. була оприлюднена атака BLESA (BLE Spoofing Attack), яка використовувала пропуск перевірки на рівні повторного встановлення захищеного з’єднання, що дозволяло зловмиснику непомітно вставити свої пакети у потік даних деяких пристроїв [9]. Інші практичні атаки включають пасивне прослуховування (passive eavesdropping) незашифрованих або слабо захищених з’єднань, MITM-атаки при паруванні з підробкою вводу-виводу, а також експлуатацію вразливостей стеків (наприклад, сімейство вразливостей SweynTooth, що дозволяють викликати зависання або перезавантаження пристроїв через некоректно сформовані пакети) [8, 9]. Окремо виділяються атаки на носимі пристрої та фітнес-трекери, зокрема стандарто-сумісні MITM-атаки на BLE-канали [10].

У відповідь на ці загрози дослідники та консорціум Bluetooth SIG вдосконалюють протокол: сучасні версії вимикають за замовчуванням небезпечні режими, рекомендують використання захищеного парування з підтвердженням (Numeric Comparison або OOB), вводять додаткові функції, як-от Resolvable Private Addresses (RPA) – динамічні адреси пристроїв, що шифруються на основі приватного ключа і змінюються кожні кілька хвилин, а також шифрування рекламних даних у нових версіях стандарту [2, 8–10].

У науковій спільноті з’явилося кілька оглядових робіт, що систематизують стан безпеки BLE. Зокрема, є робота, в якій детально аналізуються вразливості Bluetooth-технології у контексті IoT-застосувань і розглядаються практичні сценарії атак [8].

Активно досліджуються і нові можливості BLE у сфері безпеки, наприклад, використання BLE для двофакторної автентифікації, цифрових ключів (Bluetooth Smart Lock) тощо [2, 8]. Важливо, що Bluetooth 5.3/5.4 та BLE Audio (Bluetooth 5.2) також привнесли поліпшення безпеки: зокрема, методи шифрування рекламних даних і покращений контроль над процесами парування та перепідключення [2].

Загалом, безпека BLE є прийнятною для більшості сценаріїв, однак потребує правильної імплементації. Рекомендовано завжди використовувати найвищий рівень захищеності при створені пари, оновлювати прошивки пристроїв до актуальних версій і бути обережним з даними, що передаються у відкритому вигляді [2, 8–10].

1.3.2. Надійність і якість зв’язку

Надійність BLE-з’єднання залежить від багатьох факторів: відстані, перешкод, швидкості руху пристроїв, навантаження мережі, реалізації стеку тощо [1–3]. У наукових працях досліджується як покращити надійність передачі та зменшити затримки.

BLE в промислових додатках розглядається як альтернатива іншим низькопотужним технологіям: експериментальні дослідження показують, що BLE може задовольнити вимоги до затримок і пропускної здатності для багатьох сенсорних сценаріїв за умови невеликої відстані й відсутності сильних завад, часто перевершуючи ZigBee за пропускною здатністю та затримкою [1, 3, 4].

Аналіз затримок і втрат пакетів демонструє, що менший інтервал з’єднання знижує латентність обміну, але збільшує навантаження і ймовірність колізій, тому існує оптимальний баланс, що залежить від вимог до застосунку [3, 4]. У новіших версіях BLE з’явився механізм LE Isochronous Channels (Bluetooth 5.2) для передачі даних з фіксованим інтервалом (наприклад, аудіо), який також аналізується з точки зору надійності каналу [2, 4].

Щодо мобільності пристроїв, дослідження показують, що при швидкому русі стандартні параметри BLE можуть не встигати підтримувати зв’язок; налаштування меншого інтервалу реклами і сканування покращує ситуацію ціною вищого енергоспоживання [3, 5].

Окремою темою є надійність mesh-мереж BLE, де повідомлення передаються через проміжні вузли, а час доставки та ймовірність успіху істотно залежать від параметрів ретрансляції, топології та фонових завад [1–3]. На рівні GATT досліджуються способи підвищити надійність обміну даними – збільшення MTU,

використання підтверджень для критичних записів, дублювання важливих повідомлень тощо [3–5].

1.3.3. Ефективність зв’язку та майбутні удосконалення

Під ефективністю розуміють як відношення пропускної здатності до споживаної енергії, так і здатність протоколу максимально використовувати доступний спектр [1, 3, 4]. У контексті BLE багато досліджень зосереджено на аналізі пропускної спроможності різних версій протоколу. Так, Bulić та ін. експериментально виміряли фактичну швидкість передачі даних для BLE 4.2 та BLE 5 при різних параметрах і показали, що збільшення MTU та використання Data Length Extension суттєво підвищує ефективність: BLE 4.2 з MTU 23 байти дає ~0,27 Мбіт/с, а з MTU 185 байт – вже ~0,8 Мбіт/с на прикладному рівні [3,4]. BLE 5 на PHY 2M продемонстрував подвійну швидкість – до ~1,4–1,6 Мбіт/с корисних даних за оптимальних умов [4].

Еволюція стандарту суттєво покращила ефективність передачі, але такі покращення вимагають йти на компроміси: збільшення MTU потребує більше пам’яті та обчислювальних ресурсів, а PHY 2M зазвичай має менший радіус дії [1, 3, 4].

Окремим напрямом є LE Audio (Isochronous Channels), де досліджується ефективність передавання аудіопотоків BLE при одночасній підтримці кількох пристроїв (наприклад, трансляція на два навушники) з прийнятною якістю і низьким енергоспоживанням [2, 4].

В майбутньому очікується, що BLE набуватиме нових сфер – наприклад, пряма підтримка IP-пакетів, інтеграція з технологією Matter для розумного дому, де BLE використовується для первинної конфігурації пристроїв [1, 2].

Серед актуальних напрямків досліджень є застосування машинного навчання і штучного інтелекту для оптимізації роботи BLE: інтелектуальне керування енергоспоживанням, адаптивне керування потужністю передавача для економії енергії, використання BLE-маяків у комплексі зі штучним інтелектом для задач позиціонування й відстежування об’єктів у просторі [1, 6].

BLE за десятиліття існування пройшов шлях від нової експериментальної технології до зрілої платформи, яка постійно вдосконалюється. Сучасні наукові праці охоплюють усі аспекти BLE: від фізичного рівня (моделювання енергоспоживання, завадостійкість) до рівня застосувань (протоколи поверх BLE, безпека сервісів) [6–10]. Особлива увага приділяється безпеці, надійності та масштабованості, а також інтеграції BLE в медицину, промисловість 4.0 та мобільні застосунки [1, 2, 7].

Висновки до першого розділу

В першому розділі описано роль технології BLE у сучасній екосистемі безпроводових рішень. Шляхом аналізу архітектури та принципів роботи BLE продемонстровано, що ця технологія спирається на асиметричний розподіл ролей між центральними та периферійними пристроями, роботу в діапазоні 2,4 ГГц з використанням частотного перестрибування та короткі сеанси обміну даними. Такий підхід дає змогу досягати дуже низького енергоспоживання за умови правильноно налаштування інтервалів реклами, параметрів з’єднання та режимів сну, що робить BLE оптимальною технологією для сенсорних мереж, носимих пристроїв, тощо.

Однак енергоефективність BLE досягається у вигляді обмеженого радіусу дії, невисикою пропускною здатністю, залежністю від завад у ISM-діапазоні і чутливості до конфігурації мережі. Навіть з появою нових рішень ця технологія залишається орієнтованою насамперед на періодичну передачу невеликих обсягів даних. У випадку масштабування до великої кількості вузлів вимагає побудову відповідних до умов мережевих рішень.

Також, особливої уваги вимагає використання BLE на платформі Android. Хоч на цій платформі зручне API, робота з ним поєднана з жорсткими обмеженнями з боку ОС, через які розробник вимушений адаптовувати архітектуру засостунку і мережі до полік енергозбереження та керування ресурсами.

Вищенаведений огляд технології і її обмежень окреслює проблемне поле, яке і стає предметом подальших досліджень у наступних розділах роботи.

Розділ 2
РОЗРОБКА ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ МЕТОДУ МОДЕЛЮВАННЯ BLE-ПАКЕТІВ

2.1. Розбір BLE-пакетів на прикладі пристрою Xiaomi MiKettle

Першим кроком розробки методу моделювання та тестування BLE-пакетів стало детальне вивчення структури реальних рекламних кадрів ‘advertising packets’, що передаються побутовими IoT-пристроями. Для цього були перехоплені та проаналізовані BLE-пакети від смарт-чайника Xiaomi MiKettle, а також від інших пристроїв, які знаходилися в радіусі дії приймача.

Для подальшого аналізу обрано декілька сирих (raw) пакетів, представлених у вигляді HEX-послідовностей:

MiKettle:

```
02 01 06 14 16 95 FE 71 20 5C 04 06 39 4B AA 6F 7C B8 09 05 10 02 00 18 0B 03 01 AA 02 AA 03
AA 0A 18 E8 FE 05 12 06 00 80 0C 09 09 4D 69 4B 65 74 74 6C 65 00 00 00 00 00 00 00 00 00 00
```

Інший девайс:

```
1C FF 06 00 01 09 21 0A 40 C8 05 24 DD 0C 44 45 53 4B 54 4F 50 2D 41 31 4B 4B 37 52 4C 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Одночасний аналіз двох сирих пакетів обумовлено необхідністю порівняння структури та виділення спільних та відмінних полів, насамперед у секції «Service Data».

Як основу для аналізу структури кадрів було використано загальну модель BLE-пакета на фізичному рівні (BLE advertising PDU), як включає в себе наступні основні поля:

1) Preamble: преамбула (1 байт для LE 1M, 2 байти для LE 2M), що слугує для синхронізації приймача з передавачем;
2) Access Address (4 байти): ідентифікатор з’єднання/каналу; для рекламних кадрів використовується фіксоване значення 0×8E89BED6;
3) PDU (Protocol Data Unit): включає 16-бітний заголовок (header) та корисне навантаження (payload); в заголовку містяться службові біти (тип кадру,

довжина, службові прапорці), у payload – дані сервісів, ідентифікатори, ім’я пристрою тощо;

4) CRC (24 біти): контрольна сума для виявлення помилок при передачі.

Розбір перехопленого пакету від пристрою «MiKettle» має наступний вигляд:

– 0×020106 прапорці: «LE General Discoverable» + «BR/EDR Not Supported» (тобто LE-only);
– 0×1416» – початок частини пакету «Service data» 16 біт UUID (оскільки маємо байт 0×16) довжиною 0×14;
– 0×95FE – це UUID, який призначено для Xiaomi Inc.;
– Все наступне після UUID йде MiBeacon-payload(це формат Xiaomi), в якому:
    - FC = 0×2071 – «frame control» (бітове поле, яке каже чи є в пакеті MAC, capability, чи шифровано тощо);
    - ProductID – 0×045C;
    - Cnt = 0×06 (лічильник кадрів);
    - MAC (reverse) = B8:7C:6F:AA:4B:39 (MAC передається у зворотному порядку байтів);
    - Capability = 0×09 (вендорське бітове поле можливостей);
    - Object: Type = 0×1005, Len = 0×02, Value = 0×0018 – значення в термінах Xiaomi – це невідомі дані, оскільки Xiaomi їх не описало, але є можливість ознайомитись з інформацією наведеною в публікації «Reverse engineering the MiBeacon protocol» [11];
– 0×0B03 – це список шістнадцяти бітного стандарту UUID-ідентифікаторів, які вказують що це за сервіс (0×03), 5 штук: 0×AA01, 0×AA02, 0×AA03 (вендорські), 0×180A (Device Information Service), 0×FEE8 (16-бітний UUID, призначений членові SIG — Quintic Corp) [12];
– 0×05120600800C – значення яке вказує мінімальне і максимальне значення інтервалу з’єднання до центрального пристрою (0×12): min=0×0006 = 7.5 мс, max=0×0C80 = 4000 мс (одиниця – 1.25 мс);
– 0×09094D694B6574746C65 – Complete Local Name як раз дорівнює “MiKettle” (рис. 2.1).

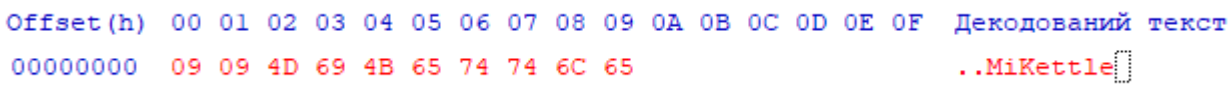


Рис. 2.1. Декодування Complete Local Name засобами HxD Editor

На основі цього аналізу було сформовано перелік полів, які можна отримати засобами стандартного BLE-стека Android через ScanResult та ScanRecord: MAC-адресу пристрою, RSSI, часові мітки, параметри PHY (1M/2M/Coded), інформацію про тип реклами (legacy, connectable, advertising SID, Tx Power тощо), а також сирі байти Service Data та UUID-ів. Разом з тим, Android-API не надає доступу до критично важливих полів фізичного рівня – преамбули, Access Address та номера каналу, що в свою чергу унеможливлює створення власної структури пакетів, яка відрізняється від загальної моделі BLE-adv.

Як проміжний висновок можна сказати, що розбір рекламного пакета MiKettle засобами Android дозволяє отримати лише частину полів BLE-кадру (переважно рівень GAP/GATT), тоді як інформація фізичного рівня, необхідна для подальшого імітаційного моделювання конфліктної взаємодії сигналів (канал, преамбула, access address), виявляється недоступною. Це вказує на необхідність використання інших апаратних платформ для глибшого доступу до радіоефіру.

2.2. Дослідження RSSI та можливостей оцінки відстані

Наступним етапом є дослідження можливості оцінки відстані до BLE-пристрою на основі рівня прийнятого сигналу RSSI, а також як поводиться похибка такої оцінки в ближній та дальній зонах антени.

2.2.1. Означення, початкові умови та вихідні дані

RSSI – показник потужності прийнятого сигналу в дБмВт, що відображає потужність сигналу відносно 1 мВт. Значення близько до мінус 30 дБмВт відповідають дуже сильному сигналу (пристрій знаходиться близько), значення близько мінус 90 дБмВт відповідно дуже слабкому сигналу, наближеному до межі чутливості приймача.

Для можливості та зручності проведення дослідження, обрано наступне апаратне забезпечення:

– плата розробника ESP32-S3 N16R8 Type-C з інтегрованим модулем Wi-Fi + BT + BLE;
– плата Ubertooth One з утилітою ubertooth-BLE для захоплення та аналізу BLE-трафіку.

Плата розробника ESP32 використана як контрольоване джерело BLE-рекламних пакетів з попередньо запрограмованим вмістом та параметрами передавання. Відповідна прошивка, код якої наведено в Додатку А, формувала рекламні кадри з довільними даними (manufacturer data) та передавала їх з фіксованою потужністю передавача.

2.2.2. Модифікація утиліти ubertooth-BLE для точних вимірювань часу

Стандартний вивід утиліти ubertooth-btle, яка використовується для взаємодії з пристроєм Ubertooth One, містить час у секундах, що унеможливлює вимірювання з необхідною точністю в експериментах, які пов’язані з часовими інтервалами між пакетами та аналізом сплесків похибки в ближній і дальній зонах. Для усунення цього недоліку було модифіковано вихідний код бібліотеки libubertooth (файл libubertooth/src/ubertooth_callback.c), після чого для кожного прийнятого пакета виводиться час у наносекундах.

На початку функції cb_BTLE(), яка є колбек-функцією, що спрацьовує при знайдені BLE-пакету, додані змінні:

long long nanoseconds; struct timespec tp;

та блок ініціалізації:

```
if (infile == NULL) {
    systime = time(NULL);
    clock_gettime(CLOCK_REALTIME, &tp);
    nanoseconds = (long long)tp.tv_sec * 1000000000LL + tp.tv_nsec;
}
```

Також, підключено заголовки:

```
#include <inttypes.h>
#include <time.h>
```

Наведені зміни дозволяють модифікувати вивід утиліти ubertooth-BTLE для підвищення точності дослідження, а саме, зміни виводу часу отримання пакету з секунд на наносекунди. Стандартний вивід програмного засобу має вигляд:

```
printf("systime=%u freq=%d addr=%08x delta_t=%.03f ms rssi=%d\n",systime,rx->channel+2402, lell_get_access_address(pkt),ts_diff / 10000.0, rx->rssi_min - 54);
```

в якому ми бачимо *systime* в секундах, що не підходить за рівнем точності для наших вимірів. Для виводу наносекунд, необхідно видозмінити вище наведений код на наступний:

```
printf("nanosystime=%lld systime=%u freq=%d addr=%08x delta_t=%.03f ms rssi=%d\n", nanoseconds, systime, rx->channel + 2402, lell_get_access_address(pkt),ts_diff / 10000.0, rx->rssi_min - 54);
```

Після виконаних дій, проєкт перезібрано згідно офіційної документації наступними командами:

```
$ mkdir build
$ cd build
$ cmake ..
$ make
$ sudo make install
```

В результаті кожен рядок виводу ubertooth-BTLE містить додаткове поле на початку з часом у наносекундах, що підвищило точність у подальших вимірюваннях та дозволило вимірювати різницю між часом передавання та приймання пакетів.

2.2.3. Схема експерименту та автоматизація збору даних

Експеримент проводився в умовах відкритого простору з використанням мобільного комп’ютера Intel NUC Kit, ESP32-S3 та Ubertooth One, які розташовувалися на висоті 1 м від землі, при цьому відстань між ними змінювалась у діапазоні від 0 до 10 м з кроком 25 см.

Порядок взаємодії був таким:

1. До мобільного комп’ютера підключалися Ubertooth One та ESP32 (рис. 2.2).
2. Керування запуском/зупинкою сканування та генерації пакетів виконувалося автоматизовано за допомогою Bash-скрипта.

3. На кожній фіксованій відстані скрипт:
   - запускав ubertooth-BLE з апаратним фільтром за MAC-адресою ESP32;
   - запускав скрипт send_packets.py, код якого наведено в Додатку Б, для генерації серії рекламних кадрів за допомогою команд, які відправлялись через Serial-порт до ESP32-S3, в якому прошивка оброблювала їх і виконувала, код якої наведено в Додатку А;
   - після завершення відправлення чекав 3 с, завершував роботу ubertooth-BLE та переміщував файли логів у окремий каталог, назва якого відповідала поточній відстані (наприклад, «50 sm»).

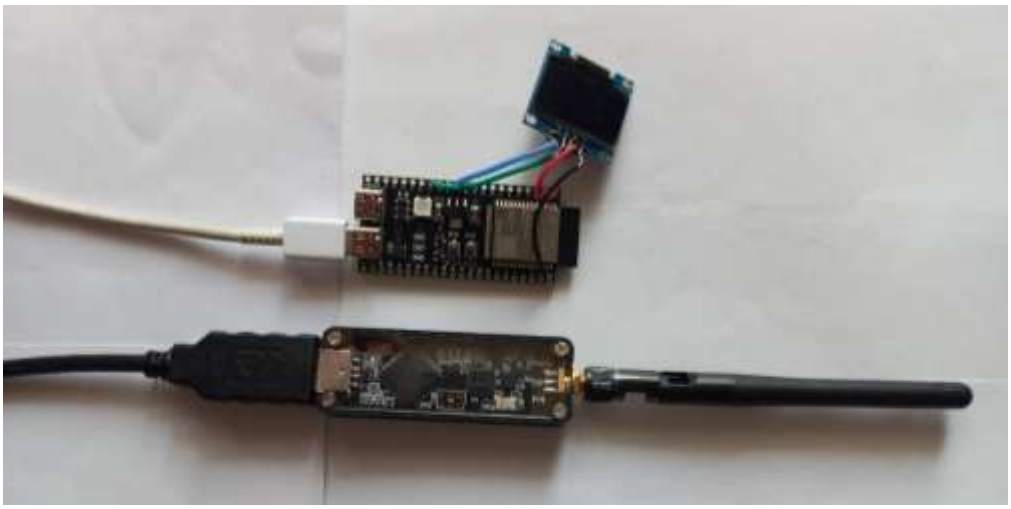


Рис. 2.2. Засоби проведення експерименту Ubertooth One і ESP32-S3

Таким чином даний експеримент дозволяє отримати великий набір даних (~41 тис. пакетів), який містить співставлення між реальною відстанню, виміряними значеннями RSSI та кількістю прийнятих пакетів.

2.2.4. Обґрунтування вибору апаратного забезпечення

Основним критерієм вибору апаратного забезпечення є: наявність SDK та прикладів коду до них, доступність в продажу та ціна. Під час оцінки всіх критеріїв при виборі доступних плат для передавача розглянуто різні пристрої, такі як: SDR, Bluetooth-адаптери та одноплатні комп’ютери, однак вибір припав на сімейство мікроконтролерів ESP32. В табл.2.1 наведено характеристики різних модифікацій мікроконтролерів [13, 14].

Таблиця 2.1. Порівняння характеристик мікроконтролерів ESP32

Модель	Процесор	Частота процесора, MHz	RAM, кБ	Діапазони частот
ESP32-C3	RISC-V, 32-bit	160	400	BLE 5 / Wi-Fi 2.4
ESP32-S3	Dual-core Xtensa LX7, 32-bit	240	512	BLE 5 / Wi-Fi 2.4
nRF52832	ARM Cortex-M4F, 32-bit	64	64	BLE 5
nRF52840	ARM Cortex-M4F, 32-bit	64	256	BLE 5 (2M/LR) / 802.15.4
HackRF One	NXP LPC4320 (M4/M0), 32-bit	204	264	1–6 ГГц

Згідно з табл.2.1, вибір плати ESP32-S3 N16R8 Type-C є вигідним: двоядерний процесор LX7 дозволяє паралельно обробляти кілька завдань (наприклад, отримувати команди та генерувати пакети), великий обсяг SRAM та flash-пам’яті дозволяє кешувати достатній обсяг даних, і програмний код може бути об’ємнішим, що робить цю плату універсальною для завдань спостереження. З усіх доступних у продажу плат, плата ESP32-S3 N16R8 Type-C є найбільш універсальною для двоядерного процесора. Крім того, деякі плати оснащені роз'ємами U.FL, які дозволяють використовувати зовнішні антени.

2.2.5. Аналіз результатів та оцінка відстані за RSSI

В результаті проведення експерименту, побудовано таблицю залежності похибки від відстані. Аналіз демонструє наявність вираженого піку помилок у зоні малої відстані, яка ще називається ближньою зоною (~1 м для частоти 2,402 ГГц). В цій області спостерігався підвищений розкид RSSI та збільшена кількість помилок і втрати пакетів. При збільшенні відстані до дальньої зони похибка зменшувалась і стабілізувалась.

Також, в Android-застосунку було використано класичну логарифмічну модель втрат для оцінки відстані на основі RSSI:

```
protected static double calculateDistanceOld(double txPower, double rssi) {
    double n = 2.0;
    if (rssi == 0) {
        return -1.0;
    }
    return Math.pow(10, (txPower - rssi) / (10 * n));
}
```

яка застосовується наступним чином:

```
calculateDistanceOld(-59, res.getRssi())
```

В математичному виражені ця функція має наступний вигляд [15]:

$$d = 10^{\frac{txPower - RSSI}{10n}} \quad (2.1)$$

де $txPower$ визначено як типове значення «Measured Power для iBeacon», мінус 59 дБмВт, n залежить від умов навколишнього середовища:

– 2,0 – вільний пустий простір;
– 2,0–2,4 – у кімнаті або офісі;
– 2,5–4 – Перешкоди, стіни, люди, дерева, тощо;
– < 2,0 – коли прямий «коридор» між антеною-прийому і передатчиком.

Отже, як проміжний висновок можна сказати, що отримані результати демонструють неможливість правильного розрахунку і аналізу BLE-позиціонування без врахування впливів ближньої і дальньої зон; побудовано легко-повторюваний експеримент, який є дешевим, простим у реалізації і показовим.

2.3. Аналіз прошивки Ubertooth One як кандидата на роль антени для Android

Під час проведення дослідження було встановлено обмеження Android-стеку, яке вказує на неможливість доступу до фізичного рівня за допомогою SDK, що наданий в середовищі операційної системи – прийняте рішення про необхідність підключення зовнішньої антени до смартфону. Так як в вище описаному експерименті USB-донгл Ubertooth One зарекомендував себе як доступний і надійний BLE-сніфер, прийнято рішення про дослідження можливості використання його як зовнішньої антени для Android-застосунку.

2.3.1. Організація середовища дослідження і відлагодження прошивки

Для прошивки та налагодження було використано одноплатний комп’ютер Raspberry Pi 4, до якого підключено JTAG-виводи Ubertooth One (рис. 2.3), після чого налаштовувався openOCD та запускалась gdb із попередньо зібраною

прошивкою без оптимізацій, але з вбудованою відлагоджувальною інформацією, що дає можливість покроково відлагоджувати прошивку.

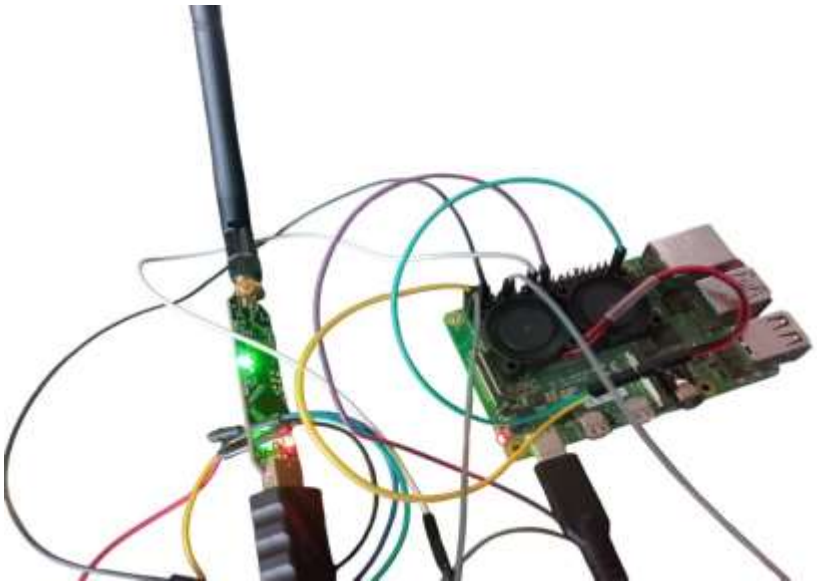


Рис. 2.3. Підключення Ubertooth One до Raspberry Pi 4

Також використовувались вихідний код прошивки і схеми PCB, які лежать в цьому репозиторії, за допомогою яких можливо будь-кому надрукувати плату (рис. 2.4) [16].

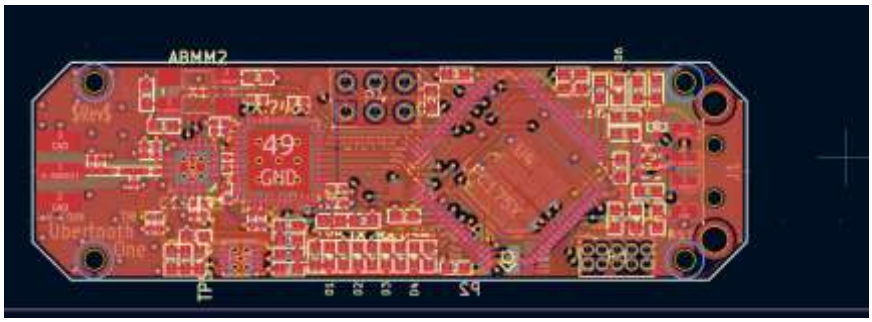


Рис. 2.4. PCB-схема Ubertooth One

Також було задіяно мультиметр, за допомогою якого замірювались супротиви, індукційності, виконувалась «продзвонка» доріжок плати – все це виконано за допомогою збільшувального скла для підвищення точності (рис. 2.5).

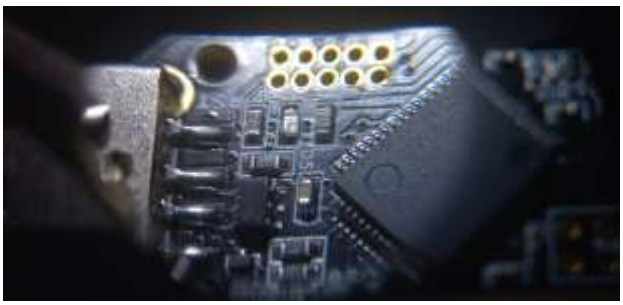


Рис. 2.5. JTAG і прийомний тракт Ubertooth One

Як результат було отримано дослідний стенд, в якому засобами покрокового відлагодження було здійснено дослідження прошивки Ubertooth One, її недоліки і переваги, принцип роботи і вузькі місця. Також, такий метод дозволяє напряму записувати прошивку на пристрій за допомогою DFU-роз'єму, без використання USB кабелю – це допомагає у випадку пошкодження прошивки пристрою при якому бутлоадер вже не запускається.

2.3.2. Структура обробки BLE-трафіку у прошивці

Проведений огляд і вивчення коду прошивки, а також, покрокове відлагодження надало розуміння, що на стороні хоста (ПК або інший керуючий пристрій) утиліти сімейства ubertooth-* формують виклики у вендорні USB-control-запити з кодами команд, наприклад, «UBERTOOTH_BTLE_SNIFFING» (режими follow/adv) або «UBERTOOTH_BTLE_PROMISC» (повний проміскуїтет). Саме ці команди переводять прошивку у режими захоплення BLE-трафіку. На стороні мікроконтролера вони обробляються у хендлері control-запитів (endpoint 0), де по полю bRequest обирається відповідна гілка обробки (cmd_poll, cmd_BTLE_sniffing, cmd_set_channel тощо – все це команди керування режимами роботи і налаштуваннями радіочастотного чипу).

Буферизація даних організована у вигляді FIFO-черг: після приймання пакета він потрапляє до внутрішньої черги обробки, а потім, після мінімальної обробки, – у чергу відправки на хост. Це дозволяє теоретично реалізувати потокову обробку значної кількості пакетів.

Було виконано спробу збільшити розмір внутрішньої черги через зміну макросу #define FIFOSIZE. Однак практичні випробування показали, що

збільшення розміру FIFO обмежується доступним обсягом оперативної пам’яті мікроконтролера: при помірному збільшенні розміру черги спостерігалось погіршення роботи у вигляді зростання втрати пакетів приблизно на 27%.

Як проміжний висновок можна сказати, що під час дослідження виникла підозра у відсутності асинхронності роботи, що проявлялось у наступному: під час отримання пакету на «хості», тобто читання з FIFO-черги, пакети в радіоефірі не оброблювались, що досить суттєво впливає на продуктивність і збільшує втрату пакетів, що є досить поганим показником і вимагає подальшого дослідження. Ubertooth One є зручним інструментом для первинного аналізу, однак у ролі високопродуктивної антени для інтеграції з Android застосунками та реалізації експериментального стенду він має обмеження, які вимагають додаткового дослідження і вдосконалення – все це є мотивацією для пошуку інших апаратних платформ.

2.4. Реалізація сканування BLE-пакетів та візуалізації інформації в інтерфейсі Android-застосунку

Після аналізу апаратних засобів реалізована програмна частини Android-застосунку для сканування BLE-рекламних пакетів та базової візуалізації отриманої інформації. Це ПЗ використовується як частина експериментального стенду для подальшого моделювання та тестування методів обробки сигналів.

2.4.1. Початкова реалізація програмного засобу сканування BLE засобами Android

Згідно з поставленим завданням – сканування BLE ефіру, прийнято рішення про створення Android-застосунку за допомогою мови програмування Java, яке:

- перевіряє наявність та стан Bluetooth-адаптера;
- запитує необхідні runtime-дозволи;
- запускає безперервне сканування BLE-пакетів;
- зчитує RSSI кожного пакету та оцінює орієнтовну відстань до пристрою;
- зупиняє сканування при переході активності у фоновий режим.

Програмний код застосунку наведено в github репозиторії [17].

Вхідною точкою застосунку є метод «onCreate()» класу «MainActivity», який має наступний вид:

```
@RequiresPermission(allOf={Manifest.permission.BLUETOOTH_CONNECT,Manifest.permission.BLUETO
OTH_SCAN, Manifest.permission.BLUETOOTH_CONNECT})
        @Override protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            startScanButton = findViewById(R.id.scanButton);
            stopScanButton  = findViewById(R.id.stopButton);
            loadAverageText = findViewById(R.id.loadAverageText);
            ListView lv = findViewById(R.id.deviceListView);
            listAdapter = new ArrayAdapter<>(this,
                    android.R.layout.simple_list_item_1,
                    new ArrayList<>());
            lv.setAdapter(listAdapter);
            lv.setOnItemClickListener((parent, view, position, id) -> {
                String text = listAdapter.getItem(position);
                String[] parts = text.split("\\s+|\\[|]");
                String mac = parts[0];
                String type = parts.length > 2 ? parts[2] : "Generic";
                Intent intent = new Intent(MainActivity.this, DeviceDetailActivity.class);
                intent.putExtra("mac", mac);
                intent.putExtra("packetType", type);
                startActivity(intent);
            });
            subscription = processor
                    .onBackpressureBuffer(1_000_000,()->Log.w(TAG, "BLE buffer overflow!"),
                     BackpressureOverflowStrategy.DROP_OLDEST)
.observeOn(Schedulers.io(), false, Runtime.getRuntime().availableProcessors())
                    .subscribe(this::heavyScanResult,throwable->Log.e(TAG,"Rx
error",throwable));
            checkEverythingAndStart();
    }
```

В першу чергу анотація «@RequiresPermission» є підказкою, що метод потребує дозволів «BLUETOOTH_CONNECT» і «BLUETOOTH_SCAN» для правильної роботи застосунку, без цих дозволів застосунок втрачає свою стабільність в роботі.

Всередині цього методу є наступні головні ділянки коду:

– setContentView(R.layout.activity_main) – підключення розмітки «activity_main.xml», яка вказує яким чином застосунок відображає користувацький інтерфейс;
– startScanButton, stopScanButton, loadAverageText – елементи користувацького інтерфейсу, на які навішується коллбек функції для можливості користувача взаємодіяти з застосунком;
– Налаштування RX підписки:
  o onBackpressureBuffer() – буфер з політикою «DROP_OLDEST» (якщо переповнення, то видаляються старі події, при цьому виводиться warning в консоль подій);
  o observeOn() – важка обробка йде в I/O-потоки, які виділені системою;
  o subscribe() – кожен ScanResult (або інша сутність) обробляється в методі heavyScanResult();
– checkEverythingAndStart() – ініціалізація самої логіки застосунку(кнопки, поведінка, тощо).

Метод checkEverythingAndStart() є завершальним етапом реалізації вхідної точки застосунку і в деякому розумінні «роздоріжжям» в подальших реалізаціях поведінки застосунку, має наступний вид:

```
    private void checkEverythingAndStart() {
    startScanButton.setOnClickListener(v -> {
        android.app.AlertDialog.Builder builder = new
android.app.AlertDialog.Builder(this);
        builder.setTitle("Виберіть режим")
       .setItems(new CharSequence[]{"Стандартний","Тест","USB CDC ESP32"}, (dialog, which) -
> {
             BluetoothAdapter bt;
             switch (which) {
                   case 0:
                         if (!havePerms()) { ActivityCompat.requestPermissions(this,
RUNTIME_PERMS, REQ_PERMS); return;}
                         if (!isLocationOn()) { askToEnableLocation(); return;}
                         bt = BluetoothAdapter.getDefaultAdapter();
                         if (bt == null) { toast("Цей пристрій не має Bluetooth"); return;
}
                         if (!bt.isEnabled()) {
                               startActivityForResult(new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE), REQ_ENABLE_BT);
                               return;
                         }
                         BleScanManager.getInstance().start(this);
                         startScanButton.setVisibility(View.GONE);
                         stopScanButton.setVisibility(View.VISIBLE);
                         break;
                   case 1:
                         if (!havePerms()) { ActivityCompat.requestPermissions(this,
RUNTIME_PERMS, REQ_PERMS); return;}
```

```
                        if (!isLocationOn()) { askToEnableLocation(); return;}
                        bt = BluetoothAdapter.getDefaultAdapter();
                        if (bt == null) { toast("Цей пристрій не має Bluetooth"); return;
}
                        if (!bt.isEnabled()) {
                              startActivityForResult(new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE), REQ_ENABLE_BT);
                              return;
                        }
                        startActivity(new Intent(this, TestModeActivity.class));
                        break;
                  case 2:
                        startActivity(new Intent(this, UsbReceiverActivity.class));
                        break;
            }
       })
       .show();
     });
     stopScanButton.setOnClickListener(v -> {BleScanManager.getInstance().stop();
stopScanButton.setVisibility(View.GONE);startScanButton.setVisibility(View.VISIBLE);});
     }
```

Даний метод виконує роль селектора різних типів активностей застосунку(тобто сутностей, які працюють за різною логікою і можуть бути застосовані для вирішення різних типів задач), реалізованого через діалогове вікно, в якому користувачу дається можливість обрати подальшу поведінку застосунку (рис. 2.6).

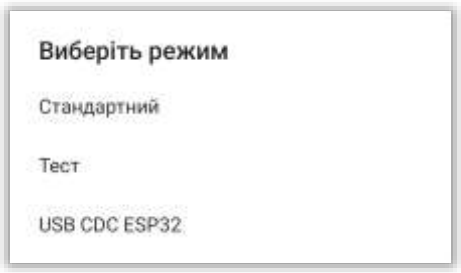


Рис. 2.6 Вибір режиму роботи Android-застосунку

Застосунок дозволяє працювати в наступних трьох режимах:

– Стандартний – в цьому режимі застосунок для сканування використовує внутрішню Bluetooth-антену телефону, при цьому перевіряючи всі «runtime-дозволи» за допомогою методу havePerms() (для доступу до функціоналу Bluetooth, геолокації тощо); вся логіка отримання BLE-пакетів реалізована і виконується самим застосунком;
– Тест – даний режим реалізовано для фізичного експерименту, який описано в Розділі 2.2.3 даної роботи, в якому відбувається підключення

до точки доступу ESP32-S3, запуск сканування і порівняння працездатності під час сканування BLE-пакетів антен Ubertooth і Anrdoid;

– USB CDC ESP32 – окремий режим сканування BLE-пакетів, в якому використовується зовнішня антена для сканування (ESP32, nRF52840 та ін).

В ході виконання роботи шляхом реалізації першого режиму під назвою «Стандартний» виникла потреба отримувати номер каналу, в якому було отримано пакет. Згідно офіційної документації від Android [18] видно, що номер рекламного каналу, на якому отримано пакет не доступний. Подальший аналіз вихідного коду прошивки, її обгорток продемонстрував, що доступ до такої інформації можливий за умови наявності «root-доступу» і виклику напряму вендорських функції – це суперечить обмеженням застосунку і поставленій задачі.

Отже, задача повноцінного BLE-сніфінгу засобами Android-застосунку була реалізована не в повній мірі через обмеження самої оболонки. Проте були отриманні інші дані, такі як: MAC, RSSI та інші параметри реклами, що дає можливість отримати поверхневу інформацію про поточний стан радіочастотного ефіру.

2.4.2. Візуалізація результатів сканування в інтерфейсі користувача

Після успішного запуску застосунку в режимі «Стандартний», наступним кроком стала реалізація виводу результатів сканування в інтерфейсі користувача. Обрано просте табличне подання результатів: список пристроїв у «ListView», кожний елемент якого містить агреговану по пристрою інформацію виду «MAC [count type packets] | RSSI | CALC_DISTANCE | TIME_DIFF | COUNT_PACKETS», в якому:

– MAC – мак-адреса пристрою;

– count type packets – кількість типів пакетів прийнятих від одного пристрою, наприклад: якщо від пристрою отримано два пакети типу «ADV_IND» і три пакети типу «SCAN_REQ», то даний параметр дорівнюватиме п’яти;

– RSSI – значення потужності отриманого сигналу останнього пакету;
– CALC_DISTANCE – приблизне значення відстані розраховане за формулою 2.1 з значень, отриманих в останньому пакеті від даного пристрою;
– TIME_DIFF – цей параметр демонструє активність пристрою шляхом розрахунку різниці між часом отримання останнього пакету і передостаннього пакету від даного пристрою;
– COUNT_PACKETS – загальна кількість пакетів, отримана від даного пристрою.

Для реалізації цієї схеми в користувацькому інтерфейсі, було створено ресурс-розмітку за шляхом «res/layout/activity_main.xml» проекту, в якому додано елемент ListView, який заповнюється адаптером ArrayAdapter<String> класу MainActivity застосунку.

Оновлення інтерфейсу виконується за допомогою методу updateListView(), який формує для кожного запису рядок і передавав список рядків до адаптера наступним за допомогою наступного функціоналу:

```
for (Entry entry : activeEntries) {
    String typeWithCount = entry.countTypePackets > 1
        ? String.format(Locale.US, "%s (%d)", entry.type, entry.countTypePackets)
        : entry.type;
    String timePart = entry.packets.size() > 1
        ? String.format(Locale.US, " | %.3f ms", entry.latest.timestamp_diff)
        : "";
    activeLines.add(String.format(Locale.US,
        "%s [%s] | %d dBm | %.2f m%s | %d packets",
        entry.mac,
        typeWithCount,
        entry.latest.rssi,
        entry.latest.distance,
        timePart,
        entry.packets.size()));
}
for (Entry entry : staleEntries) {
    String typeWithCount = entry.countTypePackets > 1
        ? String.format(Locale.US, "%s (%d)", entry.type, entry.countTypePackets)
        : entry.type;
    String timePart = entry.packets.size() > 1
        ? String.format(Locale.US, " | %.3f ms", entry.latest.timestamp_diff)
```

```
                : "";
            activeLines.add(String.format(Locale.US,
                "%s [%s] | %d dBm | %.2f m%s | %d packets",
                entry.mac,
                typeWithCount,
                entry.latest.rssi,
                entry.latest.distance,
                timePart,
                entry.packets.size()));
        }
        List<String> lines = new ArrayList<>();
        lines.addAll(activeLines);
        lines.addAll(staleLines);
        runOnUiThread(()->
{listAdapter.clear();listAdapter.addAll(lines);listAdapter.notifyDataSetChanged();});
```

Першим циклом for (Entry entry: activeEntries) відбувається формування кожного активного запису(пристрою), в якому якщо кількість пакетів більше ніж один, показується тип і кількість пакетів, якщо тип лише один, тоді просто виводиться Eddystone. Також в цьому циклі будується timePart змінна для кожного запису, в якому якщо пакетів більше ніж один, то різниця у часі між останнім і передостаннім пакетом, а якщо всього один пакет – нічого не додається.

Другим циклом for (Entry entry: activeEntries) відбувається все те саме що і для циклу активних пристроїв, але для застарілих пристроїв – тих, від яких давно не було отримано жодного пакету.

Таким чином в користувацькому інтерфейсі будується список, який сортується за активністю пристроїв, що є зручно і логічно для користувача.

В кінці методу реалізовано створення фінального списку у змінній lines, яка додається в listAdapter і виклик notifyDataSetChanged() для перемалювання списку.

Висновки до другого розділу

В другому розділі розроблено та досліджено ключові програмно-апаратну основу, необхідну для подальшого імітаційного моделювання конфліктної взаємодії інформативних та завадових BLE-сигналів.

В першу чергу розібраний рекламний пакет від пристрою «Xiaomi MiKettle» дозволив порівняти реальний розбір пакету з узагальненою моделлю «BLE

advertising PDU». Таким чином продемонстровано, що за допомогою стандартного BLE-стека Android можна отримати лише поля рівня GAP/GATT (MAC-адресу, RSSI, Service Data, UUID тощо), але такі критично важливі для моделювання параметри фізичного рівня як preable, Access Address і номер каналу є недоступними. Це обмежує можливості побудови експериментальної установки, власних форматів пакетів і моделювання процесів на рівні радіоканалу, що вимагає залучення додаткового апаратного забезпечення.

Також досліджено залежність якості приймання BLE-пакетів від відстані та рівня сигналу за допомогою використання «ESP32-S3 N16R8 Type-C» та «Ubertooth One». Вдосконалення програмного засобу ubertooth-BLE шляхом переходу формату часу від секунд до наносекунд дозволило з необхідною точністю вимірювати інтервали між подіями передавання й приймання пакетів. Автоматизація експерименту за допомогою Bash і Python скриптів забезпечила отримання великої вибірки (~41 тис. пакетів) у діапазоні відстаней 0–10 м. Аналіз результатів показав наявність вираженого піку похибок та втрат пакетів у ближній зоні антени (порядку 1 м для частоти 2,402 ГГц) і подальшу стабілізацію характеристик у дальній зоні.

Проведено аналіз прошивки пристрою «Ubertooth One» як потенційно можливого апаратного засобу, що буде використаний як зовнішня антена для Android-застосунку. Дослідження вихідного коду, організації USB-control команд і FIFO-черг, а також результати відладки через JTAG демонструють, що за наявних обмежень обсягу пам’яті та, ймовірно, відсутності асинхронності між частиною коду по взаємодії з радіоефіром й передачею даних на хост, спроби збільшити буфери призводять до втрат пакетів з черги. Таким чином, Ubertooth One виявився зручним і доступним інструментом для аналізу протоколу та лабораторних експериментів, але має суттєві обмеження для ролі зовнішньої антени в задачах інтенсивного BLE-моніторингу.

Як результат всього дослідження створено Android-застосунок для сканування BLE-пакетів і візуалізації отриманої інформації. Він може працювати в кількох режимах роботи: з використанням внутрішньої антени смартфона, тестовий режим,

з використанням зовнішньої антени по USB CDC. Попри неможливість доступу до повного набору полів фізичного рівня, отримані в межах Android-SDK дані є достатніми для поверхневої оцінки радіоефіру й інтегруються в загальний експериментальний стенд.

Отримані результати демонструють необхідність в подальшому розвитку застосунку, дослідженню можливості застосування інших пристроїв (ESP32, nRF52) в якості зовнішньої антени, а також, пристроїв Android, що мають root-доступ.

Розділ 3
АПАРАТНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ МЕТОДУ МОДЕЛЮВАННЯ BLE-ПАКЕТІВ

3.1. Дослідження обмежень апаратного забезпечення для моделювання BLE-пакетів

3.1.1. «Зовнішня» BLE-антена для Android-застосунку

Після дослідження обмеження вбудованого BLE-інтерфейсу Android-застосунку та аналізу можливостей Ubertooth One, виникла потреба у додатковому апаратному забепеченні, яке має доступ до фізичного рівня, здатне приймати BLE-пакети з ефіру, попередньо їх обробляти й передавати у зручному форматі на хост-пристрій. Обрано плату розробника ESP32-S3 з вбудованим Wi-Fi та BLE.

Метою на цьому етапі було використати ESP32 як «зовнішню антену» для Android-системи: ESP32 пасивно сканує BLE-ефір, формує з кожного рекламного пакета структурований запис (час, MAC-адреса, RSSI, сирий payload тощо), а далі передає ці дані на хост по одному з двох каналів:

– по USB CDC у Serial (як текстовий рядок);
– широкомовним UDP по Wi-Fi на порт 50000 (якщо Wi-Fi під’єднано).

Програмний код прошивки для ESP32-S3 наведено в Додатку В. Для реалізації сканування використано стек NimBLE. В прошивці ініціалізується BLE-контролер, конфігурується сканер, знайдені рекламні пакети обробляються у callback-класі NimBLEScanCallbacks. З кожного пакету формується JSON-об’єкт з такими полями:

– ts_ms – час отримання пакета в мілісекундах від моменту запуску контролера;
– mac – MAC-адреса передавача у строковому форматі;
– rssi – рівень прийнятого сигналу (RSSI, дБм);
– is_scan_response – ознака того, чи є кадр scan response;
– name – локальне ім’я пристрою (якщо присутнє у пакеті);

– payload_hex – сирий вміст рекламного кадру у вигляді HEX-строки.

Формат даних JSON був обраний для уніфікації обробки даних з боку хосту: будь-яка мова програмування здатна обробити JSON-строку і перетворити її на сутності, притаманні тільки їй – це полегшує обробку даних і дозволяє використовувати ESP32 як зовнішню антену для будь якого пристрою, який написано на будь-якій сучасній мові.

Для передачі даних по Wi-Fi прошивка працює в режимі точки доступу. В ній задається SSID, канал, пароль для доступу до точки Wi-Fi.

Перехід між режимом по USB і Wi-Fi відбувався завдяки зміні конфігурації в файлі самої прошивки і запису її на пристрій. Під час тестування виявлено суттєві обмеження ESP32-S3 при одночасному активному використанні Wi-Fi та BLE. При активному широкомовному передаванні UDP-пакетів прийом BLE-реклами помітно погіршився: спостерігалися пропуски кадрів, нерівний часовий інтервал між отриманими записами, пропуски відправки UDP-пакетів до хосту. Все це пояснюється тим, що Wi-Fi та BLE працюють через спільний радіомодуль.

В результаті прийнято рішення про відмову від Wi-Fi режиму і працювати лише через UART режим, в якому всі пакети передаються тільки через USB-serial.

3.1.2. Проблеми фільтрації BLE-пакетів за каналом засобами ESP32

Однією з вимог до дослідження є можливість фільтрації трафіку окремо за кожним з рекламним каналом (37, 38, 39). У випадку роботи з апаратним засобом ESP32 при написанні прошивки до нього помічено, що API, яке надане стеком NimBLE є високорівневим, в якому доступ до параметрів пов’язаних з частотою, конкретним каналом відсутній: сканер працює з рекламними подіями, які не мають в собі ідентифікаторів каналу та іншої необхідної інформації всередині кожного кадру.

Загалом, для того, щоб запрограмувати ESP32 існують декілька мов і фреймворків для них [19]:

- C/C++ (ESP-IDF) – це офіційний SDK від Espressif для ESP32, що дозволяє програмувати за допомогою засобів мови програмування C/C++, повний доступ до заліза й стеків (Wi-Fi, BLE, RTOS);
- C++ (Arduino) – це спрощений шар поверх ESP-IDF. Так само реалізовується на C++, але з простішим API і величезною кількістю бібліотек/прикладів. Швидкий старт, але менше контролю та гнучкості, ніж у чистому ESP-IDF;
- MicroPython – пишеться за допомогою мови Python, яка працює через вбудований інтерпретатор на мікроконтролері. Зручно для прототипів, навчання і простих експериментів, повільний і менша оптимізація по пам’яті;
- CircuitPython – це певний форк MicroPython, що створено з ухилом в простоту, але з меншим низькорівневим контролем, але простий.

Згідно документації Espressif [19], можливо фільтрувати адреси/тип адреси, RSSI, наявність/вміст конкретних AD-полів (UUID сервісу, manufacturer data), типи реклам (connectable/non-connectable), дублікатність, PHY, тощо. Але фільтр по каналу не можливо зробити, оскільки:

- Сканер завжди стрибає (хоупінг) по 3 первинних рекламних каналах (37/38/39) згідно з BLE-специфікацією, що є базовою поведінкою контролера, яка записана у внутрішній пам’яті мікроконтролера (ROM пам’ять, до якої нема доступу, що зображено на рис. 3.1 – взято з офіційного даташиту [20]);
- HCI-події реклам (Advertising Report / Extended Advertising Report) не містять номера каналу. Користувацький застосунок, який записується в SPI-flash пам’ять, яка підключена до SPI0/1 (рис. 3.1) [20], не отримує цієї інформації і не може по ній фільтрувати. Це загальне обмеження платформ, що працюють через стандартний HCI (Android, iOS, Linux, Windows, ESP32 тощо).

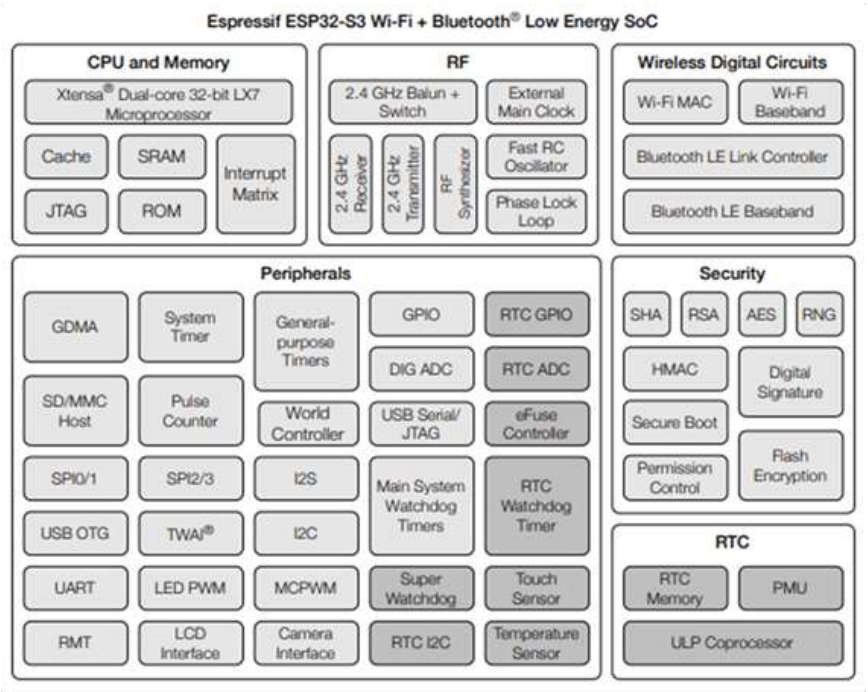


Рис. 3.1. Функціональна діаграма ESP32-S3

Також, досліджено питання про неможливість відправлення спеціальних опкодів на мікроконтролер прошивки, щоб керувати чипом. Дослідження показало що:

– у кристалі є ROM (завантажувач – внутрішня прошивка, яка записана на заводі виробника без можливості перезапису і зчитування ззовні ділянки чипу), він незмінний, виконується безпосередньо при запуску чіпа;
– користувацький код, який завантажується у плату (наприклад, засобом esptool), записується у зовнішню пам’ять(зазвичай об’єм 4-16 Мб);
– чіп за допомогою з’днання через SPI0/1 зчитує прошивку і виконує її після виконання власних запрограмованих функцій. Тобто під час включення ROM-bootloader, який вбудовано в ROM пам’ять, ініціалізує читання через ніжку чіпу SPI0/1, читає bootloader користувацької прошивки, і вже він обирає потрібний розділ для застосунку і запускає його;
– прошивка всередині чипу недоступна для читання і недокументована, що унеможливлює прямий доступ до керування радіочіпом BLE, WiFi, тощо.

Також, під час дослідження було помічено, що під час компілювання засобами platformio є можливість побачити і дослідити файл за шляхом «.pio/build/esp32-s3-

devkitm-1/firmware.map», який є linker map файлом. Всередині файлу описано де і в якій області опинились всі секції, символи прошивки, такі як:

– Memory Configuration – перелік областей пам’яті з адресами та розмірами;
– описані секції .text, .data, .flash.* – тобто розмічені області даних;
– крос-посилання символів, відкинуті секції(ті що були відкинутими після оптимізації).

Всередині цього файлу дійсно є певні «замаплені» функції, які натякають на можливість налаштування каналу, але доступу до цих функції прямого на момент дослідження не знайдено. Також існує здогадка автора про те, що навіть якщо і викликати ці функції, стандартний стек (наприклад NimBLE) перезапише налаштування, які будуть задані в обхід його.

Єдиним теоретично можливим варіантом є спробувати вгадувати час переключення між каналами на ESP32, але цей варіант ненадійний, оскільки: ймовірність засинання контролеру і зсув розрахунку наступного таймінгу відносно реального переключення між каналом.

3.1.3. Заміна базової платформи на nRF52840

Після дослідження обмеження ESP32 було обрано апаратний засіб nRF52840, яка є спеціалізованим мікроконтролером з повною підтримкою BLE 5.x, гнучкішим радіочастотним блоком та розвиненою екосистемою інструментів (SDK, SoftDevice, приклади для сканування та передавання реклами).

Основними характеристиками, які є вирішальними у виборі nRF52840 є:

– наявність інформації про канал в пакетах, які оброблюються користувацькою прошивкою;
– наявність функціоналу «follow-connection», який дозволяє перехоплювати пакети при з’єднанні BLE пристроїв один з одним і сніфити подальші пакети їхнього з’єднання;
– існує готова прошивка і плагін для Wireshark, що дозволяє працювати як Sniffer.

Основним недоліком є гірша якість антени в порівнянні з ESP32.

Як готова плата, на якій є чіп NRF52840, стояв вибір між платами «ProMicro NRF52840 Type-C» і «Seeed XIAO BLE nRF52840 Sense», було обрано «ProMicro NRF52840 Type-C» за наступними критеріями:

– ціна – від виробника «Seed» більша ніж від «ProMicro» (різниця дорівнює 769 грн.);
– інтерфейси GPIO – у «Seed» одинадцять GPIO, тоді як у «ProMicro» двадцять шість GPIO;
– flash-пам’ять – «Seed» вміщує в себе 1 МБ флеш, 256 КБ RAM і 2 МБ QSPI Flash, тоді як «ProMicro» 1 МБ флеш, 256 КБ RAM (без додаткової);
– переферія – «Seed» має вбудований цифровий мікрофон, шестиосьовий IMU, RGB-підсвітку, чип зарядки BQ25101, додатково QSPI Flash 2 МБ, тоді як «ProMicro» кнопку перезапуску, зарядка LiPo (до 300 мА), кілька GPIO;

Згідно вище описаного порівняння, було обрано ProMicro в першу чергу за ціною, для поставленої задачі така кількість переферії і велика кількість інтерфейсів GPIO непотрібні, що робить плату доцільною для подальших досліджень і експериментів.

Прошивка, яка була написана наведена в Додатку Г. Після завантаження цієї прошивки, пакети, які віддавались по Serial всередині себе містили номер каналу.

3.2. Переваги і недоліки при використанні nRF52840

3.2.1. Емулюція апаратного USB-UART чипу

Багато CDC-пристроїв (USB Communication Device Class), до яких відноситься nRF52840, не відправляють дані до тих пір, поки хост не запустить DTR і RTS (в деяких випадках) через відсутність апаратного USB-UART чипу. CDC – це клас пристроїв USB, які імітують роботи COM-інтерфейсу. Тобто всередині нього є підкласи, які повторюють і емулюють старі RS-232 порти поверх USB.

В наслідок цього з’явилась необхідність модифікації програми хосту, в нашому випадку яким є застосунок Android, у вигляді додавання двох функцій у

файлі «app/src/main/java/com/example/blesniffer/UsbReceiverActivity.java» в функції «private void openUsb() {», що відповідає за відкриття через USB послідовного порту на Андроїд після отримання і відкриття порта комунікації:

```
port.setDTR(true);
port.setRTS(true);
```

В результаті внесених вищенаведених змін отримано високопродуктивний компактний сніфер, який має функіонал фільтрування пакетів за каналом, за мак адресою, тощо. Особлива перевага полягає в можливості доступу до фізичного рівня з користувацької прошивки.

3.2.2. Порівняльний аналіз електроспоживання ESP32 і nRF52840

Оскільки для роботи Android-застосунку з використанням зовнішньої антени необхідно пристрій, що містить її, фізично підключити до смартфону, постає питання тривалості автономної роботи. В наслідок цього прийняте рішення про необхідність заміру споживання пристроїв nRF52840 і ESP32-S3.

Для цього використано цифровий мультиметр «UNI-T UT136B+» в режимі «амперметру» і заміряно споживання шляхом підключення щупів в розріз кабелю Type-C – Type-C, як зображено на рис. 3.2.

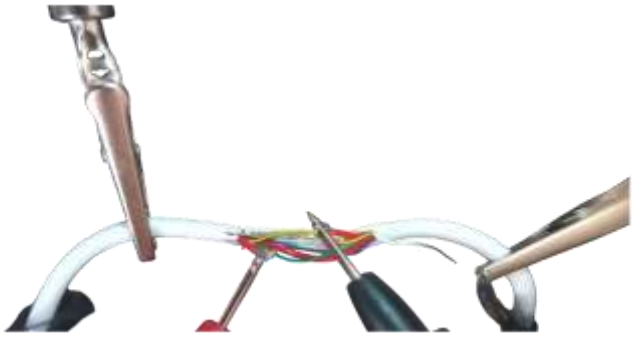


Рис. 3.2. Вимірювання споживання плат ESP32-S3 і nRF52840 щупами мультиметра

В результаті отримано наступні результати вимірювання споживання мікроконтролерів, які наведено в табл.3.1.

Таблиця 3.1. Результати вимірювання споживання електроенергії мікроконтролерів

Модель	Струм, мА	Напруга живлення, В	Потужність, мВт
ESP32-S3	2,36	5	11,8
nRF52840	0,25	5	1,25

3.3. Застосування HackRF One як інструмента аналізу та моделювання завадових сигналів

HackRF One – це програмно-визначений радіоприймач-передавач, який здатний працювати в діапазоні частот від 1 МГц до 6 ГГц у напівдуплексному режимі (передавати або прийом в один момент часу). Апаратна частина розроблена як відкритий пристрій для експериментів з радіотехнологіями. Дана плата підключається до комп’ютера через USB порт і підтримує частоту дисркетизації до 20 млн семплів на секунду при розрядності 8 біт.

Portapack – це апаратний доповнювач до HackRF One, який оснащений екраном, модулем керування (джойстик і кнопки), аудіо-гіздом, слотом карти пам’яті та акумулятором, що дозволяє йому працювати автономно від комп’ютера, загальний вид якого зображено на рис. 3.3.

Рис. 3.3. Загальний вид Portapack HackRF One

В подальших розділах описано два види дослідження інструменту: загальне ознайомлення і демонстрація функціоналу і дослідження застосування в BLE-технологіях.

Можливості Portapack HackRF One досліджуються з акцентом на прийомі сигналів і їх декодування: від простого прослуховування радіо за допомогою готових програмних інструментів (SDR++, SDR# тощо) до дослідження роботи РЕБ під час тривоги, а також, прийому рекламних пакетів за допомогою технології BLE на частоті 2402 (37 канал).

3.3.1. Дослідження прийому аналогових радіосигналів

Яскравим прикладом прийому аналогових радіосигналів є можливість прослуховування міського радіо в діапазоні частот від 88 до 120 МГц, а також незашифрованих аудіопереговорів за допомогою рацій. Це можливо завдяки наявності роз'єму для підключення динаміків в інтерфейсі Portapack і наявності розділу під назвою «Receive» в прошивці, що зображено на рис. 3.4.

Рис. 3.4. Розділ «Receive» в прошивці Portapack

3.3.2. Дослідження роботи РЕБ за допомогою HackRF One

Також даний програмно-апаратний засіб дозволяє бачити в режимі реального часу роботу РЕБ, що дозволяє ознайомитись з даною технологією на практиці і краще розуміти принцип роботи даної технології завдяки наочній демонстрації, яка зображена на рис. 3.5. Наведена ілюстрація є лише інформативною і ознайомчою, вона не може бути задіяна в реальних застосуваннях.

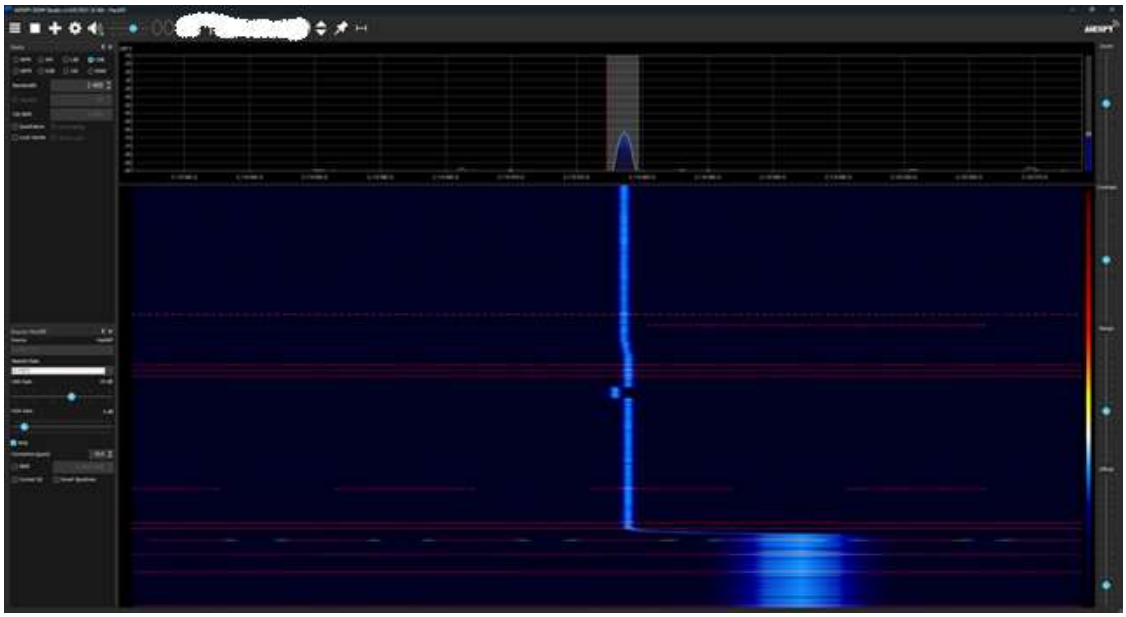


Рис. 3.5. Дослідження РЕБ за допомогою Portapack HackRF One

3.3.3. Декодування цифрових сигналів за допомогою HackRF One з використанням програмного засобу DSD+

Для повного розуміння можливостей HackRF One після дослідження прийому сигналів було приділено увагу програмному декодуванню їх. Це є основною складовою при роботі з цифровими протоколами зв’язку, які мають складну структуру кадрів, специфічні методи демодуляції та вбудованими механізмами корекції помилок. В ході даного дослідження було використано програмний пакет Digital Speech Decoder Plus.

Digital Speech Decoder Plus (DSD+) – це програмний засіб, який призначений для декодування різноманітних цифрових радіосистем, що використовуються у

службовому та цивільному секторі [21]. Він працює виключно з аудіопотоком, який подається від попередньо демодульованого радіосигналу. Він підтримує наступний перелік цифрових протоколів:

– DMR (Digital Mobile Radio)
– P25 Phase I
– NXDN
– D-STAR
– YSF (Yaesu System Fusion)
– ProVoice
– dPMR

Робота програмного засобу продемонстрована на рис. 3.6.

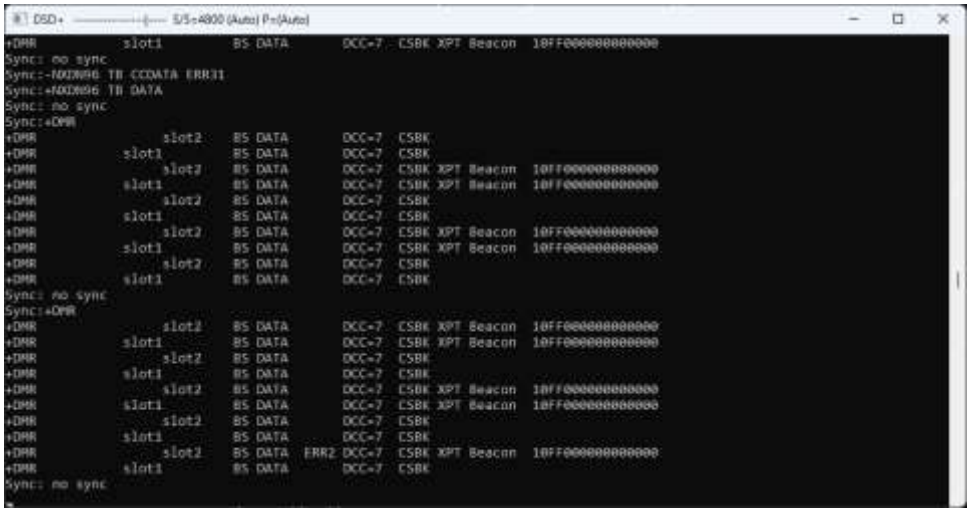


Рис. 3.6. Декодування цифрового потоку програмним засобом DSD+

3.4. Дослідження технології BLE засобами Portapack HackRF One

Дане дослідження спрямоване на низькорівневе вивчення технології BLE шляхом прийому сигналу за допомогою SDR HackRF One без використання готових утиліт. Метою є створення і налаштування програмно-апаратного забезпечення для подальшої імплементації його в роботах по створенню власних протоколів передачі даних за допомогою технології BLE.

3.4.1. Структура BLE-кадру

Для правильності виділення BLE-кадрів з ефіру, необхідно ознайомитись з базовою його структурою на фізичному рівні. Для простого випадку LE Uncoded PHY кадр має вигляд, який зображено на рис. 3.7.

Основними складовими є:

- преамбула – один байт, який має вид послідовності 0xAA або 0x55. Необхідний для приймача, щоб правильно синхронізуватися з передавачем і вказує на початок пакету;
- Access Address – чотирьох-байтовий ідентифікатор пакета, який для всіх рекламних пакетів фіксований і має значення 0x8E89BED6. Він вказує що пакет типу BLE-реклами, а не якийсь інший трафік;
- PDU (Protocol Data Unit) – корисне навантаження кадру. Всередині PDU є заголовок (header) з полями типу кадру, довжиною payload тощо, а далі саме корисне навантаження, в якому вже сховані MAC-адреси, типи кадрів, службові дані та інша інформація;
- CRC – це контрольна сума, за допомогою якої перевіряється цілісність кадру.

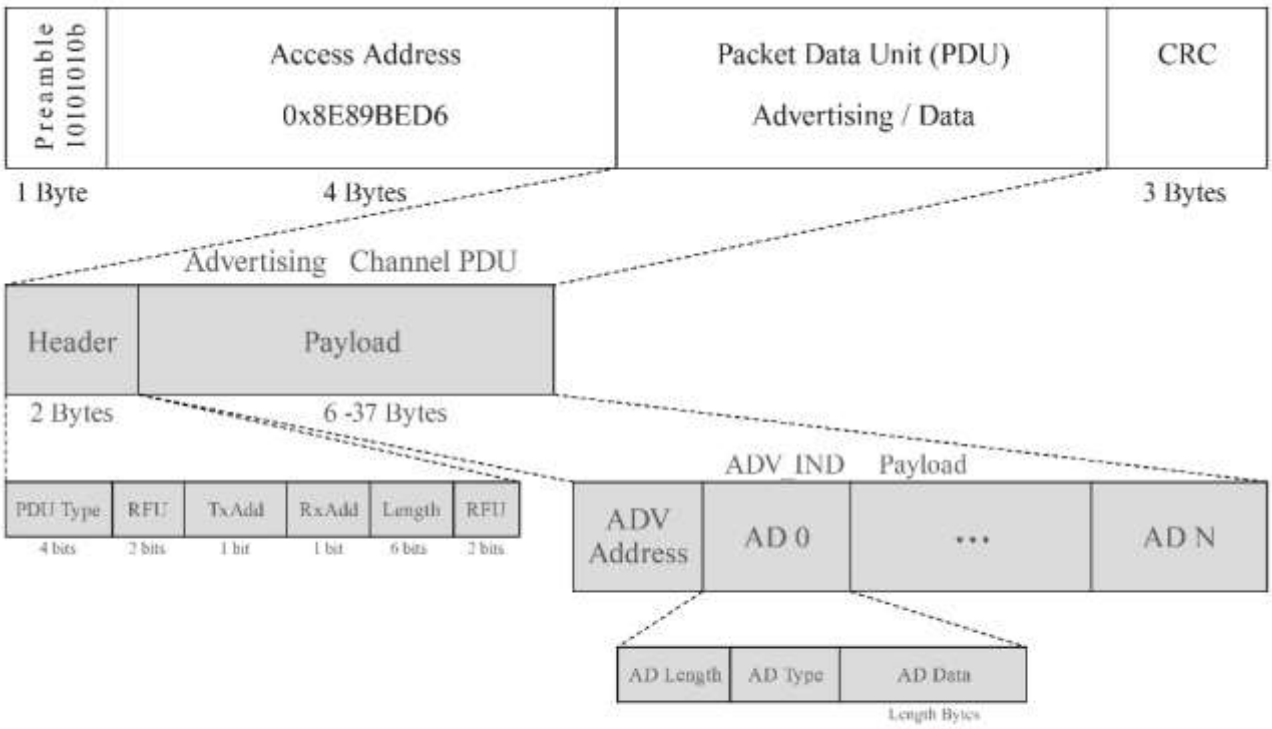


Рис. 3.7. Структура BLE-кадру на фізичному рівні

В спрощеному розумінні, на практиці це виглядає як знайти правильну послідовність байтів в потоці байтів, який отримується за допомогою HackRF, обробити і декодувати знайдений пакет.

3.4.2. Метод прийому і обробки BLE-пакетів

Для спрощення і кращої демонстративності процесу було використано програмний засіб GNU RC для взаємодії з HackRF One. В ньому побудовано функціональну схему, його також називають флоуграфом (від англійського слова flowgraph), який зображено на рис. 3.8.

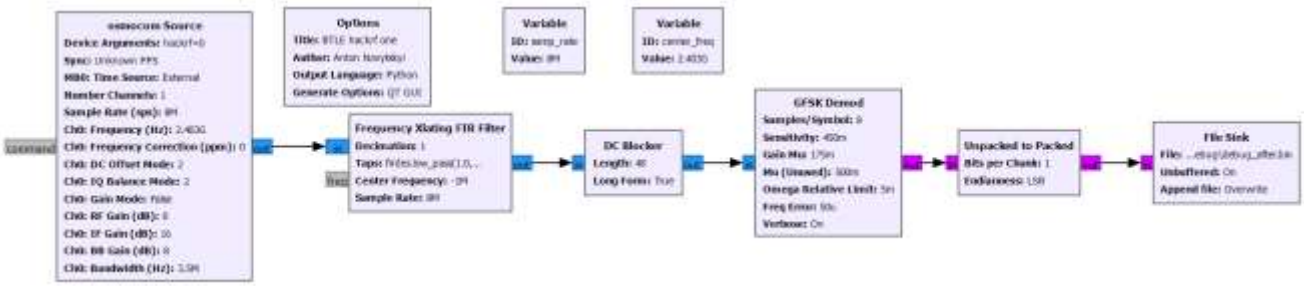


Рис. 3.8. Функціональна схема для отримання BLE-кадрів за допомогою GNU RC

В даній схемі відбувається прийом сигналу з HackRF One, виконується базова обробка і демодуляція сигналу, після якої потік бітів перетворюється у послідовність байтів. Для полегшення відладки, байтовий потік записується у файл під назвою “debug_after.bin”.

Для подальшої обробки файлу з байтовою послідовністю написано Python-скрипт, який наведено в Додатку Д. Він зчитує весь файл, займається пошуком сигнатур, вирізанням пакетів, перевіркою структури та цілісності кадрів.

В ролі маячка виступає ESP32-S3 з прошивкою, що наведено в Додатку Е. Він з певною періодичністю відправляв рекламні пакети з відомим вмістом, що дозволяло легко відлагоджувати і оцінювати правильність алгоритму.

3.4.3. Розробка експериментального стенду

Під час дослідження і створення програмно-апаратного засобу для отримання BLE-кадрів з ефіру за допомогою HackRF One помічено складність в виконанні поставленого завдання, яка полягає у наявності великої кількості рекламних пакетів від різних пристроїв в ефірі, що зображено на рис. 3.9 за допомогою програмного засобу SDR++ і HackRF One.

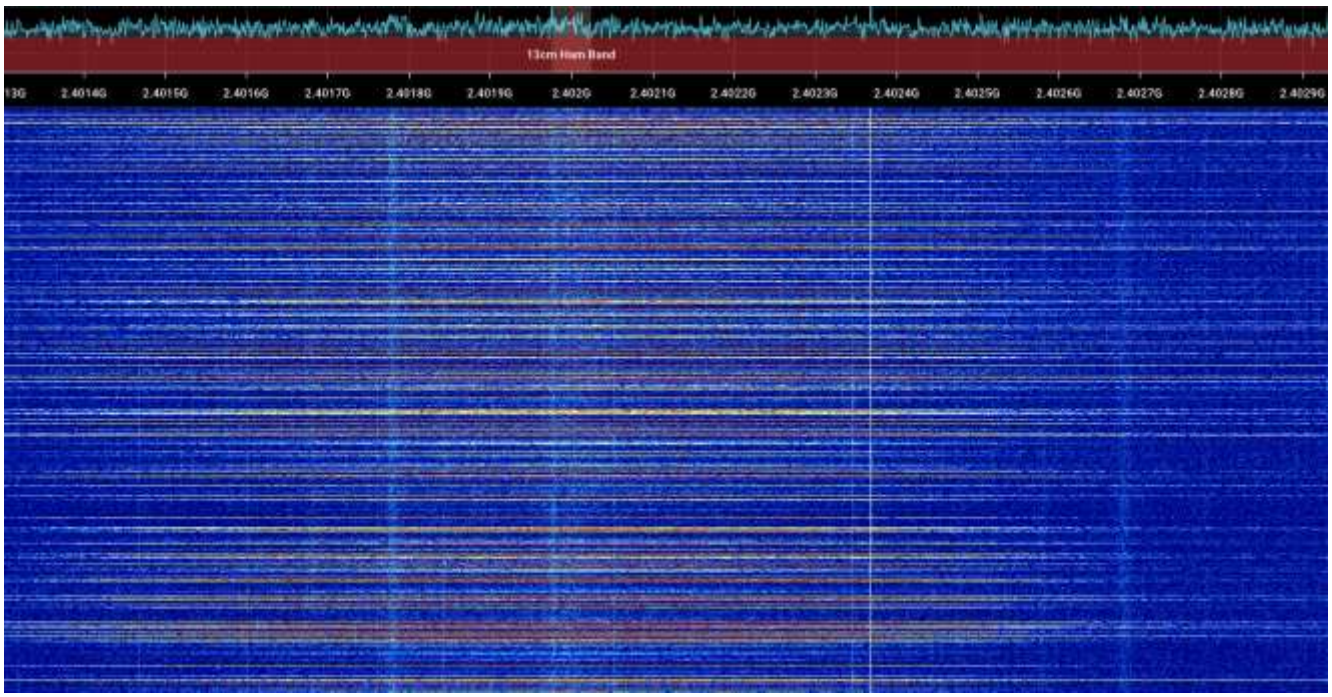


Рис. 3.9. Загальний вид активності ефіру в програмному засобі SDR++ на 37 рекламному каналі BLE

Для спрощення дослідження було створено лабораторний стенд, прицнипова схема якого зображена на рис. 3.10.

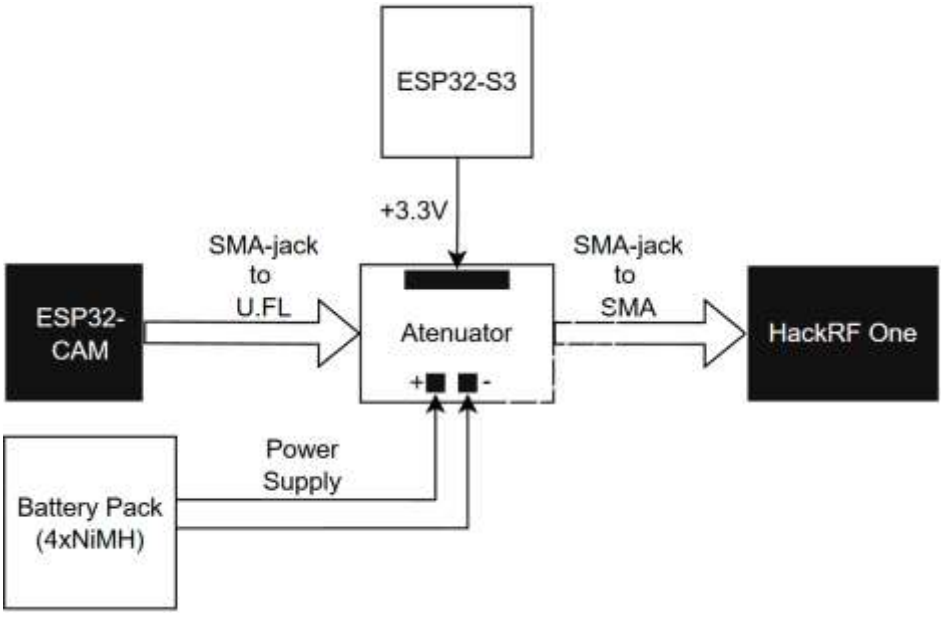


Рис. 3.10. Принципова схема експериментального стенду з ESP32-CAM і атенюатором

В ролі передавача використано модуль Wi-Fi ESP32-CAM з камерою 2MP, який на схемі зображено як «ESP32-CAM», оскільки він має роз’єм U.FL, що дозволяє підключати атенюатор і транслювати пакети через нього рекламні пакети.

Для того, аби працювала зовнішня антена замість вбудованої, треба перепаяти джампер у відповідне положення [22], як зображено на рис. 3.11.

Рис. 3.11. Перепаяний джампер в положення включення зовнішньої антени на ESP32-CAM

В ролі атенюатора використано модуль ВЧ-атенюатор PE4302 з ослабленням 31,5 дБмВт. Згідно описаної документації вибір режиму рівня ослаблення виконується за допомогою конфігураційного входу P/S. Після подачі живлення мікросхема автоматично переходить у певний стан ослаблення відповідно до вибраного режиму. Вибір режиму відбувається за наступного алгоритму, наведеного в табл.3.2 [23].

Таблиця 3.2. Схема подачі високого рівню напруги для керування рівнем атенюації [23]

C16	C8	C4	C2	C1	C0.5	Режим послаблення, дБмВт
0	0	0	0	0	0	Без атенюації
0	0	0	0	0	1	0,5
0	0	0	0	1	0	1
0	0	0	1	0	0	2
0	0	1	0	0	0	4
0	1	0	0	0	0	8
1	0	0	0	0	0	16
1	1	1	1	1	1	31,5

Як джерело високого логічного рівня на конфігураційному вході P/S атенюатора виступає ESP32-S3, підключення якого зображено на рис. 3.12, де підключено всі піни, що відповідає рівеню атенюації рівному 31,5 дБмВт.

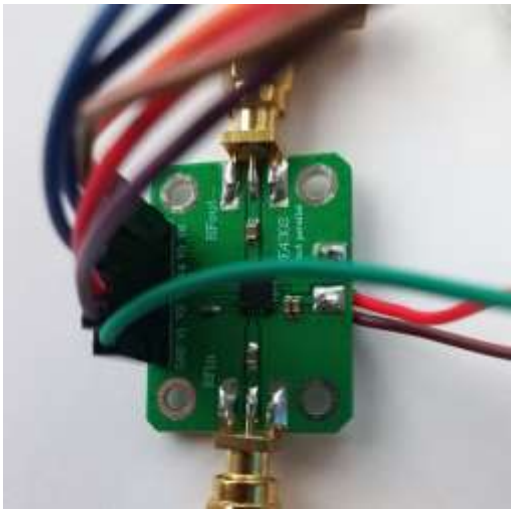


Рис. 3.12. Підключення до роз’єму P/S атенюатора для керування рівнем атенюації

В результаті отримано експериментальний стенд, який є замкненою системою дослідження методів отримання і декодування BLE-пакетів за схемою в якій передавач підключений через кабель до атенюатор, який в свою чергу через кабель підключений до HackRF One. В такій схемі сторонні сигнали відсутні, що полегшує дослідження і можливість створення власних методів передачі даних. Під час досліджень пакети були впіймані, але не декодовані через специфіку BLE – whitening.

3.4.4. Whitening і Dewhitening при декодуванні BLE

Whitening у BLE – це етап обробки даних, який застосовується перед модуляцією, суть якого полягає в наступному: якщо існує довга послідовність однакових бітів, то суттєво гіршає синхронізація та якість прийому. Для уникнення довгих однакових послідовностей, в BLE усі біти PDU перемішуються.

Алгоритм вайтнінгу має наступний вид:

– на передавачі формується псевдовипадкова бітова послідовність за допомогою 7-бітного LFSR (Linear Feedback Shift Register) – це поліном [24]:

$$x^7 + x^4 + 1 \quad (2.2)$$

– початкове значення (seed) цього LFSR залежить від номера каналу;
– над коженим бітом корисних даних перед модуляцією виконується операція XOR з відповідним бітовим потоком з LFSR.

Результатом є вже завайтнений кадр, який розпізнати на приймачі без процедури зворотного вайтнінгу неможливо, оскільки він має шумоподібний вид.

Під час дослідження було проведено досить багато часу і спроб на девайтнінг пакетів, в загальному алгоритм був наступний:

– запуск флоуграфу в GNU RC, який записує в файл потік байтів отриманного з HackRF One;
– реалізований за допомогою мови програмування Python скрипт, який наведено в Додатку Д, зчитує весь файл, шукає преабули і access address;
– після кожного знайденого співпадіння скрипт вирізає пакет потрібної довжини і проводить процедуру dewhitening, перевіряється цілісність пакету за допомогою BLE CRC і результат виводиться користувачу в термінал;
– в кінці порівнюється пакет який отримано з пакетом який був відправленим.

В результаті тестування з використанням пристрою ESP32-cam, який має мак-адресу 38:18:2B:B1:B4:96 отримані наступні дані:

```
@offset 0x1F935A
Type: SCAN_RSP, Len: 32
MAC : 55:5E:68:EF:49:0C (public)
FullHEX(140): AA D6 BE 89 8E 8D DD C1 15 8C 8C 7E 88 7D 38 22 79 A4 44 CB D1 7D 24 46
06 1B 42 03 84 00 D2 01 E8 EF 0F 11 B4 17 7E 69 39 E0 C2 04 9D F2 9C
FullDewhitenHex(42): 14 A0 0C 49 EF 68 5E 55 7B CA 85 04 B9 D4 56 D4 F4 13 7D A3 64 F0
80 A8 8B E6 03 22 CE 1D 82 93 7C 10 A9 1B 7E 27 A1 AA 58 62
AdvData(26): 7B CA 85 04 B9 D4 56 D4 F4 13 7D A3 64 F0 80 A8 8B E6 03 22 CE 1D 82 93 7C
10
CRC: A9 1B 7E || CRC OK: False
```

Згідно вище описаних результатів видно, що мак адреса не співпадає, пакет також не співпадає і CRC не співпадає – отже пакет отримано з помилками.

Після великої кількості спроб девайтнінгу і декодування пакетів, побайтового порівняння пакетів відправленого з прийнятим, де різниця є занадто великою, а також, неправильності роботи утиліти DSD+, в якій звукові дані приймались з помилками, було вирішено про необхідність перевірки HackRF One на предмет виробничого браку або поломки пристрою.

Висновки до третього розділу

В третьому розділі описано апаратне моделювання та аналіз BLE-пакетів, що дозволило визначити придатність різних апаратно-програмних платформ для досліджень на фізичному рівні. Плата розробника ESP32-S3 використана в якості зовнішньої BLE-антени для Android-застосунку, в якій розроблено прошивку зі скануванням ефіру, формуванням структурованих JSON-записів і передаванням їх на хост через USB CDC або UDP по Wi-Fi. Виявлено, що спільне використання Wi-Fi та BLE на ESP32-S3 практично неможливо або призводить до суттєвого погіршення якості прийому через спільне використання радіотракту.

Досліджено можливість ESP32 фільтрувати трафік за BLE-каналами. Продемонстровано, що використання стандартного стеку NimBLE та HCI-подій не дозволяє фіксувати номер рекламного каналу і керувати ним з користувацької прошивки, оскільки механізми керування радіоналаштуваннями приховані у ROM-коді мікроконтролера й недоступні для конфігурації. Дослідження структури прошивки через linker map-файли підтверджує, що потенційно наявні внутрішні функції налаштування каналу хоч і наявні, але не доступні у застосуванні.

Описаний результат дослідження ESP32 цілком обґрунтовано дозволив перейти до реалізації рішення на базі nRF52840. Обрано плату ProMicro NRF52840 Type-C, яка надає інформацію про номер каналу і підтримує режим «follow-connection», при цьому має нижчу вартість і надлишковий для задачі запас периферії у порівнянні з альтернативними платами. Адаптована прошивка від ESP32 забезпечила аналогічний вивід по Serial порту інформації про пакет разом із номером каналу. Також виявлено й адаптовано Anrdoid-застосунок під особливість

класу пристроїв USB CDC, через яку для коректної роботи nRF52840 з Android-застосунком необхідно програмно ініціалізувати сигнали DTR і RTS.

Також продемонстровано порівняльні вимірювання енергоспоживання ESP32-S3 та nRF52840, в яких значення вказують на те, що nRF52840 є в декілька разів економніший за EPS32-S3 - це є критичним фактором для роботи зі смартфоном в режимі зовнішньої антени, що вказує на доцільність вибору.

Останній блок розділу присвячено застосуванню HackRF One з Portapack як SDR-інструмента для аналізу ефіру на частотах роботи BLE технології. Продемонстровано його можливості для прийому й аналізу аналогових сигналів, спостереження за роботою засобів радіоелектронної боротьби і декодування цифрових протоколів. Побудовано повноцінний експериментальний стенд із ESP32-CAM з можливістю підключення зовнішньої антени, ВЧ-атенюатором та HackRF One. Розроблено флоуграф у GNU Radio та Python-скрипт для пошуку і девайтнінгу BLE-кадрів з перевіркою CRC. Попри коректний алгоритм вайтнінгу і девайтнінгу, було виявлено значні похибки декодування та невідповідність отриманих кадрів надісланим, що в свою чергу разом з некоректною роботою DSD+ наштовхнуло на думку про необхідність оцінити стан конкретного екземпляра HackRF One та відмову від його використання як інструмента аналізу BLE-пакетів.

Розділ 4
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МЕТОДУ МОДЕЛЮВАННЯ BLE-ПАКЕТІВ

4.1. Постановка експерименту для моделювання BLE-пакетів

Метою даного розділу є побудова, проведення та аналіз експериментів з моделювання та оцінювання поведінки BLE-пакетів в незакритій системі в умовах:

– радіоефіру без завад;
– наявності генератора шуму.

Експеримент спрямований на перевірку працездатності розробленого методу формування BLE-пакетів, створеного інструментарію передавання (на базі nRF52840) та засобів збору даних (Android-застосунок і TI-донгл), а також на оцінювання впливу різних параметрів пакетів (довжина, інтервал, потужність) та завадового сигналу на якість прийому[25].

Для цього розроблено:

– прошивку для передавачів nRF52840;
– Python-оркестратор, який керує всіма передавачами одночасно, максимальна кількість яких становить вісім штук;
– інфраструктуру збору даних на Android та збереження у форматі ndjson;
– процедуру обробки результатів експерименту.

4.1.1. Розробка джерела BLE-реклами

В експериментальному стенді використовується до восьми штук плат розробників TSTAR MICRO NRF52840, під які написана одна прошивка, код якої наведено в Додатку Ж. Вона написана так, щоб задовільнити будь-які умови відправлення пакетів, які можуть знадобитись в даному експерименті.

З даною прошивкою кожен передавач має можливість формувати пакети з контрольованими параметрами такими як:

– кількість пакетів;
– канал рекламування;

– довжина корисного навантаження;
– формувати пакет з лічильником кадру: працює у випадку, якщо задана довжина пакету більше за нуль;
– інтервал передачі між пакетами;
– силу сигналу рекламованих пакетів.

Прошивка реалізує налаштування інтерфейсів за допомогою наступних команд:

```
SET C=<N> CH=<37/38/39> PL=<len> I=<ms> P=<dBm>
GO
```

Першою викликається команда SET, в якій:

– C – кількість пакетів, яка буде надіслана цим пристроєм;
– CH – канал по якому будуть відправлятись пакети, дозволений лише один канал за одну команду, мультивибір відсутній;
– PL – довжина пакету, може набувати наступних значень: 0, 1, 10, 20, 30. При значенні рівному одиниці в пакеті буде міститись лише лічильник пакету без додаткових заповнювачів.
– I – інтервал між пакетами;
– P – потужність, з якою будуть надіслані пакети.

Після успішної обробки команди як результат прошивка поверне строку «OK», після чого апаратний засіб з заданими параметрами запускається командою «GO».

В результаті отримано пристрій, який дозволяє керувати налаштуваннями реклами і профілем передавання.

4.1.2. Розробка оркестратора за допомогою мови програмування Python

Для спрощення проведення експерименту, його наглядності, а також, для унеможливлення виникнення людської помилки прийнято рішення про створення інтерфейсу керування всіми підключеними BLE апаратними засобами, код якого наведено в Додатку И і після запуску має вигляд як на рис. 4.1.

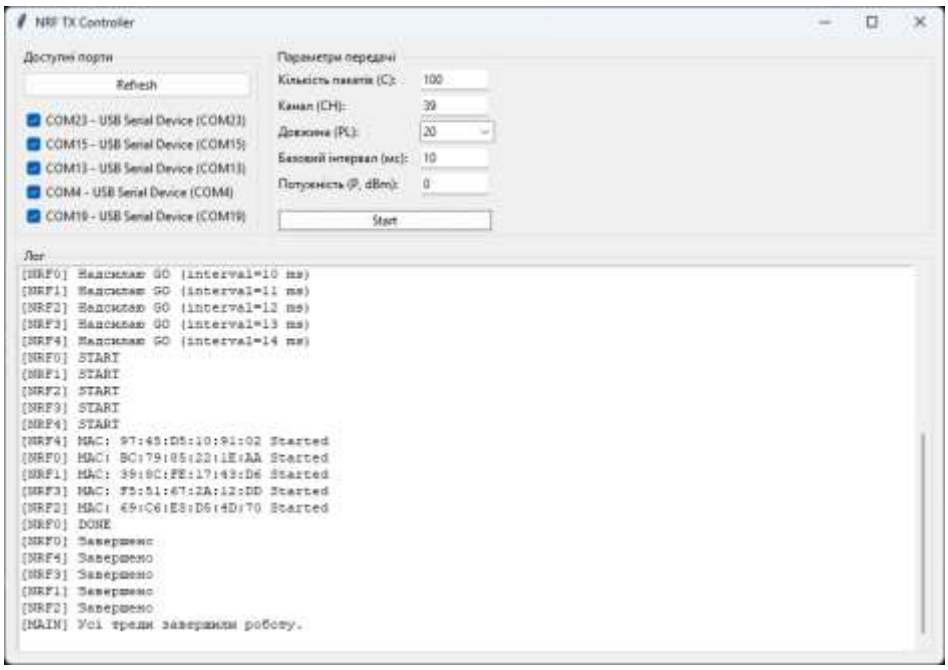


Рис. 4.1. Вид програми оркестратора для керування рекламними пристроями nRF52840

Ця Python-програма працює в багатопотоковому режимі і запускає стільки потоків, скільки обрано передавачів. Кожен з цих потоків працює за наступним алгоритмом:

– відкриває свій COM-порт для свого пристрою;
– надсилає конфігураційну команду SET;
– очікує подію запуску пристроїв, яка ініціалізовується шляхом натиснення на кнопку Start програмного засобу і після відправляє команду GO;
– запуск кожного окремого девайсу відбувається з сувом по часу, який дорівнює десяти відсоткам від значення базового інтервалу, яке налаштовується в програмному засобі для запобіганню накладанню пакетів;
– виводить лог кожного передавача окремо і паралельно.

Всі рекламні пристрої підключені до експериментального стенду за допомогою USB хабу на вісім пристроїв, як зображено на рис. 4.2.

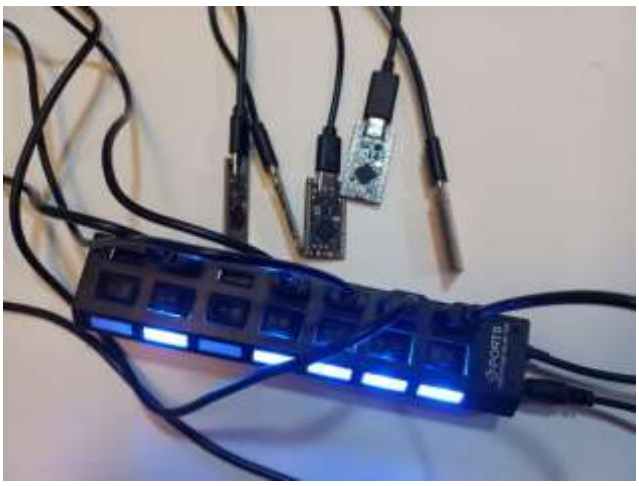


Рис. 4.2. Підключення рекламних пристроїв до експериментального стенду за допомогою USB-хабу

В результаті отримано програму-оркестратор, яка гарантує синхронізацію старту кожного з рекламних пристроїв, що є критичним для експерименту.

4.1.3. Методика проведення експерименту без завади і попередня обробка результатів

Під час проведення експерименту, передавачі розташовувались на відстані 60 см один від одного, приймачі, один з яких Android-застосунок і підключений до нього за допомогою кабелю Type-C – Type-C зовнішньої антени nRF52840, описаної в Розділі 3.3 даної роботи, другий з яких SmartRF, розташовувалися на відстані 4 метри від передавачів.

Експеримент проводиться в реальних умовах, без ізоляції від навколишнього середовища і застосування фільтрів за MAC-адресою, в два етапи: без генератора шуму і з генератором шуму. У другому експерименті генератор шуму розташовується посередині між передавачами і приймачем на відстні 2 метра від кожного.

Перший експеримент проводиться з наступними критеріями і змінними:

– канал реклами – 39;
– наявность генератора завад – ні;
– кількість пакетів прорекламована кожним девайсом – 1000;
– потужність сигналу – 0 дБмВт;

– базовий інтервал – змінний в значеннях від 10 до 40 мс;
– довжина пакету – змінна в значеннях від 1 байту до 30 байт.

Другий експеримент проводиться з наступними критеріями і змінними:

– канал реклами – 39;
– наявность генератора завад – так;
– кількість пакетів прорекламована кожним девайсом – 1000;
– потужність сигналу – в двох значеннях 0 і 8 дБмВт;
– базовий інтервал – змінний в значеннях від 10 до 40 мс;
– довжина пакету – змінна в значеннях від 1 до 30 байт.

Для кожної комбінації параметрів проводилась окрема серія вимірювань за наступним алгоритмом:

1) Всі пристрої NRF52840, на яких встановлено прошивку для генерації реклами, підключаються до USB-хабу, який в свою чергу підключається до ПК; оскільки живлення одного USB-порту не вистачає для живлення стількох девайсів, USB-хаб вимагає підключення до себе додаткового живлення;
2) Пристрій NRF52840, який є зовнішньою антеною під’єднується до Android-смартфону, що описано в Розділах 3.1–3.4 цієї роботи, запускається сканування BLE-трафіку;
3) Підключення USB-донглу SmartRF до ПК і запуск ПЗ SmartRF Packet Sniffer і старт сканування BLE-трафіку;
4) Запуск за допомогою оркестратора, який описаний в Розділі 4.3 цієї роботи всіх рекламних пристроїв;
5) Після відправки всіх пакетів за допомогою рекламних пристроїв сканування на всіх пристроях зупиняється і відбувається збереження даних:
    a. Android-застосунок для кожного прийнятого рекламного кадру формує окремий JSON-об’єкт, який у форматі ndjson, тобто по одному JSON-об’єкту на рядок, зберігається до файлу;

b. SmartRF Packet Sniffer зберігає результати у власному форматі PSD, де кожен запис містить інформацію про пакет, наприклад: номер пакета, часову мітку, довжину, статусні байти, сирий пакет у полі packetRaw;

Після проведення експерименту відбувається попередня обробка результатів. Для цього створено скрипт за допомогою мови програмування Python, код якого наведено в Додатку К, який:

– рекурсивно зчитує місце зберігання експериментальних даних та знаходить файли з розширеннями ‘ndjson’ і ‘psd’;
– для кожного ndjson-файлу читає рядки по одному JSON-об’єкту, відфільтровує записи за потрібними MAC-адресами, з JSON бере поле з корисними даними і записує його у текстовий файл як один hex-рядок на пакет;
– для кожного PSD-файлу пропускає службовий заголовок фіксованої довжини, розбиває файл на блоки, в яких один запис SmartRF з корисними даними, в них шукає поле packetRaw, виділяє BLE-кадр, відновлює з нього MAC-адресу, фільтруючи потрібні і з BLE-кадру виділяє саме рекламні дані зберігаючи їх як hex-рядок у відповідний текстовий файл.

Як результат, для кожної серії експериментів створюється пара текстових файлів, у яких в назві описано параметри експерименту.

4.2. Обробка та зіставлення даних з двох приймачів

Після проведення експерименту і передобробки отримананих даних, проводиться аналіз відповідності між пакетами, які були захоплені Android-приймачем і SmartRF. Для цього розроблено скрипт за допомогою мови програмування Python, код якого наведено в додатку Л, який проходить рекурсивно папку «results_postprocessed» (створена в попередньому розділі за допомогою програмного коду для передобробки даних), знаходить файли з пакетами з спец.назвою, та оброблює їх попарно за спільним набором параметрів.

На першому етапі обробки, скрипт перетворює знайдені текстові файли у табличне представлення, в якому:

– в Android-даних кожен рядок містить корисне навантаження пакету під назвою ‘payload_hex’ та MAC-адресу під назвою ‘mac’;
– в SmartRF-даних кожен рядок містить корисне навантаження пакету під назвою ‘col3’ і MAC-адресу під назвою ‘col2’.

Також, проведено оцінку ефіру за допомогою SDR HackRF на частоті 2,480 ГГц, що відповідає каналу 39, на якому рекламуються пристрої в експерименті. Згідно рис. 4.3. видно, що в ефірі доволі багато пристроїв інших користувачів технології BLE.

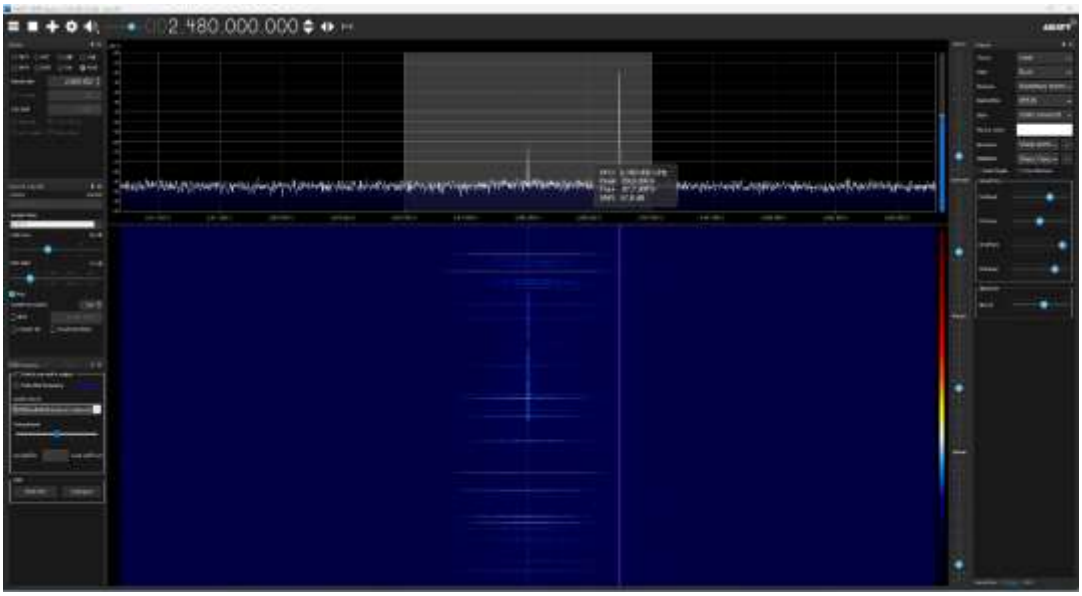


Рис. 4.3. Перевірка стану 39 каналу за допомогою HackRF

4.2.1. Відновлення номера пакетів та їхня класифікація

У BLE-пакетах перші два байти корисного навантаження використовуються як лічильник кадру, що знаходяться в діапазоні від 0 до 999. У скрипті обробки експериментальних даних цей номер відновлюється з hex-рядка за наступною логікою:

– видаляються пробіли з ‘payload_hex’;
– перші два байти інтерпретуються як 16-бітне ціле в little-endian порядку, тобто молодший байт – перший, старший – другий;

– отримане число зберігається в колонці під назвою ‘packet_no_dec’.

Таким чином, для кожного пакету з певного рекламного пристрою обидва датафрейми доповнюються декодованим номером пакету і сортуються за ним. Це дозволяє коректно зіставляти кадри між двома приймачами.

Для оцінки правильності пакета введено ознаку ‘is_clean’, значення якого відповідає за те, чи пакет вважається правильно-впійманим, якщо його payload відповідає одному з шаблонів:

– **для пакетів в яких довжина корисного навантаження дорівнює два** (тобто лише лічильник передається в пакеті): довжина hex-рядку дорівнює чотирьом символам і значення лічильнику пакету менше за 1000;
– **для пакетів в яких довжина корисного навантаження більше двох** (тобто лише лічильник і заповнювач передається в пакеті): довжина пакету більше за чотири символи, при цьому всі символи після перших чотирьох збігаються з заповнювачем, а саме ‘0×AA’.

В усіх інших випадках, таких як: неправильна довжина, відсутність номера або його неправильне значення, наявність байтів відмінних від заповнювачів – пакети вважаються поломаними.

Як результат, для кожного пакету з певною MAC-адресою і для кожного з приймачів окремо обчислюються:

– загальна кількість прийнятих пакетів;
– кількість правильних і неправильних пакетів;

На основі цих показників формується перша група узагальнюючих таблиць по всіх серіях експерименту – все це реалізовано за допомогою розробленого python-скрипту, який наведено в Додатку Л.

4.2.2. Обробка повних та часткових співпадіннь між приймачами

Окрім простого підрахунку пакетів, важливим є питання узгодженості спостережень двох приймачів. На основі цих множин визначаються два типи співпадінь:

– **повне співпадіння:** кадр з однаковим номером та ідентичним корисним навантаженням був прийнятий обома приймачами. Кількість таких кадрів позначається як «Повне співпадіння (всього)».
– **часткове співпадіння:** ситуація, коли номер кадру збігається, але вміст корисного навантаження після лічильника відрізняється. Для виявлення таких випадків payload-и групуються за номером кадру, після чого обчислюється кількість пар з однаковим лічильником та різними заповнювачами. Кількість таких кадрів позначено як «Часткове співпадіння (лічільник)».

4.3. Аналіз експериментальних даних

Всі результати для різних комбінацій параметрів (інтервал, довжина корисного навантаження, рівень потужності) зведені у підсумкові таблиці та візуалізовані у вигляді теплових карт. Це дозволяє оцінити закономірності впливу параметрів передавання на якість прийому двома незалежними приймачами nRF разом з Android та SmartRF.

4.3.1. Обробка прийнятих пакетів за допомогою Android-застосунку та SmartRF

У табл. 4.1 наведено залежність кількості прийнятих з переданих пакетів, які прийняті та правильно декодовані обома передавачами.

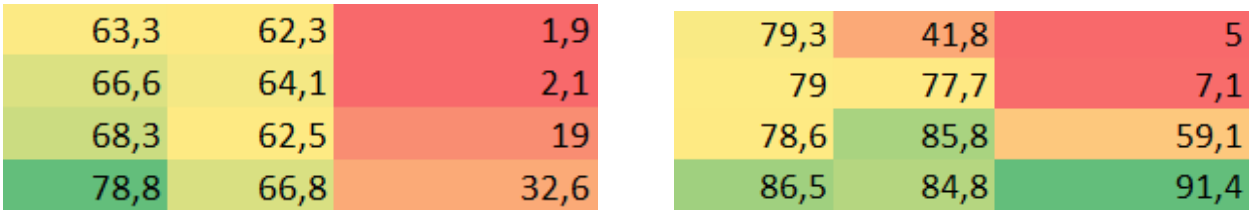
В таблиці значення подані у вигляді:

– перше число позначає середнє значення отриманих пакетів від кожного з рекламуючих пристроїв;
– друге число позначає девіацію, що відображає розкид результатів між окремими передавачами та повторними запусками.

Таблиця 4.1. Кількість отриманих правильних пакетів за допомогою nRF і SmartRF

Кількість отриманих даних, байт	Інтервал, мс	nRF, %	SmartRF, %
30	10	63,3±24,3	79,3±10,2
	20	62,3±19,4	41,8±3,2
	40	1,9±0,2	50,0±34,0
20	10	66,3±15,2	66,3±15,2
	20	64,1±29,1	64,1±29,1
	40	2,1±0,9	2,1±0,9
10	10	68,3±8,0	78,6±11,2
	20	62,5±13,9	85,8±2,2
	40	19,0±7,4	59,1±16,0
2	10	78,8±8,0	86,5±6,1
	20	66,8±11,7	84,8±79,0
	40	32,6±7,3	91,4±38,0

Для кращого розуміння, дані з табл. 4.1 візуалізованого за допомогою теплової карта, яка наведена на рис. 4.4.



а б

Рис. 4.4. Теплова карта розподілу кількості отриманих пакетів для nRF (а) і SmartRF (б)

Як видно з табл.4.1. та рис. 4.4а для Android-пристрою характерні наступні тенденції:

– при малих інтервалах в межах від 10 до 20 мс і великих за довжиною кадрах частка правильно-впійманих пакетів коливається в межах від 60 до 66%;
– при збільшенні інтервалу частка правильно-прийнятих кадрів різко падає до одиниць відсотків;
– для малих інтервалів і малих за довжиною кадрах (порядку 2 або 10 байт) відсоток правильно-прийнятих кадрів в межах від 68 до 78%.

Для SmartRF згідно табл.4.1 та рис. 4.4б видно, що:

– для довгих кадрів порядку 30 байтів частка правильно-прийнятих пакетів вища, ніж у nRF в межах від 41 до 80% при різних інтервалах реклами;
– для середніх і коротких довжин корисного навантаження в межах від 2 до 20 байтів пристрій демонструє прийом в межах від 60 до 90% правильно-прийнятих кадрів при різних інтервалах реклами;

Різке зниження кількості правильно-прийнятих кадрів на пристрої Android при великих інтервалах і довжинах кадрів пояснюється тим, що експеримент проведено в реальних умовах при великій кількості BLE-пристроїв навколо, яка рекламує себе. Оскільки пристрої, які рекламували себе за допомогою BLE-пакетів можуть змінюватись, було прийнято рішення проводити експеримент без фільтрів за MAC-адресами, що в свою чергу означає можливість втрати пакетів в точці відправлення прийнятого пакету від nRF до Android-застосунку. Наприклад: якщо на рекламному пристрої встановлено інтервал 40 мс, то в моментах між рекламними пакетах nRF оброблює і відправляє інші пакети від інших девайсів на Android-застосунок по кабелю.

На SmartRF такої тенденції не спостерігається, оскільки цей пристрій працює безпосередньо з ПК, в якому: більше ресурсів для обробки (наприклад центральних процесор), USB-порт може передавати більшу кількість даних за одиницю часу.

Виявлені недоліки системи вимагають подальшого дослідження проблеми і вдосконалення Android-застосунку і прошивки пристрою nRF52840, які описані в розділі 3 даної роботи[25-29].

4.3.2. Обробка прийнятих пакетів із помилками за допомогою Android-застосунку та SmartRF

У табл.4.2 наведено залежність кількості прийнятих з переданих пакетів, які прийняті та правильно декодовані обома передавачами.

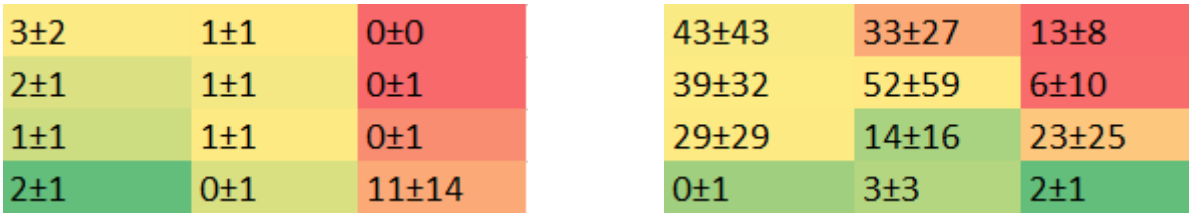
Таблиця 4.2. Кількість отриманих неправильних пакетів за допомогою nRF і SmartRF

Кількість отриманих даних, байт	Інтервал, мс	nRF, %	SmartRF, %
30	10	2,5	42,5
	20	0,8	32,8
	40	0	13,0
20	10	1,8	38,8
	20	1,0	52,0
	40	0,3	6,0
10	10	0,8	28,8
	20	1,0	13,8
	40	0,3	23,0
2	10	2,0	0,3
	20	0,3	2,5
	40	11,0	1,5

В табл. 4.2 значення подані у вигляді:

– перше число позначає середнє значення отриманих пакетів від кожного з рекламуючих пристроїв;
– друге число позначає девіацію, що відображає розкид результатів між окремими передавачами та повторними запусками.

Для кращого розуміння, дані з табл. 4.2 візуалізованого за допомогою теплової карта, яка наведена на рис. 4.5.



а б

Рис.4.5. Теплова карта розподілу кількості отриманих пакетів для nRF (а) і SmartRF (б)

Як видно з табл. 4.2. та рис.4.5а для Android-пристрою кількість неправильно-прийнятих пакетів на 1000 переданих становить:

– від 3 до 1 для більшості режимів;
– локальне зростання до 11 пакетів при інтервалі 40 мс і довжиною пакету 2 байти.

Тобто nRF або приймає пакет коректно, або не бачить його зовсім; випадків, коли payload спотворено, небагато. З огляду на абсолютні величини, частка поломаних кадрів для nRF не перевищує кількох відсотків.

Для SmartRF зігдно табл. 4.2 та рис.4.5б кількість «битих» пакетів вища: у деяких режимах значення досягають 43, 33, 52 пакетів на 1000, що також відображається червоними ділянками на тепловій карті. Це пояснюється, як було описано в Розділі 4.7.1 даної роботи тим, що SmartRF може обробити значно більше кадрів загалом, у тому числі й гранично ослаблені чи зашумлені пакети, які Android-приймач не встигає обробити, частина з яких декодується з помилками в payload.

Таким чином, даний аналіз в котре підтверджує описаний в Розділі 4.7.1 даної роботи висновок щодо необхідності подальшого дослідження даної тематики, а також вдосконалення програмного забезпечення для прошивки пристрою nRF52840.

4.3.3. Узгодження прийнятих пакетів між пристроями nRF52840 і SmartRF

Для порівняння роботи пристроїв nRF52840 і SmartRF співставимо прийняті ними пакети. Результат узгодження наведено у табл. 4.3.

Таблиця 4.3. Кількість співпадінь отриманих пакетів за допомогою nRF52840 і SmartRF

Кількість отриманих даних, байт	Інтервал, мс	Повне співпадіння, ‰	Часткове співпадіння, ‰
30	10	505	6
	20	371	11
	40	12	3
20	10	573	4
	20	352	1
	40	17	0
10	10	611	8
	20	496	24
	40	134	2
2	10	609	0
	20	530	1
	40	155	0

Для кожної комбінації параметрів порівняння отриманих пакетів між двома пристроями nRF52840 і SmartRF наведено:

– повні співпадіння – кадри, у яких лічильник і заповнювачі збігаються;
– часткові співпадіння – кадри з одаковим лічильником, але різними байтами після лічильника.

Згідно з наведеними даними в табл.4.3:

– для режимів з короткими інтервалами та недовгим пакетом кількість повних співпадінь сягає 611 кадрів;
– для режимів з довгими інтервалами і довгим пакетом кількість співпадінь зменшується до 12 кадрів, що корелює з низькою часткою правильно-прийнятих пакетів у nRF.

Кількість часткових співпадінь в усіх режимах невелика, а саме в діапазоні від 0 до 24 пакетів, що означає ситуації, коли номер пакета однаковий, а payload відрізняється між приймачами зустрічаються рідко. Тобто декодувальна здатність у пристроїв рівна.

4.4. Методика проведення експерименту з завадою

Методика експерименту з завадою повністю повторюється з експериментом без завади і описана в Розділі 4.5 за винятком наявності завади.

Пристрій, який було використано як генератор шуму – RF Generator FOR FM Radio WIFI Bluetooth Walkie-talkie Wireless communication signal. Існує багато його модифікацій, в роботі використано модифікацію ‘2.4Ghz’. Цей пристрій генерує завади з високою потужністю, а саме, 11 дБмВт, малого розміру і вимагає живлення 5 В [30].

Під час запуску генератору шуму також відслідковувався стан ефіру за допомогою SDR HackRF One, що зображено на рис. 4.6.

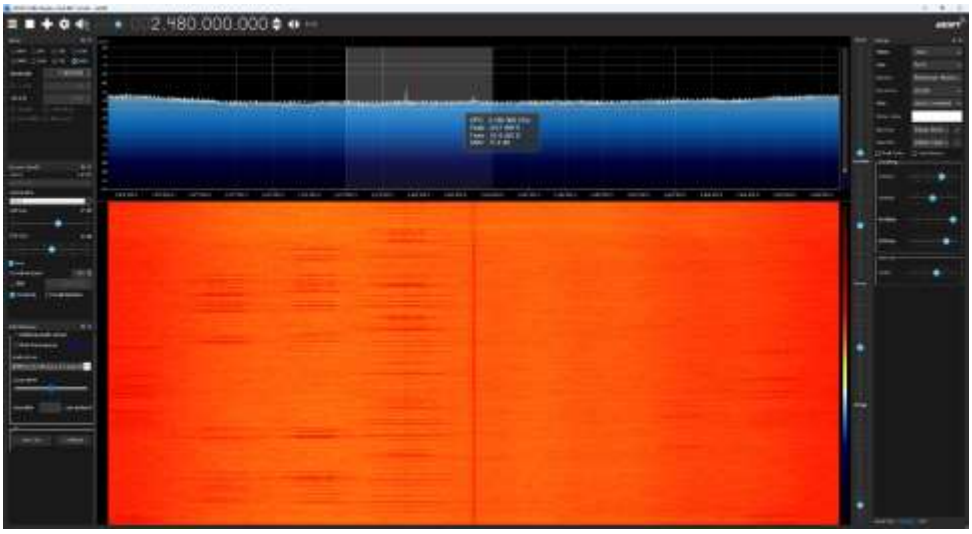


Рис.4.6. Демонстрація роботи генератора завад за допомогою SDR HackRF One

При використанні генератора завад, SNR в ефірі дорівнює 11,4 дБмВт, а згідно рис. 4.3 без завад – 57,9 дБмВт.

Оскільки генератор шуму має потужність 11 дБмВт, то було проведено передній аналіз при якому виявлено, що при потужностях передавачів nRF52840 менше ніж 8 дБмВт (найвища доступна потужність передавачів) сенсу проводити експеримент немає, оскільки приймачі взагалі нічого не ловлять.

В табл. 4.4 продемонстровано результати прийнятих пакетів при інтервалах передачі 10 мс і 20 мс, потужності передавачів 8 дБмВт і довжині пакету 2 байти.

Таблиця 4.4. Результати роботи антенту nRF52840 і SmartRF під дією завади при інтервалі передачі 10 і 20 мс

Кількість пакетів	Пристрій	Інтервал передачі, мс							
		10	20	10	20	10	20	10	20
		Передавач 1 39:8C:FE:17:43:D6		Передавач 2 69:C6:E8:D5:4D:70		Передавач 3 97:45:D5:10:91:02		Передавач 4 BC:79:85:22:1E:AA	
Відправлено	nRF52840	1000							
Отримано	nRF52840	4	1	2	2	2	1	1	—
без помилки		4	1	2	2	2	1	1	—
з помилкою		0	0	0	0	0	0	0	—
Отримано	SmartRF	0	0	0	0	0	0	0	—
без помилки		0	0	0	0	0	0	0	—
з помилкою		0	0	0	0	0	0	0	—

Згідно вищенаведених даних видно, що за наявності генератора завад радіоканал практично перестає бути придатним для передавання інформаційних BLE-пакетів за обраних параметрів.

Для інтервалу 10 мс, потужності передавачів 8 дБмВт та довжини пакета 2 байти (табл. 4.4) з 1000 переданих кадрів кожним із чотирьох пристроїв nRF52840 Android-приймач реєструє лише від 1 до 4 пакетів, причому всі вони є правильно декодованими. З цього випливає те, що окремі пакети можливо впіймати під дією завади, але частка отриманих кадрів вимірюється десятою відсотка. Донгл SmartRF у цьому режимі взагалі не реєструє жодного пакета, що вказує на повне заглушення сигналу корисних передавачів з боку генератора шуму.

Аналогічна картина спостерігається і для інтервалу 20 мс (табл. 4.5): кожен передавач nRF52840 знову відправляє 1000 кадрів, з яких Android-приймач приймає лише 1–2 пакети без помилок, тоді як SmartRF не фіксує жодного. З огляду на попередній аналіз спектру (рис. 4.7), де рівень сигналу генератора завад суттєво домінує над корисними передаваннями, такий результат є очікуваним: співвідношення сигнал/шум стає настільки низьким, що реалізовані приймальні тракти, орієнтовані на стандартний BLE-рівень потужності, не здатні надійно виділити рекламні пакети на фоні інтенсивної завади.

Таким чином, в умовах роботи потужного генератора шуму на тій самій частоті, що й рекламний 39-й канал, навіть максимальна доступна потужність передавачів nRF52840 (8 дБмВт) виявляється недостатньою для забезпечення стабільного прийому. Але, згідно вище описаного виявилось, що Android-застосунок разом з приймачем nRF52840 в умовах роботи генератора шуму працює краще, ніж донгл SmartRF.

Висновки до четвертого розділу

Четвертий розділ присвячено експериментальному дослідженню розробленого методу моделювання BLE-пакетів та створеного на його основі апаратно-програмного стенду. На базі плат TSTAR MICRO nRF52840 реалізовано джерела BLE-реклами з керованими параметрами, а за допомогою Python-

оркестратора забезпечено синхронізований запуск до восьми передавачів зі зручним налаштуванням параметрів реклами.

Застосовано два незалежних приймача: Android-застосунок із зовнішньою BLE-антеною на основі nRF52840 та апаратний сніффер SmartRF Packet Sniffer. Для кожної комбінації параметрів реклами зібрані дані у форматах ndjson та PSD були приведені до уніфікованого вигляду, автоматично оброблені й проаналізовані.

Запропонована методика обробки включає: виділення з BLE-пакетів лічильника і підрахунок кількості правильно та неправильно прийнятих кадрів, аналіз повних та часткових співпадінь між Android-приймачем та SmartRF.

Згідно цих даних побудовано підсумкові таблиці та теплові карти, які демонструють, що:

– для режимів з малими інтервалами та короткими або середніми payload’ами приймаються більшость переданих кадрів;
– збільшення інтервалу до 40 мілісекунд призводить до суттєвого падіння кількості прийнятих пакетів на Android-пристрої;
– повні співпадіння між двома приймачами для оптимальних режимів досягають 600 пакетів, кількість часткових співпадінь мала - вказує на узгодженість декодування в обох трактах та відсутності їх спотворення.

Окремо досліджено роботу системи в умовах потужної завади на робочому каналі. За наявності генератора шуму з потужністю близько 11 дБмВт та максимальній доступній потужності передавачів nRF52840 рівною 8 дБмВт Android-застосунок приймає одиниці кадрів з тисячі, а SmartRF у наведених серіях взагалі не фіксує жодного пакета. Це означає практично повне блокування каналу для корисної інформації і демонструє, що у такому режимі краще працює створений Android-застосунок, який працює разом з антеною реалізованою за допомогою nRF52840.

Отримані в цьому розділі результати демонструють необхідность розвитку і вдосконалення Android-застосунку, прошивки для антени nRF52840, а також у потребі покращення їхньої роботи в умовах наявності завад.

ВИСНОВКИ

Дана робота присвячена розробці та експериментальному дослідженню методу імітаційного моделювання конфліктної взаємодії інформативних та завадових BLE-сигналів у діапазоні 2,4 ГГц на основі спеціалізованого програмно-апаратного стенду. В результаті виконання мета була повністю досягнута.

В роботі досліджено архітектуру BLE-технології, її переваги, обмеження та актуальні напрямки наукових досліджень. Встановлено, що BLE є однією з ключових технологій завдяки низькому енергоспоживанню, розповсюдженності та простоті інтеграції. Проте робота в перенасиченому діапазоні 2,4 ГГц, обмежена пропускна здатність і специфіка реалізації стеків на мобільних платформах має проблеми пов’язані зі стійкостю зв’язку і безпекою. Окремо висвітлені принципи роботи BLE на Android, де політики енергозбереження та високорівневий доступ до стеку унеможливлюють дослідження на фізичному рівні.

На прикладі реальних рекламних кадрів від пристрою Xiaomi MiKettle розібрано структуру BLE-пакету і продемонстровано поля доступні в Android. Експериментально досліджено залежность RSSI й похибки оцінки відстані в ближній та дальній зонах антени за допомогою ESP32-S3 та Ubertooth One – неможливість правильного BLE-позиціонування без врахування ефекту ближньої зони. Модифікація утиліти ubertooth-btle дозволила отримати необхідну точність аналізу інтервалів між пакетами. Розроблений Android-застосунок для сканування та візуалізації BLE-пакетів став базисним програмним компонентом для майбутнього експериментального стенду.

Первинно в ролі зовнішньої антенти для Android застосунку використано плату розробника ESP32-S3, яка формувала JSON-записи з інформацією про пакети і передавала їх через USB CDC. Аналіз стеку NimBLE та архітектури ESP32 показав принципову неможливість отримання номера каналу та гнучкого керування радіоналаштуваннями на рівні прошивки. Цей критичний недолік став підставою для переходу на мікроконтролер nRF52840, який надає доступ до інформації про канал. На базі плати від ProMicro створено енергоефективну зовнішню антену з

доступом до фізичного рівня, адаптовано Android-застосунок до особливостей USB CDC. Порівняльні вимірювання споживання продемонстрували, що nRF52840 суттєво економніший за ESP32-S3, що є важливим для роботи на Android.

Додатково досліджено застосування HackRF One з Portapack як SDR-інструмента для приймання, аналізу й низькорівневого декодування BLE-кадрів, його придатність для навчальних завдань: візуалізації роботи РЕБ, приймання аналогових та цифрових сигналів. Водночас коректного девайтнінгу BLE-пакетів і перевірки CRC досягти не вдалось, що разом з некоректною роботою інших інструментів вказує на необхідність подальшого дослідження даного напрямку.

Для сценаріїв конфліктної взаємодії реалізовано фізичний експеримент без ізоляції від навколишнього середовища. Розроблено прошивку для плати nRF52840, в якій налаштовуються: канал, інтервал, потужність, довжина і кількість кадрів. Python-оркестратор забезпечив одночасне керування групою передавачів. Дані зібрані Android-застосунком з зовнішньою антеною nRF52840 та SmartRF автоматично оброблені, класифіковані й зіставлені між собою за номером кадру за допомогою Python-скриптів. Побудовано залежності частки коректно прийнятих пакетів від інтервалу, довжини кадру та наявності широкосмугових завад.

Наукова новизна роботи полягає в поєднанні кількох рівнів дослідження BLE-технології в єдиному методі: від аналізу протоколу та обмежень мобільних стеків до створення спеціалізованого апаратно-програмного комплексу з керованими джерелами BLE-трафіку та багатоканальним прийманням.

Практична цінність результатів полягає в тому, що розроблений стенд та програмне забезпечення можуть бути використані для тестування стійкості BLE та пристроїв працюючих за схожим принципом до завад, налаштування параметрів рекламних пакетів у масштабних розгортаннях та дослідження кібербезпеки безпроводових мереж. Побудована методика обробки даних та скрипти можуть бути адаптовані до інших джерел та типів безпроводових протоколів.

Отримані результати дають можливість розробити стійкі до завад безпроводові системи, здатних надійно працювати в умовах конфліктної взаємодії сигналів у перенасиченому ISM-діапазоні.

Оформлення результатів цього дослідження здійснювалося згідно з методичними рекомендаціями кафедри [31].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Baker, B., Woods, J., Reed, M. J., & Afford, M. (2024). A Survey of Short-Range Wireless Communication for Ultra-Low-Power Embedded Systems. Journal of Low Power Electronics and Applications, 14(2): 27. DOI: 10.3390/jlpea14020027.

2. Koulouras, G., Katsoulis, S., & Zantalis, F. (2025). Evolution of Bluetooth Technology: BLE in the IoT Ecosystem. Sensors, 25(4): 996. DOI: 10.3390/s25040996.

3. Tosi, J., Taffoni, F., Santacatterina, M., Sannino, R., & Formica, D. (2017). Performance Evaluation of Bluetooth Low Energy: A Systematic Review. Sensors, 17(12): 2898. DOI: 10.3390/s17122898.

4. Bulić, P., Kojek, G., & Biasizzo, A. (2019). Data Transmission Efficiency in Bluetooth Low Energy Versions. Sensors, 19(17): 3746. DOI: 10.3390/s19173746.

5. Siva, J., Yang, J., & Poellabauer, C. (2019). Connection-less BLE Performance Evaluation on Smartphones. Procedia Computer Science, 155: 51–58. DOI: 10.1016/j.procs.2019.08.011.

6. Eltholth, A. A. (2023). Improved Spectrum Coexistence in 2.4 GHz ISM Band Using Optimized Chaotic Frequency Hopping for Wi-Fi and Bluetooth Signals. Sensors, 23(11): 5183. DOI: 10.3390/s23115183.

7. Grigorik, I., & Gorovyi, S. (2023). Development of a Mobile Health Monitoring and Alert Application for Agricultural Workers. Inventions, 8(5): 133. DOI: 10.3390/inventions8050133.

8. Lonzetta, A.M., Cope, P., Campbell, J., Mohd, B. J., & Hayajneh, T. (2018). Security Vulnerabilities in Bluetooth Technology as Used in IoT. Journal of Sensor and Actuator Networks, 7(3): 28. DOI: 10.3390/jsan7030028.

9. Wu, J., Nan, Y., Kumar, V., Tian, D., Bianchi, A., Payer, M., & Xu, D. (2020). BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy. In: Proc. 14th USENIX Workshop on Offensive Technologies (WOOT 2020).

10. Greß, H., Krüger, B., & Tischhauser, E. (2025). The Newer, the More Secure? Standards-Compliant Bluetooth Low Energy Man-in-the-Middle Attacks on Fitness Trackers. Sensors, 25(6): 1815. DOI: 10.3390/s25061815.

11. Reverse engineering the MiBeacon protocol. *Passive BLE Monitor integration*. URL: https://home-is-where-you-hang-your-hack.github.io/ble_monitor/MiBeacon_protocol (дата звернення: 24.11.2025).

12. Device information service | bluetooth® technology website. *Bluetooth® Technology Website*. URL: https://www.bluetooth.com/specifications/specs/device-information-service-1-1/ (дата звернення: 24.11.2025).

13. Chip Series Comparison – ESP32-S3 – – ESP-IDF Programming Guide v5.0 documentation. Technical Documents | Espressif Systems. URL: https://docs.espressif.com/projects/esp-idf/en/v5.0/esp32s3/hw-reference/chip-series-comparison.html (дата звернення: 24.11.2025).

14. ESP Product Selector. ESP Product Selector. URL: https://products.espressif.com/#/product-comparison?type=SoC&names=ESP32-C6,ESP32-H2FH4S (дата звернення: 24.11.2025).

15. STMicroelectronics. Social Distancing Detection using Bluetooth® Low Energy: Application Note AN5508 [Електронний ресурс]. – Версія 1.0. – 2020. – Режим доступу: https://www.st.com/resource/en/application_note/an5508-social-distancing-detection-using-bluetooth-low-energy-stmicroelectronics.pdf (дата звернення: 24.09.2025).

16. GitHub. greatscottgadgets/ubertooth: Software, firmware, and hardware designs for Ubertooth. GitHub. URL: https://github.com/greatscottgadgets/ubertooth (дата звернення: 24.11.2025).

17. GitHub. Antohapura/android_ble_scaner: Android BLE sniffer with 3 mode: onboad, test, external type-c. GitHub. URL: https://github.com/Antohapura/android_ble_scaner (дата звернення: 24.11.2025).

18. ScanResult | API reference | Android Developers. Android Developers. URL: https://developer.android.com/reference/android/bluetooth/le/ScanResult (дата звернення: 24.11.2025).

19. API Reference – ESP32-S3 – ESP-IDF Programming Guide v5.0 documentation. Technical Documents | Espressif Systems. URL:

https://docs.espressif.com/projects/esp-idf/en/v5.0/esp32s3/api-reference/index.html (дата звернення: 11.06.2025).

20. Espressif Documentation. Wireless SoCs, Software, Cloud and AIoT Solutions | Espressif Systems. URL: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf (дата звернення: 24.11.2025).

21. DSDPlus Fast Lane Program – DSDPlus. DSDPlus – Digital Decoder. URL: https://www.dsdplus.com/dsdplus-fast-lane-program/ (дата звернення: 24.11.2025).

22. Staff, L. E. Getting Started With ESP32-CAM: A Beginner's Guide. Last Minute Engineers. URL: https://lastminuteengineers.com/getting-started-with-esp32-cam/#connecting-an-external-antenna-to-esp32cam (дата звернення: 24.11.2025).

23. Peregrine Semiconductor Corp. PE4302 – 6-bit, 31.5 dB, DC–4.0 GHz, 50 Ω RF Digital Attenuator: Product Specification [Електронний ресурс]. – Document No. 70-0056-04. – San Diego, CA: Peregrine Semiconductor Corp., 2008. – 11 с.

24. GitHub. Nikeshbajaj/Linear_Feedback_Shift_Register: PyLFSR. GitHub. URL: https://github.com/Nikeshbajaj/Linear_Feedback_Shift_Register (дата звернення: 24.11.2025).

25. Соколов, В., Новицький, А., & Бодненко, Д. (2025). Імітаційне моделювання конфліктної взаємодії BLE-пакетів. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 2(30), 662–681. https://doi.org/10.28925/2663-4023.2025.30.997 (Категорія Б).

26. Крючкова, Л., & Цмоканич, І. (2023). Удосконалення захисних впливів на небезпечні сигнали високочастотного нав'язування. Кібербезпека: освіта, наука, техніка, 3(19), 243–253. https://doi.org/10.28925/2663-4023.2023.19.243253

27. Крючкова, Л., & Леонтюк, Н. (2024). Програмно-апаратна реалізація алгоритму швидкої оцінки потужності Wi-Fi сигналу в точках простору урбанізованого приміщення. Кібербезпека: освіта, наука, техніка, 4(24), 241–256. https://doi.org/10.28925/2663-4023.2024.24.241256

28. Довженко, Н., Складанний, П., Іваніченко, Є., & Жильцов, О. (2025). Модель динамічної взаємодії безпілотних літальних апаратів із сенсорною

мережею для енергоефективного моніторингу. Кібербезпека: освіта, наука, техніка, 3(27), 466–478. https://doi.org/10.28925/2663-4023.2025.27.766

29. Novytskyi, A., Sokolov, V., Kriuchkova, L., & Skladannyi P. (2025). Determining the Error Distribution of BLE Beacons at Antenna Near and Far Fields. In Proceedings of the 4th Int. Conf. on Cyber Hygiene & Conflict Management in Global Information Networks (CH&CMiGIN'2025), Kyiv, vol. 4024, 133–143

30. INNOTION. YSGM232508: VCO, 8 dBm, 2010–2570 MHz: Datasheet. – Version 3.0. – Innotion, Inc., 2025. – 4 p. – Режим доступу: https://www.lcsc.com/datasheet/C52043361.pdf (дата звернення: 24.11.2025).

31. Жданова, Ю. Д., Складанний, П. М., & Шевченко, С. М. (2023). Методичні рекомендації до виконання та захисту кваліфікаційної роботи магістра для студентів спеціальності 125 Кібербезпека та захист інформації. https://elibrary.kubg.edu.ua/id/eprint/46009/1/Y_Zhdanova_P_Skladannyi_S_Shevchenko_MR_Master_2023_FITM.pdf

Додаток А

Програмний код прошивки плати розробника ESP32 яка використана як контрольоване джерело BLE-рекламних пакетів

```
#include <Arduino.h>
#include <WiFi.h>
#include <NimBLEDevice.h>
#include <NimBLEExtAdvertising.h>

#define BOOT_BTN_PIN 0

const char* AP_SSID = "ESP32_TestAP";
const int   AP_CHANNEL = 1;
const int   SERVER_PORT = 1234;
const int   BLE_PACKET_COUNT = 500;
const unsigned long BLE_ADV_DURATION = 500;
const unsigned long BLE_ADV_TIMEOUT = 500;
const unsigned long AFTER_CMD_DELAY = 3000;
const unsigned long CONFIRM_TIMEOUT = 10000;

static NimBLEExtAdvertising* pExtAdv;

WiFiServer cmdServer(SERVER_PORT);
bool     apActive = false;

void startSoftAP() {
  WiFi.softAP(AP_SSID);
  WiFi.softAPConfig(IPAddress(192,168,4,1), IPAddress(192,168,4,1),
IPAddress(255,255,255,0));
  cmdServer.begin();
  apActive = true;
  Serial.println("[WiFi] SoftAP started, waiting for start_test...");
}

void stopSoftAP() {
  cmdServer.close();
  WiFi.softAPdisconnect(true);
  apActive = false;
  Serial.println("[WiFi] SoftAP stopped");
}

void sendBLEPacket(int packetNum) {
  NimBLEExtAdvertisement extAdv;
  extAdv.setLegacyAdvertising(true);
```

```
  String payload = String(packetNum);
  extAdv.setName(payload.c_str());
  extAdv.setPrimaryChannels(true, true, true);
  if (pExtAdv->isAdvertising()) {
    pExtAdv->stop();
    delay(10);
  }
  pExtAdv->removeInstance(0);
  if (!pExtAdv->setInstanceData(0, extAdv)) {
    Serial.println("[BLE] Failed to set packet data");
    return;
  }
  if (pExtAdv->start(0, /*duration*/0, /*maxEvents*/1)) {
    unsigned long startMs = millis();
    while (pExtAdv->isAdvertising() && (millis() - startMs < BLE_ADV_TIMEOUT)) {
      delay(10);
    }
    if (!pExtAdv->isAdvertising()) {
      Serial.printf("[BLE] Packet %d advertised\n", packetNum);
    } else {
      Serial.printf("[BLE] Packet %d timed out, stopping...\n", packetNum);
      pExtAdv->stop();
    }
  } else {
    Serial.printf("[BLE] Failed to advertise packet %d\n", packetNum);
  }
}
void setup() {
  Serial.begin(115200);
  while (!Serial) delay(10);
  NimBLEDevice::init("Raw Test");
  NimBLEDevice::setPower(ESP_PWR_LVL_P21, NimBLETxPowerType::All);
  pExtAdv = NimBLEDevice::getAdvertising();
  startSoftAP();

  pinMode(BOOT_BTN_PIN, INPUT_PULLUP);
}
void loop() {
  if (apActive) {
    WiFiClient client = cmdServer.available();
    if (client) {
      Serial.println("[WiFi] Client connected");
      client.setTimeout(3000);
      unsigned long t0 = millis();
      while (!client.available() && millis() - t0 < 3000) {
```

```
            delay(10);
          }
          String cmd = client.readStringUntil('\n');
          cmd.trim();
          if (cmd.length() == 0 && client.available()) {
            char buf[64] = {0};
            int len = client.readBytes(buf, sizeof(buf)-1);
            if (len > 0) {
              cmd = String(buf);
              cmd.trim();
            }
          }
          Serial.printf("[WiFi]   Received   command:   '%s'   (len=%d)\n",   cmd.c_str(),
cmd.length());
          if (cmd == "start_test") {
            String mac = NimBLEDevice::getAddress().toString().c_str();
            mac.toUpperCase();
            client.printf("%s\n", mac.c_str());
            Serial.printf("[WiFi] Sent MAC: %s\n", mac.c_str());
            Serial.println("[WiFi] Waiting for 'received' confirmation...");
            unsigned long t1 = millis();
            String confirm = "";
            bool ok = false;
            while (millis() - t1 < 15000) {
              if (client.available()) {
                confirm = client.readStringUntil('\n');
                confirm.trim();
                Serial.printf("[WiFi] Got confirmation candidate: '%s'", confirm.c_str());
                if ( confirm.equalsIgnoreCase("received")) {
                  ok = true;
                  break;
                } else {
                  Serial.println("[WiFi] Not equal confirmation");
                }
              }
              delay(10);
            }
            if (ok) {
            Serial.printf("[WiFi] Confirmation: '%s'\n", confirm.c_str());
            client.stop();
            stopSoftAP();
            WiFi.mode(WIFI_OFF);

    Serial.println("[WiFi] Wi-Fi radio OFF");
            delay(3000);
            for (int i = 1; i <= BLE_PACKET_COUNT; i++) {
              sendBLEPacket(i);
```

```
          delay(50);
        }
        WiFi.mode(WIFI_AP);
        startSoftAP();
      } else {
        Serial.println("[WiFi] ERROR: no valid 'received' confirmation, aborting test");
        client.stop();
      }
      } else {
        client.stop();
      }
    }
  }
  static bool prevBtnState = HIGH;
  bool btnState = digitalRead(BOOT_BTN_PIN);
  if (prevBtnState == HIGH && btnState == LOW) {
    Serial.println("[BTN] BOOT натиснуто! Запуск BLE-послідовності.");
    for (int i = 1; i <= BLE_PACKET_COUNT; i++) {
      sendBLEPacket(i);
      delay(50);
    }
    while (digitalRead(BOOT_BTN_PIN) == LOW) {
      delay(10);
    }
    Serial.println("[BTN] BLE-тест завершено, чекаємо новий натиск.");
  }
  prevBtnState = btnState;
}
```

Додаток Б

Програмний код керуванням відправленням пакетів за допомогою ESP32-S3

```
import serial
import time

# налаштування
SERIAL_PORT   = '/dev/ttyACM0'
# SERIAL_PORT   = 'COM11'
BAUDRATE      = 115200
MAC_FILTER    = '10:20:ba:4b:b1:41'
ITERATIONS    = 1000
idle_timeout  = 10
def measure_one(ser, num_iter):
    sent_ts = time.time_ns()
    ser.write(f"{sent_ts}\n".encode())
    last_try = time.time()
    num_try = 0
    while True:
        num_try += 1
        print(f"Try #{num_try} for {sent_ts} ## {num_iter} item")
        timeout = idle_timeout - int((time.time() - last_try))
        if timeout <= 0:
            break
        line = ser.readline()  # читаємо байти до '\n'
        if line and str(line).startswith("b'Advertised done"):
            break
    return 1

def main():
    ser = serial.Serial(SERIAL_PORT, BAUDRATE, timeout=1)
    try:
        for i in range(ITERATIONS):
            measure_one(ser, i)
    finally:
        ser.close()

if __name__ == '__main__':
    main()
```

Додаток В

Програмний код ESP32-S3 для використання як зовнішньої антени

```
#include <Arduino.h>
#include <NimBLEDevice.h>
#include <ArduinoJson.h>
#include "esp_bt.h"
#include "esp_coexist.h"
typedef enum {
  SCAN_ALL_CHANNLE,
  ONLY_SCAN_CHANNEL_37,
  ONLY_SCAN_CHANNEL_38,
  ONLY_SCAN_CHANNEL_39,
  SCAN_MODE_TYPE_ERROR,
} scan_mode_config_t;
extern "C" uint8_t esp_ble_scan_channel_setting(scan_mode_config_t scan_mode);
static const bool   SEND_OVER_USB       = true;
static const bool   SEND_OVER_USB_DEBUG = true;
static const size_t JSON_BUF            = 768;
static const uint32_t SERIAL_BAUD       = 921600;
static int gAdvChannel = 0;
static bool applyFixedScanChannel(int ch) {
  uint8_t mode = SCAN_ALL_CHANNLE;
  if      (ch == 37) mode = ONLY_SCAN_CHANNEL_37;
  else if (ch == 38) mode = ONLY_SCAN_CHANNEL_38;
  else if (ch == 39) mode = ONLY_SCAN_CHANNEL_39;
  else if (ch != 0)  return false;
  const uint8_t rc =
esp_ble_scan_channel_setting(static_cast<scan_mode_config_t>(mode));
  if (SEND_OVER_USB_DEBUG)
    Serial.printf("{\"dbg\":\"scan_ch_set\",\"ch\":%d,\"rc\":%u}\n", ch, rc);
  return (rc == 0);
}

static inline String toHex(const uint8_t* data, size_t len) {
  static const char* HEXCHARS = "0123456789abcdef";
  String out; out.reserve(len*2);
  for (size_t i=0;i<len;++i){ out += HEXCHARS[(data[i]>>4)&0xF]; out +=
HEXCHARS[data[i]&0xF]; }
  return out;
}
static inline String jsonEscape(const std::string& s) {
  String out; out.reserve(s.size());
  for (char c: s){
    switch(c){
      case '\"': out+="\\\""; break; case '\\': out+="\\\\"; break;
```

```
            case '\b': out+="\\b";  break; case '\f': out+="\\f";  break;
            case '\n': out+="\\n";  break; case '\r': out+="\\r";  break;
            case '\t': out+="\\t";  break;
            default: if((uint8_t)c<0x20){ char buf[7];
snprintf(buf,sizeof(buf),"\\u%04x",(unsigned char)c); out+=buf; } else out+=c;
          }
        }
        return out;
      }
      static inline String normMac(String s) {
        s.toUpperCase();
        String out; out.reserve(12);
        for (size_t i=0;i<s.length();++i){ char c=s[i]; if(c!=':' && c!='-') out+=c; }
        return out;
      }
      static NimBLEScan* gScan = nullptr;
      static bool        gScanRunning = false;
      static bool        gActiveScan  = true;
      static int         gRssiMin     = -100;
      static std::vector<String> gAllowMacs;
      static std::vector<String> gBlockMacs;
      static portMUX_TYPE gCfgMux = portMUX_INITIALIZER_UNLOCKED;
      static bool addrInList(const String& mac12, const std::vector<String>& lst){
        for (auto &m: lst) if (mac12 == m) return true;
        return false;
      }
      static bool passFilters(const NimBLEAdvertisedDevice* dev) {
        int rssi = dev->getRSSI();
        if (rssi < gRssiMin) return false;

        String mac12 = normMac(String(dev->getAddress().toString().c_str()));
        taskENTER_CRITICAL(&gCfgMux);
        bool allowHas = !gAllowMacs.empty();
        bool allowHit = allowHas && addrInList(mac12, gAllowMacs);
        bool blockHas = !gBlockMacs.empty();
        bool blockHit = blockHas && addrInList(mac12, gBlockMacs);
        taskEXIT_CRITICAL(&gCfgMux);

        if (allowHas)   return allowHit;
        if (blockHas)   return !blockHit;
        return true;
      }
      class AdvCallbacks : public NimBLEScanCallbacks {
      public:
        void onResult(const NimBLEAdvertisedDevice* dev) {
          if (!passFilters(dev)) return;
```

```
        const auto& payloadVec = dev->getPayload();
        const uint8_t* p       = payloadVec.data();
        const size_t   plen    = payloadVec.size();

        bool isScanResp = false;
    #ifdef BLE_HCI_ADV_RPT_EVTYPE_SCAN_RSP
        isScanResp = (dev->getAdvType() == BLE_HCI_ADV_RPT_EVTYPE_SCAN_RSP);
    #else
        isScanResp = (dev->getAdvType() == 0x04);
    #endif
        const int rssi            = dev->getRSSI();
        const std::string nameStd = dev->haveName() ? dev->getName() : "";
        const String nameEsc      = jsonEscape(nameStd);
        const std::string addr    = dev->getAddress().toString();
        const String payloadHex   = toHex(p, plen);
        const unsigned long ts_ms = millis();

        char buf[JSON_BUF];
        int n = snprintf(buf, sizeof(buf),
          "{\"ts_ms\":%lu,\"mac\":\"%s\",\"rssi\":%d,\"is_scan_response\":%s,"
          "\"name\":\"%s\",\"payload_hex\":\"%s\"}",
          ts_ms, addr.c_str(), rssi, isScanResp ? "true" : "false",
          nameEsc.c_str(), payloadHex.c_str());

        if (n < 0 || (size_t)n >= sizeof(buf)) {
          size_t L = strlen(buf);
          if (L >= 2) { buf[sizeof(buf)-2] = '}'; buf[sizeof(buf)-1] = '\0'; }
        }
        if (SEND_OVER_USB) Serial.println(buf);
      }
      void onScanStart() { if (SEND_OVER_USB_DEBUG)
Serial.println("{\"event\":\"onScanStart\"}"); }
      void onScanEnd(NimBLEScanResults) { if (SEND_OVER_USB_DEBUG)
Serial.println("{\"event\":\"onScanEnd\"}"); }
    };
    static AdvCallbacks gAdvCb;
    static void startBleScanIfNeeded() {
      esp_coex_preference_set(ESP_COEX_PREFER_BT);
      if (!gScan) { if (SEND_OVER_USB_DEBUG) Serial.println("{\"err\":\"no_scan_obj\"}");
return; }
      if (gScanRunning) { if (SEND_OVER_USB_DEBUG)
Serial.println("{\"event\":\"scan_already_running\"}"); return; }

      gScan->setDuplicateFilter(false);
      gScan->setMaxResults(0);
      gScan->setActiveScan(gActiveScan);
      gScan->setInterval(16);
```

```
        gScan->setWindow(16);
        bool chApplied = applyFixedScanChannel(gAdvChannel);
        if (SEND_OVER_USB_DEBUG) {
          Serial.printf("{\"dbg\":\"apply_channel\",\"ch\":%d,\"ok\":%s}\n",
                        gAdvChannel, chApplied ? "true":"false");
        }
        gScan->start(0, false, false);
        delay(5);
        gScanRunning = gScan->isScanning();
        if (SEND_OVER_USB_DEBUG) {
          Serial.printf("{\"event\":\"ble_scan_started\",\"ok\":%s}\n", gScanRunning ?
"true" : "false");
        }
      }
      static void stopBleScanIfNeeded() {
        if (!gScan || !gScanRunning) return;
        gScan->stop();
        gScanRunning = false;
        if (SEND_OVER_USB_DEBUG) Serial.println("{\"event\":\"ble_scan_stopped\"}");
      }
      static void initBLE() {
        esp_bt_controller_mem_release(ESP_BT_MODE_CLASSIC_BT);
        NimBLEDevice::init("");
        gScan = NimBLEDevice::getScan();
        gScan->setScanCallbacks(&gAdvCb);
      }
      static String gLine;
      static void sendAck(const char* type, const char* msg) {
        StaticJsonDocument<192> doc;
        doc[type] = msg;
        String s; serializeJson(doc, s);
        Serial.println(s);
      }
      static void sendAckKV(const char* type, JsonDocument& payload) {
        StaticJsonDocument<256> wrapper;
        wrapper[type] = true;
        for (auto kv : payload.as<JsonObjectConst>()) {
          wrapper[String(kv.key().c_str())] = kv.value();
        }
        String s; serializeJson(wrapper, s);
        Serial.println(s);
      }
      static bool parseMacList(JsonVariant v, std::vector<String>& out) {
        if (!v.is<JsonArray>()) return false;
        out.clear();
        for (JsonVariant m : v.as<JsonArray>()) {
          String mac = normMac(m.as<const char*>());
```

```
      if (mac.length() == 12) out.push_back(mac);
    }
    return true;
  }
  static void handleJsonCommand(const String& line) {
    StaticJsonDocument<512> doc;
    DeserializationError err = deserializeJson(doc, line);
    if (err) { sendAck("err", "bad_json"); return; }
    const char* cmd = doc["cmd"] | "";
    if (!*cmd) { sendAck("err", "missing_cmd"); return; }

    if (strcmp(cmd, "set_adv_channel")==0) {
      int v = doc["ch"] | 0;
      if (!(v==0 || v==37 || v==38 || v==39)) { sendAck("err","bad_adv_channel");
return; }
      stopBleScanIfNeeded();
      taskENTER_CRITICAL(&gCfgMux);
      gAdvChannel = v;
      taskEXIT_CRITICAL(&gCfgMux);
      bool chApplied = applyFixedScanChannel(gAdvChannel);
      startBleScanIfNeeded();
      StaticJsonDocument<128> p;
      p["adv_channel"] = gAdvChannel;
      p["adv_channel_applied"] = chApplied;
      sendAckKV("ack", p);
      return;
    }
    if (strcmp(cmd, "set_mac_filter")==0) {
      std::vector<String> allow, block;
      bool okAllow = !doc.containsKey("allow") || parseMacList(doc["allow"], allow);
      bool okBlock = !doc.containsKey("block") || parseMacList(doc["block"], block);
      if (!okAllow || !okBlock){ sendAck("err","bad_mac_list"); return; }

      int reqCh = 0; bool hasCh = false;
      if (doc.containsKey("adv_channel")) {
        int v = doc["adv_channel"] | 0;
        if (v==0 || v==37 || v==38 || v==39) { reqCh = v; hasCh = true; }
        else { sendAck("err","bad_adv_channel"); return; }
      }
      stopBleScanIfNeeded();
      taskENTER_CRITICAL(&gCfgMux);
      gAllowMacs = std::move(allow);
      gBlockMacs = std::move(block);
      if (hasCh) gAdvChannel = reqCh;
      taskEXIT_CRITICAL(&gCfgMux);
      bool chApplied = true;
      if (hasCh) {
```

```
    chApplied = applyFixedScanChannel(gAdvChannel);
  }
  startBleScanIfNeeded();
  StaticJsonDocument<192> payload;
  payload["allow_size"] = (int)gAllowMacs.size();
  payload["block_size"] = (int)gBlockMacs.size();
  if (hasCh) {
    payload["adv_channel"] = gAdvChannel;
    payload["adv_channel_applied"] = chApplied;
  }
  sendAckKV("ack", payload);
  return;
}
if (strcmp(cmd, "add_mac")==0 || strcmp(cmd,"remove_mac")==0) {
  const char* list = doc["list"] | "allow";
  const char* macs = doc["mac"] | "";
  String mac1 = normMac(String(macs));
  if (mac1.length()!=12){ sendAck("err","bad_mac"); return; }

  stopBleScanIfNeeded();
  taskENTER_CRITICAL(&gCfgMux);
  auto& target = (strcmp(list,"block")==0) ? gBlockMacs : gAllowMacs;
  if (strcmp(cmd,"add_mac")==0) {
    if (!addrInList(mac1, target)) target.push_back(mac1);
  } else {
    target.erase(std::remove_if(target.begin(), target.end(),
                  [&](const String& s){ return s==mac1; }), target.end());
  }
  taskEXIT_CRITICAL(&gCfgMux);
  startBleScanIfNeeded();
  StaticJsonDocument<160> payload;
  payload["list"] = list;
  payload["size"] = (int)target.size();
  sendAckKV("ack", payload);
  return;
}
if (strcmp(cmd, "clear_filters")==0) {
  stopBleScanIfNeeded();
  taskENTER_CRITICAL(&gCfgMux);
  gAllowMacs.clear(); gBlockMacs.clear();
  taskEXIT_CRITICAL(&gCfgMux);
  startBleScanIfNeeded();
  sendAck("ack","cleared");
  return;
}
if (strcmp(cmd, "set_rssi_min")==0) {
  int v = doc["rssi"] | -100;
```

```
        if (v<-127 || v>20){ sendAck("err","rssi_out_of_range"); return; }
        stopBleScanIfNeeded();
        gRssiMin = v;
        startBleScanIfNeeded();
        StaticJsonDocument<96> payload; payload["rssi"]=gRssiMin; sendAckKV("ack",
payload);
        return;
      }
      if (strcmp(cmd, "scan_start")==0) { startBleScanIfNeeded();
sendAck("ack","scan_started"); return; }
      if (strcmp(cmd, "scan_stop")==0)  { stopBleScanIfNeeded();
sendAck("ack","scan_stopped"); return; }
      if (strcmp(cmd, "set_active_scan")==0) {
        bool v = doc["active"] | true;
        stopBleScanIfNeeded();
        gActiveScan = v;
        startBleScanIfNeeded();
        StaticJsonDocument<96> p; p["active"]=gActiveScan; sendAckKV("ack", p);
        return;
      }
      if (strcmp(cmd, "set_scan_params")==0) {
        uint16_t interval = doc["interval"] | 16;
        uint16_t window   = doc["window"]   | 16;
        stopBleScanIfNeeded();
        gScan->setInterval(interval);
        gScan->setWindow(window);
        startBleScanIfNeeded();
        StaticJsonDocument<128> p; p["interval"]=interval; p["window"]=window;
sendAckKV("ack", p);
        return;
      }
      sendAck("err","unknown_cmd");
    }
    static void pumpSerial() {
      while (Serial.available() > 0) {
        int c = Serial.read();
        if (c == '\r') continue;
        if (c == '\n') {
          if (gLine.length()) {
            handleJsonCommand(gLine);
            gLine = "";
          }
        } else {
          if (gLine.length() < 512) gLine += (char)c;
          else { gLine = ""; sendAck("err","line_too_long"); }
        }
      }
```

```
}
void setup() {
  Serial.begin(SERIAL_BAUD);
  delay(300);
  initBLE();
  startBleScanIfNeeded();
  if (SEND_OVER_USB_DEBUG)
Serial.println("{\"event\":\"started\",\"mode\":\"UART_ONLY\"}");
}
void loop() {
  pumpSerial();
  delay(2);
}
```

Додаток Г

Програмний код nRF52840 для використання як зовнішню антену для Android-застосунку

```
#include <Arduino.h>
#undef min
#undef max
#include <vector>
#include <string>
#include <algorithm>
#include <bluefruit.h>
#include <ArduinoJson.h>
static constexpr bool   SEND_OVER_USB       = true;
static constexpr bool   SEND_OVER_USB_DEBUG = false;
static constexpr size_t JSON_BUF            = 768;
static constexpr uint32_t SERIAL_BAUD       = 921600;
#define USE_SOFTDEVICE_CHANNEL_MASK 1
static volatile bool gScanRunning = false;
static bool  gActiveScan = true;
static int   gRssiMin    = -100;
static int   gAdvChannel = 0;

static std::vector<String> gAllowMacs;
static std::vector<String> gBlockMacs;
static inline String toHex(const uint8_t* data, size_t len) {
  static const char* hex = "0123456789abcdef";
  String out; out.reserve(len*2);
  for (size_t i=0;i<len;++i){ out += hex[(data[i]>>4)&0xF]; out += hex[data[i]&0xF]; }
  return out;
}
static inline String jsonEscape(const std::string& s) {
  String out; out.reserve(s.size());
  for (char c: s){
    switch(c){
      case '\"': out+="\\\""; break;
      case '\\': out+="\\\\"; break;
      case '\b': out+="\\b"; break;
      case '\f': out+="\\f"; break;
      case '\n': out+="\\n"; break;
      case '\r': out+="\\r"; break;
      case '\t': out+="\\t"; break;
      default:
        if ((uint8_t)c < 0x20) { char b[7]; snprintf(b,sizeof(b),"\\u%04x",(unsigned
char)c); out+=b; }
        else out+=c;
    }
```

```
    }
    return out;
  }
  static inline String normMac(String s) {
    s.toUpperCase();
    String out; out.reserve(12);
    for (size_t i=0;i<s.length();++i){ char c=s[i]; if (c!=':' && c!='-') out+=c; }
    return out;
  }
  static bool addrInList(const String& mac12, const std::vector<String>& lst){
    for (auto &m: lst) if (mac12 == m) return true;
    return false;
  }
  static void sendAck(const char* type, const char* msg) {
    StaticJsonDocument<192> doc; doc[type] = msg;
    String s; serializeJson(doc, s); Serial.println(s);
  }
  static void sendAckKV(const char* type, JsonDocument& payload) {
    StaticJsonDocument<256> wrap; wrap[type] = true;
    for (auto kv : payload.as<JsonObjectConst>()) wrap[String(kv.key().c_str())] =
kv.value();
    String s; serializeJson(wrap, s); Serial.println(s);
  }
  static bool parseMacList(JsonVariant v, std::vector<String>& out) {
    if (!v.is<JsonArray>()) return false;
    out.clear();
    for (JsonVariant m : v.as<JsonArray>()) {
      String mac = normMac(m.as<const char*>()); if (mac.length()==12)
out.push_back(mac);
    }
    return true;
  }
  extern "C" uint32_t sd_ble_gap_scan_start(ble_gap_scan_params_t const * p_scan_params,
ble_data_t const * p_adv_report_buffer);
  extern "C" uint32_t sd_ble_gap_scan_stop(void);
  static ble_gap_scan_params_t g_sdScanParams;
  static uint8_t g_sdScanBuffer[256];
  static ble_data_t g_sdReportBuf = { .p_data = g_sdScanBuffer, .len =
sizeof(g_sdScanBuffer) };
  static void configureScannerBasic() {
    Bluefruit.Scanner.setRxCallback(nullptr);
    Bluefruit.Scanner.useActiveScan(gActiveScan);
  }
  static void stopBleScanIfNeeded() {
    if (gScanRunning) {
      sd_ble_gap_scan_stop();
      gScanRunning = false;
```

```
      if (SEND_OVER_USB_DEBUG) Serial.println("{\"event\":\"ble_scan_stopped(sd)\"}");
    }
    Bluefruit.Scanner.stop();
  }
  static void setChannelMaskFromChoice(int ch, ble_gap_scan_params_t& p) {
    memset(p.channel_mask, 0, sizeof(p.channel_mask));
    if      (ch == 37) p.channel_mask[4] = (1u<<6) | (1u<<7);
    else if (ch == 38) p.channel_mask[4] = (1u<<5) | (1u<<7);
    else if (ch == 39) p.channel_mask[4] = (1u<<5) | (1u<<6);
  }
  static void attachScanCallback();
  static void startBleScanIfNeeded() {
    memset(&g_sdScanParams, 0, sizeof(g_sdScanParams));
    g_sdScanParams.active   = gActiveScan ? 1 : 0;
    g_sdScanParams.interval = 16;
    g_sdScanParams.window   = 16;
    g_sdScanParams.timeout  = 0;
    setChannelMaskFromChoice(gAdvChannel, g_sdScanParams);
    Bluefruit.Scanner.stop();
    sd_ble_gap_scan_stop();
    attachScanCallback();
    uint32_t err = sd_ble_gap_scan_start(&g_sdScanParams, &g_sdReportBuf);
    gScanRunning = (err == NRF_SUCCESS);
    if (SEND_OVER_USB_DEBUG) {
Serial.printf("{\"event\":\"ble_scan_started(sd)\",\"ok\":%s,\"err\":%u,\"mask4\":%u}\n",
                  gScanRunning?"true":"false", (unsigned)err,
(unsigned)g_sdScanParams.channel_mask[4]);
    }
  }
  static bool passFilters(const ble_gap_evt_adv_report_t* report,
                          String& mac12, int& rssi, bool& isScanResp,
                          uint8_t& chOut, String& nameEsc, String& payloadHex)
  {
    rssi = report->rssi;
    if (rssi < gRssiMin) return false;
    uint8_t ch = report->ch_index;
    chOut = ch;
    if ((gAdvChannel == 37 || gAdvChannel == 38 || gAdvChannel == 39) && ch !=
gAdvChannel)
      return false;
    const ble_gap_addr_t& a = report->peer_addr;
    char macbuf[18];
    snprintf(macbuf, sizeof(macbuf), "%02X:%02X:%02X:%02X:%02X:%02X",
             a.addr[5], a.addr[4], a.addr[3], a.addr[2], a.addr[1], a.addr[0]);
    mac12 = normMac(String(macbuf));
    if (!gAllowMacs.empty()) {
      if (!addrInList(mac12, gAllowMacs)) return false;
```

```
    } else if (!gBlockMacs.empty()) {
      if (addrInList(mac12, gBlockMacs)) return false;
    }
    isScanResp = (report->type.scan_response != 0);
    std::string nameStd;
    {
      const uint8_t* p = report->data.p_data;
      size_t len = report->data.len;
      while (len > 2) {
        uint8_t dlen = p[0];
        if (dlen == 0 || dlen + 1 > len) break;
        uint8_t type = p[1];
        if (type == 0x09 || type == 0x08) {
          nameStd.assign((const char*)(p+2), (size_t)(dlen-1));
          break;
        }
        size_t advlen = (size_t)dlen + 1;
        p   += advlen;
        len -= advlen;
      }
    }
    nameEsc = jsonEscape(nameStd);
    payloadHex = toHex(report->data.p_data, report->data.len);
    return true;
  }
  static void rx_callback(ble_gap_evt_adv_report_t* report) {
    Bluefruit.Scanner.resume();
    String mac12, nameEsc, payloadHex;
    int rssi = 0; bool isScanResp = false; uint8_t ch = 0;
    if (!passFilters(report, mac12, rssi, isScanResp, ch, nameEsc, payloadHex)) return;
    unsigned long ts_ms = millis();
    char macOut[18];
    snprintf(macOut, sizeof(macOut), "%02X:%02X:%02X:%02X:%02X:%02X",
             report->peer_addr.addr[5], report->peer_addr.addr[4], report-
>peer_addr.addr[3],
             report->peer_addr.addr[2], report->peer_addr.addr[1], report-
>peer_addr.addr[0]);
    char chField[16];
    snprintf(chField, sizeof(chField), ",\"ch\":%u", (unsigned)ch);
    char buf[JSON_BUF];
    int n = snprintf(buf, sizeof(buf),
      "{\"ts_ms\":%lu,\"mac\":\"%s\",\"rssi\":%d,\"is_scan_response\":%s,"
      "\"name\":\"%s\",\"payload_hex\":\"%s\"%s}",
      ts_ms, macOut, rssi, isScanResp ? "true" : "false",
      nameEsc.c_str(), payloadHex.c_str(), chField
    );
```

```
      if (n < 0 || (size_t)n >= sizeof(buf)) { size_t L = strlen(buf); if (L >= 2) {
buf[sizeof(buf)-2] = '}'; buf[sizeof(buf)-1] = '\0'; } }

      if (SEND_OVER_USB) Serial.println(buf);
    }
    static void attachScanCallback() {
      Bluefruit.Scanner.setRxCallback(rx_callback);
    }
    static String gLine;
    static void handleJsonCommand(const String& line) {
      StaticJsonDocument<512> doc;
      DeserializationError err = deserializeJson(doc, line);
      if (err) { sendAck("err","bad_json"); return; }
      const char* cmd = doc["cmd"] | "";
      if (!*cmd) { sendAck("err","missing_cmd"); return; }
      if (strcmp(cmd, "set_adv_channel")==0) {
        int v = doc["ch"] | 0;
        if (!(v==0 || v==37 || v==38 || v==39)) { sendAck("err","bad_adv_channel");
return; }
        stopBleScanIfNeeded();
        gAdvChannel = v;
        startBleScanIfNeeded();
        StaticJsonDocument<128> p;
        p["adv_channel"] = gAdvChannel;
        p["adv_channel_applied"] = true;
        sendAckKV("ack", p);
        return;
      }
      if (strcmp(cmd, "set_mac_filter")==0) {
        std::vector<String> allow, block;
        bool okAllow = !doc.containsKey("allow") || parseMacList(doc["allow"], allow);
        bool okBlock = !doc.containsKey("block") || parseMacList(doc["block"], block);
        if (!okAllow || !okBlock){ sendAck("err","bad_mac_list"); return; }
        int reqCh = 0; bool hasCh = false;
        if (doc.containsKey("adv_channel")) {
          int v = doc["adv_channel"] | 0;
          if (v==0 || v==37 || v==38 || v==39) { reqCh = v; hasCh = true; }
          else { sendAck("err","bad_adv_channel"); return; }
        }
        stopBleScanIfNeeded();
        gAllowMacs = std::move(allow);
        gBlockMacs = std::move(block);
        if (hasCh) gAdvChannel = reqCh;
        startBleScanIfNeeded();
        StaticJsonDocument<192> p;
        p["allow_size"] = (int)gAllowMacs.size();
        p["block_size"] = (int)gBlockMacs.size();
```

```
        if (hasCh) { p["adv_channel"] = gAdvChannel; p["adv_channel_applied"] = true; }
        sendAckKV("ack", p);
        return;
      }
      if (strcmp(cmd, "add_mac")==0 || strcmp(cmd,"remove_mac")==0) {
        const char* list = doc["list"] | "allow";
        const char* macs = doc["mac"]  | "";
        String mac1 = normMac(String(macs));
        if (mac1.length()!=12){ sendAck("err","bad_mac"); return; }
        stopBleScanIfNeeded();
        auto& target = (strcmp(list,"block")==0) ? gBlockMacs : gAllowMacs;
        if (strcmp(cmd,"add_mac")==0) {
          if (!addrInList(mac1, target)) target.push_back(mac1);
        } else {
          target.erase(std::remove_if(target.begin(), target.end(), [&](const String& s){
return s==mac1; }), target.end());
        }
        startBleScanIfNeeded();
        StaticJsonDocument<160> p; p["list"]=list; p["size"]=(int)target.size();
sendAckKV("ack", p);
        return;
      }
      if (strcmp(cmd, "clear_filters")==0) {
        stopBleScanIfNeeded();
        gAllowMacs.clear(); gBlockMacs.clear();
        startBleScanIfNeeded();
        sendAck("ack","cleared");
        return;
      }
      if (strcmp(cmd, "set_rssi_min")==0) {
        int v = doc["rssi"] | -100;
        if (v<-127 || v>20){ sendAck("err","rssi_out_of_range"); return; }
        stopBleScanIfNeeded();
        gRssiMin = v;
        startBleScanIfNeeded();
        StaticJsonDocument<96> p; p["rssi"]=gRssiMin; sendAckKV("ack", p);
        return;
      }
      if (strcmp(cmd, "scan_start")==0) { startBleScanIfNeeded();
sendAck("ack","scan_started"); return; }
      if (strcmp(cmd, "scan_stop")==0)  { stopBleScanIfNeeded();
sendAck("ack","scan_stopped"); return; }
      if (strcmp(cmd, "set_active_scan")==0) {
        bool v = doc["active"] | true;
        stopBleScanIfNeeded();
        gActiveScan = v;
        startBleScanIfNeeded();
```

```
        StaticJsonDocument<96> p; p["active"]=gActiveScan; sendAckKV("ack", p);
        return;
      }
      if (strcmp(cmd, "set_scan_params")==0) {
        uint16_t interval = doc["interval"] | 16;
        uint16_t window   = doc["window"]   | 16;
        stopBleScanIfNeeded();
        g_sdScanParams.interval = interval;
        g_sdScanParams.window   = window;
        startBleScanIfNeeded();
        StaticJsonDocument<128> p; p["interval"]=interval; p["window"]=window;
sendAckKV("ack", p);
        return;
      }
      sendAck("err","unknown_cmd");
    }
    static void pumpSerial() {
      while (Serial.available() > 0) {
        int c = Serial.read();
        if (c == '\r') continue;
        if (c == '\n') {
          if (gLine.length()) { handleJsonCommand(gLine); gLine = ""; }
        } else {
          if (gLine.length() < 512) gLine += (char)c;
          else { gLine = ""; sendAck("err","line_too_long"); }
        }
      }
    }
    void setup() {
      Serial.begin(SERIAL_BAUD);
      delay(300);
      Bluefruit.begin();
      Bluefruit.setTxPower(0);
      Bluefruit.setName("nRF52840_Sniffer");
      configureScannerBasic();
      startBleScanIfNeeded();
      if (SEND_OVER_USB_DEBUG)
Serial.println("{\"event\":\"started\",\"mode\":\"UART_ONLY\",\"platform\":\"nRF52840\"}");
    }
    void loop() {
      pumpSerial();
      delay(2);
    }
```

Додаток Д

Програмний код засобами мови Python для пошуку і декодування BLE-пакетів отриманих з ефіру за допомогою HackRF One

```
import argparse, re, sys, binascii
from proto import *
WHITEN_SEED = {37: 0x25, 38: 0x2B, 39: 0x33}
def crc24_ble(data: bytes, init=0x555555) -> int:
    crc = init
    for byte in data:
        for k in range(8):                # LSB-first
            bit = (byte >> k) & 1
            c0 = crc & 1
            crc >>= 1
            if bit ^ c0:
                crc ^= 0x00065B
    return crc & 0xFFFFFF
def read_input(path: str, is_hex: bool) -> bytes:
    data = open(path, 'rb').read()
    if not is_hex:
        return data
    hex_str = re.sub(rb'[^0-9A-Fa-f]', b'', data)
    if len(hex_str) % 2:
        raise ValueError("Непарна кількість hex-символів у файлі")
    return bytes.fromhex(hex_str.decode())
def brev_byte(x: int) -> int:
    x = ((x & 0x55) << 1) | ((x >> 1) & 0x55)
    x = ((x & 0x33) << 2) | ((x >> 2) & 0x33)
    return ((x & 0x0F) << 4) | ((x >> 4) & 0x0F)
def dewhiten(payload: bytes, seed: int) -> bytes:
    s = seed & 0x7F
    out = bytearray()
    for byte in payload:
        o = 0
        for k in range(8):
            w = s & 1
            bit = (byte >> k) & 1
            o |= ((bit ^ w) << k)
            fb = ((s ^ (s >> 3)) & 1)
            s = ((s >> 1) | (fb << 6)) & 0x7F
        out.append(o)
    return bytes(out)
def parse_adv(pkt_dec: bytes, full_raw_hex: bytes):
    if len(pkt_dec) < 2:
        return None
    hdr0, hdr1 = pkt_dec[0], pkt_dec[1]
```

```
    pdu_type = hdr0 & 0x0F
    txadd = (hdr0 >> 6) & 1
    rxadd = (hdr0 >> 7) & 1
    length = hdr1 & 0x3F
    if length < 6 or length > 80:
        return None
    need = 2 + length + 3
    if len(pkt_dec) < need:
        return None
    advA = pkt_dec[2:8][::-1]
    adv_data = pkt_dec[8:8+(length-6)]
    crc = pkt_dec[8+(length-6):8+(length-6)+3]
    return {
        "pdu_type": pdu_type,
        "txadd": txadd, "rxadd": rxadd,
        "length": length,
        "mac": ":".join(f"{b:02X}" for b in advA),
        "adv_data": adv_data,
        "crc": crc,
        "total_len": need,
    }
def try_decode_once(raw_after: bytes, seed: int, brev_first: bool):
    if brev_first:
        tmp = bytes(brev_byte(x) for x in raw_after)
        dec_full = dewhiten(tmp, seed)
    else:
        tmp = dewhiten(raw_after, seed)
        dec_full = bytes(brev_byte(x) for x in tmp)
    parsed = parse_adv(dec_full, raw_after)
    if not parsed:
        return None
    parsed['full_dewhiten_raw_hex'] = dec_full
    hdr_payload = dec_full[:2 + parsed["length"]]
    crc_val = int.from_bytes(parsed["crc"], "little")
    crc_ok = (crc24_ble(hdr_payload) == crc_val)
    # if not crc_ok:
    #     return None
    parsed['crc_ok'] = crc_ok
    return parsed
def find_packets(b: bytes, ch: int):
    marker = bytes.fromhex("AA D6 BE 89 8E")
    i = 0
    seed = WHITEN_SEED[ch]
    while True:
        j = b.find(marker, i)
        if j < 0:
            break
```

```
            raw_after = b[j+5:j+5+42]
            if len(raw_after) < 5:
                break
            parsed = (try_decode_once(raw_after, seed, brev_first=False))
            if parsed:
                parsed['full_raw_hex'] = b[j: j+5+42].hex(" ").upper()
                yield j, parsed
                i = j + 5 + parsed["total_len"]
            else:
                i = j + 1
    def find_packets_worked(b: bytes, ch: int):
        marker = bytes.fromhex("AA D6 BE 89 8E")  # preamble + AccessAddress (LSB-порядок)
        i = 0
        while True:
            j = b.find(marker, i)
            if j < 0:
                break
            raw_after = b[j+5:j+5+42]
            if len(raw_after) < 5:
                break
            dec_full = bytes(brev_byte(x) for x in dewhiten(raw_after, WHITEN_SEED[ch]))
            parsed = parse_adv(dec_full, b[j:j+5+42])
            if parsed:
                yield j, parsed
                i = j + 5 + parsed["total_len"]
            else:
                i = j + 1
    def main():
        ap = argparse.ArgumentParser(description="Виділення та декодування BLE ADV з
.bin")
        ap.add_argument("path", help="шлях до .bin")
        ap.add_argument("--ch", type=int, default=37, choices=[37,38,39], help="ADV канал
(37/38/39)")
        ap.add_argument("--hex", action="store_true", help="файл містить ASCII-hex замість
сирих байтів")
        args = ap.parse_args()
        data = read_input(args.path, args.hex)
        found = False
        for idx, pkt in find_packets(data, args.ch):
            found = True
            pt = pkt["pdu_type"]
            types = {0: "ADV_IND", 2: "ADV_NONCONN_IND", 6: "ADV_SCAN_IND", 4: "SCAN_RSP",
1: "ADV_DIRECT_IND", 5: "CONNECT_REQ", 7: "ADV_EXT_IND"}
            print(f"\n@offset 0x{idx:X}")
            print(f" Type: {types.get(pt, f'PDU_{pt}')}, Len: {pkt['length']}")
            print(f" MAC : {pkt['mac']} ({'public' if pkt['txadd']==0 else 'random'})")
            ad = pkt["adv_data"]
```

```
            print(f" FullHEX({len(pkt['full_raw_hex'])}): {pkt['full_raw_hex']}")
            print(f" FullDewhitenHex({len(pkt['full_dewhiten_raw_hex'])}):
{pkt['full_dewhiten_raw_hex'].hex(" ").upper()}")
            print(f" AdvData({len(ad)}): {ad.hex(' ').upper()}")
            print(f" CRC: {pkt['crc'].hex(' ').upper()} || CRC OK: {pkt['crc_ok']}")
            k = 0
            while k < len(ad):
                L = ad[k]
                if L == 0 or k+1+L > len(ad): break
                t = ad[k+1]
                v = ad[k+2:k+1+L]
                print(f"  AD: len={L-1:2d} type=0x{t:02X} value={v.hex(' ').upper()}")
                k += 1 + L
        if not found:
            print("Пакети не знайдені. Перевірте канал (--ch), наявність преамбули та
формат файлу.")
    if __name__ == "__main__":
        main()
```

Додаток Е

Програмний код для ESP32-CAM для генерування реклами

```
#include <Arduino.h>
#include <NimBLEDevice.h>
static const char* DEVICE_NAME = "3123123";
static std::string bleMac;
void setup() {
  Serial.begin(115200);
  NimBLEDevice::init(DEVICE_NAME);
  NimBLEDevice::setOwnAddrType(BLE_OWN_ADDR_PUBLIC);
  NimBLEDevice::setPower(ESP_PWR_LVL_N0);
  NimBLEAdvertisementData advData;
  advData.setName(DEVICE_NAME);
  NimBLEAdvertising* adv = NimBLEDevice::getAdvertising();
  adv->setAdvertisementData(advData);
  adv->start();
  Serial.printf("[BLE] Advertising started as \"%s\"\n", DEVICE_NAME);
  bleMac = NimBLEDevice::getAddress().toString();
  Serial.printf("[BLE] Adapter MAC: %s\n", bleMac.c_str());
}
void loop() {
  delay(1000);
}
```

Додаток Ж

Програмний код прошивки для генерації параметризованої реклами за допомогою апаратного засобу TSTAR MICRO NRF52840

```
// ATTENTION!! After clean cache or before clean start(build) you have to edit file
//          .pio/libdeps/adafruit_feather_nrf52840/SdFat          -          Adafruit
Fork/src/common/ArduinoFiles.h
// line 52: void flush() override { BaseFile::sync(); } => void flush() {
BaseFile::sync(); }
#undef min
#undef max
#include <Arduino.h>
#include <nrf.h>
#include <cstring>
#include <cstdio>
struct TxConfig {
    uint32_t num_packets;
    uint8_t  channel;
    uint8_t  payload_len;
    uint32_t interval_ms;
    int8_t   tx_power_dbm;
};
static TxConfig g_cfg = {
    100,
    39,
    20,
    10,
    0
};
static volatile bool g_start_flag   = false;
static uint8_t      g_packet_counter = 0;
static uint8_t g_ble_packet[3 + 6 + 30];
static uint8_t g_adv_addr[6];
static void print_mac_started()
{
    uint32_t devaddr0 = NRF_FICR->DEVICEADDR[0];
    uint32_t devaddr1 = NRF_FICR->DEVICEADDR[1];
    uint8_t mac[6];
    mac[0] = (uint8_t)(devaddr0 & 0xFF);
    mac[1] = (uint8_t)((devaddr0 >> 8) & 0xFF);
    mac[2] = (uint8_t)((devaddr0 >> 16) & 0xFF);
    mac[3] = (uint8_t)((devaddr0 >> 24) & 0xFF);
    mac[4] = (uint8_t)(devaddr1 & 0xFF);
    mac[5] = (uint8_t)((devaddr1 >> 8) & 0xFF);
    char buf[64];
    snprintf(buf, sizeof(buf),
```

```
            "MAC: %02X:%02X:%02X:%02X:%02X:%02X Started",
            mac[5], mac[4], mac[3], mac[2], mac[1], mac[0]);
    Serial.println(buf);
}
static void init_adv_addr_from_ficr()
{
    uint32_t devaddr0 = NRF_FICR->DEVICEADDR[0];
    uint32_t devaddr1 = NRF_FICR->DEVICEADDR[1];
    g_adv_addr[0] = (uint8_t)(devaddr0 & 0xFF);
    g_adv_addr[1] = (uint8_t)((devaddr0 >> 8) & 0xFF);
    g_adv_addr[2] = (uint8_t)((devaddr0 >> 16) & 0xFF);
    g_adv_addr[3] = (uint8_t)((devaddr0 >> 24) & 0xFF);
    g_adv_addr[4] = (uint8_t)(devaddr1 & 0xFF);
    g_adv_addr[5] = (uint8_t)((devaddr1 >> 8) & 0xFF);
}
static void radio_set_tx_power(int8_t dbm)
{
    uint32_t val;
    if (dbm >= 8) {
        val = RADIO_TXPOWER_TXPOWER_Pos8dBm;
    } else if (dbm >= 7) {
        val = RADIO_TXPOWER_TXPOWER_Pos7dBm;
    } else if (dbm >= 6) {
        val = RADIO_TXPOWER_TXPOWER_Pos6dBm;
    } else if (dbm >= 5) {
        val = RADIO_TXPOWER_TXPOWER_Pos5dBm;
    } else if (dbm >= 4) {
        val = RADIO_TXPOWER_TXPOWER_Pos4dBm;
    } else if (dbm >= 3) {
        val = RADIO_TXPOWER_TXPOWER_Pos3dBm;
    } else if (dbm >= 2) {
        val = RADIO_TXPOWER_TXPOWER_Pos2dBm;
    } else if (dbm >= 0) {
        val = RADIO_TXPOWER_TXPOWER_0dBm;
    } else if (dbm >= -4) {
        val = RADIO_TXPOWER_TXPOWER_Neg4dBm;
    } else if (dbm >= -8) {
        val = RADIO_TXPOWER_TXPOWER_Neg8dBm;
    } else if (dbm >= -12) {
        val = RADIO_TXPOWER_TXPOWER_Neg12dBm;
    } else if (dbm >= -16) {
        val = RADIO_TXPOWER_TXPOWER_Neg16dBm;
    } else if (dbm >= -20) {
        val = RADIO_TXPOWER_TXPOWER_Neg20dBm;
    } else {
        val = RADIO_TXPOWER_TXPOWER_Neg40dBm;
    }
```

```
        NRF_RADIO->TXPOWER = (val << RADIO_TXPOWER_TXPOWER_Pos);
    }
    static void radio_set_channel(uint8_t ch)
    {
        uint8_t freq;
        uint8_t chan_index;
        if (ch == 37) {
            freq       = 2;
            chan_index = 37;
        } else if (ch == 38) {
            freq       = 26;
            chan_index = 38;
        } else if (ch == 39) {
            freq       = 80;
            chan_index = 39;
        } else {
            freq       = ch;
            chan_index = ch;
        }
        NRF_RADIO->FREQUENCY   = freq;
        NRF_RADIO->DATAWHITEIV = chan_index;
    }
    static void radio_config_payload(uint8_t payload_len)
    {
        (void)payload_len;
        NRF_RADIO->MODE = RADIO_MODE_MODE_Ble_1Mbit << RADIO_MODE_MODE_Pos;
        const uint32_t PACKET_S0_FIELD_SIZE      = 1;
        const uint32_t PACKET_S1_FIELD_SIZE_BITS = 2;
        const uint32_t PACKET_LENGTH_FIELD_SIZE  = 6;

        NRF_RADIO->PCNF0 =
            ((PACKET_S0_FIELD_SIZE      << RADIO_PCNF0_S0LEN_Pos) & RADIO_PCNF0_S0LEN_Msk)
|
            ((PACKET_S1_FIELD_SIZE_BITS << RADIO_PCNF0_S1LEN_Pos) & RADIO_PCNF0_S1LEN_Msk)
|
            ((PACKET_LENGTH_FIELD_SIZE  << RADIO_PCNF0_LFLEN_Pos) & RADIO_PCNF0_LFLEN_Msk);
        const uint32_t BLE4_PACKET_BASE_ADDRESS_SIZE = 3;
        const uint32_t BLE4_PACKET_PAYLOAD_MAX_SIZE  = 37;
        NRF_RADIO->PCNF1 =
            ((RADIO_PCNF1_WHITEEN_Enabled       <<        RADIO_PCNF1_WHITEEN_Pos)        &
RADIO_PCNF1_WHITEEN_Msk) |
            ((RADIO_PCNF1_ENDIAN_Little             <<    RADIO_PCNF1_ENDIAN_Pos)         &
RADIO_PCNF1_ENDIAN_Msk)  |
            ((BLE4_PACKET_BASE_ADDRESS_SIZE       <<        RADIO_PCNF1_BALEN_Pos)        &
RADIO_PCNF1_BALEN_Msk)   |
            ((0  /*STATLEN*/                                <<  RADIO_PCNF1_STATLEN_Pos)  &
RADIO_PCNF1_STATLEN_Msk)|
```

```
            ((BLE4_PACKET_PAYLOAD_MAX_SIZE       <<       RADIO_PCNF1_MAXLEN_Pos)       &
RADIO_PCNF1_MAXLEN_Msk);
          NRF_RADIO->PREFIX0 = 0x8E;
          NRF_RADIO->BASE0   = 0x89BED600;
          NRF_RADIO->TXADDRESS   = 0;
          NRF_RADIO->RXADDRESSES        =        (RADIO_RXADDRESSES_ADDR0_Enabled        <<
RADIO_RXADDRESSES_ADDR0_Pos);
          NRF_RADIO->CRCCNF =
              (RADIO_CRCCNF_LEN_Three << RADIO_CRCCNF_LEN_Pos) |
              (RADIO_CRCCNF_SKIPADDR_Skip << RADIO_CRCCNF_SKIPADDR_Pos);
          NRF_RADIO->CRCINIT = 0x555555;
          NRF_RADIO->CRCPOLY = 0x00065B;
          NRF_RADIO->SHORTS = 0;
      }
      static void radio_init()
      {
          NRF_CLOCK->EVENTS_HFCLKSTARTED = 0;
          NRF_CLOCK->TASKS_HFCLKSTART = 1;
          while (NRF_CLOCK->EVENTS_HFCLKSTARTED == 0) {}
          NRF_RADIO->POWER = 1;
          init_adv_addr_from_ficr();
          radio_set_channel(g_cfg.channel);
          radio_set_tx_power(g_cfg.tx_power_dbm);
          radio_config_payload(g_cfg.payload_len);
          NRF_RADIO->MODECNF0 =
              (RADIO_MODECNF0_RU_Fast << RADIO_MODECNF0_RU_Pos);
      }
      static void radio_send_packet(uint8_t *adv_data)
      {
          uint8_t pl = g_cfg.payload_len;
          if (pl > 30) pl = 30;
          uint8_t adv_len = 6 + pl;
          if (adv_len > 37) adv_len = 37;
          g_ble_packet[0] = 0x02;
          g_ble_packet[1] = adv_len;
          g_ble_packet[2] = 0x00;
          memcpy(&g_ble_packet[3], g_adv_addr, 6);
          memcpy(&g_ble_packet[3 + 6], adv_data, pl);
          NRF_RADIO->PACKETPTR = (uint32_t)g_ble_packet;
          NRF_RADIO->EVENTS_READY = 0;
          NRF_RADIO->TASKS_TXEN   = 1;
          while (NRF_RADIO->EVENTS_READY == 0) {}
          NRF_RADIO->EVENTS_END = 0;
          NRF_RADIO->TASKS_START = 1;
          while (NRF_RADIO->EVENTS_END == 0) {}
          NRF_RADIO->TASKS_DISABLE = 1;
          while (NRF_RADIO->EVENTS_DISABLED == 0) {}
```

```
    NRF_RADIO->EVENTS_DISABLED = 0;
}
static void apply_config()
{
    if (g_cfg.payload_len != 0  &&
        g_cfg.payload_len != 1  &&
        g_cfg.payload_len != 10 &&
        g_cfg.payload_len != 20 &&
        g_cfg.payload_len != 30)
    {
        g_cfg.payload_len = 20;
    }
    radio_set_channel(g_cfg.channel);
    radio_set_tx_power(g_cfg.tx_power_dbm);
    radio_config_payload(g_cfg.payload_len);
}
static void handle_set_command(char *args)
{
    while (*args == ' ') args++;
    char *token = strtok(args, " ");
    while (token != NULL) {
        if (strncmp(token, "C=", 2) == 0) {
            g_cfg.num_packets = (uint32_t)strtoul(token + 2, NULL, 10);
        } else if (strncmp(token, "CH=", 3) == 0) {
            g_cfg.channel = (uint8_t)strtoul(token + 3, NULL, 10);
        } else if (strncmp(token, "PL=", 3) == 0) {
            g_cfg.payload_len = (uint8_t)strtoul(token + 3, NULL, 10);
        } else if (strncmp(token, "I=", 2) == 0) {
            g_cfg.interval_ms = (uint32_t)strtoul(token + 2, NULL, 10);
        } else if (strncmp(token, "P=", 2) == 0) {
            g_cfg.tx_power_dbm = (int8_t)strtol(token + 2, NULL, 10);
        }
        token = strtok(NULL, " ");
    }
    apply_config();
    Serial.println("OK");
}
static void process_line(char *line)
{
    if (strncmp(line, "SET", 3) == 0) {
        handle_set_command(line + 3);
    } else if (strcmp(line, "GO") == 0) {
        g_packet_counter = 0;
        g_start_flag = true;
        Serial.println("START");
        print_mac_started();
    } else {
```

```
        Serial.println("ERR");
    }
}
void setup()
{
    Serial.begin(921600);
    delay(100);
    Serial.println("NRF52840 RAW BLE TX READY");
    radio_init();
}
void loop()
{
    static char   line_buf[80];
    static uint8_t idx = 0;
    while (Serial.available() > 0) {
        char c = (char)Serial.read();
        if (c == '\r' || c == '\n') {
            if (idx > 0) {
                line_buf[idx] = '\0';
                process_line(line_buf);
                idx = 0;
            }
        } else {
            if (idx < sizeof(line_buf) - 1) {
                line_buf[idx++] = c;
            }
        }
    }
    if (g_start_flag) {
        g_start_flag = false;
        uint8_t payload[32];
        uint32_t t_next = micros();
        for (uint32_t i = 0; i < g_cfg.num_packets; ++i) {
            t_next += (uint32_t)g_cfg.interval_ms * 1000UL;
            if (g_cfg.payload_len > 0) {
                payload[0] = g_packet_counter++;
                if (g_cfg.payload_len > 1) {
                    memset(payload + 1, 0xAA, g_cfg.payload_len - 1);
                }
            }
            radio_send_packet(payload);
            while ((int32_t)(micros() - t_next) < 0) {
            }
        }
        Serial.println("DONE");
    }
}
```

Додаток И

Програмний код оркестратора для керування генерації реклами на декількох TSTAR MICRO NRF52840 засобами мови програмування Python

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import threading
import time
import sys
import queue
from dataclasses import dataclass
import serial
import serial.tools.list_ports
import tkinter as tk
from tkinter import ttk, messagebox
from tkinter.scrolledtext import ScrolledText
BAUDRATE = 921600
@dataclass
class DeviceConfig:
    name: str
    port: str
    index: int
    num_packets: int
    channel: int
    payload_len: int
    base_interval_ms: int
    tx_power_dbm: int
class NRFWorker(threading.Thread):
    def __init__(self, cfg: DeviceConfig, start_event: threading.Event, log_queue:
queue.Queue):
        super().__init__(name=cfg.name, daemon=True)
        self.cfg = cfg
        self.start_event = start_event
        self.log_queue = log_queue
    def log(self, msg: str):
        self.log_queue.put(f"[{self.cfg.name}] {msg}")
    def run(self):
        interval_ms = int(self.cfg.base_interval_ms * (1.0 + 0.1 * self.cfg.index))
        try:
            ser = serial.Serial(self.cfg.port, baudrate=BAUDRATE, timeout=1)
        except serial.SerialException as e:
            self.log(f"Не можу відкрити {self.cfg.port}: {e}")
            return
        cmd = (
            f"SET"
            f" C={self.cfg.num_packets}"
```

```
            f" CH={self.cfg.channel}"
            f" PL={self.cfg.payload_len}"
            f" I={interval_ms}"
            f" P={self.cfg.tx_power_dbm}\n"
        )
        self.log(f"Надсилаю конфіг: {cmd.strip()}")
        try:
            ser.write(cmd.encode("ascii"))
            ser.flush()
        except Exception as e:
            self.log(f"Помилка при записі SET: {e}")
            ser.close()
            return
        time.sleep(0.1)
        try:
            line = ser.readline().decode(errors="ignore").strip()
            if line:
                self.log(f"Відповідь: {line}")
        except Exception as e:
            self.log(f"Помилка читання ACK: {e}")
        self.log("Очікую глобальний START")
        self.start_event.wait()
        self.log(f"Надсилаю GO (interval={interval_ms} ms)")
        try:
            ser.write(b"GO\n")
            ser.flush()
        except Exception as e:
            self.log(f"Помилка при записі GO: {e}")
            ser.close()
            return
        while True:
            try:
                line = ser.readline().decode(errors="ignore").strip()
            except Exception as e:
                self.log(f"Помилка читання: {e}")
                break
            if not line:
                break
            self.log(line)
            if line.strip() == "DONE":
                break
        ser.close()
        self.log("Завершено")
class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("NRF TX Controller")
```

```
            self.geometry("900x600")
            self.log_queue = queue.Queue()
            self.port_vars = {}
            self._build_ui()
            self.after(100, self._poll_log_queue)
        def _build_ui(self):
            top_frame = ttk.Frame(self)
            top_frame.pack(fill="x", padx=10, pady=10)
            ports_frame = ttk.LabelFrame(top_frame, text="Доступні порти")
            ports_frame.pack(side="left", fill="y", padx=(0, 10))
            self.ports_frame = ports_frame
            btn_refresh = ttk.Button(ports_frame, text="Refresh",
command=self.refresh_ports)
            btn_refresh.pack(fill="x", pady=5, padx=5)
            self.ports_list_frame = ttk.Frame(ports_frame)
            self.ports_list_frame.pack(fill="both", expand=True, padx=5, pady=5)
            config_frame = ttk.LabelFrame(top_frame, text="Параметри передачі")
            config_frame.pack(side="left", fill="both", expand=True)
            row = 0
            ttk.Label(config_frame, text="Кількість пакетів (C):").grid(row=row, column=0,
sticky="w", padx=5, pady=2)
            self.var_count = tk.IntVar(value=100)
            ttk.Entry(config_frame, textvariable=self.var_count, width=10).grid(row=row,
column=1, sticky="w", padx=5, pady=2)
            row += 1
            ttk.Label(config_frame, text="Канал (CH):").grid(row=row, column=0, sticky="w",
padx=5, pady=2)
            self.var_channel = tk.IntVar(value=39)
            ttk.Entry(config_frame, textvariable=self.var_channel, width=10).grid(row=row,
column=1, sticky="w", padx=5, pady=2)
            row += 1
            ttk.Label(config_frame, text="Довжина (PL):").grid(row=row, column=0,
sticky="w", padx=5, pady=2)
            self.var_length = tk.IntVar(value=20)
            length_combo = ttk.Combobox(
                config_frame,
                textvariable=self.var_length,
                values=[0, 1, 10, 20, 30],
                width=8,
                state="readonly",
            )
            length_combo.grid(row=row, column=1, sticky="w", padx=5, pady=2)
            row += 1
            ttk.Label(config_frame, text="Базовий інтервал (мс):").grid(row=row, column=0,
sticky="w", padx=5, pady=2)
            self.var_interval = tk.IntVar(value=10)
```

```
            ttk.Entry(config_frame, textvariable=self.var_interval, width=10).grid(row=row,
column=1, sticky="w", padx=5, pady=2)
            row += 1
            ttk.Label(config_frame, text="Потужність (P, dBm):").grid(row=row, column=0,
sticky="w", padx=5, pady=2)
            self.var_tx_power = tk.IntVar(value=0)
            ttk.Entry(config_frame, textvariable=self.var_tx_power, width=10).grid(row=row,
column=1, sticky="w", padx=5, pady=2)
            row += 1
            btn_start = ttk.Button(config_frame, text="Start", command=self.on_start)
            btn_start.grid(row=row, column=0, columnspan=2, sticky="ew", padx=5, pady=(10,
2))
            log_frame = ttk.LabelFrame(self, text="Лог")
            log_frame.pack(fill="both", expand=True, padx=10, pady=(0, 10))
            self.txt_log = ScrolledText(log_frame, wrap="word")
            self.txt_log.pack(fill="both", expand=True)
            self.refresh_ports()
        def refresh_ports(self):
            for child in self.ports_list_frame.winfo_children():
                child.destroy()
            self.port_vars.clear()
            ports = list(serial.tools.list_ports.comports())
            if not ports:
                lbl = ttk.Label(self.ports_list_frame, text="Портів не знайдено")
                lbl.pack(anchor="w")
                return
            for p in ports:
                var = tk.BooleanVar(value=True)
                text = f"{p.device} - {p.description}"
                cb = ttk.Checkbutton(self.ports_list_frame, text=text, variable=var)
                cb.pack(anchor="w", padx=2, pady=1)
                self.port_vars[p.device] = var
        def _poll_log_queue(self):
            try:
                while True:
                    msg = self.log_queue.get_nowait()
                    self._append_log(msg)
            except queue.Empty:
                pass
            self.after(100, self._poll_log_queue)
        def _append_log(self, msg: str):
            self.txt_log.insert("end", msg + "\n")
            self.txt_log.see("end")
        def on_start(self):
            selected_ports = [port for port, var in self.port_vars.items() if var.get()]
            if not selected_ports:
                messagebox.showwarning("Увага", "Не вибрано жодного порту.")
```

```
                return
            try:
                num_packets = int(self.var_count.get())
                channel = int(self.var_channel.get())
                payload_len = int(self.var_length.get())
                interval_ms = int(self.var_interval.get())
                tx_power_dbm = int(self.var_tx_power.get())
            except ValueError:
                messagebox.showerror("Помилка", "Переконайся, що всі числові поля заповнені
коректно.")
                return
            if num_packets <= 0:
                messagebox.showerror("Помилка", "Кількість пакетів (C) має бути > 0.")
                return
            if interval_ms <= 0:
                messagebox.showerror("Помилка", "Інтервал (I) має бути > 0.")
                return
            self.txt_log.delete("1.0", "end")
            self._append_log("=== Старт експерименту ===")
            start_event = threading.Event()
            workers = []
            for idx, port in enumerate(selected_ports):
                cfg = DeviceConfig(
                    name=f"NRF{idx}",port=port, index=idx,
                    num_packets=num_packets, channel=channel,
                    payload_len=payload_len, base_interval_ms=interval_ms,
                    tx_power_dbm=tx_power_dbm,
                )
                w = NRFWorker(cfg, start_event, self.log_queue)
                w.start()
                workers.append(w)
                time.sleep(interval_ms * 0.1 / 1000)
            def waiter():
                time.sleep(0.2)
                self.log_queue.put("[MAIN] Глобальний старт (GO)")
                start_event.set()
                for w in workers:
                    w.join()
                self.log_queue.put("[MAIN] Усі треди завершили роботу.")
            threading.Thread(target=waiter, daemon=True).start()
    def main():
        app = App()
        app.mainloop()
    if __name__ == "__main__":
        main()
```

Додаток К

Програмний код для уніфікації формату експериментальних даних під час післяобробки

```
import os
import json
import argparse
import binascii
import struct
root_path = os.getcwd() + "/../реалізація експерименту/"
PSD_HEADER_SIZE = 0
PSD_STRUCT_FMT = "<B I Q H B 255s"
PSD_RECORD_SIZE = struct.calcsize(PSD_STRUCT_FMT)
POSSIBLE_PACKET_KEYS = ("packet", "data", "payload", "raw", "bytes")
ALLOWED_MAC    =    ["39:8C:FE:17:43:D6",    "F5:51:67:2A:12:DD",    "BC:79:85:22:1E:AA",
"97:45:D5:10:91:02", "69:C6:E8:D5:4D:70"]
def find_files(root_dir, extensions=(".ndjson", ".psd")):
    for dirpath, _, filenames in os.walk(root_dir):
        for name in filenames:
            lower = name.lower()
            if lower.endswith(extensions):
                yield os.path.join(dirpath, name)
def extract_from_ndjson(path: str):
    base_out_path = os.path.splitext(path)[0] + ".txt"
    file_name = os.path.basename(base_out_path)
    ndjson_dir = os.path.dirname(path)
    dir_name = os.path.basename(ndjson_dir)
    name_of_experiment = os.path.basename(
        os.path.dirname(ndjson_dir)
    )
    out_dir = os.path.join(
        os.curdir,
        "results_postprocessed",
        name_of_experiment,
        dir_name,
    )
    os.makedirs(out_dir, exist_ok=True)
    out_path = os.path.join(out_dir, file_name)
    with open(path, "r", encoding="utf-8") as src, \
         open(out_path, "w", encoding="utf-8") as dst:
        for line_no, line in enumerate(src, 1):
            line = line.strip()
            if not line:
                continue
            try:
                obj = json.loads(line)
```

```
            except json.JSONDecodeError:
                dst.write(f"# line {line_no} (invalid JSON): {line}\n")
                continue
            if 'mac' not in obj or obj['mac'] not in ALLOWED_MAC:
                continue
            if isinstance(obj, dict):
                chosen = None
                for key in POSSIBLE_PACKET_KEYS:
                    if key in obj:
                        chosen = obj[key]

                if chosen is not None:
                    if isinstance(chosen, (bytes, bytearray)):
                        dst.write(binascii.hexlify(chosen).decode("ascii") + "\n")
                    else:
                        dst.write(str(chosen) + "\n")
                    continue
            dst.write(json.dumps(obj, ensure_ascii=False) + "\n")
    print(f"[NDJSON] {path} -> {out_path}")
def extract_from_psd(path: str,
                     header_size: int = PSD_HEADER_SIZE,
                     record_size: int = PSD_RECORD_SIZE):
    base_out_path = os.path.splitext(path)[0] + ".txt"
    file_name = os.path.basename(base_out_path)
    ndjson_dir = os.path.dirname(path)
    dir_name = os.path.basename(ndjson_dir)
    name_of_experiment = os.path.basename(
        os.path.dirname(ndjson_dir)
    )
    out_dir = os.path.join(
        os.curdir,
        "results_postprocessed",
        name_of_experiment,
        dir_name,
    )
    os.makedirs(out_dir, exist_ok=True)
    out_path = os.path.join(out_dir, file_name)
    with open(path, "rb") as f:
        with open(out_path, "w", encoding="utf-8") as out:
            while True:
                chunk = f.read(PSD_RECORD_SIZE)
                if not chunk:
                    break
                if len(chunk) != PSD_RECORD_SIZE:
                    print(
                        f"Попередження: урізаний запис "
                        f"({len(chunk)} байт замість {PSD_RECORD_SIZE}), зупинка!"
```

```
                    )
                    break
                (
                    packetInfo,
                    packetNumber,
                    timeStamp,
                    payloadLength,
                    packetLen,
                    packetRaw,
                ) = struct.unpack(PSD_STRUCT_FMT, chunk)
                mac = extract_mac_from_packet(packetRaw, packetLen)
                if mac not in ALLOWED_MAC:
                    continue
                if packetLen < 2:
                    print(f"Пропускаю пакет {packetNumber}: packetLen={packetLen} < 2")
                    continue
                payload = extract_advdata_from_packet_data_hex(packetRaw.hex())
                line = f"{packetNumber};{mac};{payload}\n"
                out.write(line)
def extract_advdata_from_packet_data_hex(packet_data_hex: str) -> str | None:
    cleaned = "".join(packet_data_hex.split())
    data = bytes.fromhex(cleaned)
    if len(data) < 1 + 4 + 2 + 6 + 3 + 2:
        return None
    packet_len = data[5]
    adv_data = data[12 : 12 + packet_len - 6].hex()
    return adv_data
def extract_mac_from_packet(packet_raw: bytes, packet_len: int) -> str | None:
    pkt = packet_raw[:packet_len]
    mac = ":".join(list(reversed(pkt.hex(" ").upper().split(" ")[6:12])))
    return mac
def main():
    parser = argparse.ArgumentParser(
        description="Знайти .ndjson і .psd (TI) у директорії, "
                    "витягнути пакети в .txt файли."
    )
    parser.add_argument(
        "--root",
        type=str,
        default=root_path,
        help="Коренева директорія для пошуку файлів"
    )
    parser.add_argument(
        "--psd-header-size",
        type=int,
        default=PSD_HEADER_SIZE,
        help=f"Розмір заголовку PSD в байтах (default: {PSD_HEADER_SIZE})"
```

```
    )
    parser.add_argument(
        "--psd-record-size",
        type=int,
        default=PSD_RECORD_SIZE,
        help=f"Розмір одного запису (пакету) PSD в байтах (default: {PSD_RECORD_SIZE})"
    )
    args = parser.parse_args()
    for path in find_files(args.root):
        if path.lower().endswith(".ndjson"):
            extract_from_ndjson(path)
        elif path.lower().endswith(".psd"):
            extract_from_psd(path,
                             header_size=args.psd_header_size,
                             record_size=args.psd_record_size)
if __name__ == "__main__":
    main()
```

Додаток Л

Програмний код для формування Excel таблиць при обробці даних отриманих при реалізації фізичного експерименту

```
#!/usr/bin/env python3
import os
import re
import json
import argparse
from typing import Dict, List, Tuple, Any, Iterable
import pandas as pd
BLE_PATTERN = re.compile(
    r"^ble_packets_all_(\d+)packets_(\d+)ch_pl(\d+)_int(\d+)_P(\d+)\.txt$"
)
TEXAS_PATTERN = re.compile(
    r"^texas_(\d+)packets_(\d+)ch_pl(\d+)_int(\d+)_P(\d+)\.txt$"
)
def parse_ble_file(path: str) -> List[Dict[str, Any]]:
    rows: List[Dict[str, Any]] = []
    with open(path, "r", encoding="utf-8") as f:
        for i, line in enumerate(f, 1):
            line = line.strip()
            if not line:
                continue
            try:
                obj = json.loads(line)
            except json.JSONDecodeError:
                print(f"[WARN] {path}: рядок {i} – невалідний JSON, пропускаю")
                continue
            payload_hex = obj.get("payload_hex")
            mac = obj.get("mac")
            if payload_hex is None or mac is None:
                print(
                    f"[WARN] {path}: рядок {i} – немає payload_hex або mac, пропускаю"
                )
                continue
            rows.append({"payload_hex": payload_hex, "mac": mac})
    return rows
def parse_texas_file(path: str) -> List[Dict[str, Any]]:
    rows: List[Dict[str, Any]] = []
    with open(path, "r", encoding="utf-8") as f:
        for i, line in enumerate(f, 1):
            line = line.strip()
            if not line:
                continue
            parts = line.split(";")
```

```
                if len(parts) <= 2:
                    print(
                        f"[WARN] {path}: рядок {i} – менше 3 полів після split(';'), пропускаю"
                    )
                    continue
                col2 = parts[1].strip()
                col3 = parts[2].strip()
                rows.append({"col2": col2, "col3": col3})
        return rows
    def first_hex_byte_to_dec(hex_str):
        if not isinstance(hex_str, str):
            return None
        cleaned = "".join(hex_str.split())
        if len(cleaned) < 2:
            return None
        try:
            lo = int(cleaned[0:2], 16)
            if len(cleaned) < 4:
                return lo
            hi = int(cleaned[2:4], 16)
            return lo | (hi << 8)
        except ValueError:
            return None
    def _normalize_payload(hex_str):
        if not isinstance(hex_str, str):
            return None
        cleaned = "".join(hex_str.split()).lower()
        return cleaned if cleaned else None
    def is_clean_packet(hex_str: Any, packet_no_dec: Any) -> bool:
        cleaned = _normalize_payload(hex_str)
        if cleaned is None:
            return False
        import math
        if packet_no_dec is None or (isinstance(packet_no_dec, float) and math.isnan(packet_no_dec)):
            num = None
        else:
            try:
                num = int(packet_no_dec)
            except (TypeError, ValueError):
                num = None
        length = len(cleaned)
        if length == 4:
            if num is None:
                return False
            return num < 1000
```

```
    if length > 4:
        tail = cleaned[4:]
        return tail != "" and all(ch == "a" for ch in tail)
    return False
def extract_pairs_and_counters(df: pd.DataFrame, payload_col: str) -> Tuple[set, set]:
    pairs = set()
    counters = set()
    if "packet_no_dec" not in df.columns:
        return pairs, counters
    for _, row in df.iterrows():
        payload = _normalize_payload(row.get(payload_col))
        if payload is None:
            continue
        val = row.get("packet_no_dec")
        try:
            if pd.isna(val):
                continue
        except TypeError:
            if val is None:
                continue
        try:
            cnt = int(val)
        except (TypeError, ValueError):
            continue
        counters.add(cnt)
        pairs.add((cnt, payload))
    return pairs, counters
def find_same_counter_different_tail(
    src_packets: Iterable[str],
    dst_packets: Iterable[str],
) -> List[Tuple[str, str]]:
    def clean(p: str) -> str:
        return "".join(str(p).split()).lower()
    def group_by_counter(packets: Iterable[str]):
        groups = {}
        for raw in packets:
            cleaned = clean(raw)
            if len(cleaned) < 4:
                continue
            counter = cleaned[:4]
            groups.setdefault(counter, []).append(cleaned)
        return groups
    src_groups = group_by_counter(src_packets)
    dst_groups = group_by_counter(dst_packets)
    mismatched: List[Tuple[str, str]] = []
    common_counters = src_groups.keys() & dst_groups.keys()
    for counter in common_counters:
```

```
            for s in src_groups[counter]:
                tail_s = s[4:]
                for d in dst_groups[counter]:
                    tail_d = d[4:]
                    if tail_s != tail_d:
                        mismatched.append((s, d))
    return mismatched
def calc_directional_match_stats(
    src_pairs: set,
    src_counters: set,
    dst_pairs: set,
    dst_counters: set,
) -> Tuple[int, int, int]:
    full_pairs = src_pairs & dst_pairs
    full_total = len(full_pairs)
    clean_pairs = {
        (cnt, payload)
        for (cnt, payload) in full_pairs
        if is_clean_packet(payload, cnt)
    }
    full_clean = len(clean_pairs)
    src_payloads = [t[1] for t in src_pairs]
    dst_payloads = [t[1] for t in dst_pairs]
    partial_cnt = len(
        find_same_counter_different_tail(src_payloads, dst_payloads)
    )
    return full_total, full_clean, partial_cnt
def main():
    parser = argparse.ArgumentParser(
        description=(
            "Шукає файли ble_packets_all_*.txt і texas_*.txt в ієрархії, "
            "парсить їх та створює Excel-и для кожної директорії окремо "
            "по спільних параметрах (packets/ch/pl/int/P)."
        )
    )
    parser.add_argument(
        "--root",
        type=str,
        default="./results_postprocessed",
        help="Коренева директорія для пошуку файлів (default: поточна)",
    )
    parser.add_argument(
        "--out-dir",
        type=str,
        default="excel_results",
        help="Куди зберігати Excel-файли (default: ./excel_results)",
    )
```

```
    args = parser.parse_args()
    root_dir = os.path.abspath(args.root)
    out_root = os.path.abspath(args.out_dir)
    os.makedirs(out_root, exist_ok=True)
    results_by_dir: Dict[str, Dict[Tuple[str, str, str, str, str], Dict[str, Any]]] =
{}
    for dirpath, dirnames, filenames in os.walk(root_dir):
        rel_dir = os.path.relpath(dirpath, root_dir)
        for fname in filenames:
            ble_m = BLE_PATTERN.match(fname)
            texas_m = TEXAS_PATTERN.match(fname)
            full_path = os.path.join(dirpath, fname)
            if ble_m:
                packets, ch, pl, intval, pval = ble_m.groups()
                key = (packets, ch, pl, intval, pval)
                print(f"[INFO] [{rel_dir}] BLE-файл: {fname} (key={key})")
                ble_rows = parse_ble_file(full_path)
                if not ble_rows:
                    print(f"[WARN] {fname}: не отримано жодного рядка BLE, пропускаю")
                    continue
                dir_results = results_by_dir.setdefault(rel_dir, {})
                res = dir_results.setdefault(key, {})
                if "ble" in res:
                    print(
                        f"[WARN] [{rel_dir}] ключ {key} вже має BLE-таблицю, "
                        f"перезаписую новою ({fname})"
                    )
                res["ble"] = ble_rows
            elif texas_m:
                packets, ch, pl, intval, pval = texas_m.groups()
                key = (packets, ch, pl, intval, pval)
                print(f"[INFO] [{rel_dir}] TEXAS-файл: {fname} (key={key})")
                texas_rows = parse_texas_file(full_path)
                if not texas_rows:
                    print(f"[WARN] {fname}: не отримано жодного рядка TEXAS, пропускаю")
                    continue
                dir_results = results_by_dir.setdefault(rel_dir, {})
                res = dir_results.setdefault(key, {})
                if "texas" in res:
                    print(
                        f"[WARN] [{rel_dir}] ключ {key} вже має TEXAS-таблицю, "
                        f"перезаписую новою ({fname})"
                    )
                res["texas"] = texas_rows
    if not results_by_dir:
        print("[INFO] Не знайдено жодних даних для обробки.")
        return
```

```
        for rel_dir, results in results_by_dir.items():
            if rel_dir == ".":
                out_dir = out_root
            else:
                out_dir = os.path.join(out_root, rel_dir)
            os.makedirs(out_dir, exist_ok=True)
            print(f"\n[INFO] Обробка директорії: {rel_dir} -> {out_dir}")
            for key, data in results.items():
                packets, ch, pl, intval, pval = key
                out_name                                                    =
f"results_{packets}packets_{ch}ch_pl{pl}_int{intval}_P{pval}.xlsx"
                out_path = os.path.join(out_dir, out_name)
                print(f"[INFO]  Створюю Excel: {out_name}")
                if "ble" in data:
                    df_ble = pd.DataFrame(data["ble"]).drop_duplicates()
                else:
                    df_ble = pd.DataFrame(columns=["payload_hex", "mac"])
                if "texas" in data:
                    df_texas = pd.DataFrame(data["texas"]).drop_duplicates()
                else:
                    df_texas = pd.DataFrame(columns=["col2", "col3"])
                with pd.ExcelWriter(out_path, engine="xlsxwriter") as writer:
                    summary_rows: List[Dict[str, Any]] = []
                    base_sheet = "data"
                    df_ble.to_excel(
                        writer,
                        sheet_name=base_sheet,
                        startrow=0,
                        startcol=0,
                        index=False,
                    )
                    startcol_texas = df_ble.shape[1] + 2
                    df_texas.to_excel(
                        writer,
                        sheet_name=base_sheet,
                        startrow=0,
                        startcol=startcol_texas,
                        index=False,
                    )
                    if "mac" in df_ble.columns:
                        unique_macs = sorted(
                            m
                            for m in df_ble["mac"].dropna().unique()
                            if str(m).strip() != ""
                        )
                    else:
                        unique_macs = []
```

```
for mac in unique_macs:
    safe_mac = str(mac).replace(":", "-").replace(" ", "_")
    sheet_name = f"MAC_{safe_mac}"
    if len(sheet_name) > 31:
        sheet_name = sheet_name[:31]
    df_ble_mac = df_ble[df_ble["mac"] == mac].copy()
    if "col2" in df_texas.columns:
        df_texas_mac = df_texas[df_texas["col2"] == mac].copy()
    else:
        df_texas_mac = df_texas.copy()
    if df_ble_mac.empty and df_texas_mac.empty:
        continue
    if not df_ble_mac.empty and "payload_hex" in df_ble_mac.columns:
        df_ble_mac["packet_no_dec"] = df_ble_mac["payload_hex"].apply(
            first_hex_byte_to_dec
        )
        df_ble_mac = (
            df_ble_mac.sort_values("packet_no_dec")
            .reset_index(drop=True)
        )
        df_ble_mac["is_clean"] = df_ble_mac.apply(
            lambda row: is_clean_packet(
                row["payload_hex"], row["packet_no_dec"]
            ),
            axis=1,
        )
    if not df_texas_mac.empty and "col3" in df_texas_mac.columns:
        df_texas_mac["packet_no_dec"] = df_texas_mac["col3"].apply(
            first_hex_byte_to_dec
        )
        df_texas_mac = (
            df_texas_mac.sort_values("packet_no_dec")
            .reset_index(drop=True)
        )
        df_texas_mac["is_clean"] = df_texas_mac.apply(
            lambda row: is_clean_packet(
                row["col3"], row["packet_no_dec"]
            ),
            axis=1,
        )
    startrow = 2
    df_ble_mac.to_excel(
        writer,
        sheet_name=sheet_name,
        startrow=startrow,
        startcol=0,
        index=False,
```

```
)
startcol_texas_mac = df_ble_mac.shape[1] + 2
df_texas_mac.to_excel(
    writer,
    sheet_name=sheet_name,
    startrow=startrow,
    startcol=startcol_texas_mac,
    index=False,
)
ws = writer.sheets[sheet_name]
title = f"Фільтр за MAC: {mac} (відсортовано за номером пакету)"
ws.write(0, 0, title)
total_sent = int(packets)
nrf_total = int(len(df_ble_mac)) if not df_ble_mac.empty else 0
nrf_clean = (
    int(df_ble_mac["is_clean"].sum())
    if "is_clean" in df_ble_mac.columns
    else 0
)
nrf_broken = nrf_total - nrf_clean
sm_total = int(len(df_texas_mac)) if not df_texas_mac.empty else 0
sm_clean = (
    int(df_texas_mac["is_clean"].sum())
    if "is_clean" in df_texas_mac.columns
    else 0
)
sm_broken = sm_total - sm_clean
ble_pairs, ble_counters = extract_pairs_and_counters(
    df_ble_mac, "payload_hex"
)
texas_pairs, texas_counters = extract_pairs_and_counters(
    df_texas_mac, "col3"
)
base_col = 10
val_col = 13
row0 = 2
ws.write(row0, base_col, "Загальна кільк. відправлених")
ws.write(row0, val_col, total_sent)
ws.write(row0 + 1, base_col, "Кільк. отриманих nRF")
ws.write(row0 + 1, val_col, nrf_total)
ws.write(row0 + 2, base_col, "Кільк. отриманих Smart RF")
ws.write(row0 + 2, val_col, sm_total)
row_check_header = row0 + 5
ws.write(row_check_header, base_col, "Перевірка пакету")
r = row_check_header + 1
ws.write(r, base_col, "Загальна кільк. відправлених")
ws.write(r, val_col, total_sent)
```

```
r += 1
ws.write(r, base_col, "Кільк. отриманих nRF (всього)")
ws.write(r, val_col, nrf_total)
r += 1
ws.write(r, base_col, "Кільк. отриманих nRF (чисті)")
ws.write(r, val_col, nrf_clean)
r += 1
ws.write(r, base_col, "Кільк. отриманих nRF (биті)")
ws.write(r, val_col, nrf_broken)
r += 1
ws.write(r, base_col, "Кільк. отриманих Smart RF (всього)")
ws.write(r, val_col, sm_total)
r += 1
ws.write(r, base_col, "Кільк. отриманих SmartRF (чисті)")
ws.write(r, val_col, sm_clean)
r += 1
ws.write(r, base_col, "Кільк. отриманих SmartRF (биті)")
ws.write(r, val_col, sm_broken)
full_total_a2g, full_clean_a2g, partial_a2g = (
    calc_directional_match_stats(
        ble_pairs, ble_counters, texas_pairs, texas_counters
    )
)
full_total_g2a, full_clean_g2a, partial_g2a = (
    calc_directional_match_stats(
        texas_pairs, texas_counters, ble_pairs, ble_counters
    )
)
header_row_matches = r + 2
col_a2g = 14
col_g2a = 15
ws.write(header_row_matches, col_a2g, "A -> G")
ws.write(header_row_matches, col_g2a, "G -> A")
rm = header_row_matches + 1
ws.write(rm, base_col, "Повне співпадіння (всього)")
ws.write(rm, col_a2g, full_total_a2g)
ws.write(rm, col_g2a, full_total_g2a)
rm += 1
ws.write(rm, base_col, "Повне співпадіння (чисті)")
ws.write(rm, col_a2g, full_clean_a2g)
ws.write(rm, col_g2a, full_clean_g2a)
rm += 1
ws.write(rm, base_col, "Часткове співпадіння (лічільник)")
ws.write(rm, col_a2g, partial_a2g)
ws.write(rm, col_g2a, partial_g2a)
summary_rows.append(
    {
```

```
                                "MAC": mac,
                                "Загальна кільк. відправлених": total_sent,
                                "Кільк. отриманих nRF (всього)": nrf_total,
                                "Кільк. отриманих nRF (чисті)": nrf_clean,
                                "Кільк. отриманих nRF (биті)": nrf_broken,
                                "Кільк. отриманих Smart RF (всього)": sm_total,
                                "Кільк. отриманих SmartRF (чисті)": sm_clean,
                                "Кільк. отриманих SmartRF (биті)": sm_broken,
                                "Повне співпадіння (всього) A->G": full_total_a2g,
                                "Повне співпадіння (чисті) A->G": full_clean_a2g,
                                "Часткове співпадіння (лічільник) A->G": partial_a2g,
                                "Повне співпадіння (всього) G->A": full_total_g2a,
                                "Повне співпадіння (чисті) G->A": full_clean_g2a,
                                "Часткове співпадіння (лічільник) G->A": partial_g2a,
                            }
                        )
                    if summary_rows:
                        df_summary = pd.DataFrame(summary_rows)
                        df_summary_t = df_summary.set_index("MAC").T.reset_index()
                        df_summary_t.rename(columns={"index": "Показник"}, inplace=True)
                        df_summary_t.to_excel(
                            writer,
                            sheet_name="summary",
                            startrow=0,
                            startcol=0,
                            index=False,
                        )
                print(f"[OK]   Збережено: {out_path}")
    print("\nГотово.")
if __name__ == "__main__":
    main()
```