ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Universidad Politécnica de Cartagena

Trabajo Fin de Grado

GRADO EN INGENIERÍA TELEMÁTICA

Diseño y desarrollo de una aplicación web de monitorización de dispositivos Bluetooth Low Energy

AUTOR: Daniel Ros López

DIRECTOR: Alejandro S. Martínez Sala

Septiembre / 2021

Autor	Daniel Ros López
E-mail del autor	griefros@gmail.com
Director	Alejando S. Martínez Sala
E-mail del director	alejandros.martinez@upct.es
Título del TFE	Diseño y desarrollo de una aplicación web de monitorización de dispositivos Bluetooth Low Energy
Resumen	Diseño e implementación de una aplicación web en NodeJS para detección y monitorización de dispositivos Bluetooth Low Energy utilizando placas configurables ESP32 como escaners, mecanismo de obtención de las tramas en formato CSV para análisis. Estudio de los protocolos Bluetooth y MQTT. Puesta en marcha de pruebas para verificar el funcionamiento en diferentes entornos públicos. Análisis de resultados obtenidos del servidor.
Titulación	Grado en Ingeniería de Telecomunicación.
Departamento	Tecnología de la Información y las Comunicaciones.
Fecha de presentación	Septiembre - 2021

Agradecimientos

Para empezar, me gustaría agradecer al tutor de mi TFG, Alejandro Santos Martínez, por brindarme la oportunidad de hacer este trabajo y otros proyectos con los que he podido crecer y aprender más.

Agradecer por supuesto a mis padres el sacrificio diario que realizan para que yo haya podido llegar hasta aquí.

A mis hermanos Francisco José y Mayca, a mis amigos de mi ciudad, en especial a Alejandro Preciado por aguantarme en mis horas más bajas y siempre estar ahí, a María del Carmen Martínez, Luis y Juan Antonio, por darme tantos consejos de cara a la universidad y a Carmen por su optimismo.

Para finalizar, aunque no menos importante, agradecer a mis compañeros de la carrera, pues son con los que más he compartido la experiencia recibida durante el trayecto en mis años de universidad. En especial a Miguel Ángel, Pedro Giménez y Alba Martínez muchas gracias por estar ahí.

Índice

Índice de figura

Índice de tablas

Capítulo 1. Introducción y objetivos

1.1 Introducción y objetivos

Actualmente hay una gran variedad sistemas, aplicaciones y servicios IoT que hacen uso de la tecnología Bluetooth Low Energy (BLE) para poder utilizarse, desde relojes inteligentes y "fitbands" hasta sensores.

En BLE hay un mecanismo único y simple para darse a conocer en el medio, los llamados **advertisement,** que son beacons que se emiten periódicamente por el dispositivo BLE. Mediante estas tramas un dispositivo BLE con el rol scanner (p.e un app del móvil) puede detectarlos y conectarse a ellos.

Lo que se busca en este trabajo es desarrollar los medios para poder detectar esas tramas advertisement de dispositos BLE en cobertura, monitorizarlas y almacenar la información que contienen, además se busca que dichos datos estén disponibles a través de un portal web para el posterior análisis de las tramas.

Para lograr dichos medios se va a desarrollar un escáner BLE [3] utilizando el microcontrolador ESP32, el cual se encargará de enviar los datos sin filtrar a un servidor mediante protocolo MQTT, el cual deberemos desarrollar también, dichos mensajes llegarán a un servidor que se encarga de guardarlos en una base de datos para su posterior ilustración en un front-end. Se incluye un esquema simplificado y de alto nivel para entender el contexto del sistema, una primera introducción.

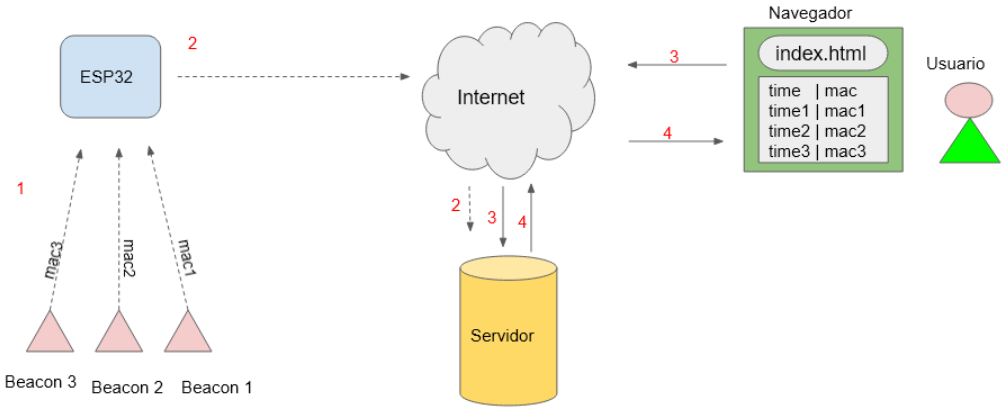


Figura 1. Esquema simplificado solución escáner con envío de datos

El sistema se evalúa y prueba en varios escenarios y casos prácticos reales para comprobar la capacidad de escaneo BLE y el funcionamiento global del sistema. Lo escenarios objetivo son la biblioteca de la UPCT, un supermercado y un autobús.

1.2 Herramientas software y elementos usados

Enumerar todas las herramientas SW, librerías (con sus versiones), versiones entornos desarrollo, etc., usados en el proyecto. Incluir referencias y enlaces en bibliografía.

Para el desarrollo de este trabajo se han utilizado los siguientes programas y librerías.

- 232 Analyzer Free Version 5.2.1 Commfront. Esta herramienta es útil para poder ver el comportamiento serial de los programas desarrollados en ESP32.
- App Beacon Simulator. Aplicación para móviles Android e iOS que hace que tu teléfono emita tramas beacon.
- App iBKS Config Tool. La aplicación que nos permite configurar las balizas iBKS.
- App nRF Connect Nordic. Una aplicación que escanea las tramas BLE de tu entorno por medio de un Smartphone
- ESP-IDF v4.2 Espressif. Es el entorno de desarrollo del microcontrolador ESP32, contiene las librerías que este utiliza, así como las herramientas para poder compilar los programas, cargarlos y monitorizarlos. A continuación, se listan las librerías usadas para estos los proyectos
    - FreeRTOS.h
    - task.h
    - event_groups
    - nvs_flash.h
    - esp_event.h
    - esp_log.h
    - string.h
    - esp_bt.h
    - esp_bt_main.h
    - esp_gap_ble_api.h
    - esp_wifi.h
    - lwip/err.h
    - lwip/sys.h
    - cJSON.h
    - driver/uart.h
    - mqtt_client.h
- iBKS Beacon. Dispositivos que, conectados a una pila, envían de forma continua tramas beacon mediante Bluetooth.
- MQTTX v1.5.2. Una aplicación de escritorio que se conecta a un servidor MQTT y permite enviarle tramas a los topics que se quiera.
- Mosquitto Eclipse Foundation v2.0.11. Un programa que, cargado en un ordenador con Linux o Windows permite iniciar un servidor MQTT, así como monitorizar las tramas que entran para un topic u otro.
- NodeJS 14.17.3 LTS. Es un framework del lenguaje de programación Javascript que se especializa en la creación de backends, se ha utilizado para dar forma al servidor que escucha las tramas MQTT y las guarda en BBDD para su uso futuro, a continuación, se listan las librerías usadas

  - csv-express v1.2.2
  - dotenv v 10.0.0
  - express 4.17.1
  - MQTT v4.2.6
  - sqlite3 v5.0.1
- Packet sniffer v2.18.0 Texas Instrument. Software utilizado con un Donger USB para en un PC poder escuchar las tramas BLE que llegan.
- Postman 8.6.2. Software para realizar peticiones al back-end.
- Python. Lenguaje utilizado para elaboración de gráficas.
- Router Fibra óptica Movistar
- Sqlite3 v3.35.5. Base datos relacional que utiliza la misma sintaxis del lenguaje SQL. Es un software más pequeño que no requiere de instalación.
- Linux Ubuntu 18.04 LTS. Sistema operativo de libre uso, bastante funcional y útil a la hora de realizar aplicaciones como la que se quiere en este trabajo.
- Visual Studio Code IDE Microsoft. Este es el entorno de desarrollo (IDE) con el que se realizan los programas, ya sea en C para el microcontrolador ESP32 o en Javascript.
- Windows 10. El sistema operativo de Microsoft, utilizado en general para el desarrollo de la mayoría de la programación, así como uso de la mayoría de los programas.

1.3 Estructura y organización del proyecto

A lo largo de este trabajo vamos a estudiar diferentes conceptos, protocolos, tecnologías y software que nos permita indagar en el protocolo de Bluetooth Low Energy.

Se va a utilizar software de terceros para estudiar cómo se envían las tramas normalmente, estudiaremos el funcionamiento y hardware del microcontrolador ESP32 para poder emular dicho comportamiento utilizando las herramientas que esta tiene.

Para finalizar se desarrollará una aplicación de back-end en NodeJS[2] para recopilar la información capturada por el programa desarrollado en ESP32 así como poder acceder a esta mediante un Front-end que permita además la descarga de este contenido.

Finalmente, en el capítulo de pruebas se evalúa el desempeño del sistema con capturas en entornos reales en la biblioteca de la UPCT, un supermercado y un autobús. Se hace un primer análisis de los datos y finalmente se establecen las conclusiones y trabajos futuros.

Capítulo 2. Tecnologías Empleadas

2.1 Conceptos básicos de Bluetooth Low Energy

Bluetooth Low Energy [3] es un estándar para **BLE** (Bluetooth Low Energy) es el nuevo estándar Bluetooth 4.0 (versiones 4.1, 4.2) y la reciente versión BT 5. Por primera vez se especifica un mecanismo muy simple de transmisión de beacons para que dispositivos detectados y localizados. Este mecanismo de **advertising de beacons** es la base para servicios de localización. En BLE también se define el **scanner** de advertising como un sniffer que recibe los mensajes de los beacons BLE en cobertura.

Este nuevo estándar opera en la banda ISM de 2.4GHz, una de las principales características es el costo considerablemente reducido que esta emplea respecto a su predecesor. Ya que esté permanece en modo suspensión constantemente excepto cuando se inicia una conexión. Es este detalle, es el que lo hace adecuado para aplicaciones IoT.

A continuación, vamos a estudiar ambos mecanismos.

Advertising de Beacons

Es necesario entender lo que esto significa, pues la palabra Advertising es inglesa, significa promoción y beacon significa baliza, esta palabra se usa incluso en el estándar de WiFi, ahí las tramas Beacon son las que se encargan promocionar en el medio el SSID para el punto de acceso WiFi, en el caso de Bluetooth es muy similar, pues promocionan en el medio un dispositivo, en ocasiones con la intención de que un usuario pueda escanear dicho dispositivo y conectarse a él mediante Bluetooth.

Al final el advertising de Beacons es la promoción de un dispositivo o en ocasiones servicios por medio de las tramas Beacon utilizando la arquitectura de Bluetooth Low Energy.

Para la utilización del medio se separa el espectro de 2.4 en un total de 40 canales, de los cuales 3 son utilizados para el advertising de beacons, estos canales son el 37, 38 y 39.

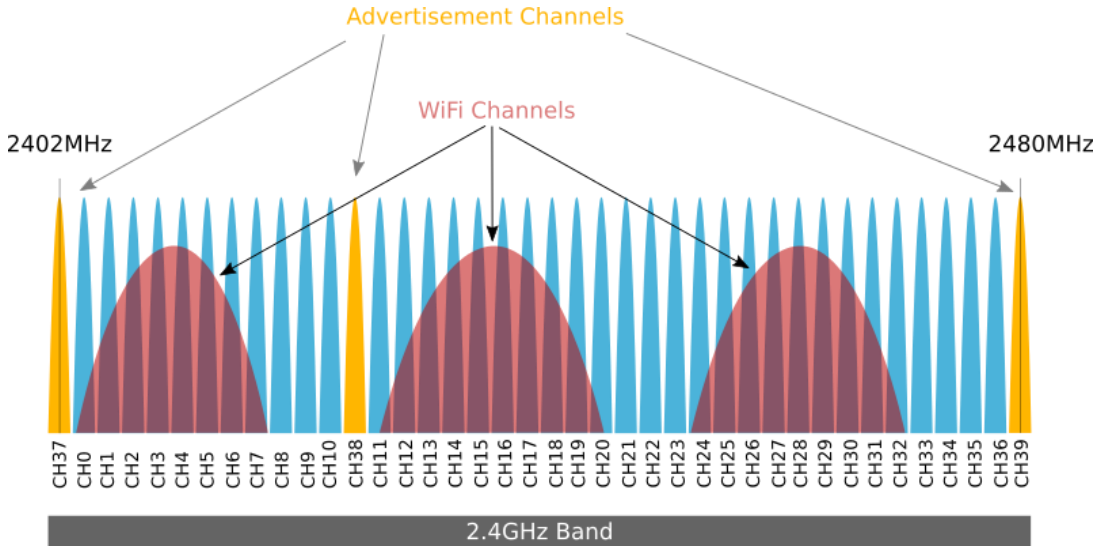


Figura 2. Canales Banda 2.4 ISM

Una trama beacon tiene los siguientes campos:

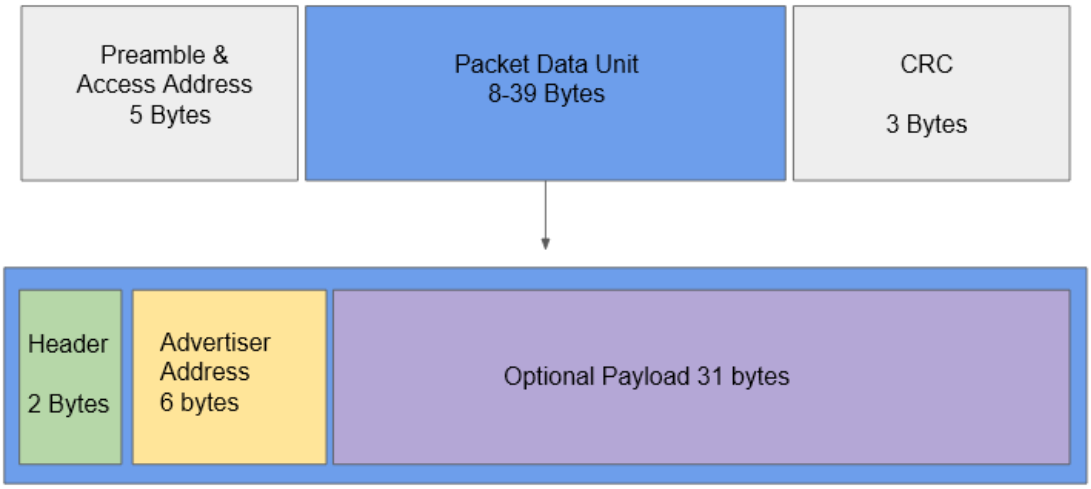


Figura 3. Cabecera trama BLE. Imagen de Bluetooth Core.

El preámbulo es una serie fija de bits que sirven para poder detectar la trama como una trama Bluetooth es patrón donde los 8 primeros bits son 10101010.

El Access Address para tramas en promoción, debe ser siempre 0x8E89BED6 de acuerdo a la especificación de Bluetooth[3].

CRC es un Código de Redundancia cíclica y sirve para comprobar que los datos dentro del Packet Data Unit son correctos.

El campo Packet Data Unit contiene a su vez:

- **El Header** o cabecera, compuesto por 2 Bytes. Los campos más importantes de la cabecera son:

- **Type** o tipo de Advertisement que está definido en los primeros 4 bits del primer Byte:
  - ADV_IND → 0x0000.
  - ADV_DIRECT_IND → 0x0001.
  - ADV_NONCONN_IND → 0x0010
  - ADV_SCAN_IND → 0x0110
  - SCAN_REQ → 0x0011
  - SCAN_RSP → 0x0100
  - CONNECT_IND → 0x0101
- bit de tipo **RFU**, Reserved for Future Use, es decir, que aún no se utiliza.
- **MAC Type**, el tipo de MAC utiliza los dos últimos bits del primer byte de la cabecera. Define si la dirección que acompaña la cabecera es publica o aleatoria, es decir, si los tres primeros bytes identifican la empresa manufacturera o no.
- **Longitud 1 byte.** Denota la longitud total entre campo Optional Payload y el Advertiser Address. Se codifica como un entero sin signo.

- **Advertiser Address** y **optional Payload**, dependiendo del tipo estos pueden varias tanto de tamaño como de nombre:
  - **ADV_IND, ADV_NONCONN_IND, ADV_SCAN_IND**. Este campo ocupará 6 bytes y representará la dirección de Advertising (AdvA) o dirección del dispositivo que se promociona en el medio. El siguiente campo ocupará de 0 a 31 bytes y será el campo Advertisement (AdvData), contiene información.
  - **ADV_DIRECT_IND**. Este campo ocupará 6 bytes y representará la dirección de Advertising. El siguiente campo ocupará 6 bytes y será la Target Address. La dirección de Advertising es igual que en el caso anterior y en cuanto a la Target Address contiene la dirección del dispositivo con la que el dispositivo promotor se desea conectar.
  - **SCAN_REQ**. Este campo ocupará 6 bytes y representará la Scan Address, que contiene la dirección del que realiza el Scan. El siguiente campo ocupará de 6 bytes y será la Advertisement Address.
  - **SCAN_RSP**, los primeros 6 bytes se corresponderan con la Advertisement Address, los siguientes bytes que pueden ser de 0 a 31 serán el ScanRspData, que puede contener información del dispositivo promotor.
  - **CONNECT_IND**, si es de este tipo, este campo será de 6 bytes y será un InitA, el siguiente campo se dividira en un AdvA de 6 bytes y el LLData de 22 bytes, el LLData tiene está estructura: No vamos a profundizar en este último.

- El **optional Payload 31 Bytes**, en el caso de que el tipo de trama sea ADV_IND, ADV_NONCONN_IND o ADV_SCAN_IND. Es totalmente personalizable en función de la aplicación o servicio que se quiera llevar. Pero es importante seguir la estructura:

  Longitud Tipo Dato

  - Longitud denota la cantidad de bytes usados en total entre el byte de Tipo y los bytes del Dato
  - Tipo, está definido según el GAP[5]
  - Dato, es lo que se quiere enviar de acuerdo al Tipo escogido.

  Se pueden codificar varios mensajes con los mismos Bytes de tipo o diferentes en el Payload. Se deben codificar uno a continuación de otro.

A partir de la estructura ya creada de Bluetooth Low Energy para la emisión de tramas se desarrollaron unos tipos de tramas específicos, algunos ejemplos son las iBeacon de Apple, las Eddystone de Google o las tramas Exposure que son las tramas que envía la aplicación Radar covid.

Estos tipos de trama tienen un formato definido por sus creadores:

- **iBeacon.** Este formato de trama fue desarrollado por Apple[6] y es el más utilizado, muchos dispositivos actuales utilizan este formato de trama. Tiene 3 campos configurables:
  - **UUID (16 Bytes)**
  - **Major (2 Bytes)**
  - **Minor (2 Bytes)**

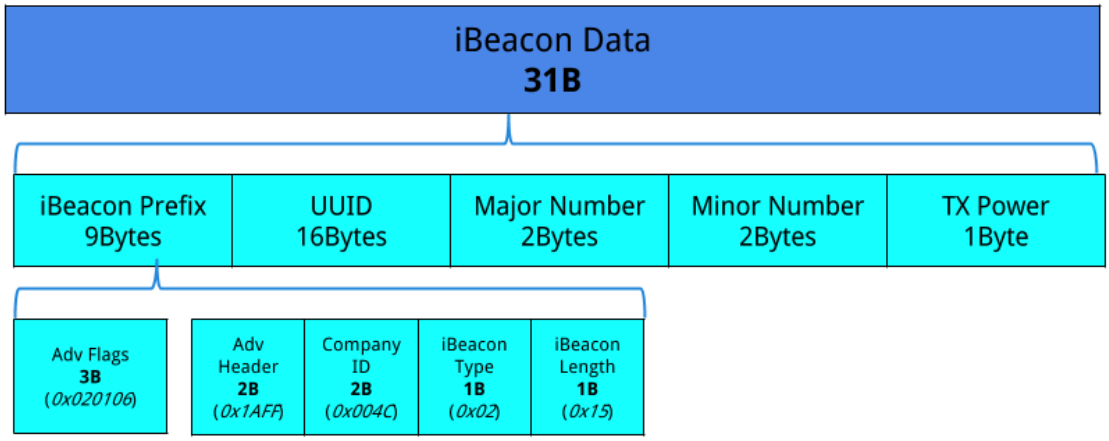


Figura 4. Formato mensaje BLE iBeacon

- Eddystone, este tipo fue desarrollado por Google[7] y no tiene un solo tipo si no que son 4 tipos distintos de tramas para la funcionalidad que se quiera. El formato de una trama Eddystone es el siguiente:

Flags 3 Bytes 0x02011A	Services 4 Bytes 0x0303AAFE	Length 1 Byte	Type 1 Byte 0x16	UUID 2 Bytes 0xAAFE	Eddystone frame 20 Bytes

Figura 5. Formato trama Eddystone

Donde el byte "Type" es el que indica el tipo de trama Eddystone el cual puede ser:

- Eddystone-UID. Tramas para localizar objetos en interiores
- Eddystone-URL. Tramas pensadas para mostar URLs en los dispositivos que las detecten.
- Eddystone-TLM. Tramas para dar información sobre el propio beacon, tiene una versión encriptada
- Eddystone-EID. Funcionamiento similar al de UID pero con la diferencia de que el ID está encriptado.

- Exposure frame. Este tipo de trama fue desarrollada por Google y Apple[8] en conjunto y se utilizó en la app radar Covid.

  Estas tramas poseen un Identificador que identifica un usuario, pero dicho identificador se genera con una clave que varía cada día por lo que es una tarea complicada el identificar una persona a través de esta aplicación.

  La idea de estos identificadores es que las aplicaciones guarden constancia de en qué hora y cuánto tiempo han estado detectándolos. En caso de que la persona dueña de ese identificador resulte positivo por COVID-19, con su consentimiento se revelan las claves temporales de las semanas anteriores y, a partir de ingeniería inversa, tu dispositivo puede detectar si has estado en contacto con una persona contagiada.

  La estructura de estas tramas es la siguiente:

Flags 3 Bytes 0x02011A	Length 1 Byte 0x03	Type 1 B 0x03	UUID 0x6FFD	Length 0x17	Type 0x16	UUID 2 B 0xFD6F	Rolling proximity identifier (16B)	Metadata 1 byte

Figura 6. Formato trama Exposure.

Existen soluciones que se basan en emitir estas tramas, como también existen aplicaciones que hacen uso del Bluetooth de tu móvil para transmitirlas. Vamos a repasar a continuación los dispositivos iBKS de Accent System, una solución que utiliza una pila de botón y que emite dichas tramas, las cuales son configurables a través de la propia aplicación.

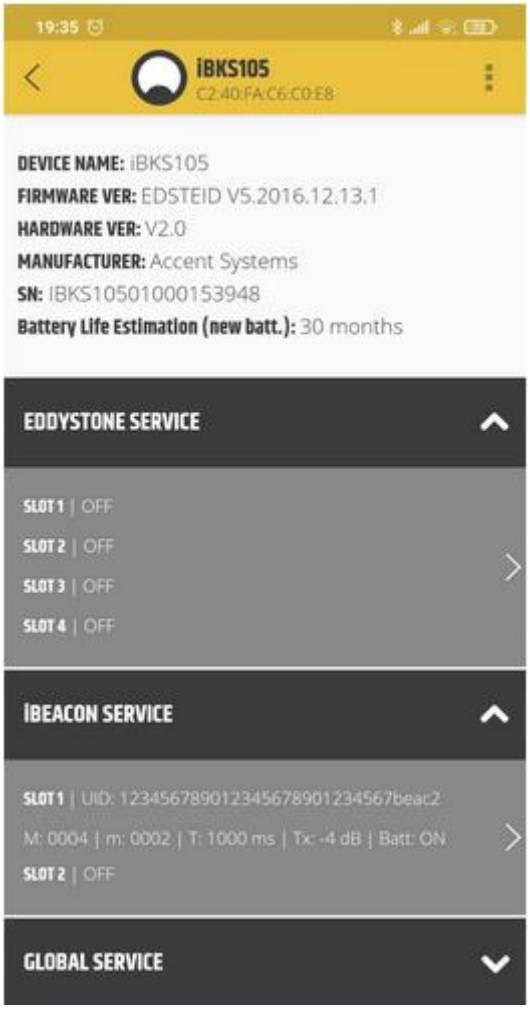


Figura 7. Captura aplicación iBKS.

La aplicación está disponible en Android e IoS, permite configurar las tramas Eddystone e iBeacon, permitiendo configurar los correspondientes campos, UUID, minor, major…

Se puede apreciar por la Figura 7 cuales son los parámetros configurados en el momento. Estos serán los parámetros con los que se promociona en el medio y serán los que escuchemos en el siguiente apartado.

Figura 8. Imagen beacon iBKS comercial.

Escaneo de tramas Beacon

De la misma forma que un dispositivo BLE se promociona al medio utilizando los conceptos del apartado anterior, también pueden escanearlo en busca de estas tramas. Existen dos formas de escanear tramas BLE:

- Escaneo activo. Para este tipo de scan primero se escucha el medio y luego se envían tramas BLE del tipo SCAN_REQ a todos los dispositivos detectados, las cuales los dispositivos responden con tramas tipo SCAN_RSP que tengan configuradas.

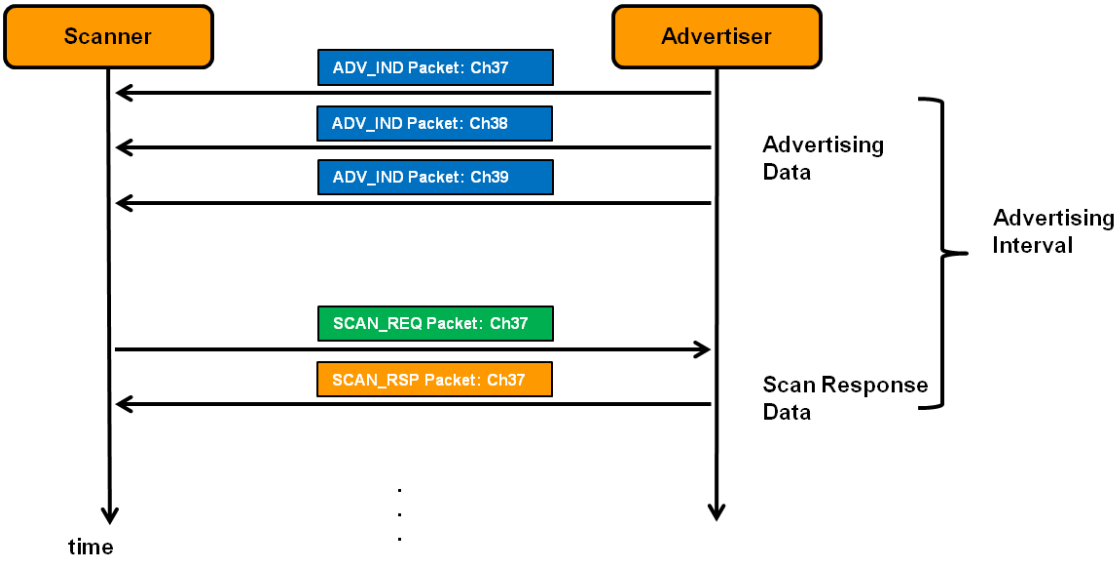


Figura 9. Cronograma escaneo activo.

- Escaneo pasivo. Aquí no se envían tramas, sino que se escucha el medio para ver los tipos de tramas que los dispositivos envían.

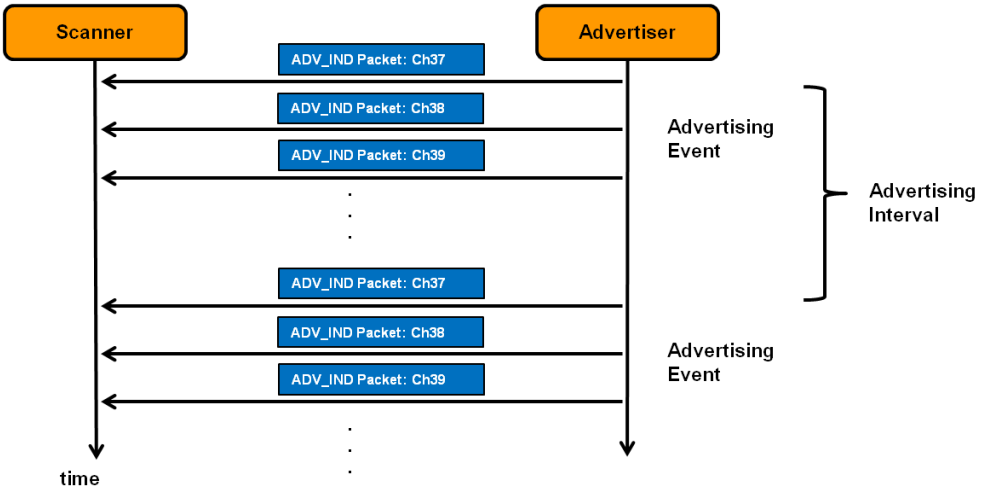


Figura 10. Cronograma escaneo pasivo

En la siguiente figura se hace una demostración de cómo funciona el advertising, emisión de tramas beacons y el escaneo de éstas.

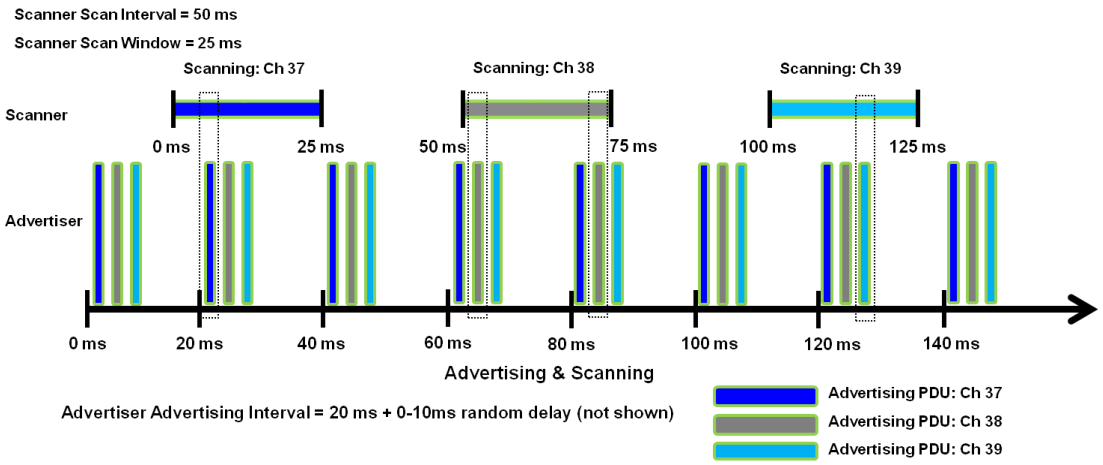


Figura 11. Ventana de escaneo y promoción.

El tiempo para el escaneo está parametrizado, de forma que el escáner comprueba durante un tiempo determinado los tres canales donde puede haber tramas de advertising.

Durante la recepción de advertisings hay un intervalo de escaneo y ventana de escaneo.

- El **Intervalo de escaneo**. Es el periodo con el cual se realizan los escaneos.
  - Rango: 0x0004 a 0x4000

- T = N x 0.625 ms siendo N el valor introducido.

- La **Ventana de escaneo**. Es el periodo de tiempo que ocupa el escaneo, debe ser menor o igual al intervalo de escaneo
  - Rango: 0x0004 a 0x4000
  - T = N x 0.625 ms siendo N el valor introducido.

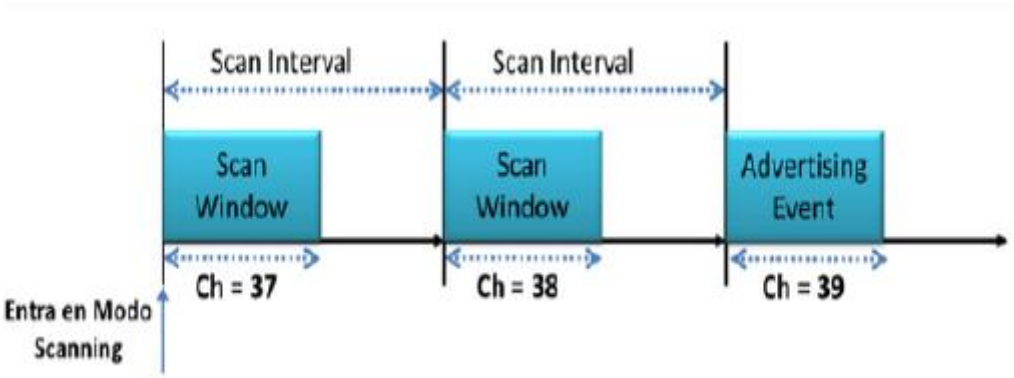


Figura 12. Ventana de escaneo.

En esta imagen se puede ver cada parámetro que efecto tiene, siendo el scan window el tiempo dentro del intervalo de escaneo en el que está escuchando y el Scan Interval, el intervalo de tiempo en el que escanea.

A continuación, vamos a ver ejemplos de escaneo de tramas partir de dos aplicaciones ya hechas.

- Escaneo por medio de Texas instrument packet sniffer

  La aplicación fue desarrollada por Texas instrument[10] A para la visualización de tramas advertisement, esta requiere del USB donger el cual es un dispositivo mediante el cual se reciben las tramas.

Figura 13. Imagen del USB Donguer.

Este dispositivo se conecta al PC como si fuera una unidad USB de almacenamiento.

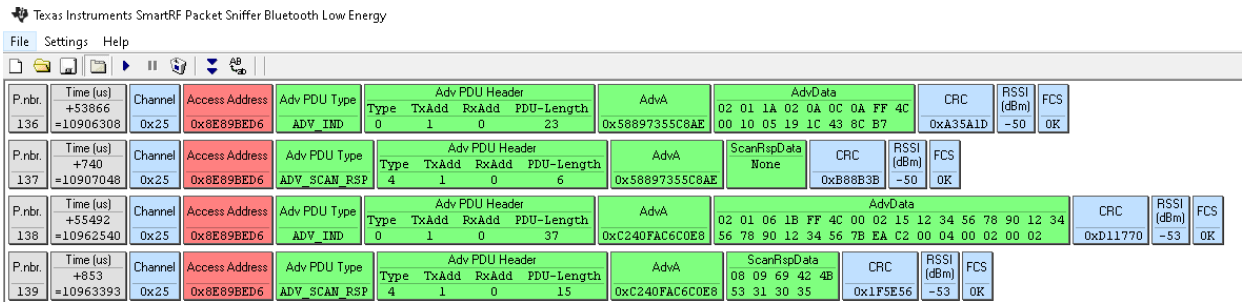


Figura 14. Captura aplicación.

Lo que muestra son las tramas capturadas, estas contienen los campos repasados en el apartado 3.1, vamos a tomar una de las tramas que se ven en la imagen como ejemplo:

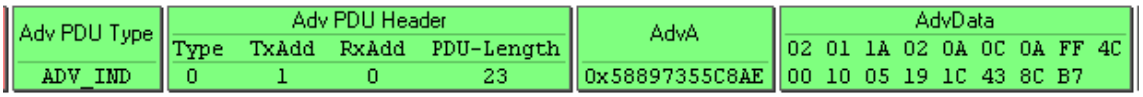


Figura 15. Ejemplo trama capturada.

La aplicación de por sí ya hace una clasificación de los bytes, es por ello que aparece la cabecera, la MAC y el optional payload en campos separados, además, como dentro de la cabecera se detecta el byte de tipo a 0, la aplicación nos indica el tipo de trama: ADV_IND.

En un ejemplo de captura, analizando los bytes OUI del fabricante la dirección MAC se comprueba que es aleatoria:

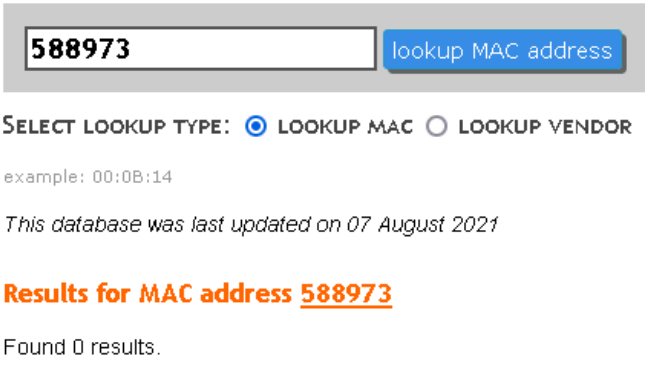


Figura 16. Comprobación OUI fabricante MAC.

- Escaneo por medio de App nRF Connect

En este apartado vamos a utilizar una aplicación de smartphone con la que vamos a escuchar el emisor de beacons visto en el apartado de Advertising de Beacons. La aplicación se encuentra disponible en IoS y Android y se llama nRF Connect, creada por Nordic[11].

Al igual que el programa anterior lo que este programa hace es escanear las tramas BLE del medio, nos ofrece una interfaz al tiempo que nos da la interpretación de estas,

pero sin necesitar del USB Donguer pues esta utiliza la antena Bluetooth de nuestro Smartphone.

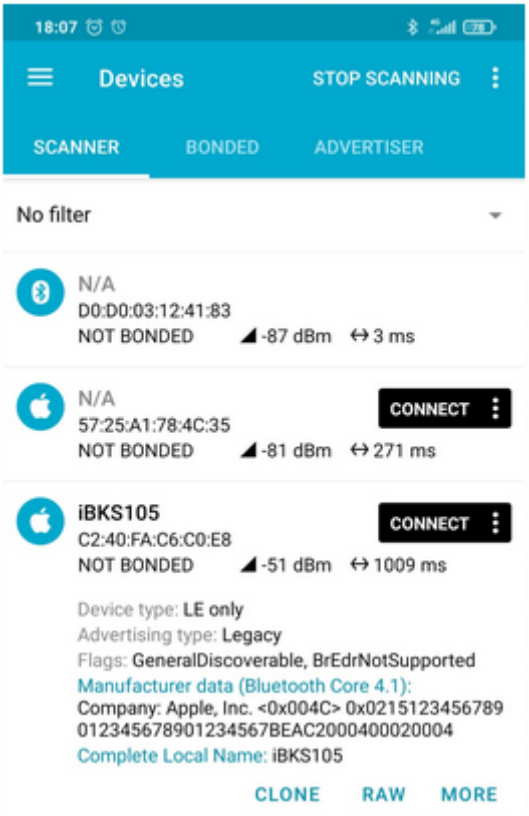


Figura 17. Captura aplicación nRF Connect.

Como se puede apreciar la aplicación desde el menú principal lo que nos da son los distintos dispositivos encontrados, si se pincha en cualquiera de ellos nos da información encontrada en las tramas, información que repasaremos a continuación. También los dice la potencia con la que se detecta y el intervalo. El dispositivo que nos interesa es el iBKS105, este es el beacon iBKS.

Además, si se pincha en RAW, podemos ver información más detallada:

Figura 18. Ventana RAW nRF Connect.

Se puede ver en la Figura 18 como la propia aplicación desglosa cada dato por detalles. Los dos primeros campos contienen exactamente la información que debe tener una trama iBeacon con la salvedad de que contiene el nivel de batería, lo que sube la longitud del paquete de 1A a 1B, tal y como se configuró en la Figura 7.

Además, existe otro campo con el tipo 9, el cual nos da el nombre, dentro del campo Value que se ve en la Figura 17 se puede leer, en Ascii el nombre iBKS105, este campo no es que se envíe junto a la trama iBeacon si no que es otra trama que se envía al mismo pero como la envía el mismo dispositivo la aplicación la junta.

Adv PDU Type	Adv PDU Header				AdvA	ScanRspData	CRC	RSSI (dBm)	FCS
	Type	TxAdd	RxAdd	PDU-Length					
ADV_SCAN_RSP	4	1	0	15	0xC240FAC6C0E8	08 09 69 42 4B 53 31 30 35	0x1F5E56	-52	OK

Adv PDU Type	Adv PDU Header				AdvA	AdvData	CRC	RSSI (dBm)	FCS
	Type	TxAdd	RxAdd	PDU-Length					
ADV_IND	0	1	0	37	0xC240FAC6C0E8	02 01 06 1B FF 4C 00 02 15 12 34 56 78 90 12 34 56 78 90 12 34 56 7B EA C2 00 04 00 02 00 64	0x8F05F0	-52	OK

Figura 19. Captura de Texas instrument.

En la figura 19 se puede ver como son dos tramas diferentes de distinto tipo además, una conteniendo el mensaje con la trama iBeacon y la otra conteniendo el nombre.

2.2 Módulo ESP32

El módulo ESP32 [11] es un System on a Chip (SoC) programable, de bajo costo y consumo diseñado por la Empresa Espressif Systems. El módulo es característico por tener antena integrada para WiFi y Bluetooth así como diversos periféricos y pines configurables para distintos usos.

Figura 20. Imagen placas ESP32 utilizadas, izquierda modulo WROOM, derecha WROVER

En cuanto a la arquitectura del ESP32 es importante destacar:

- Antena integrada para WiFi y bluetooth
  - WiFi 802.11 b/g/n
  - Bluetooth V4.2 BR/EDR and BLE
- Dos Núcleos
- Núcleo de bajo consumo (ULP)
- Memoria ROM 448 KB ROM para funciones principales
- Memoria SRAM 520 KB para datos e instrucciones
- 16 KB SRAM en RTC (Deep sleep)
- 3 UARTs, una de ellas reservada para configuración.
- Tiene módulos específicos para criptografía como un RNG (Random Number Generator)
- 32 pines, 12 de ellos de propósito general y el resto reservados para funciones específicas.

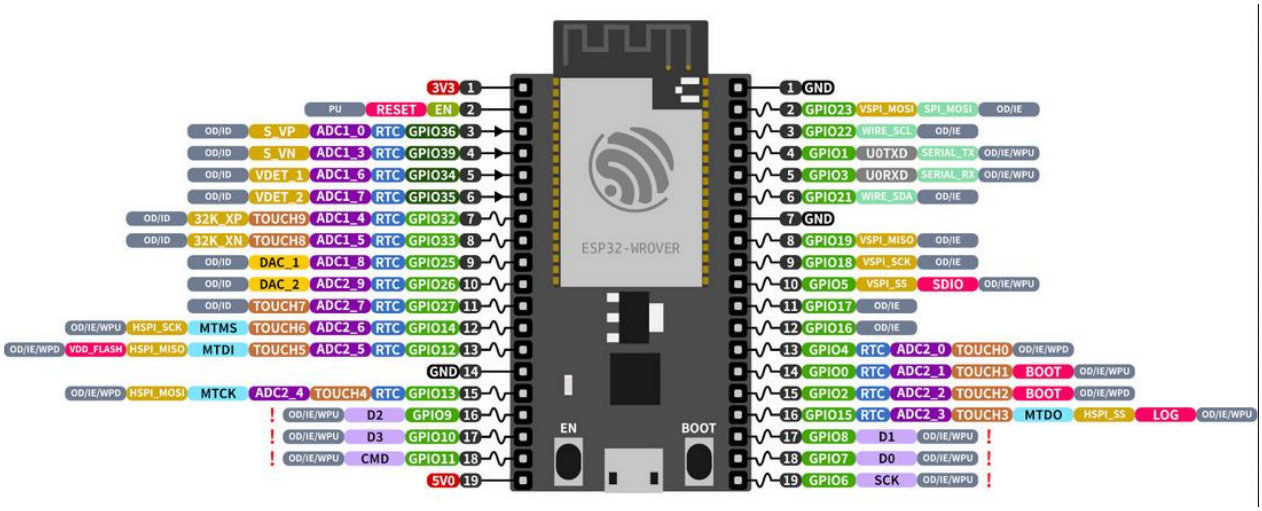


Figura 21. Esquema pines placa Devkitc V4 WROVER

Puesto que se trata de un chip programable, dadas las características y su precio lo hace compatible para un gran número de aplicaciones.

2.3 Conceptos básicos de MQTT

El protocolo MQTT Message Queing Telemetry Transport[4] es un estándar para el envío de datos de un sensor a un servidor, la conexión se abre una vez y se reutiliza las veces que haga falta.

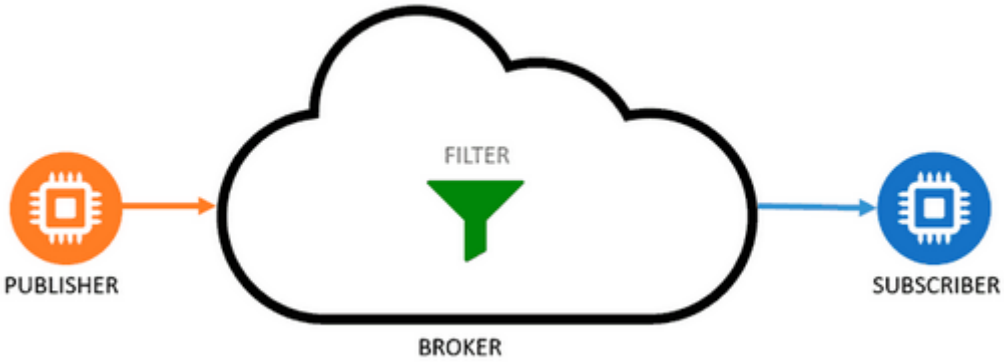


Figura 22. Esquema general MQTT

Los elementos que conforman este protocolo son:

- Publisher. El que envía la trama a un bróker, dicha trama llevará un topic que se puede escoger libremente.
- Subscriber. Es el otro extremo de la comunicación, aquel que se suscribe a un topic y recibe los mensajes que llegan al bróker bajo dicho topic.
- Topic. Es un concepto que identifica como se envían y reciben los mensajes, un Subscriber solo recibirá los mensajes de los topics a los que está suscrito.
- Broker. El que recibe tramas bajo diferentes topics y se encarga de que llegue a aquellos suscritos a los topics correspondientes.

Los mensajes son enviados al Broker bajo diferentes topics, donde un cliente puede enviar un mensaje a un topic concreto y todas las maquinas suscritas a ese topic lo van a recibir.

La siguiente imagen es un esquema de funcionamiento de una aplicación MQTT cualquiera.

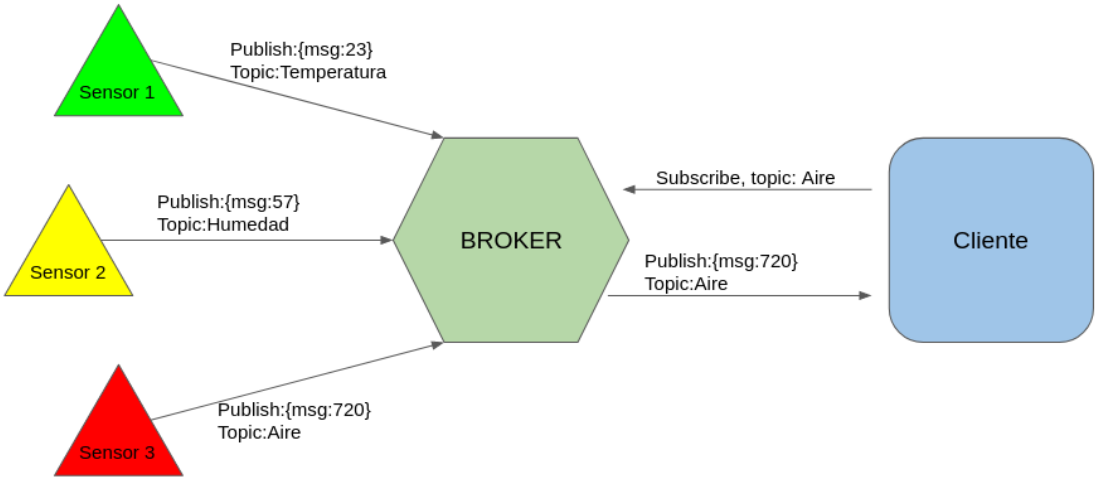


Figura 23. Diagrama ejemplo funcionamiento MQTT

En dicha imagen podemos ver tres sensores: Sensor 1, Sensor 2…, que hacen de **Publishers,** un **broker** que es la pieza intermedia y la que se encarga de recibir lo mensajes y reenviarlos y un cliente o **Susbcriber** que es el extremo final, puede haber varios clientes suscritos a los mismos o a diferentes topics.

Los clientes solo recibirán mensajes de aquellos topics a los que se suscriban y solo recibirán los recientes.

Cada mensaje MQTT tiene la siguiente estructura:

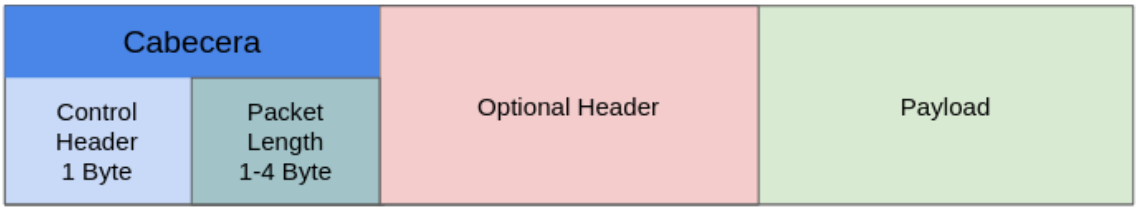


Figura 24. Trama MQTT.

La cabecera es el campo obligatorio del mensaje pues contiene el byte control que identifica el tipo de mensaje, así como el campo longitud del mensaje enviado.

El campo “optional header” es opcional y es usa cuando hace falta información adicional en determinadas situaciones.

Finalmente, el campo Payload que contiene el mensaje que se quiere enviar, el protocolo específico hasta 256 Mb, pero en aplicaciones reales se suele usar de entre 2 a 4 Kb.

A continuación, vamos a ver algunos de los bytes de control más importantes.

Message	Code	Description
Connect	0x10	Se envía cuando un cliente quiere establecer conexión con un broker.
Connack	0x20	Mensaje de reconocimiento en respuesta a un Connect.
Publish	0x30	Este indica que la información es un mensaje.
Puback	0x40	Si el mensaje enviado tiene un QoS superior a 0, el broker envía este reconocimiento con el ID del mensaje
Subscribe	0x80	Lo envía un cliente para suscribirse a un topic y poder recibir mensajes publish que vengan a este topic
Suback	0x90	Mensaje de reconocimiento para un mensaje de tipo Subscribe
Unsuscribe	0xA0	Mensaje que envía un cliente para indicarle al broker que no desea seguir recibiendo mensajes sobre un topic.
Unsuback	0xB0	Mensaje de reconocimiento a un mensaje de tipo Unsubscribe
Pingreq	0xC0	Mensaje que debe enviar un cliente para hacerle saber al broker que sigue escuchando para mantener la conexión
Pingresp	0xD0	Mensaje respuesta del Broker al cliente
Disconnect	0xE0	Mensaje enviado para terminar la conexión con el broker.

Tabla 1. Bytes control MQTT.

MQTT posee además un mecanismo para asegurar que los mensajes son recibidos, este mecanismo lo hemos mencionado con el tipo suback y se trata del QoS:

- QoS Como mucho una vez. El mensaje se envía una vez y puede que no llegue.
- QoS 1 Como mínimo una vez. El mensaje se envía y el emisor espera ACK si no lo recibe lo envía otra vez, el mensaje debe llegar como mínimo una vez, puede darse el caso de que mensajes retardados lleguen más tarde y se reciban duplicados.
- QoS 2. Exactamente una vez. Garantizado para que llegue exactamente una vez.

El desarrollo de la comunicación entre Publishers, Broker y Cliente se describe a continuación en la figura 25.

Figura 25. Cronograma Funcionamiento MQTT.

Como se puede observar se utilizan los comandos descritos en todo momento. Los publishers no necesitan estar suscritos al topic al que envían sus mensajes. Pero si requieren estar conectados al broker tal y como hace el cliente. El cliente solo recibirá los mensajes más recientes con forme estos lleguen al broker.

Capítulo 3. Diseño e implementación de un scanner y beacon BLE basado en ESP32

De forma resumida, los elementos principales del sistema son:

1. Scanner BLE basado en ESP3.
    a. Scannea los dispositivos BLE en cobertura.
    b. Cuando detecta un dispositivo genera un mensaje JSON que envía por MQTT al Server usando la interfaz Wifi del ESP32.
2. Server
    a. Recibe datos en bruto del scanner BLE por MQTT.
    b. Registra y guarda en BBDD.
    c. Un usuario puede hacer para visualizar el listado de dispositivos o descarga de un fichero CSV.

En la plataforma ESP32 se ha implementado un scanner BLE con envío de datos MQTT por Wifi y también un beacon para poder realizar pruebas controladas de recepción de datos y de cobertura con el scanner BLE.

Para el desarrollo sobre el ESP32 ha sido necesaria la documentación oficial de Espressif sobre las librerías a utilizar:

- Bluetooth: esp_bt.h[12]
- MQTT: mqtt_client.h [13]
- WiFi: esp_wifi.h [14]

además de la documentación de otras librerías de C como puede ser cJSON[15]

3.1 Scanner BLE usando el ESP32

El programa de escaneo BLE está diseñado para el ESP32, mediante este programa el módulo se encarga de:

1) Escanear el medio en busca de tramas advertisement
2) Ante un evento de advertisment formar mensajes JSON con la información obtenida
3) Enviar los mensajes JSON mediante el protocolo MQTT a un servidor mosquitto usando la interfaz Wifi del ESP32. El servidor mosquitto se configura y se ejecuta en un PC independiente.

El siguiente diagrama describe como sucede dicha comunicación:

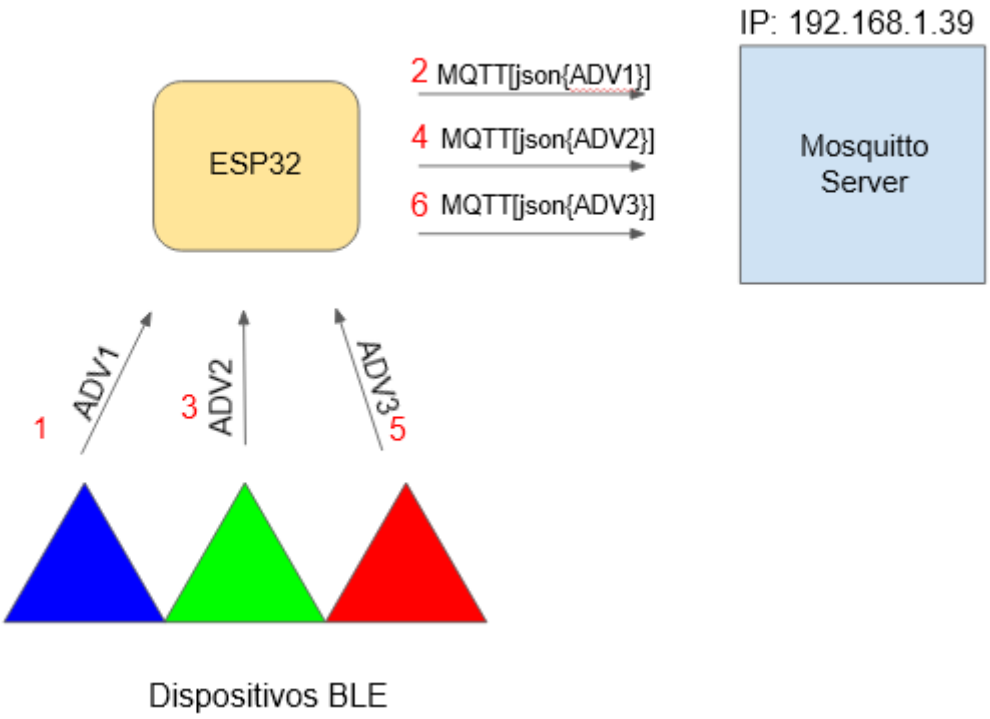


Figura 26. Esquema comunicación ESP32.

Se tienen unos dispositivos BLE en cobertura, estos pueden ser desde SmartTVs, beacons o Smart bands, es decir, cualquier dispositivo BLE que envíe tramas advertisement al medio. El ESP32 capta dichos mensajes, obtiene diferentes campos a partir de ellos como son la MAC del dispositivo que ha enviado cada trama, así como el campo Optional Payload con estos datos forma un mensaje en formato JSON, haciendo uso de la librería cJSON.

El mensaje JSON que se construye tras cada evento de advertisment incluye los siguientes campos:

- **Número de secuencia**: Contador cíclico que gestiona el propio ESP32 y que se incrementa con cada trama encontrada.
- **IdScanner**: Un identificador para el dispositivo que ha encontrado la trama.
- **TipoMAC**: Este valor, que puede ser 0 o 1, nos indica si la MAC es pública (conocida) o no es aleatoria respectivamente.
- **MAC**: La dirección física del dispositivo que se está promocionando.
- **TipoAdv**: Este nos indica el tipo de advertisement que se ha recibido:
    1. ADV_IND
    2. ADV_DIRECT_IND
    3. ADV_SCAN_IND
    4. ADV_NONCONN_IND
    5. SCAN_RSP
- **ADV**: Este es el campo optional payload.
- **RSSI**: La potencia con la que se ha recibido la trama.

Una vez formado el mensaje lo envía con QoS 0 al servidor MQTT que se le ha programado mediante la conexión WiFi que se le configure. Hay que señalar que la URL del servidor MQTT puede ser privada (estar en una LAN privada) o pública. En el caso de que el servidor MQTT sea público, el mismo ESP32 se encarga de buscar por DNS la dirección IP correspondiente.

El servidor en particular se trata de un mosquitto hub[16], no requiere programación, únicamente instalarlo y ponerlo en marcha:

```
sudo apt update
sudo apt upgrade
sudo apt-get install mosquitto mosquitto-clients
```

Instalación

```
sudo systemctl enable mosquitto.service
```

Puesta en marcha servicio MQTT Mosquitto

También es posible en el lado del servidor mediante el mosquitto hub escuchar un ***topic*** en particular:

```
mosquitto_sub -d -h localhost -p 1883 -t "ESPBLE21"
```

Subscripción

O incluso enviar mensajes ***publish***:

```
>mosquitto_pub -d -h localhost -p 1883 -t "ESPBLE21" -m "Hola Mundo"
```

El programa del scanner BLE va a seguir grosso modo el siguiente **ciclo de trabajo:**

1. Se pone la radio BLE en modo SCAN BLE.
2. En el momento que se detecta un evento de Advertisment, es decir la recepción de un mensaje beacon, se procesa el dato y se construye el mensaje JSON.
3. A continuación se habilita la interfaz Wifi y se envía un mensaje MQTT al server.
4. Se vuelve a activar la radio BLE en modo SCAN BLE y se repite el ciclo de trabajo.

Una limitación de la solución sobre ESP32 y envío de datos MQTT es que la antena del módulo ESP32 está compartida por la radio Bluetooth Low Energy y la radio Wifi. Por tanto, las radios BLE y Wifi no pueden tener un funcionamiento simultáneo, sólo puede alternarse su uso. Como consecuencia, mientras se envía un mensaje MQTT por Wifi no se están escaneando potenciales beacons BLE en cobertura.

A continuación se va a ver en detalle el funcionamiento del programa explicando su diagrama de flujo.

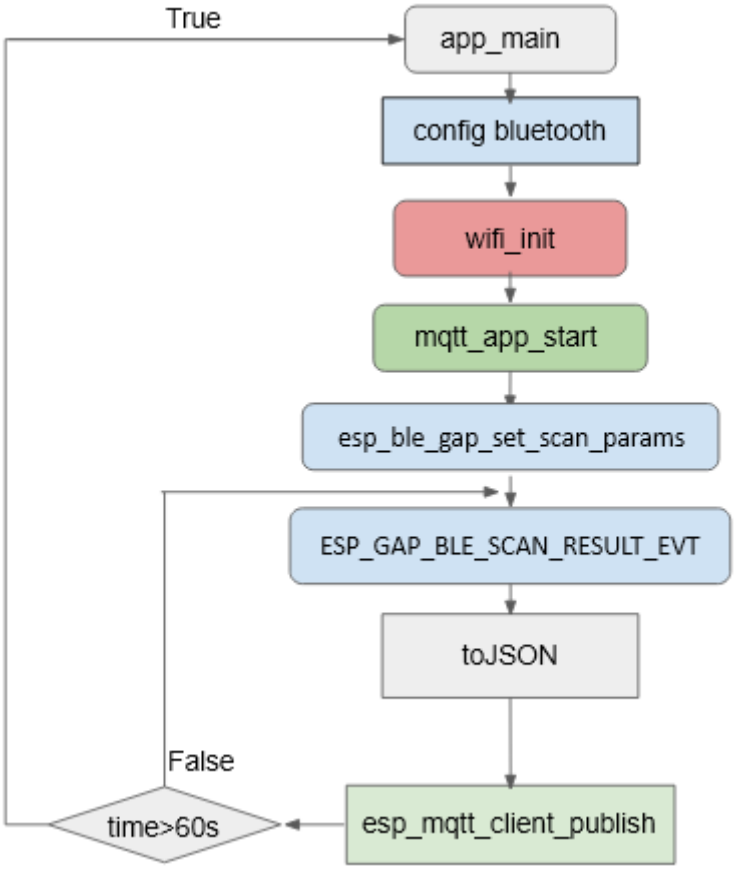


Figura 27. Diagrama flujo programa escáner BLE

El programa empieza en una función llamada app_main, para los scripts hechos en C este es el punto de partida, desde esta misma función se configuran las librerías de Bluetooth Low Energy, la configuración básica que este va a usar y entonces se llama a Wifi_init, función que se encarga de establecer conexión con el punto de acceso WiFi que se le haya configurado.

Primero se inicializa WiFi, cargando la configuración por defecto y configurando el SSID y la contraseña de este. Se configura además el modo Estación.

Cuando el ESP32 consigue conectarse al WiFI obtendrá su IP por DHCP. Entonces ahora se llama a la función mqtt_app_start para configurar MQTT de una forma similar, pasando como parámetro la dirección del servidor al cuál conectarse.

```
char topic[] = "ESPBLE21";
#define EXAMPLE_ESP_WIFI_SSID      "SSID"
#define EXAMPLE_ESP_WIFI_PASS      "PASS"
#define EXAMPLE_ESP_MAXIMUM_RETRY  10
#define MQTT_SERVER "mqtt://192.168.1.112:1883"
```

Una vez el ESP32 se conecta al servidor MQTT se llama a la función esp_ble_gap_set_scan_params, que toma como parámetros los parámetros que utilizará Bluetooth durante el escaneo, que son los siguientes:

```
static esp_ble_scan_params_t ble_scan_params = {
        .scan_type              = BLE_SCAN_TYPE_PASSIVE,
        .own_addr_type          = BLE_ADDR_TYPE_PUBLIC,
        .scan_filter_policy     = BLE_SCAN_FILTER_ALLOW_ALL,
        .scan_interval          = 0xA0,
        .scan_window            = 0xA0
    };
```

El tipo de escaneo es pasivo, es decir, el ESP32 se limitará a escuchar el medio, utilizará su propia MAC y el intervalo y ventana de escaneo se fijan a 100 ms (0xA0), para de esta forma garantizar que está escaneando durante dicho tiempo.

A partir de este momento se llama a otra función que hace de callback, para el escáner Bluetooth, la cual se ha configurado antes de llamar a la función WiFi_Init:

```
esp_ble_gap_register_callback(esp_gap_cb);
```

Siendo esp_gap_cb el nombre de la función que hace de callback.

Cada evento que suceda: Escaner en marcha, escaner parado, trama encontrada… Se gestiona dentro de este callback. En concreto cada vez que se encuentra una trama, el evento será de tipo: ESP_GAP_BLE_SCAN_RESULT_EVT

Lo que se hace es crear un JSON y, a partir de la variable param, la cuál contiene todo lo relacionado a la trama, obtener los datos que queremos mandar como JSON, ejemplo:

```
// creamos un objeto JSON
cJSON *data_BLE_json;

data_BLE_json = cJSON_CreateObject();

//Le incorporamos cada campo
cJSON_AddItemToObject(data_BLE_json, "Numero", cJSON_CreateNumber(found_frames));
cJSON_AddItemToObject(data_BLE_json, "IdScanner", cJSON_CreateNumber(IdScanner));
cJSON_AddItemToObject(data_BLE_json, "TipoMac", cJSON_CreateNumber(param->scan_rst.ble_addr_type));
```

Mencionar que IdScanner y found_frames son dos variables creadas, una con el objetivo de dar ID al que envía los datos por MQTT, ya que este de por si no tiene dicho mecanismo y el otro es para poder dar un número de secuencia.

Cuando se introduce cada dato al JSON, lo que hacemos es transformarlo en una cadena de caracteres:

```
//Se transforma a cadena de char
out = cJSON_Print(data_BLE_json);
```

Para su envío por MQTT, además se vacía las variables mediante la función free(), ya que C no incluye el mecanismo de recolección de basura como si incluyen otros lenguajes.

```
int status = esp_mqtt_client_publish(client, topic, out, 0, 0, 0);

if(status < 0){
    esp_restart();
}
free(out);

free(data_BLE_json);
```

La función esp_mqtt_client_publish es la que envía el mensaje MQTT, esta función retorna un entero, si este es menor que 0 es que ha ocurrido un error.

El escaneo repite los mismos pasos durante un minuto, transcurrido dicho tiempo el ESP32 se reinicia y vuelve a empezar todo el ciclo. De esta forma se limpia su memoria.

Tanto WiFi como MQTT, así como Bluetooth cuentan con sus respectivas funciones en el código que gestiona todos los eventos.

Instalación del código en el ESP32

Para la instalación de este y todos los programas relacionados con Bluetooth en el ESP32 hay que seguir los siguientes pasos:

En la ventana de comandos del ESP-IDF hay que situarse en el directorio del programa y ejecutar los siguientes comandos:

```
C:\Users\grief\Desktop\BLE_ADV>idf.py set-target esp32
```

Con este comando se inicializa el proyecto y se carga la configuración por defecto

Cuando el comando finalice se introduce:

```
C:\Users\grief\Desktop\BLE_ADV>idf.py menuconfig
```

Al introducir este comando la terminal tomará la siguiente forma:

Figura 28. Imagen de la interfaz menuconfig del ESP-IDF

Este menú permite modificar la configuración básica del ESP32.

Hay que navegar entre las opciones: Component Config/Bluetooth y una vez dentro habilitar Bluetooth dándole a la tecla direccional derecha, aparecerá un asterisco [*]

Cuando esté hecho, volver al menú raíz pulsando la tecla direccional izquierda, y navegar a Partition Table/Partition Table(Single factory, no OTA) y seleccionar Custom partition table CSV con la tecla direccional derecha. Cuando se haya acabado para salir pulsa la tecla Q y la tecla Y para guardar cambios.

A continuación, hay que introducir el comando:

```
C:\Users\grief\Desktop\BLE_ADV>idf.py build
```

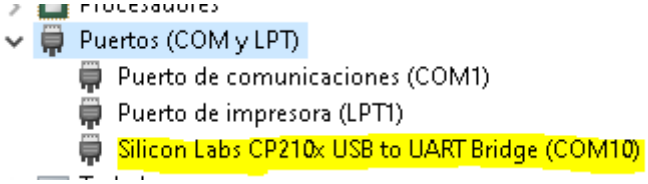
El comando puede tardar varios minutos dependiendo de cuan potente sea tu equipo, tiene que llevar a cabo una serie de procesos. Este es el comando que compila el programa para generar el archivo que tiene que cargar en el ESP32.

Cuando haya acabado se introduce el comando:

```
C:\Users\grief\Desktop\BLE_ADV>idf.py -p COMX flash
```

Para cargar el programa en el ESP32, la X dentro de COM es el número de puerto, para averiguarlo:

- En Windows 10, botón derecho sobre **Este equipo,** y seleccionar administrar. Cuando cargue la ventana de Administrador de Equipos, hacer click en administrador de dispositivos y desplegar Puertos (COM y LPT) de la ventana del centro.



- En Linux, abrir una terminal e introducir el comando:

  ls dev/ttyUSB*

Una vez hecho el comando flash se carga el programa en el dispositivo.

Si durante el proceso en la pantalla se muestran muchos puntos y barras, tal que:

```
esptool.py v3.0
Serial port COM10
Connecting........_____.....______.....______.....
```

Deberá pulsar el botón de la derecha, figura 29, de la placa del ESP32 que tiene el nombre “Boot”, hasta que la terminal cambie.

Figura 29. Imagen botón placa.

El ESP32 cuando recibe el código se conecta a la red WiFi dada y empieza a enviar los datos a la dirección del servidor mosquitto, la cuál es http://scanble21.ddns.net:1443/

El servidor Mosquitto recibe todas las tramas:

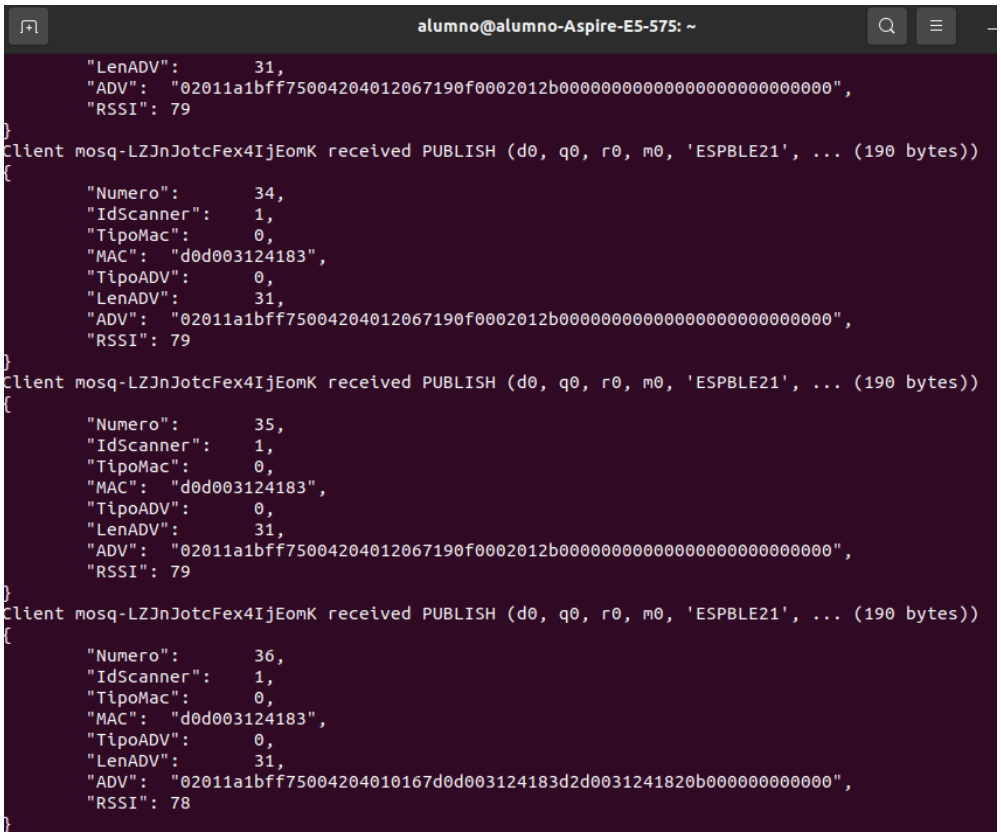


Figura 30. Captura terminal Linux con Mosquitto recibiendo las tramas.

3.2 Beacon BLE usando ESP32

Este programa para el ESP32 le da la funcionalidad de un emisor de tramas de Advertisement, un beacon, la utilidad principal de este programa es la de emitir

tramas, estas pueden contener el mensaje que se desee, siempre y cuando se respete la estructura de mensajes advertisement: Longitud, tipo y dato, dicha estructura se describió en el apartado de Advertising de beacons del Capítulo 2.

La utilidad principal de este programa con el modo beacon BLE es para poder controlar la cantidad de mensajes transmitidos y poder realizar y evaluar pruebas de cobertura y tasa de éxito en la recepción de mensajes del programa scanner BLE.

El funcionamiento del programa beacon BLE ESP32 se encuentra descrito en el siguiente diagrama de flujo:

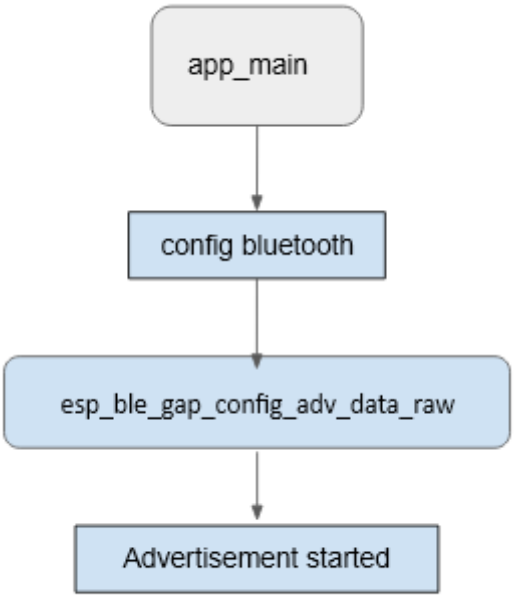


Figura 31. Diagrama flujo programa beacon BLE.

Como todos los scripts de C, se empieza por la función app_main, a partir de esta se configuran las librerías relacionadas a Bluetooth, así como los parámetros para el intervalo de promoción, la MAC a utilizar, el tipo de trama advertisement…

```
static esp_ble_adv_params_t ble_adv_params = {

    .adv_int_min = 0xA0,    //100 ms intervalo de mensajes
    .adv_int_max = 0xA0,
    .adv_type = ADV_TYPE_NONCONN_IND,
    .own_addr_type  = BLE_ADDR_TYPE_PUBLIC,
    .channel_map = ADV_CHNL_ALL,
    .adv_filter_policy  = ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY,
};
```

El intervalo se fija a 100 ms, el tipo de trama es ADV_NONCONN, utilizará su propia MAC y se va a promocionar en todos los canales. El intervalo máximo y mínimo es el mismo, para así asegurar que se va a promocionar con ese periodo.

También se asigna cuál será la función de callback que se encargue de gestionar los eventos:

```
// register GAP callback function
esp_ble_gap_register_callback(esp_gap_cb);
printf("- GAP callback registered\n\n");
```

Cuando toda la configuración ha acabado se lanza la función esp_ble_gap_config_adv_data_raw, que toma como parámetro un array de bytes el cuál será el mensaje con el que se promocione en el medio. A continuación podemos ver un mensaje de ejemplo que se utilizará para las pruebas de cobertura:

```
static uint8_t adv_raw_data[17] = {0x02,0x01,0x06,0x03,0x03,0xE1,0xFF,0x09,0x09,0x45,
                                   0x53,0x50,0x33,0x32,0x41,0x44,0x56};
```

El mensaje contiene 3 tipos distintos en él:

- **0x020106**, estos bytes codifican el tipo flag, y los que utiliza son LE General Discoverable Mode y BR/EDR Not Supported.

Data Type	Octet	Bit	Description
«Flags»	0	0	LE Limited Discoverable Mode
	0	1	LE General Discoverable Mode
	0	2	BR/EDR Not Supported. Bit 37 of LMP Feature Mask Definitions (Page 0)
	0	3	Simultaneous LE and BR/EDR to Same Device Capable (Controller). Bit 49 of LMP Feature Mask Definitions (Page 0)
	0	4	Simultaneous LE and BR/EDR to Same Device Capable (Host). Bit 66 of LMP Feature Mask Definitions (Page 1)
	0	5..7	Reserved for future use

Figura 32. Imagen del Bluetooth Core, tabla con los Bytes que conforman el tipo Flag

En concreto los bits 1 y 2 se sabe que son esos porque de los 5 disponibles que podemos apreciar en la tabla son los únicos que suman 6:

$$2^1 + 2^2 = 2 + 4 = 6$$

- **0x0303E1FF**, estos bytes indican indican una longitud de 3 y el tipo service UUID 16 bit. Por ende el siguiente dato contiene dicho UUID de 16 bits pero este se encuentra codificado en Little-endian tal y como exige el Bluetooth Core.
- **0x09094553503332414456**, El tipo es el long name, este tipo se usa para transmitir un nombre, el formato de los datos es ascii y se puede leer “ESP32ADV”

Una vez se configura el payload a promocionar desde dentro de la función gestora de eventos se llama a la función esp_ble_gap_start_advertising:

```
case ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT:

    printf("ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT\n");
    esp_ble_gap_start_advertising(&ble_adv_params);
    break;
```

El siguiente evento que ocurre será el inicio de la promoción de tramas:

```
case ESP_GAP_BLE_ADV_START_COMPLETE_EVT:

    printf("ESP_GAP_BLE_ADV_START_COMPLETE_EVT\n");
    if(param->adv_start_cmpl.status == ESP_BT_STATUS_SUCCESS) {
        printf("Advertising started\n\n");
    }
    else printf("Unable to start advertising process, error code %d\n\n", param->scan_start_cmpl.status);
    break;
```

A partir de que se da este evento, el ESP32 ya funciona como un beacon y por tanto las tramas que este envía se pueden escuchar tanto por el escáner BLE que hemos descrito en el apartado anterior, como otras aplicaciones.

3.2.1 Beacon BLE usando módulo WROVER Devkit C

La instalación es idéntica para este código, con la diferencia de que **debes saltarte** la parte de la configuración del partition table dentro del menuconfig. El resto de configuración: activar Bluetooth, los comandos, son necesarios.

Para visualizar las tramas se va a utilizar nRF Connect tal y como hicimos con los iBKS en el apartado de Escaneo BLE.

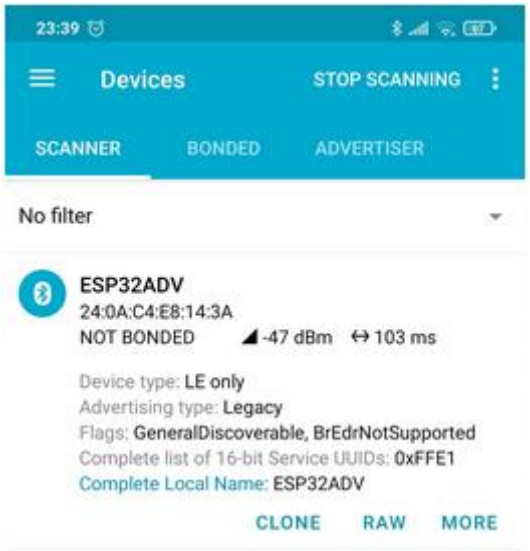


Figura 33. Captura nRF Connect

En la Figura 33 se puede ver que la aplicación detecta todos los parámetros configurados, desde el nombre, hasta el UUID.

Capítulo 4. Diseño e implementación de la arquitectura del sistema

Para el desarrollo de esta arquitectura se han utilizado tres elementos principales, uno es el ESP32 con el programa escáner descrito en el capítulo anterior, el otro es un programa llamado Mosquitto, el cual es el servidor al que el ESP32 envía los mensajes por medio del protocolo MQTT. El último elemento es un back-end desarrollado con el framework de Javascript NodeJS.

4.1 Arquitectura global del sistema

Como se ha expuesto, son tres los elementos principales de este sistema:

- ESP32. Se utiliza con el programa de escáner de beacons, este será el encargado de escuchar el medio en busca de tramas de advertisement y utilizando un punto de acceso, el cuál puede ser un router o la conexión compartida de un Smartphone, enviará los datos por medio de MQTT a un servidor de MQTT.
- Mosquitto Hub. Es el servidor MQTT con el que se comunica el ESP32, su función principal es escuchar a los publishers y reenviar los datos a los subscribers.
- Back-end. Posee dos funcionalidades:
    - La primera es suscribirse al Mosquitto Hub en el mismo topic al que el ESP32 está volcando su información, la información que le llegue la almacenará en una Base de datos relacionar, para este trabajo se utiliza SQLite.
    - La segunda funcionalidad es alojar un servicio REST que espera peticiones HTML, de esta forma un usuario puede solicitar acceso a una pagina gestionada en este mismo servidor, a partir de la cual podrá ver cuales son las últimas tramas recibidas por los escáneres o descargarse un fichero CSV con datos de tramas capturadas en el instante temporal que él desee.

En la Figura 34 se muestra la arquitectura global del sistema:

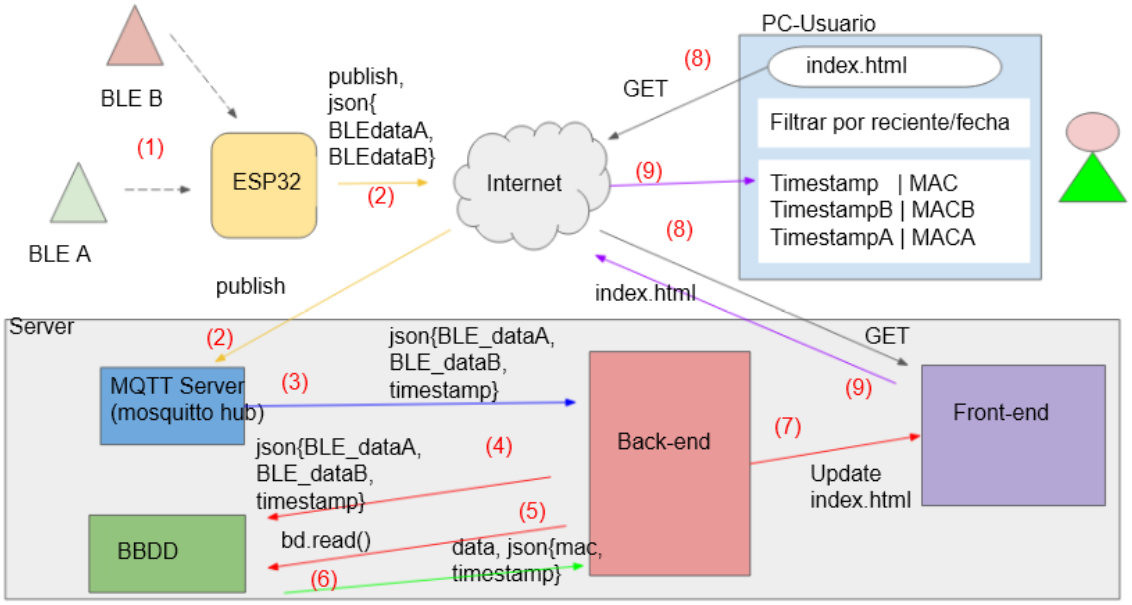


Figura 34. Arquitectura del sistema implementado

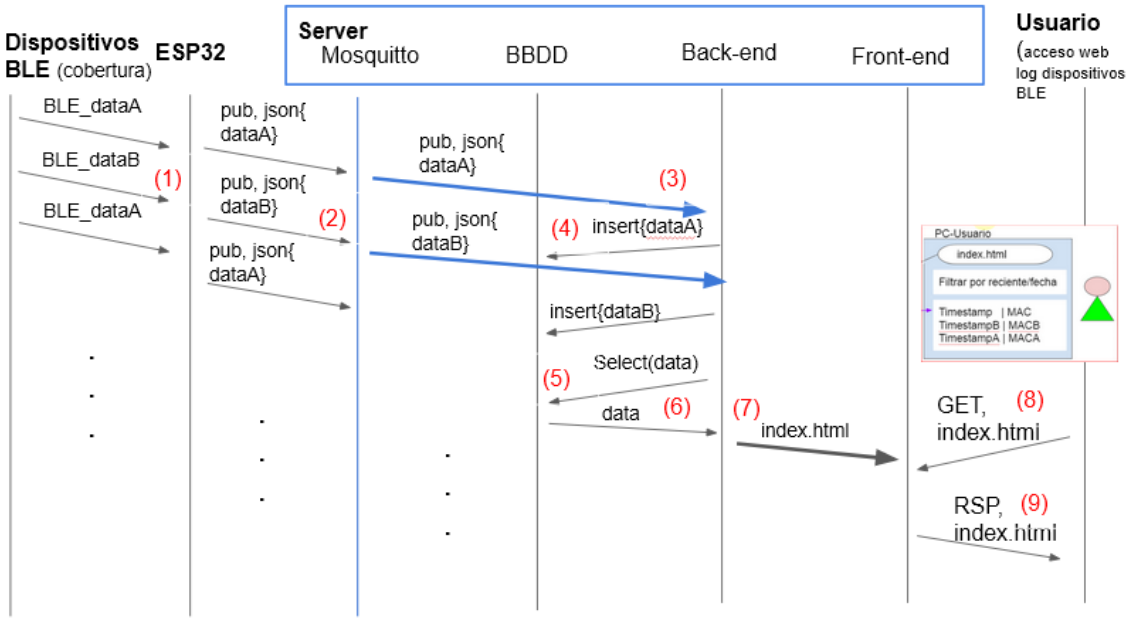


Figura 35. Cronograma alto nivel con la secuencia de mensajes entre módulos

Se describe a continuación los pasos seguidos en el cronograma y diagrama anteriores.

1. Los **Dispositivos BLE** envían tramas advertisement con un periodo definido por sus fabricantes. Estas tramas son recibidas por el ESP32 que permanece a la escucha.
2. Las tramas conforme son captadas por el **ESP32**, son ensambladas en formato JSON dentro de una trama MQTT y enviadas al Server Mosquitto.
3. El **Server Mosquitto** es un broker MQTT, por tanto cuando una trama llega a un topic, la reenvía a todos aquellos suscritos a ese topic y en este caso aquel que está suscrito a ese topic es el back-end

4. Cuando el **Back-end** recibe tramas del broker lo que hace es almacenarlas en la Base de datos (BBDD), almacenará todos los datos enviados por el ESP32 además de la fecha y hora en la que se recibió la trama. Esta fecha y hora es proporcionada por Javascript.
5. La **BBDD**, almacena todos los datos enviados por el ESP32 y esta información queda accesible para otros usos
6. Cuando sea necesario el Back-end puede pedir información a la base de datos sobre unas tramas especificas o directamente sobre las últimas tramas que han llegado.
7. Con la información más reciente, el propio **front-end** hace peticiones cada cierto tiempo para recargar la página que el usuario esté viendo y muestre las tramas más recientes. Si en el momento de hacer la petición hay una trama nueva, la pagina se actualizará inmediatamente, en caso contrario, pasados 5 minutos se actualiza igual.
8. Cada vez que un **Usuario** nuevo se conecta, envía una petición al Back-end, el cual le devolverá una página que está programa para mandar al back-end peticiones para generar un CSV o recargarse si es necesario.

4.2 Diseño back-end NodeJS y BBDD del servidor

A continuación se va a explicar de forma superficial y con pseudocódigo el funcionamiento del back-end.

- Diagrama bloques programa PC/servidor/Gateway
- Diagrama de flujo pseudocódigo
- Formato y tipo de mensajes entre los módulos

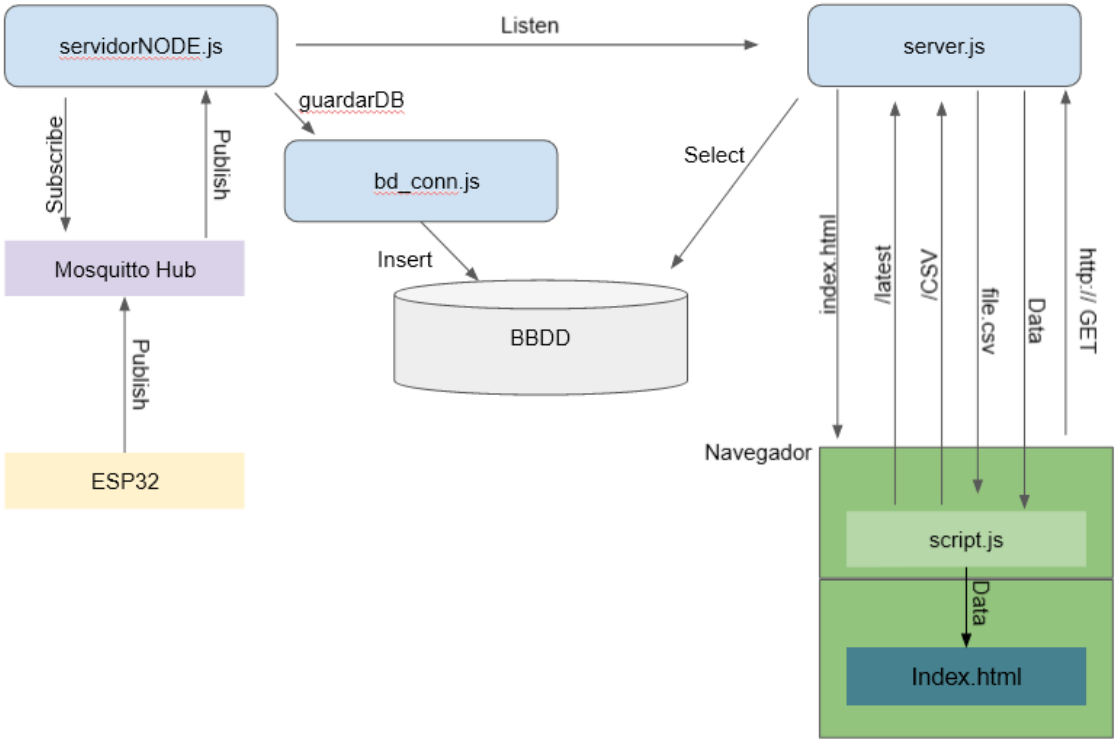


Figura 36. Interconexión módulos servidor

El esquema de arriba nos describe el comportamiento entre los módulos que componen el código del back-end, como interactúan entre ellos, con el mosquitto hub y con el ESP32.

El ESP32 se conecta al bróker de mosquitto y por medio de protocolo MQTT manda tramas publish que contienen la información de lo que este ha escaneado en el medio en el que se encuentra, entonces el bróker lo reenvía a todos los dispositivos suscritos al mismo topic que él. El back-end, en el script servidorNODE.js que tiene un código que se suscribe al bróker y escucha las tramas, este script además llama a bd_Conn.js, en concreto a la función guardarDB, que lo que hace es tomar como argumento el mensaje recibido y guardarlo en la base de datos de SQLite, mediante **comandos SQL**.

El módulo servidorNODE.js además activa el script server.js para que este escuche peticiones, estas peticiones pueden venir de **usuarios** que desean ver la página o para actualizar la página que ya se está visualizando, utilizando un script solicita nueva información para actualizar lo que el usuario está viendo o para generar un archivo CSV a petición de este.

En el caso de que sea un usuario conectándose para ver la página este le devolverá un archivo html y un script que se encarga de darle contenido al html y actualizarlo periódicamente por medio de peticiones al servidor.

Cuando el script del servidor recibe una petición tipo **/latest**, dicha petición deberá llevar hora y fecha del intervalo que desea recibir, entonces el servidor activa una **búsqueda SQL** para encontrar las tramas encontradas en ese intervalo y se las devuelve. El script actualizará la tabla que el usuario ve por sí mismo.

Si el servidor recibe una petición **/datosCSV,** realizará una consulta del intervalo indicado en la petición y creará un archivo .csv que podrá ser descargado por el usuario.

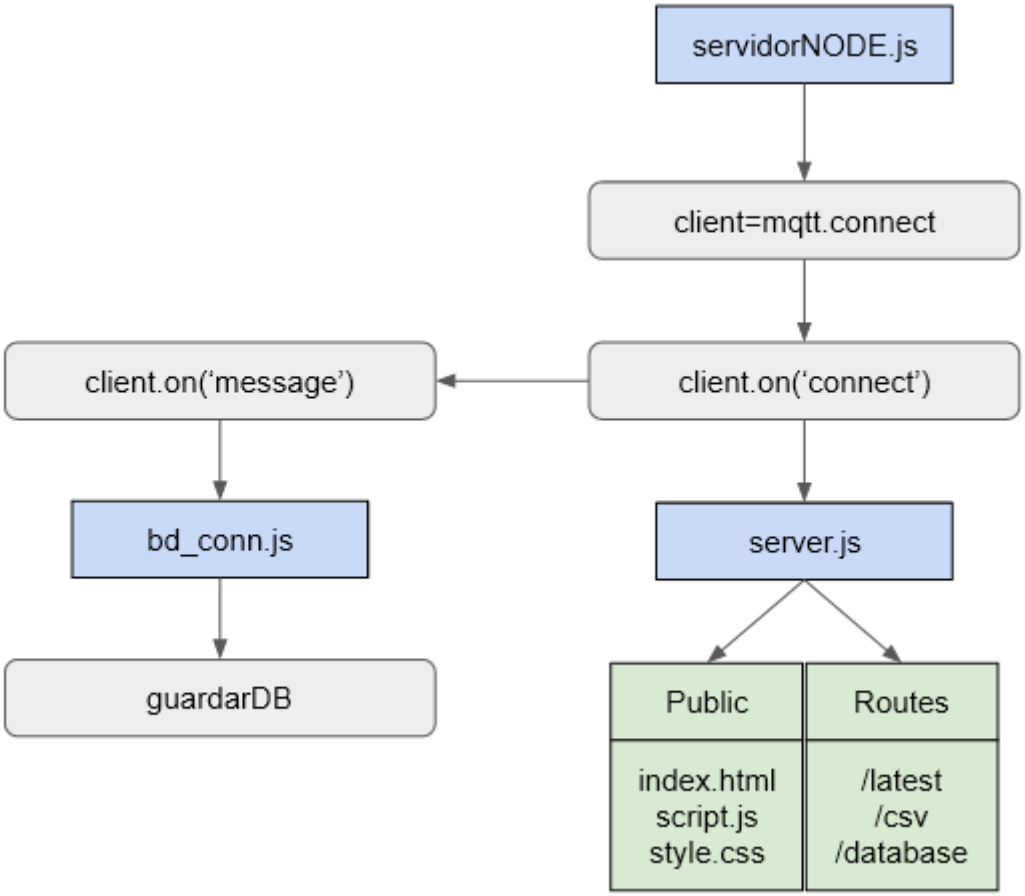


Figura 37. Diagrama flujo funciones

El servidor se pone en marcha ejecutando el comando:

```
/NODE-MQTT$ node servidorNODE.js
```

Una vez se pone en marcha se conecta al servidor MQTT y a la base de datos, la conexión se realiza a través de la librería mqtt, en concreto:

```
const client  = mqtt.connect('mqtt://localhost:1883',options)

let topic = 'ESPBLE21';

/*Esta parte se pone a la escucha del servidor local
en mosquitto y actualiza la base de datos
*/
client.on('connect', function () {
  client.subscribe(topic, function (err) {

    if(err){
        console.log(err);
    }

  })
})

client.on('message', function (topic, message) {

  guardarDB(message);

})
```

Con esas tres llamadas se conecta al servidor, se suscribe al ***topic*** y se queda a la escucha de mensajes que lleguen al ***topic***.

Cada vez que llegue un mensaje se llamará a la función guardarDB, ubicada en el script BD_conn.js a la que le pasará el mensaje que le ha llegado. La función en si misma se encarga de conectarse a la base de datos y de insertar los datos.

Una vez MQTT está preparado el servidor ya escucha todos los mensajes que llegan del ESP32 al mosquitto, es entonces cuando se inicia el servicio REST, el cual se invoca mediante un constructor de la clase Server.

Dentro de esta clase se inicializa el servidor, conectándolo también a la base de datos para que pueda buscar la información cuando esta se solicite, cuando el servidor está preparado puede o bien escuchar peticiones de conexión a las cuales les devuelve una página html con un script y un CSS, el script se encarga de darle contenido a la página realizando solicitudes al servidor. Solicitudes que pueden ser:

- **/latest** esta petición deberá contener entre sus argumentos el intervalo de fecha y hora del que se quiera la medida. Devolverá un array con las MACs que se han encontrado, la hora en la que se encontraron, el ID del scanner que lo encontró, las veces que se registró y el fabricante.
- **/datosCSV** con esta petición que requiere de los mismos parámetros y un nombre para darle al archivo, se devuelve un archivo CSV para que el usuario lo pueda descargar. El archivo contiene el ID del escáner, fecha, hora, MAC, campo payload y fabricante.
- **/database** esta petición se usa para restringir la fecha de la que el usuario puede pedir un archivo CSV, es decir, impedirá que la página del usuario

intente pedir una fecha de la cual no se tiene medida alguna en la base datos. La información que se devuelve es la fecha mínima y máxima de la que se tienen datos.

4.3 Diseño del front-end e interfaz de usuario

En este apartado se va a explicar como funciona el front-end, como realiza las peticiones para obtener información desde el back-end

El Front-end es la parte que se ejecuta en el dispositivo del cliente, este contiene entre otras cosas el código que hace que se pueda visualizar la información del ESP32 que llega al servidor.

El código, como se ha mencionado anteriormente también se encarga de preguntar al back-end por si hay nueva información destacable y poder actualizarse con dicha información.

A continuación, vamos a repasar el diagrama de flujo del front-end.

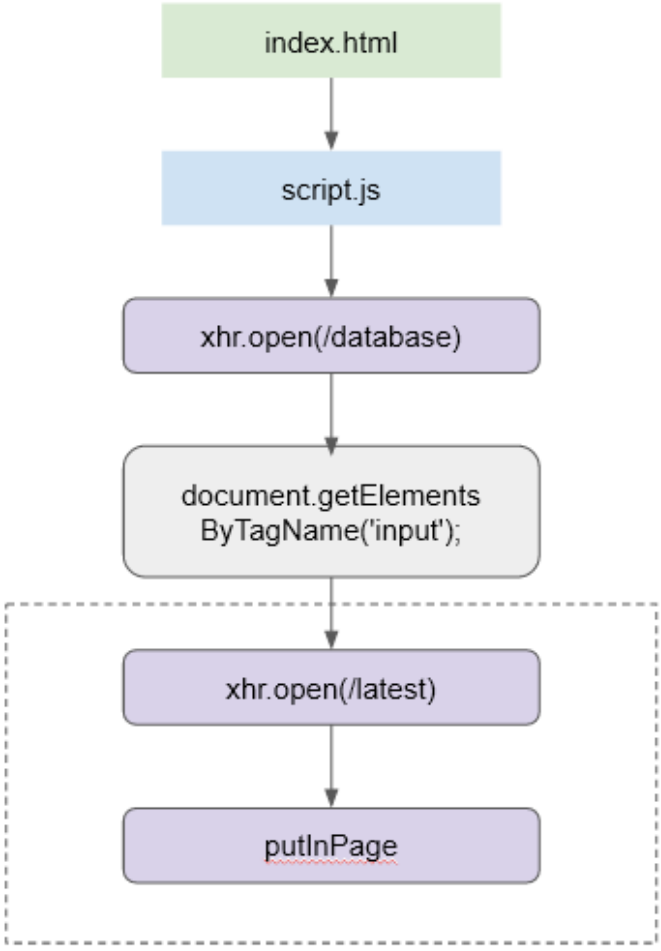


Figura 38. Diagrama flujo front-end

Como es de esperar el navegador lo primero que ejecuta es el código html, pues este contiene el esqueleto de la página, contiene también un script interno de javascript que es el que se encarga de comprobar las peticiones del usuario para generar un CSV y llevarla a cabo.

Cuando el navegador ha terminado de cargar el html, llega el turno del script.js, este contiene el código que se pone en contacto con el servidor, realizando peticiones al servicio REST de este, para así obtener información sobre las últimas tramas allegadas y así formar una tabla que será la que vea el Usuario:

Figura 39. Imagen de la página

Lo primero que hace el script es hacer la consulta "**/database**", la cual restringe los días en los que el usuario puede pedir medición:

Figura 40. Selección de fecha para CSV

La información que le llega se la pasa a los botones, que en html reciben la etiqueta (tag) de input, por tanto en javascript se llama a la función DOM:

```
let fecha =  document.getElementsByTagName('input');
fecha[0].setAttribute('min',datos[0]['min']);
fecha[0].setAttribute('max',datos[0]['max']);
fecha[2].setAttribute('min',datos[0]['min']);
fecha[2].setAttribute('max',datos[0]['max']);
```

Y dando el atributo min y max a los elementos. Esto establecerá la fecha mínima y la máxima a la que el usuario puede acceder.

Cuando ha acabado el script entra en el siguiente bucle:

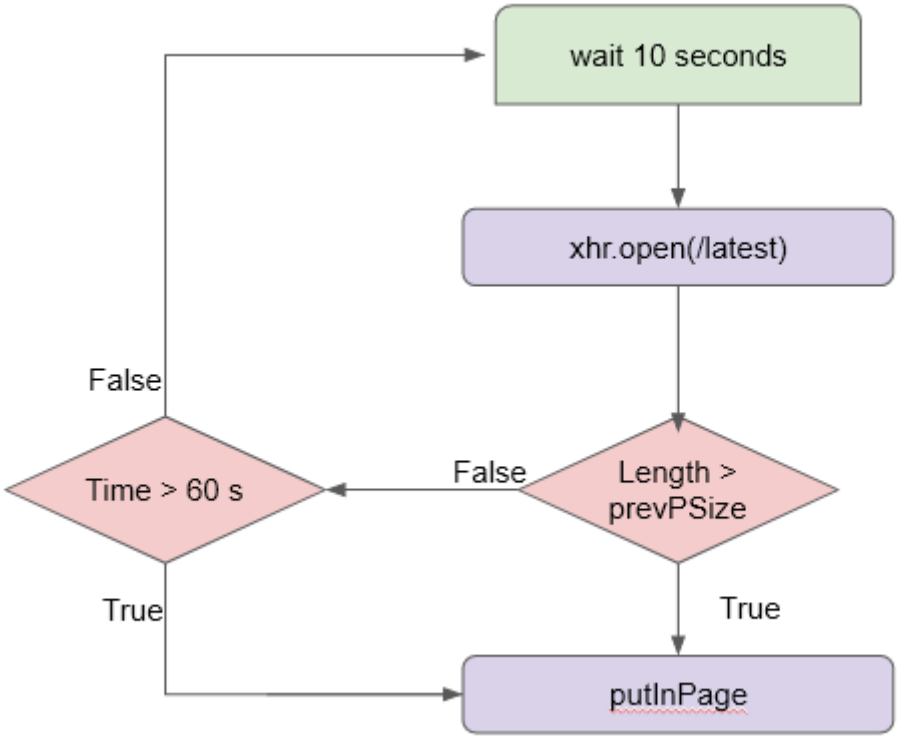


Figura 41. Diagrama flujo script principal de la página

El cliente realiza la petición /latest cada 10 segundos, si la respuesta que llega es superior a la anterior respuesta entonces se actualizará la página pues esto significa que se ha encontrado un nuevo dispositivo BLE, en caso contrario si no han pasado 60 segundos en total no actualizará.

4.4 Acceso remoto al servicio

Para poder acceder al servidor incluso fuera de casa se ha utilizado un servicio gratuito de DNS: No-IP, el inconveniente con este servicio es que cada treinta días hay que actualizar el estado del dominio.

Se ha creado el dominio http://scanble21.ddns.net/ para poder acceder desde cualquier sitio a la página web. Y se ha configurado el cliente de DNS dinámico en el router de la siguiente forma:

Figura 42. Configuración DNS en router

Además, se ha configurado en el router la redirección de puerto por NAT para que el servidor de forma externa responda bajo el puerto 80, aunque internamente lo haga bajo otro puerto, además de facilitar un puerto de entrada para Mosquitto para que de esta forma el ESP32 pueda acceder al broker desde una conexión a internet distinta.

Server Name	External Port Start	External Port End	Protocol	Internal Port Start	Internal Port End	Server IP Address	WAN Interface	Remove
HTTP	80	80	TCP	8080	8080	192.168.1.93	ppp0.1	☐
MQTT	1883	1883	TCP	1883	1883	192.168.1.93	ppp0.1	☐

Figura 43. Puertos enlazados en router

Y ya que el servidor MQTT es accesible desde internet se ha creado un usuario para Mosquitto de forma que solo sea posible acceder a través de dichas credenciales.

Para crear un usuario en Mosquitto, primero se debe restringir que no entren aquellos que no usen usuario, esto se hace en un archivo de configuraciones que en Ubuntu está ubicado en la ruta /etc/mosquitto/mosquitto.conf

Hay que añadir las siguientes líneas:

```
allow_anonymous false
password_file /etc/mosquitto/mpass.txt
```

La primera es la que evita que entren usuarios sin contraseña, mientras que la segunda lo que hace es indicarle donde está el archivo con los nombres de usuario y contraseñas.

El archivo hay que generarlo introduciendo el siguiente comando en el terminal de Ubuntu:

```
~$ sudo mosquitto_passwd -c /etc/mosquitto/mpass.txt upct_telem2021
```

Es necesario hacerlo como super usuario, pues este comando crea un archivo en la ruta especificada, en este caso se ha generado en /etc/mosquitto/mpass.txt y además hay que indicarle el usuario, el cual hemos elegido como upct_telem2021, cuando se le da a la tecla de intro nos pedirá además la contraseña.

Dentro del archivo se almacena el usuario y un hash de la contraseña que se le haya introducido.

Capítulo 5. Pruebas y resultados

Para el desarrollo de las siguientes pruebas se ha utilizado el programa escáner en el ESP32 junto con el servidor, además del programa Beacon para las pruebas de cobertura y alcance.

Se han realizado dos pruebas distintas, una prueba de cobertura en la cual se mide la efectividad y alcance de la antena integrada del ESP32, la otra, se ha llevado el ESP32 con una batería portátil a sitios públicos como son la biblioteca de la universidad, el autobús y un supermercado, con el objetivo de ver cuantas tramas se encuentran.

Los datos serán tratados utilizando Python3 para graficar y visualizar los datos obtenidos.

En la Figura 44 se muestra el setup del sistema utilizado para las pruebas.

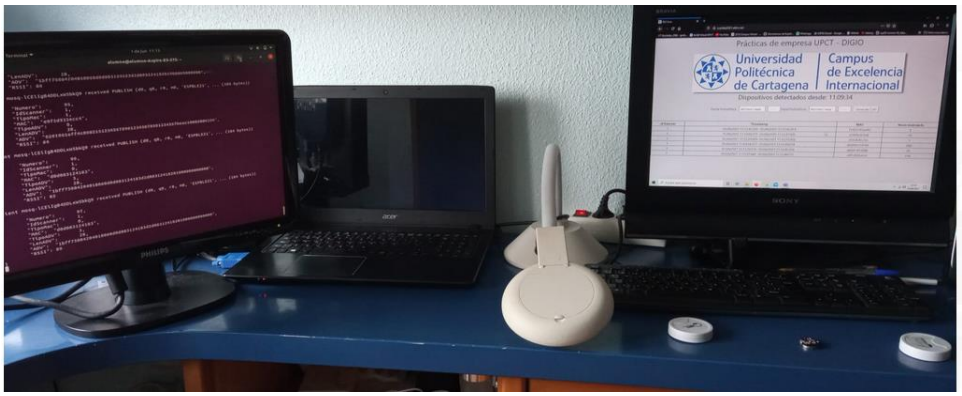


Figura 44. Setup del sistema

5.1 Pruebas de cobertura y alcance máximo

Para la realización de estas pruebas se han utilizado el escáner BLE desarrollado y el módulo ESP32 con el modo beacon BLE.

El ESP32 que se encarga de escanear estará conectado a corriente y enviando datos al servidor con normalidad, mientras que el emisor de tramas BLE estará conectado a una batería y se irá alejando del escáner para así probar la efectividad, así como el alcance del programa.

Para todas las pruebas se ha mantenido ambos dispositivos a la misma altura y fijos, estando la antena receptora (izquierda) mirando al suelo y la antena emisora (derecha) mirando hacía receptor en todo momento, como se puede ver en la siguiente figura:

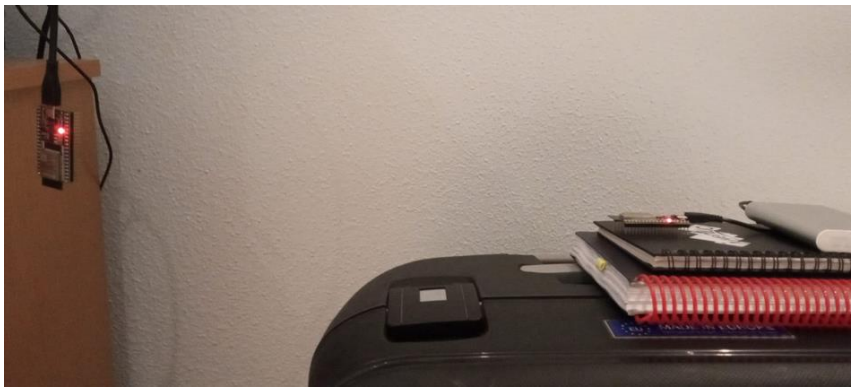


Figura 45. Imagen escenario pruebas cobertura

Para la primera prueba para comprobar cuál es el alcance verdadero del dispositivo, se ha utilizado el mismo programa escáner, pero sin conectarse a WiFi ni enviar por MQTT si no que envía los resultados por el puerto serie a través de su UART. Enviando los datos JSON por la UART no se bloquea el uso de la radio BLE y se obtiene un límite teórico superior de la cantidad máxima de datos que se pueden recibir si el 100% del tiempo el módulo BLE tiene el control de la antena.

Se han tomado medidas **cada minuto**, durante dicho intervalo el ESP32 se encontraba a una distancia fija, y variando en un metro una vez terminada la medición para volver a empezar, así hasta conseguir una distancia **total de 12 metros.** El medio utilizado es una distancia recta desde una habitación hasta otra, los dos módulos pueden verse en todo momento y el único posible obstáculo es un armario que no impide que se vean.

Los resultados obtenidos se pueden ver en la figura 46:

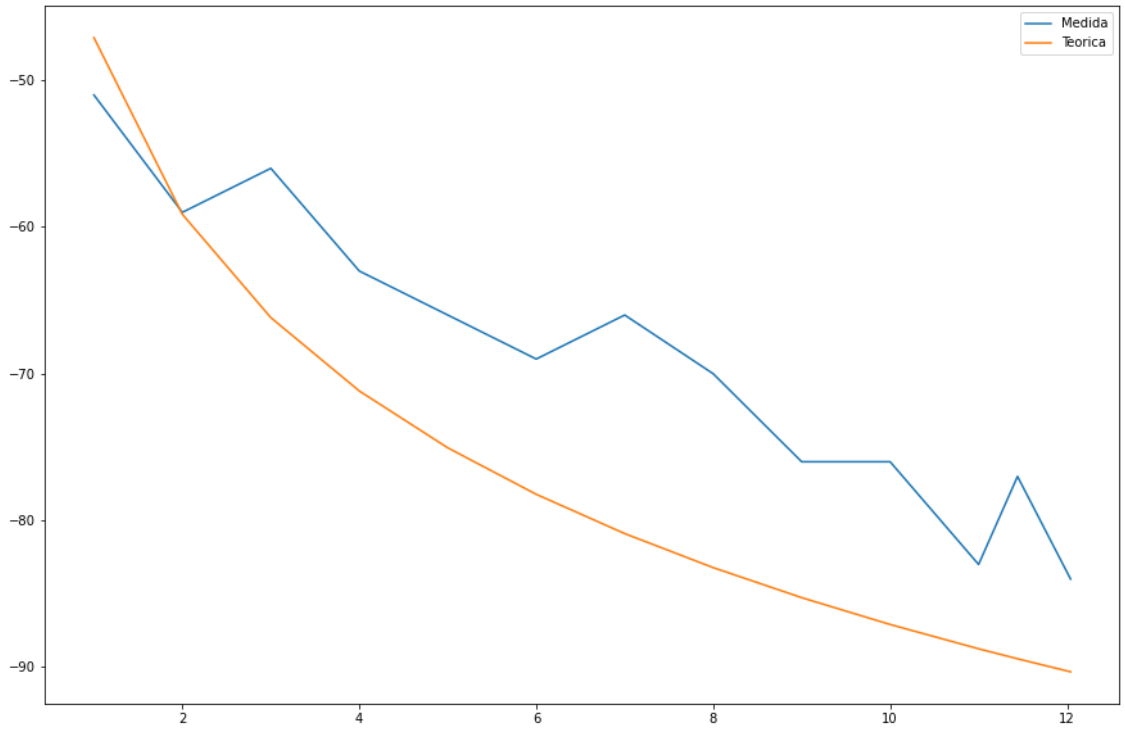


Figura 46. Relación rssi distancia.

Se puede ver que conforme aumenta la distancia el RSSI tiende a disminuir siguiendo por poco la curva teórica del cálculo del RSSI. Para el cálculo teórico del RSSI se ha utilizado el modelo de espacio libre de Friis:

$$P_r = \frac{P_t * G_t * G_r * \lambda^2}{(4\pi)^2 * d^2 * L}$$

Donde tenemos:

- P_r y G_r como potencia y ganancia respectivamente en la antena receptora. Dado que este es un ESP32 la ganancia según la especificación para recepción es de 0 dB
- P_t y G_t siendo respectivamente Potencia y ganancia de la antena emisora, para un ESP32 en transmisión la ganancia es de 3dB y la potencia por defecto es 3 dB
- λ es la longitud de onda, la calculamos con la relación entre Velocidad de la luz ($c = 3*10^8$) y frecuencia, que al estar en Bluetooth y usar la banda de 2.4 GHz ISM asumimos la frecuencia, f es 2.4 GHz
- L, son las pérdidas en el espacio libre, se calculan con la siguiente expresión:

$$L_p = \left(\frac{4 * \pi * D * f}{c}\right)^2$$

Durante toda esta prueba el intervalo de emisión de beacons era de 100 ms, por lo que si se escaneó durante 1 minutos, el beacon ha emitido teóricamente unas 600 tramas en total.

El porcentaje de tramas recibidas según la distancia es:

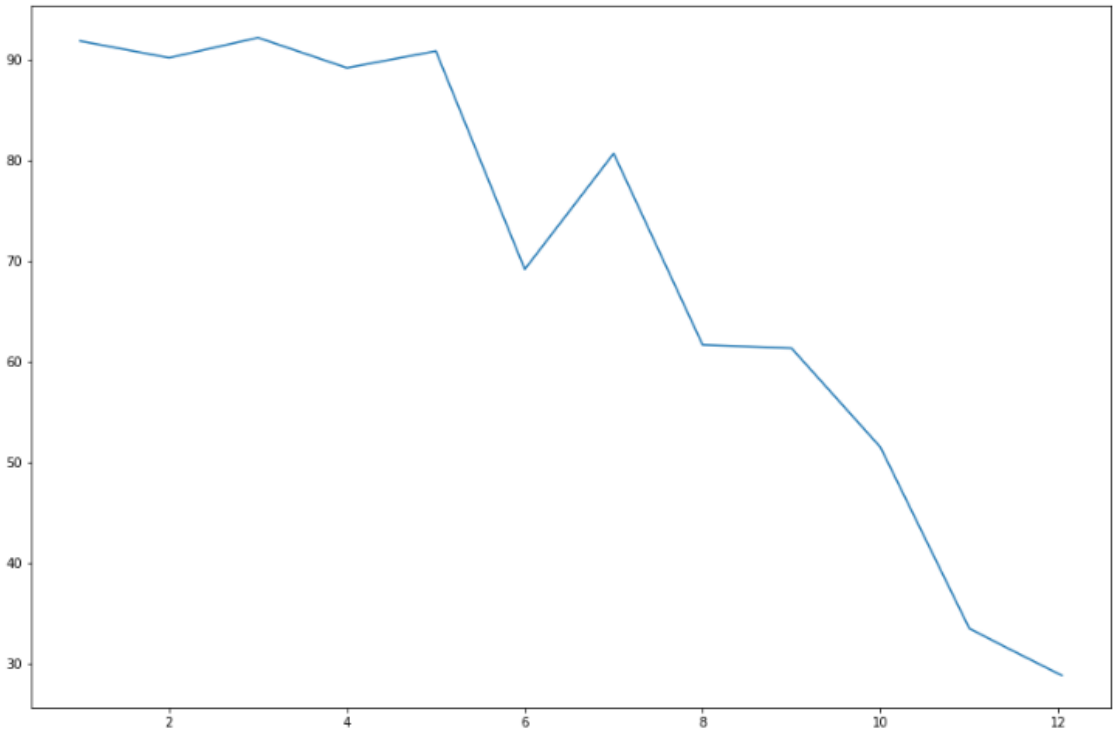


Figura 47. Relación Número de paquetes y RSSI.

Se puede apreciar que conforme la distancia se hace mayor (mayor RSSI) el número de paquetes disminuye.

Para la segunda prueba a la que he sometido el ESP32 era situar el emisor de Beacons justo delante y, cada **diez segundos**, ir alejándome 60 centímetros. Así durante 3 minutos. Logrando una distancia total de 12 metros.

La frecuencia tanto de emisor como de receptor es de 100 ms. Es decir, el escáner escanea el medio cada 100 ms y el emisor de beacons envía cada 100 ms

El resultado ha sido el siguiente:

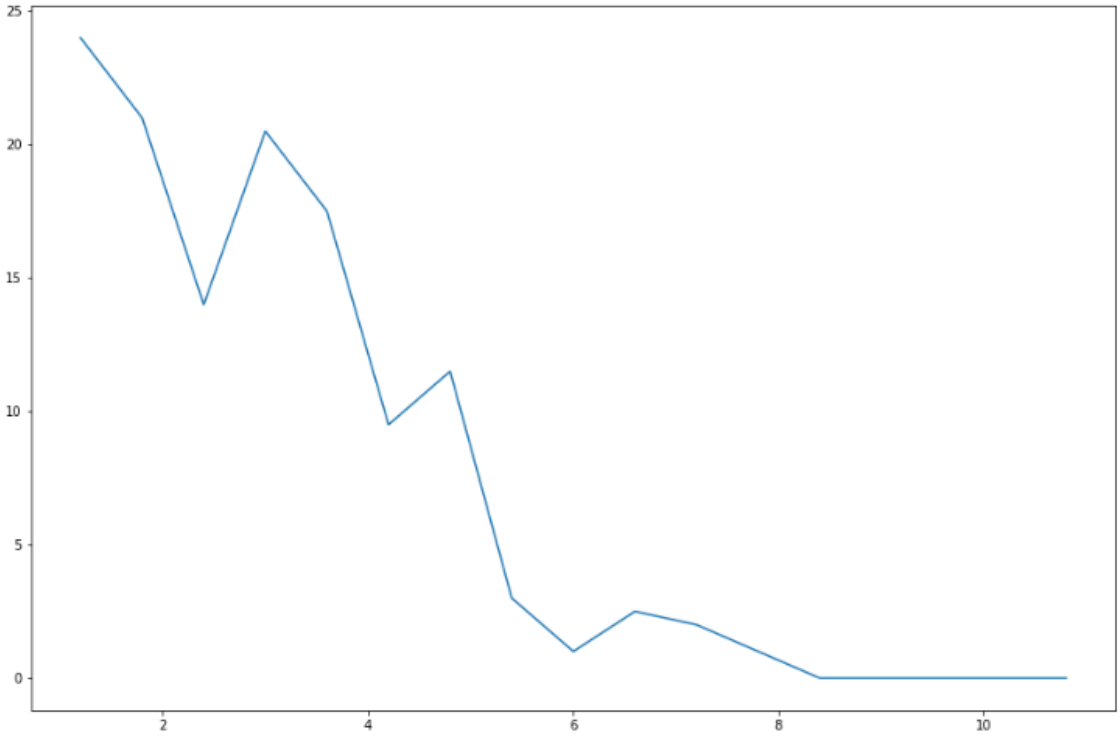


Figura 48. Relación advertisements perdidos módulo beacon ESP32 distancia en el servidor.

En este gráfico se puede apreciar la eficiencia del sistema, la cual ha sido algo irregular, pero se puede apreciar que a mayor distancia la curva va bajando hasta quedarse en 0, distancia a partir de la cual deja de recibirse mensaje alguno del módulo beacon ESP32.

Se puede apreciar que incluso cuando la distancia es menor la eficiencia no supera ni el 25% esto es debido a diversos factores, el principal es que hay otros dispositivos BLE en cobertura que están transmitiendo también advertisements y se pueden ver como interferencias en el canal inalámbrico. Se comprueba que en el entorno de pruebas de una casa puede haber un número variable de dispositivos BLE que emiten con bastante frecuencia.

En concreto de 1059 tramas capturadas durante los tres minutos que ha durado la prueba la gran mayoría provenían de un dispositivo del cual no se tenía control:

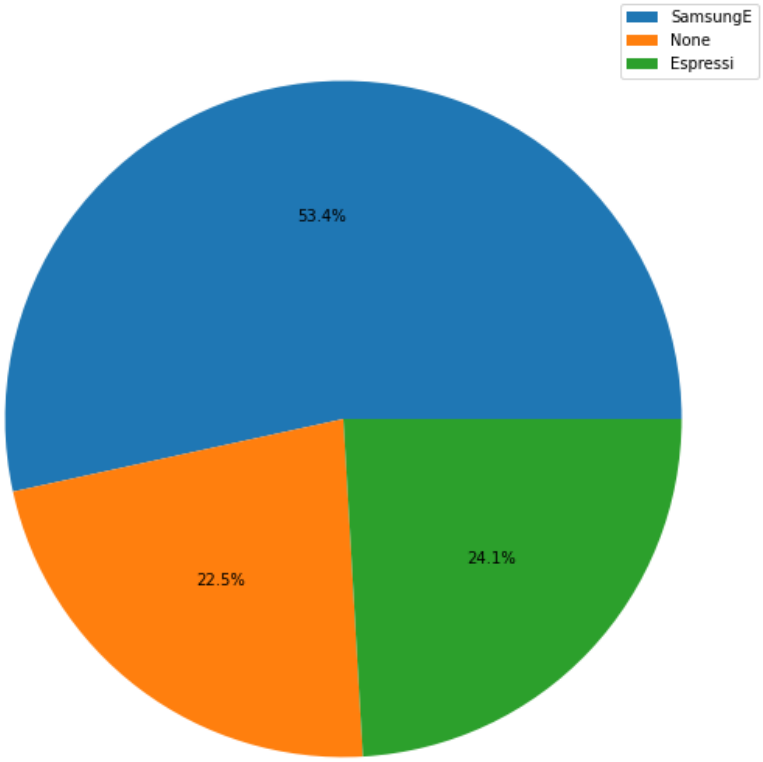


Figura 49. Diagrama porcentaje de tramas según pertenencia a los dispositivos detectados

Este gráfico representa que porcentaje de las tramas pertenecía a los distintos aparatos, se puede apreciar como la gran mayoría de estas era de un Dispositivo Samsung, mientras que casi un cuarto de ellas era de nuestro emisor de Beacons.

Esta apreciación es necesaria ya que el ESP32 es al final un dispositivo con recursos limitados y tanto Bluetooth como WiFi comparten antena, por lo que en el tiempo en el que se hace uso de esta para transmitir una trama anterior, se pueden estar perdiendo advertisements de otros dispositivos BLE que pasan en cobertura del scanner BLE.

Para medir la cobertura real, se ha obtenido también el RSSI o nivel de señal percibido, en la siguiente gráfica se puede ver como este decae con la distancia:

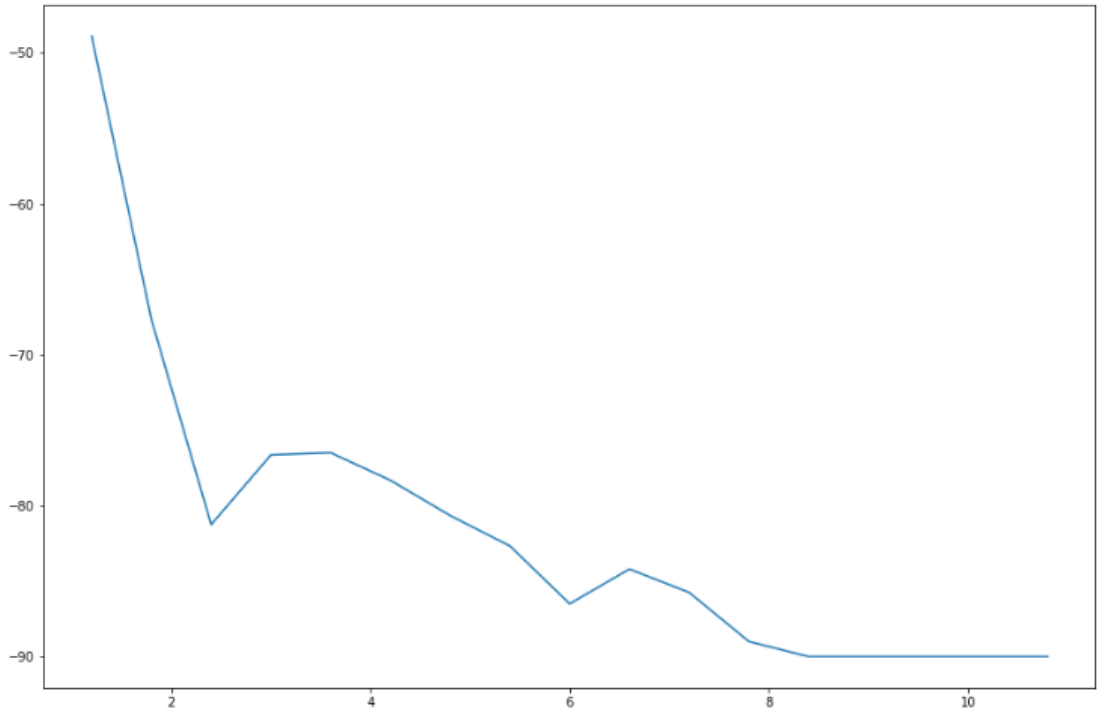


Figura 50. Relación nivel de señal recibida frente a distancia en el servidor.

La sensibilidad nominal del ESP32 según su fabricante Espressif es de -90 dBm, es por ello que superado dicho umbral deja de poder percibirse las tramas del emisor, se puede ver que ambas gráficas coinciden en que cuando dejan de recibirse paquetes es porque la señal está por debajo de la sensibilidad.

La prueba mide como decae la señal sin obstáculos importantes entre ambos ESP32. Se ha medido además como decae la señal con obstáculos como por ejemplo una pared, que puede introducir unas pérdidas de señal de 3-5 dB. Podemos ver en la siguiente gráfica qué tan rápido decae con una pared entre ambos ESP32.

En la gráfica de la figura 50 se compara la cantidad de paquetes que se pierden para una distancia determinada. Libre implica que los ESP32 se pueden ver en todo momento y la pared con la que se ha realizado la medición es una pared de ladrillo de 12 cm de grosor.

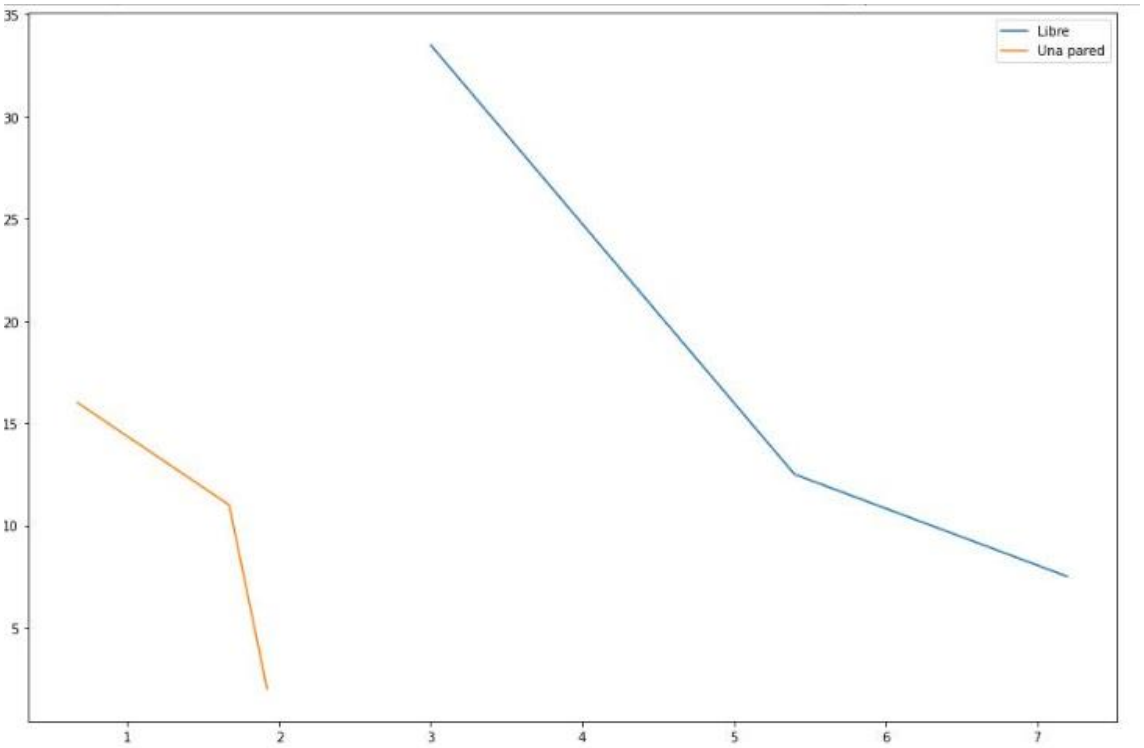


Figura 51. Comparación caída paquetes con pared y sin pared. Eje X distancia. Eje Y % recepción advertisements.

5.2 Setup de pruebas para los escenarios reales

Para la realización de las pruebas en escenarios reales, se han hecho ligeros retoques del diagrama principal:

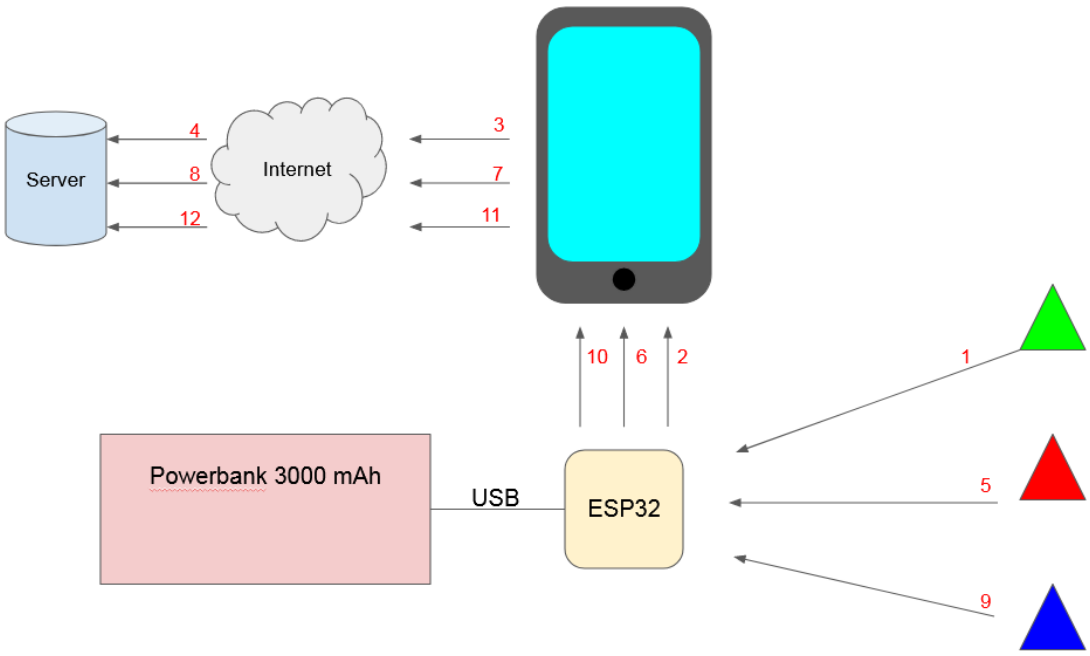


Figura 52. Esquema setup para pruebas reales

El ESP32 pasará a estar conectado a una powerbank de 3000 mAh para tener suministro eléctrico y poder operar, como punto de acceso se usará la conexión compartida de mi propio Smartphone, un Xiaomi Redmi 9, el cuál no tendrá conectada la antena Bluetooth para así evitar interferencias no deseadas.

El programa escáner será modificado para que se conecte a este nuevo punto de acceso. Las tramas seguirán llegando al servidor ya que el ESP32 permite el uso de DNS, es por ello que se configura el router para que sea capaz de redirigir al servidor MQTT en mosquitto, Figura 43. La dirección DNS es la misma para el servidor MQTT como la página, la única diferencia es el puerto al que acceden. Siendo el 80 para el servicio web y el 1883 para el servidor MQTT.

El funcionamiento durante las pruebas será el siguiente, el ESP32 configurado en modo pasivo, alimentado por powerbank y conectado a internet por medio del Smartphone, escuchará el medio en busca de tramas Bluetooth, cuando las capte, las enviará por medio de la conexión compartida al servidor MQTT que se encuentra en el mismo PC que el servidor web.

Para poder analizar los resultados haremos uso de la funcionalidad de la página para descargar ficheros CSV además de un bloc de Python. Para generar un fichero CSV de la fecha que se desee se ha de seleccionar fecha inicial y fecha final usando los selectores vistos en la Figura 40. Hay que especificar también las horas. Hecho esto se debe dar al botón de **Generate CSV!** Para que la página valide la solicitud y desbloquee el enlace al archivo CSV. En mi caso como las tres pruebas se hicieron en la misma mañana me he descargado el fichero entero. Posteriormente en el script de Python se hace un segundo filtrado por tiempo.

A continuación, habiendo descargado el fichero CSV, se requerirá el uso de un bloc de Python3 con las siguientes librerías:

- Pandas
- Matplotlib
- Collections

Se va a describir a continuación el funcionamiento del bloc de Python para su uso.

Al tratarse de un bloc se pueden ejecutar las celdas una a una pero existe una opción para ejecutarlas todas de una vez.

Antes de comenzar la ejecución hay que modificar la primera celda, donde se llaman las librerías a utilizar y desde donde se llaman las variables, la celda es la siguiente:

```
import pandas as pd
from os import path
from matplotlib import pyplot as plt
from collections import Counter


DATA_DIRECTORY = path.join('..','data')

#VARIABLES
filename = 'Dispositivos_BLE_2021-07-29_2021-07-29.csv'
f1 = '2021-07-29 08:20:20:774'
f2 = '2021-07-29 08:47:59:710'
```

Figura 53. Célula configuración

Las variables para modificar son el filename, donde se ha de poner el nombre del archivo con el que trabajar, el cuál **debe** ser guardado en una carpeta llamada data, fuera de la carpeta contenedora del código. Las variables f1 y f2 son para seleccionar el intervalo de tiempo sobre el que se quiere trabajar dentro del archivo.

A continuación, hay que seleccionar la opción “run all”:

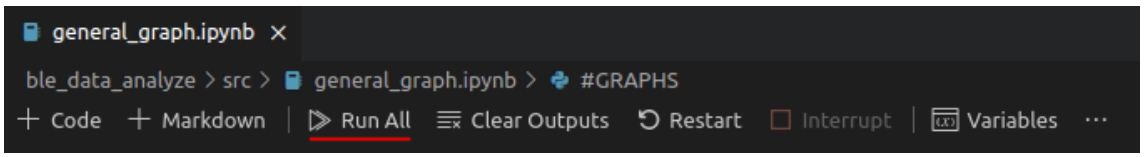


Figura 54. Botones bloc python

Para explicar el resto del código repasaremos el siguiente diagrama:

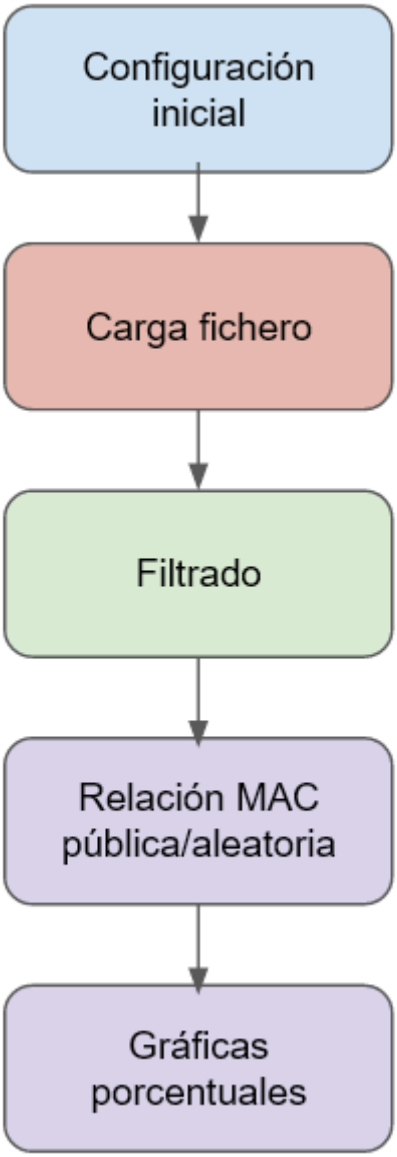


Figura 55. Flujograma script python

La configuración inicial es dada en la primera célula vista en la Figura 52, lo siguiente que se hace es la carga del fichero introducido en la variable *filename*, filtrar con las fechas y una vez hecho esto el fichero está disponible para sacar las gráficas.

En concreto este fichero nos da la siguiente información:

1. Número total de tramas obtenidas en ese periodo.
2. Relación del total de MACs aleatorias y públicas, así como representación gráfica del porcentaje de tramas que tienen MAC pública y privada
3. Relación gráfica de las tramas según pertenencia a los manufacturadores
4. Relación gráfica según pertenencia de direcciones MAC a los manufacturadores

5.3 Demo en la biblioteca de la UPCT

Se ha realizado una prueba de escaneo de tramas BLE en la biblioteca de la universidad, para ello el ESP32 se ha configurado para conectarse a la red de datos de mi smartphone y enviar los datos MQTT utilizando esta conexión. El ESP32 estuvo conectado a una batería portátil durante el tiempo de escaneo, el cuál fue desde las 9:00 hasta las 11:45 tiempo durante el cual solo hubo un único reinicio rápido de la conexión del smartphone.

Durante ese tiempo se hallaron un total de 40489 tramas, estas provenientes de 80 dispositivos diferentes.

La mayoría de los dispositivos, el 97.5% de los encontrados tenían una MAC anónima, es decir, que los primeros Bytes no contienen el OID de la empresa manufacturera, el porcentaje restante son MACs pertenecientes a las empresas 1More y Texas Instrument. En el siguiente gráfico se puede ver qué porcentaje de las **tramas** no anónimas y a qué manufacturador corresponden

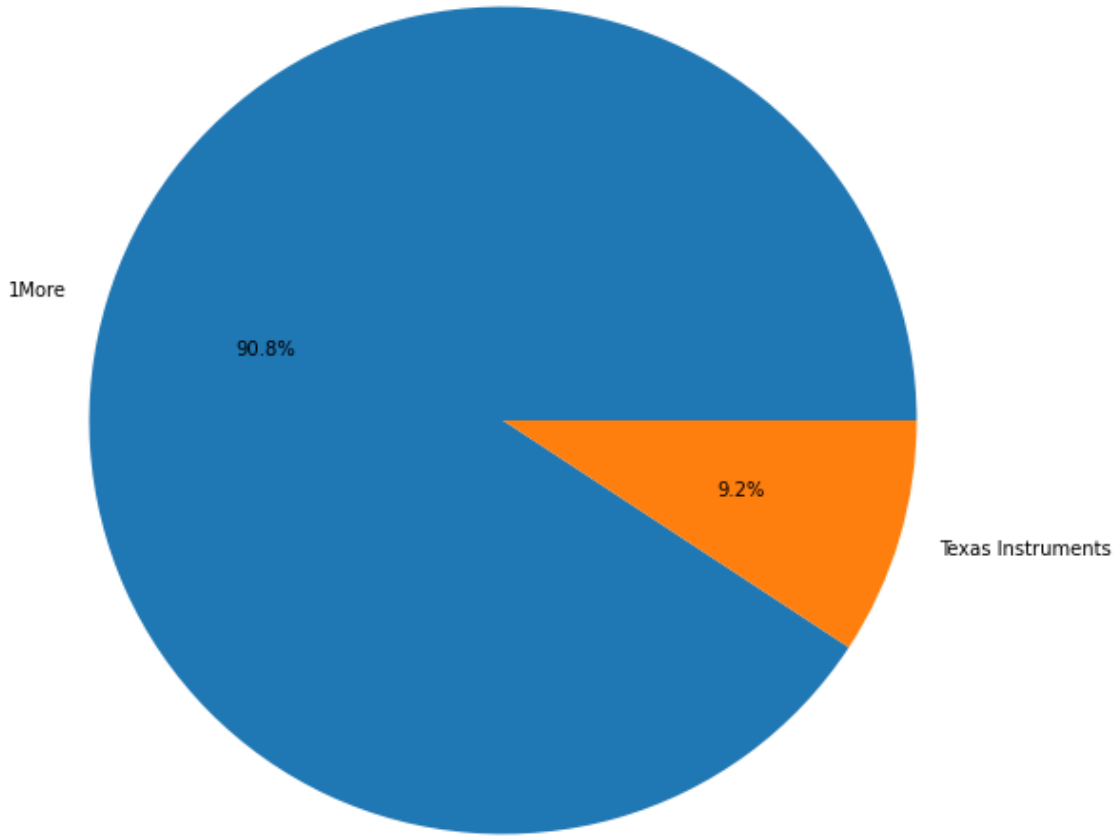


Figura 56. Porcentaje de tramas de cada fabricante.

5.4 Demo en supermercado

Poner escáner con powerbank y compartiendo datos con el móvil.

Para esta prueba se ha llevado el escáner a un supermercado, durante un tiempo de 24 minutos, tiempo en el cual el escáner ha encontrado 5224 tramas, provenientes de 106 direcciones físicas distintas.

De las 106 MACs encontradas 88 de ellas son anónimas y el resto públicas, es decir, podemos saber cual es el fabricante de estas.

En el siguiente gráfico podemos ver que MACs pertenecían a qué fabricante:

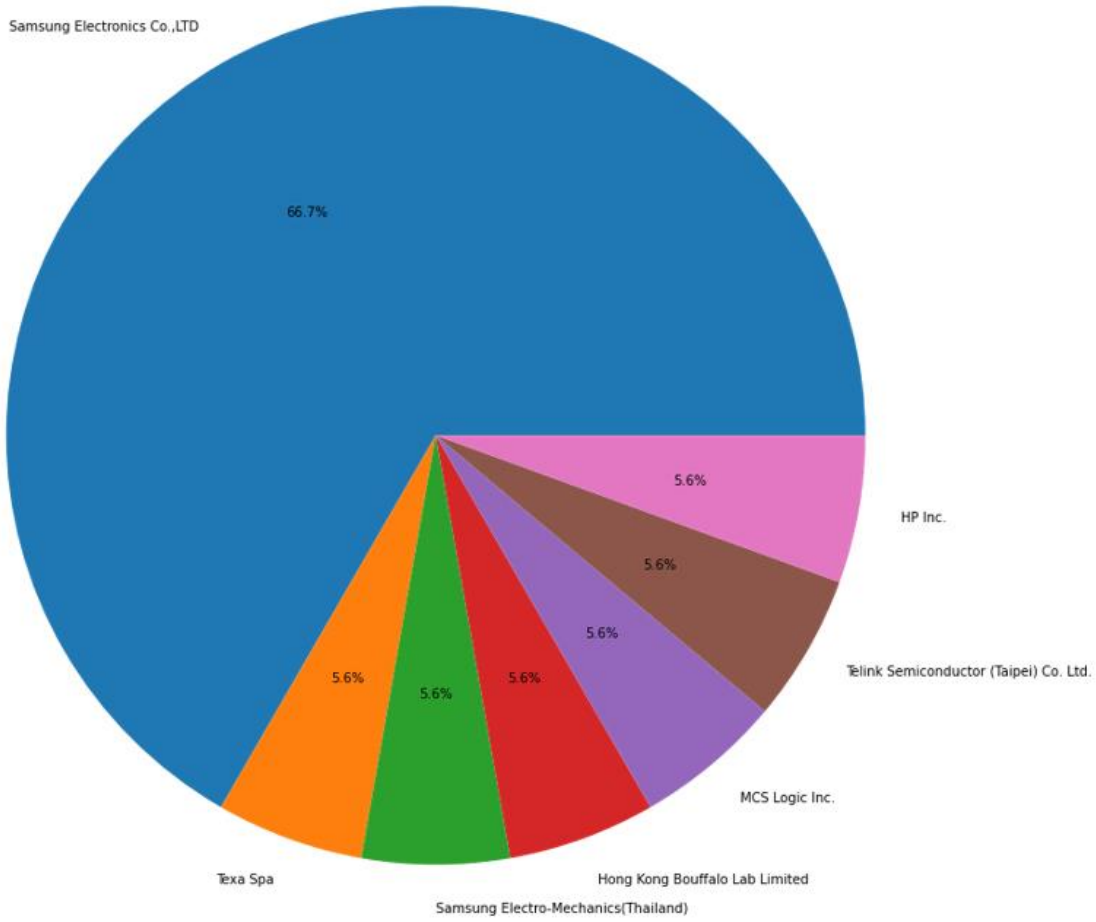


Figura 57. Porcentaje de MACs según pertenencia a fabricante

Se puede apreciar que hay una mayor cantidad de MACs de la empresa Samsung, la cual posee un 72.3% del total de las MACs contando con su subsidiaria, Samsung Electro-Mechanics, esto son 13 de las 18 MACs públicas.

5.5 Demo en Autobus

Se ha utilizado el escáner a bordo de un autobús de línea al que durante el trayecto se van subiendo hasta 23 personas, el trayecto duró 23 minutos y durante ese tiempo se detectó un total de 3307 tramas, procedentes de 197 direcciones físicas (MAC) distintas.

De estas MACs 132 son MACs aleatorias y el resto, las 65 restantes pertenecen a las siguientes empresas:

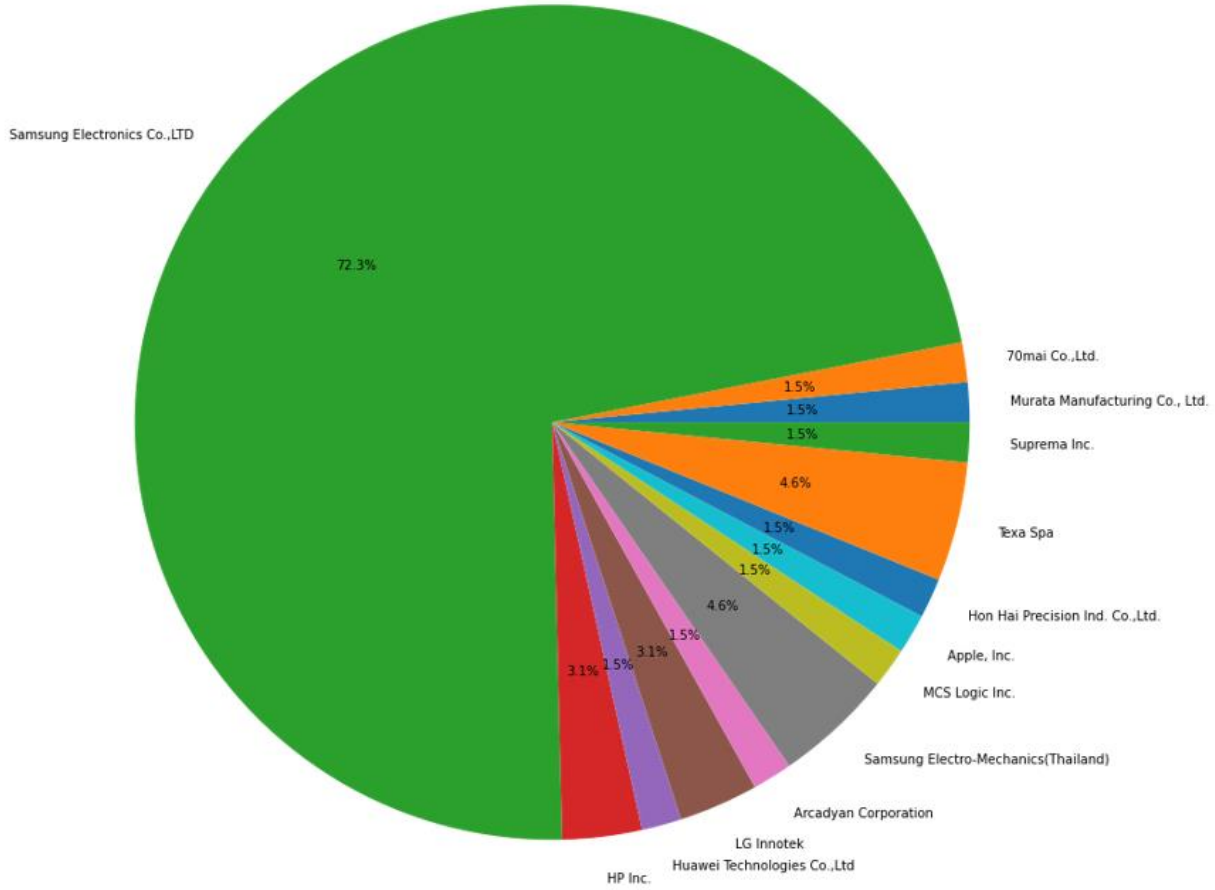


Figura 58. Porcentaje de MACs que pertenecen a los distintos fabricantes

El gráfico muestra que porcentajes de las MACs no aleatorias eran de qué empresas.

Capítulo 6. Conclusiones y trabajos futuros

6.1 Conclusiones

Para este trabajo se ha utilizado en gran medida un *System-on-chip*, el ESP32. Este módulo tiene como ventaja el gran rango de aplicaciones que puede cubrir a un precio bastante asequible, pero por supuesto tiene una desventaja y esta es los recursos que maneja, que son más limitados que en un PC de sobremesa.

Gracias al ESP32 he podido profundizar más allá de los conceptos aprendidos en la carrera sobre la arquitectura de Bluetooth, así como poder hacer prototipos que funcionan basándose en este.

Hemos podido estudiar de cerca la cantidad de aparatos que hay transmitiendo en un medio cualquiera, se puede apreciar con la cantidad de dispositivos que hay que tan utilizada es la arquitectura, pues no solo se usa para poder conectarse a un smartwatch o al manos-libres del coche, sino que en esta arquitectura los dispositivos pueden intercambiarse mensajes, un ejemplo es la aplicación radar Covid, que envía tramas para que otros dispositivos las detecten y las almacenen, para así poder llevar control de los contactos en caso de contagio.

Bluetooth Low Energy es por ende una arquitectura que permite un gran número de aplicaciones sin necesidad de cables es por ello por lo que se adaptó para bajo consumo, bajo el nombre de Bluetooth Low Energy, para así poder permitir a los dispositivos funcionar por más tiempo.

Este primer prototipo es funcional y completo y ha permitido entender con profundidad las tecnologías en uso. La limitación del dispositivo scanner BLE es que la radio Bluetooth Low Energy y la radio Wifi comparten antena y no pueden funcionar simultáneamente.

6.2 Grado de consecución de los objetivos y formación adquirida

Los objetivos que se han logrado en este trabajo han sido los siguientes:

1. Aprendizaje del entorno de desarrollo ESP32 e implementación de una aplicación IoT .
2. Formación en la tecnología Bluetooth Low Energy.
3. Diseño de un programa para el ESP32 basado en Bluetooth y WiFi con capacidad para escanear el medio en busca de tramas Beacon y enviar los datos encontrados utilizando el protocolo MQTT.
4. Diseño de un programa para el ESP32, basado en Bluetooth que envíe tramas Beacon específicas al medio.
5. Diseño de un Back-end y Front-end en NodeJS para consulta y obtención de los datos obtenidos por el escáner previamente.

6. Realización de pruebas de rendimiento y cobertura del sistema en su conjunto.

A lo largo de los anteriores apartados se ha ido narrando la forma en la que estos objetivos han sido completados.

Curva de aprendizaje y principales dificultades encontradas

El punto que mayor dificultad implicaba en este trabajo ha sido el aprendizaje y utilización del lenguaje C para el uso de las librerías del ESP32, pues este lenguaje no cuenta con las características que otros lenguajes modernos como Python o Javascript si poseen.

Para este trabajo también ha sido importante entender la documentación de las librerías del ESP32, documentación que está disponible de forma oficial y en ocasiones cuenta con códigos dados en forma de ejemplo por parte de Espressif, la empresa creadora del ESP32.

Reflexión sobre la formación adquirida en conocimientos, metodología y competencias

Para la realización de este trabajo ha sido necesario aprender un poco más acerca de cómo es la arquitectura de Bluetooth en concreto, como funcionan las tramas BLE, los bytes que estas contienen.

Además de aprender a utilizar el ESP32 y el lenguaje sobre el que este funciona, C. Para poder programar el ESP32 ha sido necesario familiarizarme con las librerías de Espressif, como se utilizan, los métodos que estas tienen…

Por último y no menos importante, está el desarrollo de un back-end en NodeJS, utilizando conocimientos obtenidos en dos de las últimas asignaturas del grado cursado y adquiriendo otros tantos en el camino.

6.3 Trabajos futuros

Los principales trabajos futuros serían:

- Mejorar el desempeño del scanner BLE: hacer una cola de recepción de mensajes de eventos de advertisment y reducir el uso de la Wifi. Como la radio es compartida, si se reduce el uso de la tarjeta Wifi se podría tener más tiempo la radio BLE en modo SCAN y posiblemente obtener más advertisment de dispositivos en cobertura.
- Implementar una versión cloud del server, por ejemplo usando Azure o AWS, para no tener que realizar el mantenimiento del hardware del server y mejorar la formación en las tecnologías cloud.

Bibliografía y referencias

[1] 232 Analyzer sniffer Puerto serie: https://www.commfront.com/

[2] NodeJS. Javascript framework: https://nodejs.org/en/

[3] BLE. Bluetooth Low Energy: https://www.bluetooth.com/specifications/specs/core-specification/

[4] MQTT: The Standard for IoT Messaging https://mqtt.org/

[5] Bluetooth GAP: https://btprodspecificationrefs.blob.core.windows.net/assigned-numbers/Assigned%20Number%20Types/Generic%20Access%20Profile.pdf

[6] Apple iBeacon: https://developer.apple.com/ibeacon/

[7] Google Eddystone: https://github.com/google/eddystone

[8] Exposure frame: https://blog.google/documents/70/Exposure_Notification_-_Bluetooth_Specification_v1.2.2.pdf

[9] Texas instrument packet sniffer: https://www.ti.com/tool/PACKET-SNIFFER

[10]nRFConnect: https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=es&gl=US

[11] ESP32 documentation Espressif: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/

[12] Librería Bluetooth ESP32: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/esp_gap_ble.html

[13] Librería MQTT ESP32: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/mqtt.html

[14] Librería WiFi ESP32: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html

[15] librería cJSON: https://github.com/DaveGamble/cJSON

[16] Mosquitto: https://mosquitto.org/

Anexos

Anexo 1. Estructura y organización del código

El código tanto el del servidor como el del escáner BLE se encuentran subidos a github: https://github.com/digio-upct/MQTT_Back-end_Server

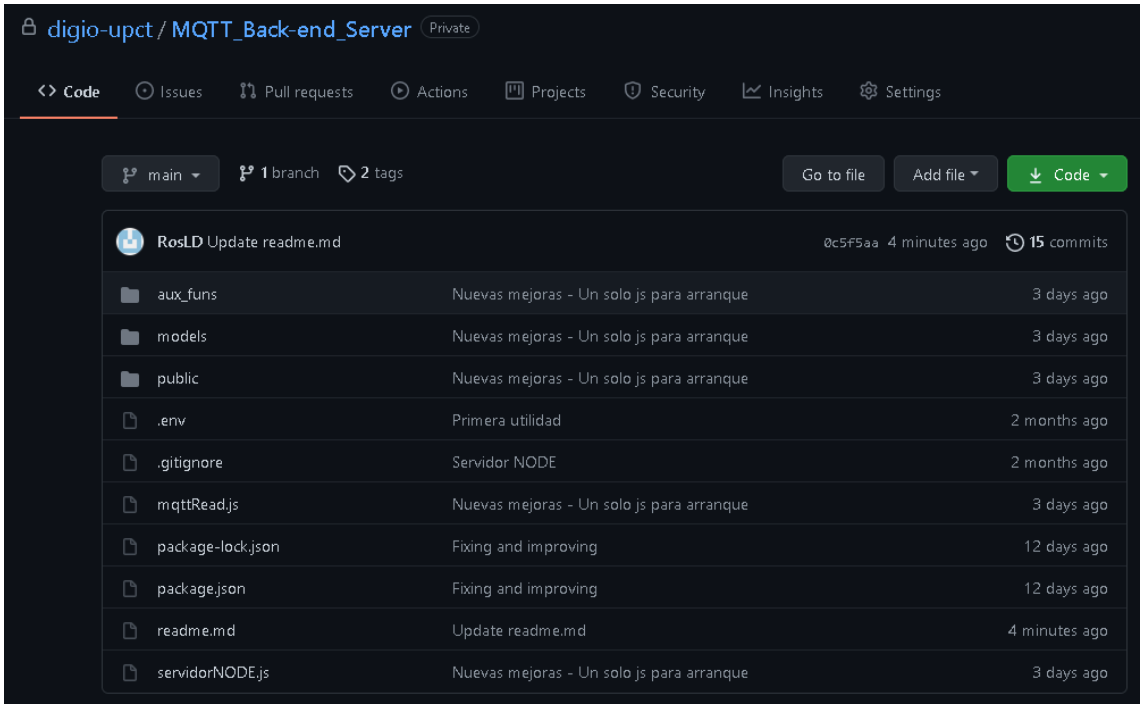


Figura 59. Captura del directorio github Back-end

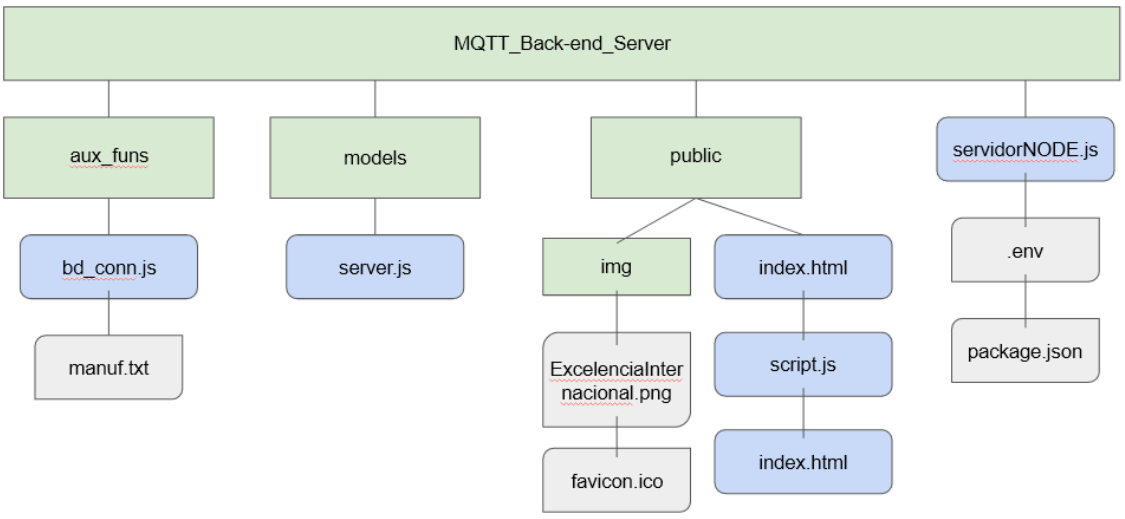


Figura 60. Árbol de directorios del Back-end

En la figura 56 se ha representado el árbol con las carpetas y subcarpetas así como los códigos que conforman el servidor del Back-end.

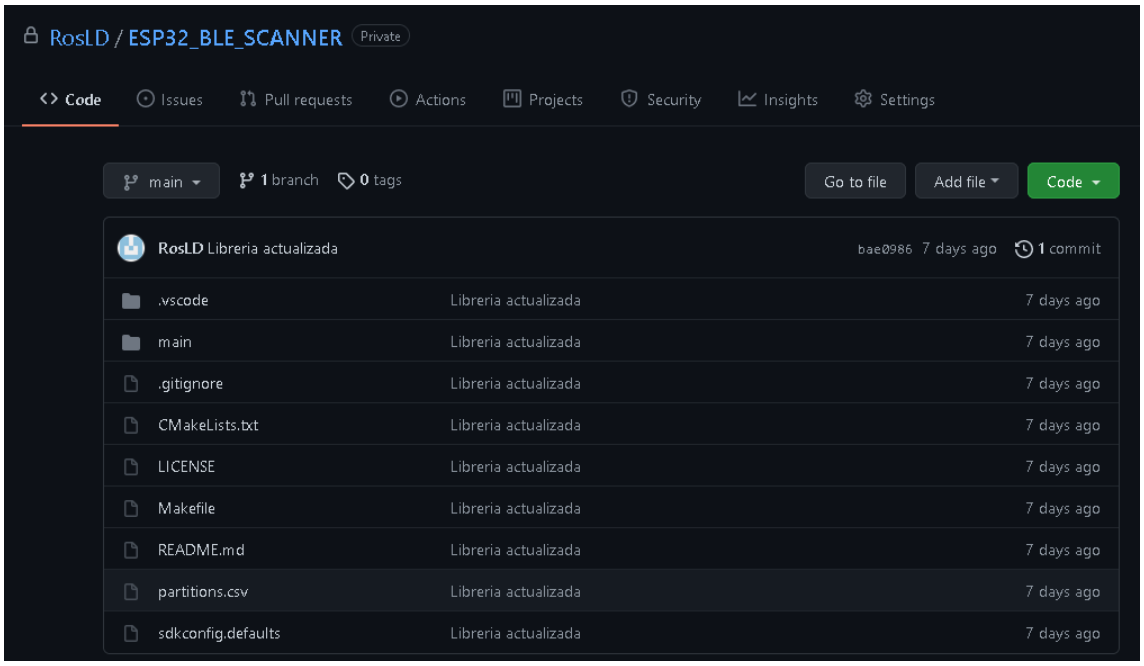


Figura 61. Captura del directorio github del Scanner BLE