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 un gateway IoT edge-computing basado en Raspberry Pi

AUTORA: Alba Martínez Meroño

DIRECTOR: Alejandro S. Martínez Sala

Septiembre / 2021

Autor	Alba Martínez Meroño
E-mail del autor	alba_bvb@hotmail.com
Director	Alejando S. Martínez Sala
E-mail del director	alejandros.martinez@upct.es
Título del TFM	Diseño y desarrollo de un Gateway IoT edge-computing basado en Raspberry Pi
Resumen	Existen numerosas aplicaciones de IoT donde los gateways recopilan los datos en bruto de sensores y los envían a una nube para su procesamiento. Una alternativa consiste en hacer gran parte del procesamiento en el propio gateway y minimizar el envío de datos y la carga computacional en la nube; esta estrategia se denomina edge-computing y puede lograr un mayor ahorro de energía y reducción del ancho de banda empleado en las comunicaciones. El objetivo del proyecto es diseñar e implementar un gateway edge-computing basado en Raspberry Pi usando Python y tecnologías estándar; el gateway edge-computing recopila datos de numerosos sensores IoT y hace un preprocesamiento y tratamiento de dichos datos minimizando el envío de datos a una nube. Se propone un caso de uso basado en sensores IoT de temperatura y humedad usando comunicación Bluetooth Low Energy.
Titulación	Grado en Ingeniería de Telemática.
Departamento	Tecnología de la Información y las Comunicaciones.
Fecha de presentación	Septiembre - 2021

Agradecimientos

Ante todo, agradecerles a mis padres el haber llegado aquí, su sacrificio y el apoyo que me han dado durante estos años, tanto en la carrera como en el instituto, pues nunca han dudado sobre mí y han estado siempre que los he necesitado.

A mis amigos, los cuales me han animado dándome consejos pese a cualquier adversidad y animándome a salir para despejarme.

A mis compañeros de la carrera, con los que he compartido la gran experiencia de realizar un grado el cual veía muy difícil nada más entrar, pero todos hemos conseguido lo que nos propusimos ayudándonos mutuamente. En especial mencionar a Daniel Ros, mi compañero de prácticas durante la mayor parte de asignaturas, que siempre ha estado ahí para ayudarme. También mencionar a mi compañero Jorge Gálvez, pues gracias a él he podido terminar esta carrera con ganas de seguir aprendiendo.

Índice

Índice de figuras

Índice de tablas

Índice de gráficas

Capítulo 1. Introducción

1.1 Antecedentes y objetivos

Actualmente hay varios sistemas y aplicaciones de IoT donde los Gateways recopilan datos en bruto de sensores y los envían a la nube para su procesamiento -recolectan información pero no hacen nada con ella, la envían a la nube donde grandes centros de datos la procesan para obtener ciertas conclusiones-. Una alternativa se basa en realizar gran parte del procesamiento en el propio Gateway para así minimizar el envío de datos y la carga computacional en la nube. Este planteamiento se denomina edge-computing y con él es posible lograr un mayor ahorro de energía y reducción del ancho de banda empleado en comunicaciones.

El objetivo de este proyecto es diseñar e implementar un Gateway edge-computing basado en **Raspberry Pi** desarrollado en el lenguaje de programación Python [6] y tecnologías estándar.

El Gateway edge-computing recopila datos de numerosos sensores IoT y hace un preprocesamiento y tratamiento de dichos datos minimizando el envío de datos a la nube, para ello se plantea el caso de uso centrado en sensores IoT de temperatura y humedad usando comunicación **Bluetooth Low Energy** -BLE- [3].

La principal tarea es implementar un escáner Bluetooth en la Raspberry Pi: cada sensor está conectado a un módulo ESP32 que encapsula los datos y los envía por Bluetooth como un mensaje advertisement al Gateway. Éste los procesa y envía mediante protocolo MQTT [2] a un servidor que los almacena en una base de datos para su posterior visualización en un portal web.

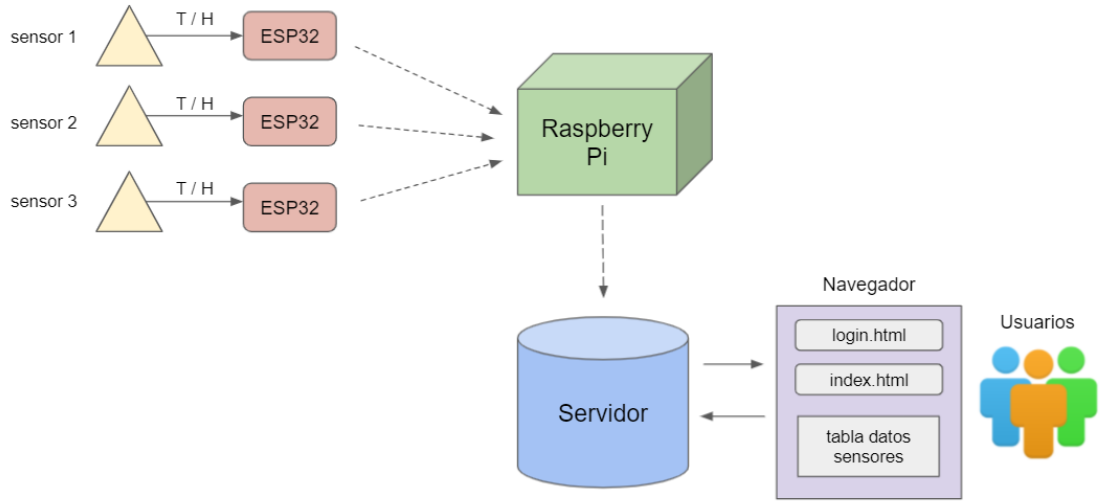


Figura 1. Esquema simplificado arquitectura Gateway IoT de sensores ESP32 y server de almacenamiento y visualización de datos

1.2 Herramientas software y elementos usados

Para el desarrollo de este proyecto se han utilizado los siguientes programas, entornos de desarrollo y librerías:

- App nRF Connect [5]. Aplicación para Smartphone que permite escanear y explorar dispositivos Bluetooth Low Energy y comunicarse con ellos.
- BlueZ [7]. Librería que permite comunicarse con el chip Bluetooth desde Linux.
- DB Browser for SQLite [14]. Herramienta de código abierto, visual y de alta calidad para crear, diseñar y editar archivos de base de datos compatibles con SQLite.
- GIT. Sistema de control de versiones distribuido de código abierto y gratuito diseñado para manejar proyectos con velocidad y eficiencia.
- Hcitool. Herramienta conjunta de BlueZ, puede utilizar la utilidad para buscar dispositivos y enviar comandos/datos para Bluetooth y BLE. Se comunica a través de un puerto hci común a sus dispositivos bluetooth.
- MQTTX v1.5.2. Aplicación de escritorio que se conecta a un servidor MQTT y permite enviar tramas a los topics que se quiera.
- Mosquitto v2.0.11 [13]. Programa que permite iniciar un servidor MQTT y monitorizar las tramas que aparecen.
- NodeJS 14.17.3 LTS [4]. Framework del lenguaje de programación Javascript especializado en la creación de backends. Utilizado para crear el servidor que escucha las tramas MQTT y guardar los datos en una Base de Datos para su posterior uso, además del diseño del frontend. Librerías y dependencias utilizadas:
    - Body-parser v1.19.0
    - Bootstrap v5.1.0
    - Connect-flash v0.1.1
    - Cookie-parser v1.4.5
    - Crypto v1.0.1
    - Csv-express v1.2.2
    - Express v4.17.1
    - Express-session v1.17.2
    - Mqtt v4.2.8
    - Passport v0.4.1
    - Passport-local v1.0.0
    - Sesión v0.1.0
    - Sqlite3 v5.0.2
- Npm. Sistema de gestión de paquetes por defecto para Node.js, un entorno de ejecución para JavaScript.

- Pip. Sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python.
- Postman 8.6.2. Software para realizar peticiones al servidor.
- Python [6]. Lenguaje utilizado para el desarrollo del código de escaneo en Raspberry Pi.
- Raspberry Pi. Placa de microordenador de pequeñas dimensiones, utilizado para el desarrollo del proyecto.
- Raspbian 10. Sistema operativo que utiliza la Raspberry Pi.
- Visual Studio Code. Editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. Este es el entorno de desarrollo en el que se realizan los programas.
- Windows 10. Sistema operativo de Microsoft, utilizado para la mayoría de programas.

1.3 Estructura y organización del proyecto

A lo largo de este proyecto vamos a estudiar diferentes conceptos, protocolos, tecnologías y software que nos permitan entender mejor el escáner Bluetooth Low Energy [3], así como el uso de Python [6] y Raspberry Pi. El proyecto se divide en varias fases:

- Formación en el entorno de desarrollo de Raspberry Pi e implementación de programas ejecutables en Python.
- Estudios y pruebas de Bluetooth Low Energy para comunicación con sensores IoT.
- Estudio y pruebas de envío de información a través del protocolo MQTT [2].
- Diseño de la arquitectura del Gateway IoT y procesamiento edge-computing de los datos de los sensores.
- Diseño de una arquitectura servidor que recibe los datos enviados por la Raspberry Pi y se visualizan en una página web.
- Implementación de un caso de uso.
- Depuración y pruebas de la solución.
- Documentación.

Capítulo 2. Tecnologías empleadas

2.1 Conceptos básicos de Bluetooth Low Energy

Bluetooth Low Energy [3] es el nuevo estándar Bluetooth 4.0 -versiones 4.1 y 4.2- y la reciente versión Bluetooth 5. Se especifica un mecanismo sencillo de transmisión de beacons para que dispositivos muy simples sean detectados y localizados.

Este nuevo estándar opera en la banda ISM de 2.4GHz, su principal característica es el coste energético notablemente reducido que emplea respecto a su predecesor, ya que solamente se transmite durante una ventana de tiempo muy reducida.

El mecanismo de advertising de beacons es la base para servicios de localización, que tiene como fin determinar la presencia de otro dispositivo. También se define el scanner de tramas beacon como un sniffer que recibe los mensajes de los beacons BLE en cobertura. Los vemos a continuación.

2.1.1 Advertising de Beacons

Advertising viene del habla inglesa, cuyo significado es promoción al medio, mientras que beacon significa baliza. La finalidad del advertising de beacons es la promoción de un dispositivo o servicio por medio de tramas beacon empleando la arquitectura BLE.

Beacon también se usa en el estándar Wi-Fi, donde sus tramas son las que se encargan de promocionar al medio el SSID para el punto de acceso. En Bluetooth es muy similar, pues se promociona en el medio un dispositivo para que el usuario pueda escanearlo y conectarse a él mediante Bluetooth.

En el medio físico se separa el espectro de 2.4GHz en 40 canales, de los cuales se utilizan exclusivamente para el advertising de beacons 3 canales: 37,38 y 39. Nos centraremos en ellos.

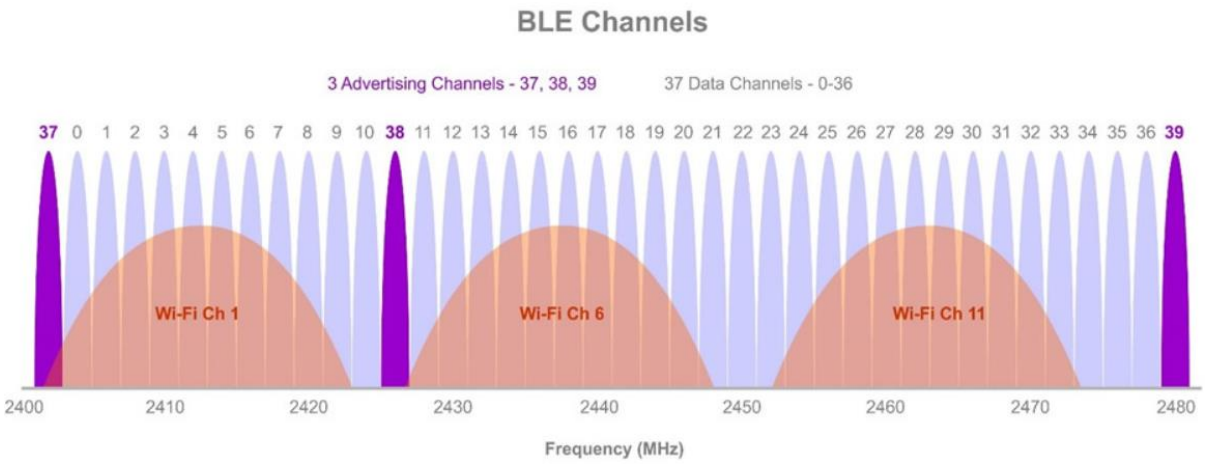


Figura 2. Distinción canales banda 2.4GHz

El formato de paquetes beacon [10] se muestra a continuación:

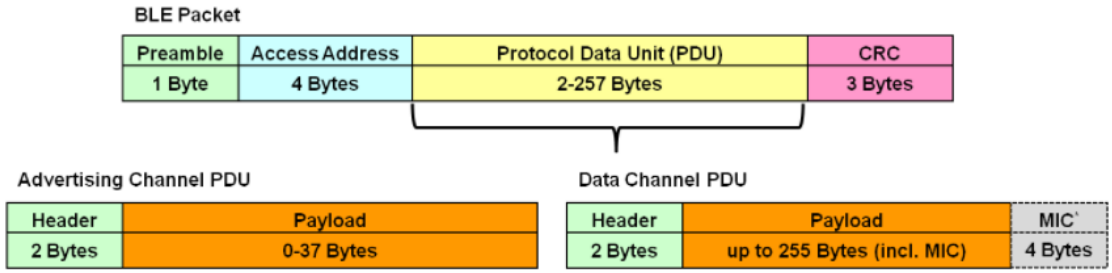


Figura 3. Formato de paquetes Beacon

- Preámbulo: lo utiliza el receptor para la sincronización -tiempo, frecuencia- y para realizar AGC -Control automático de ganancia-. Es un patrón predefinido de 1 byte de tamaño conocido por el receptor. Tanto el paquete advertising como el de datos utilizan los bits "10101010" en binario.
- Access Address: para todos los paquetes advertisement se utiliza el patrón fijo "0x8E89BED6" en forma hexadecimal con un tamaño de 4 octetos o 32 bits. Para los paquetes de datos, consta de un valor aleatorio de 32 bits generado por el dispositivo BLE en "estado de inicio". El mismo valor se utiliza en un mensaje de solicitud de conexión -CONNECT_REQ-.
- CRC: código de redundancia cíclica. Tiene un tamaño de 24 bits. Se calcula sobre PDU. Se utiliza para la detección de errores del paquete. El CRC se calcula utilizando un polinomio de la forma $x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x + 1$.
- PDU: formado por Advertising Channel PDU o Data Channel PDU como se muestra en la figura.

El uso del advertising channel es el siguiente:

- Device Discovery. Funcionalidad mínima de detectar y monitorizar advertisements sin asociación ni intercambio de datos.

- Connection Establishment. Dispositivos que se tienen que descubrir y enlazar para intercambio de datos.

- Broadcast Transmissions. Por ejemplo, un sensor no conectable envía datos periódicamente dentro del payload.

El uso del data channel es para la comunicación bidireccional entre dispositivos conectados.

2.1.2 Scanner de tramas Beacon

De igual manera que un dispositivo BLE se puede promocionar al medio, también se puede escanear para buscar esas tramas. Este escaneo puede ser de dos formas:

- Escaneo pasivo. No se envían tramas, el escáner simplemente se mantiene escuchando el medio para obtener los tipos de tramas que envían los dispositivos.

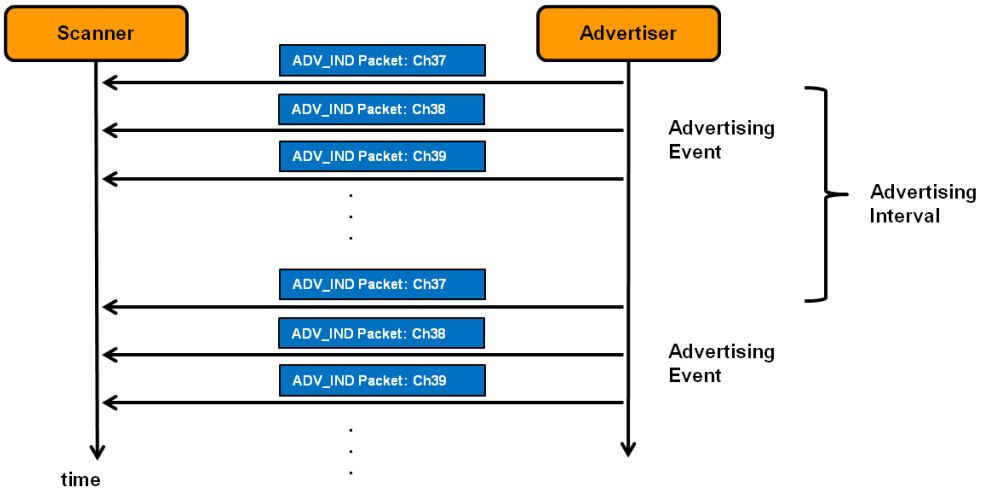


Figura 4. Cronograma escaneo pasivo

- Escaneo activo. El escáner escucha el medio para detectar dispositivos y posteriormente envía tramas SCAN_REQ a todos ellos. Estos responden con tramas SCAN_RSP.

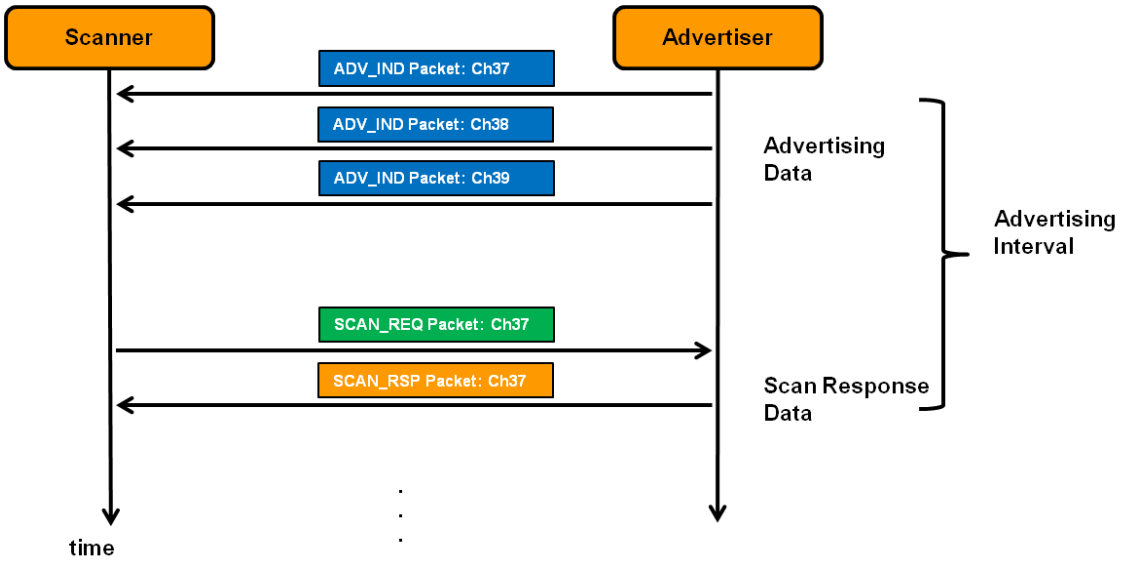


Figura 5. Cronograma escaneo activo

En la siguiente figura se muestra el intervalo de promoción y el intervalo de escaneo de las tramas:

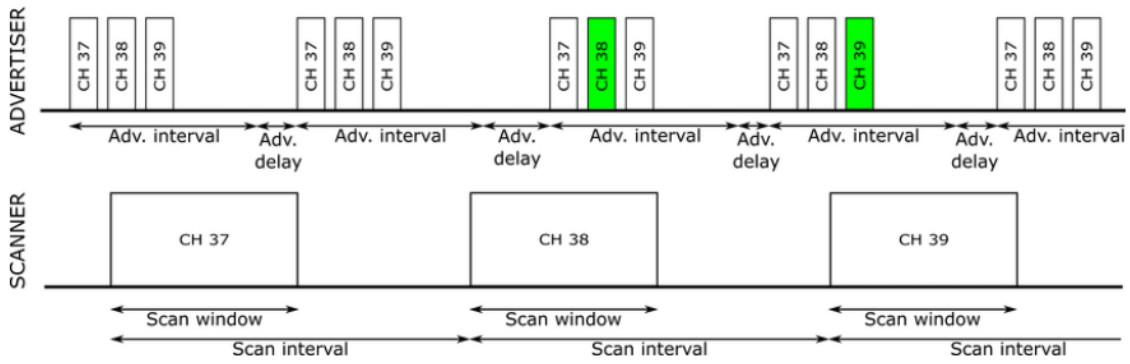


Figura 6. Advertising interval y Scan interval

Como podemos observar, el intervalo de escaneo viene parametrizado, de modo que el escáner comprueba periódicamente los canales en busca de tramas advertising en un tiempo determinado.

Funcionamiento beacon -advertiser-:

- Evento transmisión: se transmite una ráfaga de mensajes advertisment en cada canal.
- Parámetro Adv. interval: cada cuánto tiempo se transmiten mensajes advertisment. Mínimo 100ms.
- Adv. delay: retardo aleatorio entre 0-10mseg definido en estándar para reducir probabilidad de colisión.

Funcionamiento scanner:

- Escanea cíclicamente en los 3 canales.
- Scan window: tiempo de escucha en un canal.
- Scan Interval: periodo para cambiar de canal ADV. Tiempo dentro del intervalo de escaneo en el que está escuchando. T = N*0.625 ms, siendo N el valor introducido.
- Para localización: Scan interval = Scan window -en cuanto se acaba Scan Window en un canal se pasa al siguiente canal-.
- Ciclo de escaneo: tiempo necesario para escuchar los 3 canales. 3*Scan interval.

A modo de ejemplo, se explicará cómo utilizar la aplicación para Smartphone **nRF Connect** [5] que nos proporciona información acerca de los dispositivos BLE a su alcance:

- Se debe activar la ubicación y el bluetooth del dispositivo para comenzar el escaneo.
- Una vez pulsado el botón Scan, empezarán a aparecer diversos dispositivos BLE.

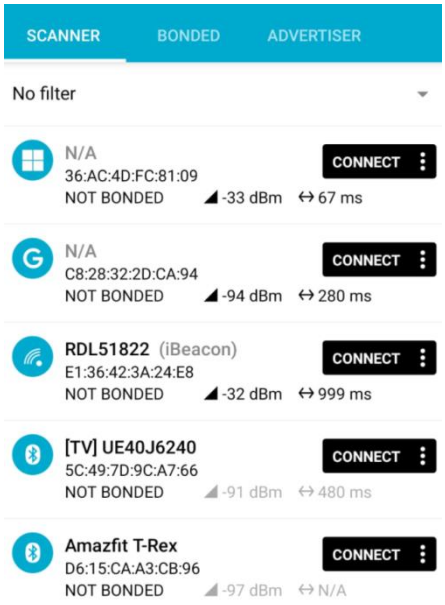


Figura 7. Captura menú principal aplicación nRF Connect

Observamos los distintos dispositivos detectados por la aplicación en su menú principal. Si pinchamos sobre RDL51822 obtenemos la siguiente información:

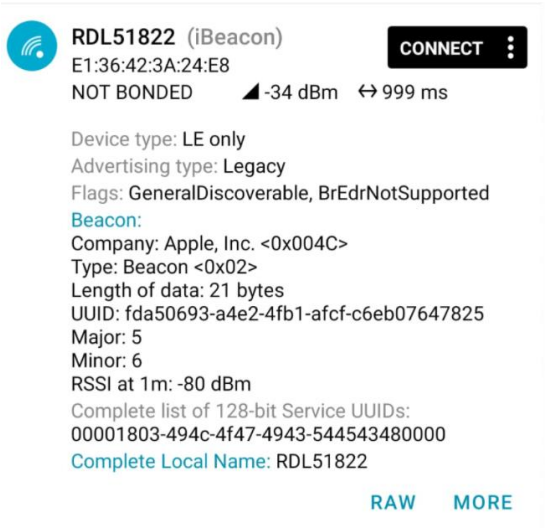
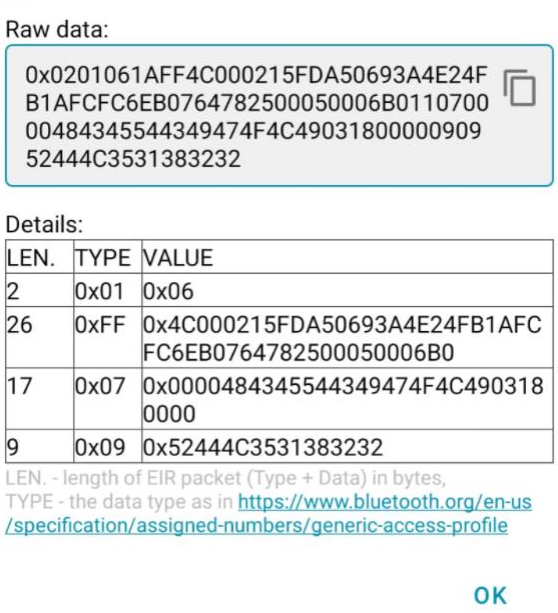


Figura 8. Captura beacon aplicación nRF Connect

Si pinchamos una vez más sobre el botón RAW aparecerá el desglose de los campos con la información más detallada, con el formato Longitud – Tipo – Valor:


Raw data:

0x0201061AFF4C000215FDA50693A4E24F
B1AFCFC6EB0764782500050006B0110700
00484345544349474F4C49031800000909
52444C3531383232

Details:

LEN.	TYPE	VALUE
2	0x01	0x06
26	0xFF	0x4C000215FDA50693A4E24FB1AFC FC6EB0764782500050006B0
17	0x07	0x0000484345544349474F4C490318 0000
9	0x09	0x52444C3531383232

LEN. - length of EIR packet (Type + Data) in bytes,
TYPE - the data type as in https://www.bluetooth.org/en-us/specification/assigned-numbers/generic-access-profile

OK

Figura 9. Captura detalles beacon aplicación nRF Connect

- En el caso de los tres primeros bytes 0x020106:

  La longitud es de 0x02 → 1 byte de tipo 0x01, que pertenece al tipo Flags, y 1 byte de valor 0x06, General Discoverable Mode y BR/EDR Not Supported.

  Esto también se puede visualizar en los campos que aparecen en el desplegable, en el apartado Flags:

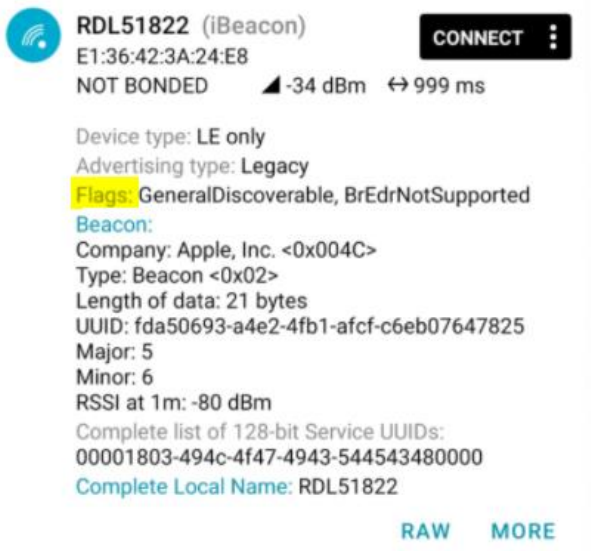


Figura 10. Captura desplegable aplicación nRF Connect

- Los siguientes bytes 0x1AFF4C000215..00050006B0 significan:

  0x1A = Longitud 26 bytes.

  0xFF = Tipo Manufacturer Specific Data.

  0x4C00 = Special Interest Group, Identificador de la compañía, en este caso es Apple, en formato Little Endian.

  0x0215 = Beacon type.

  0xFD...0x25 = Proximity UUID, información configurable por el usuario.

  0x0005 = Major, configurable por el usuario.

  0x0006 = Minor, configurable por el usuario.

  0xB0 = Power.

- Seguido los siguientes bytes 0x1107000048...180000:

  0x11 = Longitud 17.

  0x07 = Tipo Complete List of 128-bit Service Class UUIDs.

  0x000048...180000 = Identificador Universal Unico de 16bytes, sin documentar.
  Si fuera del tipo 0x03: Complete List of 16-bit Service Class UUIDs, tendríamos documentado de qué es -temperatura, presión, humedad, etc-

- Los posteriores bytes 0x090952444C3531383232 significan:

  0x09 = Longitud 9.

  0x09 = Tipo Complete Local Name.

  0x52444C3531383232 = Nombre en ASCII del dispositivo: RDL51822.

2.2 Módulo sensor de temperatura y humedad basado en ESP32

ESP32 es un sistema en un chip -SoC- programable, de bajo coste y consumo diseñado por la empresa Espressif Systems [18]. Integra las características de Wi-Fi -banda de 2.4 GHz-, Bluetooth, coprocesador de energía ultrabaja y múltiples periféricos entre otras cosas.

Con tecnología de 40 nm, ESP32 proporciona una plataforma robusta y altamente integrada, que ayuda a satisfacer las demandas continuas de uso eficiente de la energía, diseño compacto, seguridad, alto rendimiento y confiabilidad. Los ESP32 utilizados en este proyecto son del modelo WROOM.

Espressif proporciona recursos básicos de hardware y software para ayudar a los desarrolladores de aplicaciones a realizar sus ideas utilizando el hardware de la serie ESP32. El marco de desarrollo de software de Espressif está destinado al desarrollo de aplicaciones de Internet de las cosas –IoT- con Wi-Fi, Bluetooth, administración de energía y varias otras características del sistema.

Qué necesitamos para utilizar ESP32:

- Hardware: una placa ESP32, un cable USB - USB A / micro USB B y un ordenador con Windows, Linux o macOS.

- Software: se tiene la opción de descargar e instalar el software manualmente:
  Toolchain para compilar código para ESP32, herramientas de compilación: CMake y Ninja para crear una aplicación completa para ESP32 y ESP-IDF que esencialmente contiene una API -bibliotecas de software y código fuente- para ESP32 y scripts para operar la cadena de herramientas.
  O siguiendo el proceso de incorporación utilizando los complementos oficiales para entornos de desarrollo integrados –IDE-: Eclipse Plugin y VS Code Extension.

El módulo es característico por tener antena integrada para Wi-Fi y Bluetooth así como diversos periféricos y pines configurables para distintos usos.

Al ser programables estos módulos, lo hemos utilizado para conectarlo a unos sensores de temperatura y humedad.

Cada ESP32 se conecta por medio de cuatro cables al sensor y se comunica con él: el sensor le envía al ESP32 la temperatura y humedad captada para que posteriormente el ESP32 la mande a la Raspberry Pi por BLE para su procesamiento.

El formato de los mensajes advertisement que envía el ESP32 a la Raspberry es el siguiente:

Flags			UUID								2		1	1	2	2	2	2	2	2	2
0x02	0x01	0x06	0x03	0x03	0xE2	0xFF	0x17	0x16	0xE2	0xFF	LS_ID	MS_ID	Nº Seq	Battery	T1	H1	T2	H2	T3	H3	H4	T4

Figura 11. Especificación formato payload ADV ESP32

Este formato de payload advertisement se explica detalladamente en el apartado 3.1 del Capítulo 3.

- Se ha creado un programa trucado de sensor ESP32 para el desarrollo del Gateway, este programa consiste en un único módulo ESP32 con sensor de temperatura y humedad que actúa como si fuese N sensores, con la diferencia de que esos N sensores compartirán la misma MAC.

  Para este prototipo se han utilizado los siguientes elementos:

  - Modulo ESP32, WROOVER o WROOM indistintamente.
  - Sensor Sensirion SHT3X.

  El programa tiene el siguiente funcionamiento:

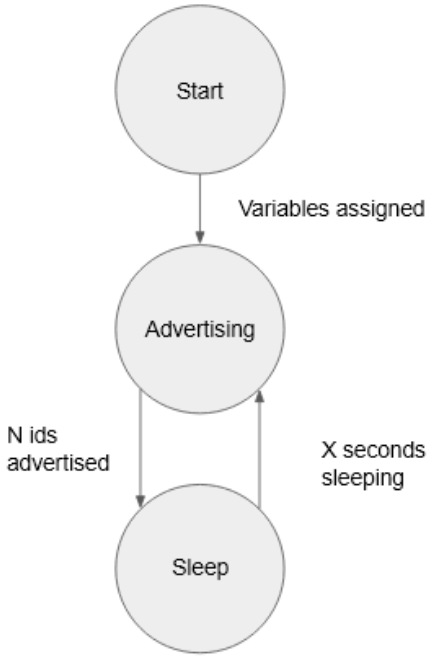


Figura 12. Funcionamiento programa trucado sensor ESP32

- El programa arranca e inicializa las variables, como el periodo de advertising, el tiempo de advertising de los IDs y el tiempo en el cual estará inactivo, así como el número N de IDs a utilizar y los pines de SDA y SCL en los que se conectará el sensor.

- Una vez se han asignado dichas variables se inicializan las librerías de Bluetooth e I2C. El programa empieza promocionándose con ID 1 y el contador empieza en 0 la primera vez. Además el programa promociona las N IDs con el mismo contador, aunque diferentes medidas de temperatura y humedad.

- Una vez promociona las N IDs el programa detiene el advertising durante X segundos, para, pasado el tiempo promocionar otra vez las N IDs empezando desde la ID 1.
  Cada vez que se termina de promocionar las N IDs se incrementa el número de secuencia hasta que este llegue a 255, una vez se llega a dicho número se restablece a 1 de nuevo.

- También se han creado dos modos de funcionamiento del ESP: estado configuración y estado operación muestreando con ahorro de energía.

- Estado configuración:

  Para esta tarea se busca realizar un prototipo que permita configurar los parámetros que utilizará el ESP32 para muestrear, desconectarse, encenderse o transmitir. Dicho prototipo debe hacer uso de la UART del ESP32 para mostrar los parámetros que ya tiene guardados y pasado un tiempo escuchar nuevos parámetros para configurarse si así se desea. Los nuevos parámetros se guardaran en memoria EEPROM, asegurándose así que una vez se desconecte el ESP32 mantenga los parámetros hasta que se vuelvan a reconfigurar. El prototipo únicamente hace uso del modo activo del ESP32.

  El funcionamiento de este primer prototipo es el siguiente:

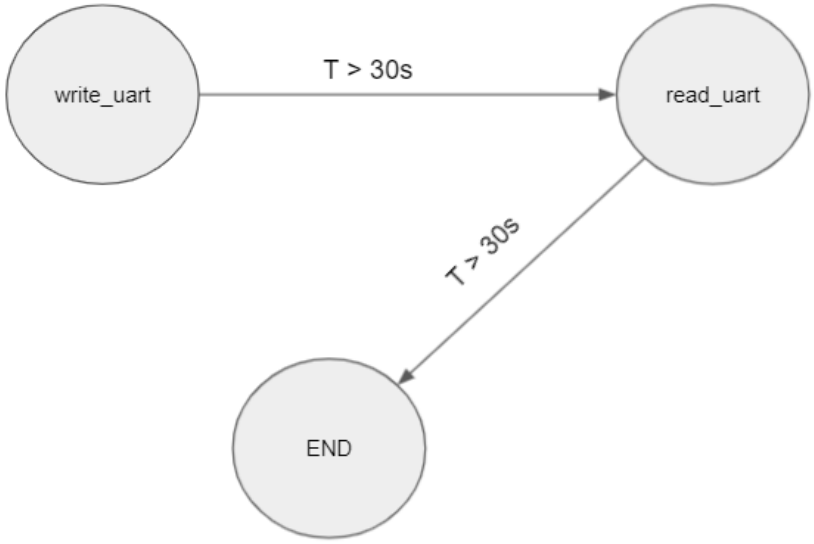


Figura 13. Máquina estados sensor de temperatura y humedad con ESP32 en estado configuración

Empieza en el estado write_uart, donde durante 30 segundos el ESP32 nos comunicará los siguientes parámetros de configuración que ya tenía guardados:

- ID -2 Bytes-
- MAC -6 Bytes-
- TxPow - Potencia de transmisión -1 Byte-
- Nadv - Numero de advertisings -1 Byte-
- NSC - Numero de ciclos de muestreo -1 Byte-
- Toff -4 Bytes-
- Advertisement interval -1 Byte-
- Tipo Sensor CO_2 -1 Byte-: 1 sensirion, 2 senseair, 3 sunrise, 4 SHT3X, 0 Sin sensor.

Pasados los 30 segundos escuchará una trama que contenga los nuevos parámetros que introducirá el servidor durante otros 30 segundos y finalizará este modo de configuración.
En total este modo de configuración se realiza en 1 minuto: 30 segundos para pasar configuración de "fábrica" y 30 segundos para escuchar los nuevos valores que pueden introducirse.
Se puede observar también que aunque se desconecte el puerto -se extraiga y se vuelva a introducir- este tendrá la misma información guardada que antes le hemos pasado.

- Estado operación muestreando con ahorro de energía:

  El funcionamiento del programa se encuentra descrito en el siguiente diagrama:

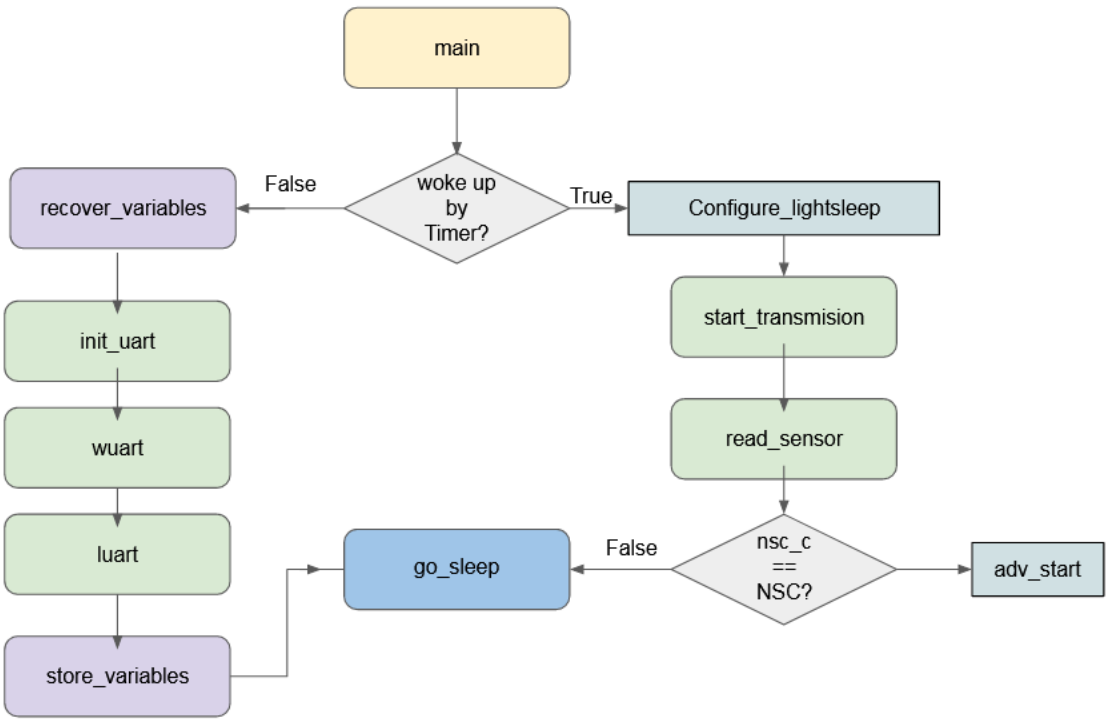


Figura 14. Diagrama flujo sensor con ESP32 en estado configuración y operación

Como todo programa del lenguaje C se empieza en el método main, nada más arrancar se comprueba que ha hecho que el ESP32 se activara, si ha despertado del sueño por Timer o si ha sido otra razón.
Cuando el ESP32 se enciende por primera vez esta comprobación sale como False, indicando que es la primera vez que se enciende, lo que el ESP32 hará ahora es recuperar las variables mediante la función recorver_variables, para a continuación, mediante la UART, comunicar los parámetros que tenga configurados en memoria así como su MAC. Estos parámetros son los siguientes:

- ID 2 Bytes
- Potencia de transmisión 1 Byte
  - 0 (-12 dBm)
  - 1 (-9 dBm)
  - 2 (-6 dBm)
  - 3 (-3 dBm)
  - 4 (0 dBm)
  - 5 (3 dBm)
  - 6 (6 dBm)
  - 7 (9 dBm)
- Número de advertisements 1 Byte
- Número de ciclos de muestreo 1 Byte. Máximo 2.
- Tiempo de sueño 4 Bytes
- Advertisement interval 1 Byte
- Sensor 1 Byte
  - 1 Sensirion
  - 2 Senseair
  - 3 Sunrise
  - 4 SHT3X
  - 0 No usar sensor

Si el ESP detecta que se han introducido nuevos parámetros llamará a la función store_variables para guardar la configuración nueva establecida.

Posteriormente establecerá en tiempo de sueño Toff en la función go_sleep y “dormirá”.

Una vez haya transcurrido el tiempo de sueño saltará el Timer y volveremos al main del código.

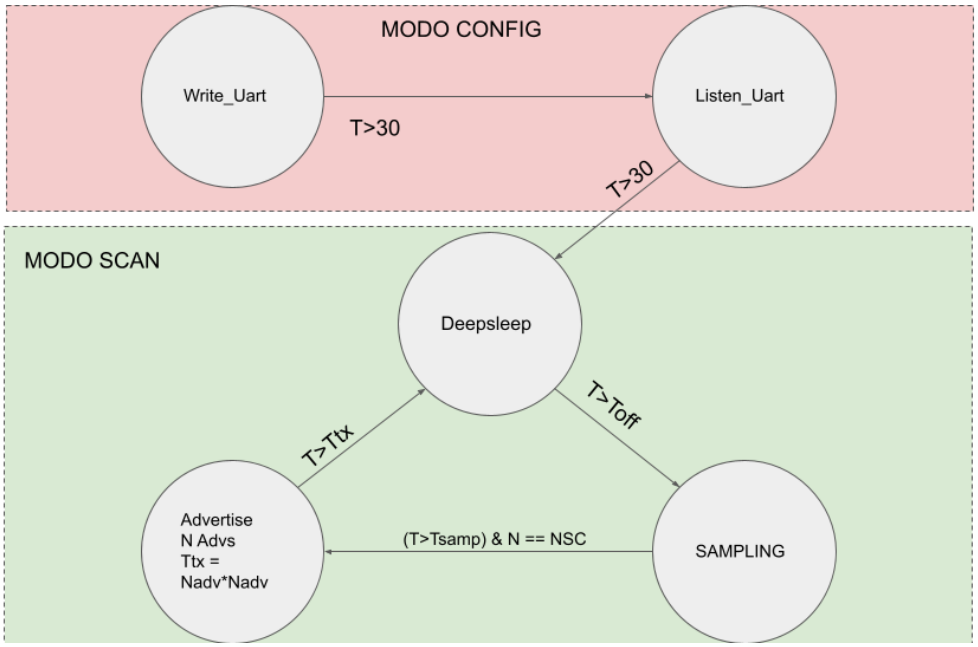


Figura 15. Máquina estados sensor con ESP32 en modo configuración y scan

El ESP32 durante la primera fase de la UART envía 5 mensajes a lo largo de 30 segundos que contienen los parámetros que tiene configurados, junto a su MAC, seguido de una cabecera y un final que permiten distinguirlo.

Cuando acaba de enviar las tramas por la UART lo siguiente que hace es escuchar por otros 30 segundos, por si se le quiere dar una configuración.

Acabados los 30 segundos de configuración el ESP32 guarda los datos en su memoria RTC y memoria física. La primera para tener los datos accesibles en todo momento desde su puesta en marcha, la segunda para tenerlos disponibles en caso de que se desconecte de la corriente. De esta forma se puede dejar configurado con un PC mediante 232Analyzer o un script de python/nodejs, desconectarlo y conectarlo a una batería, en la cual conservará dichos datos.

El ESP32 “dormirá” por unos 60 segundos por defecto, este tiempo es uno de los que se pueden configurar y se configura en un RTC Timer mediante la función esp_sleep_enable_timer_wakeup(t_in_us). Pasado ese tiempo, se configura en modo light-sleep, en este modo se mantienen apagadas la mayor cantidad de periféricos como sea posible para que el ESP32 pueda usar el núcleo principal reduciendo el consumo.

Dentro de la función start_transmision se toma una medida del sensor, si el NSC o número de ciclos de muestreo es igual al total de muestras tomadas, entonces enviará los datos obtenidos por medio de un mensaje advertisement de Bluetooth LE. En caso contrario entonces se guarda el dato en la memoria RTC y se va a “dormir”.

El mensaje Advertisement que envía tiene la estructura payload de la Figura 12.

El ESP32 además incorpora un mecanismo para las pérdidas, mediante el cual envía los datos de las últimas mediciones en los 8 últimos bytes.

En el caso de que se quieran dos ciclos de muestreo, la primera medición se muestra en los bytes T2 y H2, las segunda medición, en T1 y H1. Cuando se toman las dos nuevas muestras estos bytes pasan a los bytes T3, T4, H3 y H4 tal que:

T1->T4
T2->T3

Si se quiere 1 ciclo de muestreo, la temperatura y la humedad empiezan en el Byte T1 y H1, cuando se transmita y se adquiera otra medición, la anterior pasa a T2 y H2 respectivamente y así sucesivamente.

Cuando se tengan los ciclos suficientes el programa envía por medio de mensaje advertisement BLE durante el tiempo necesario para enviar los N advertisements deseados. Cuando acaba se va a “dormir” para volver a iniciar otro ciclo de muestreo.

2.3 Conceptos básicos del protocolo MQTT

El protocolo MQTT -Message Queing Telemetry Transport- [2] es un protocolo de comunicación Machine to Machine -M2M-. Está basado en la pila TCP/IP como base para la comunicación.

En el caso de MQTT cada conexión se mantiene abierta y se "reutiliza" en cada comunicación. Es una diferencia, por ejemplo, a una petición HTTP 1.0 donde cada transmisión se realiza a través de conexión.

El funcionamiento del MQTT es un servicio de mensajería push con patrón publicador/suscriptor -pub-sub-. En este tipo de infraestructuras los clientes se conectan con un servidor central denominado broker.

Para filtrar los mensajes que son enviados a cada cliente, los mensajes se disponen en topics organizados jerárquicamente. Un cliente puede publicar un mensaje en un determinado topic. Otros clientes pueden suscribirse a este topic, y el broker le hará llegar los mensajes suscritos.

Los clientes inician una conexión TCP/IP con el broker, el cual mantiene un registro de los clientes conectados. Esta conexión se mantiene abierta hasta que el cliente la finaliza. Por defecto, MQTT emplea el puerto 1883 y el 8883 cuando funciona sobre TLS.

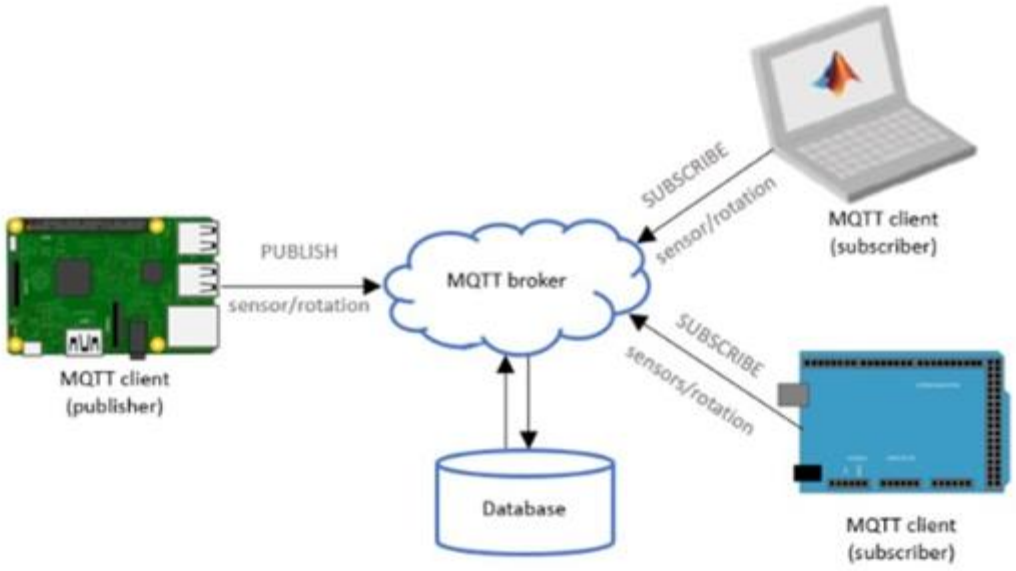


Figura 16. Esquema general MQTT

Este protocolo se compone de los siguientes elementos:

- Broker: servidor que distribuye la información a los clientes interesados conectados a este. Recibe tramas bajo diferentes topics y las distribuye a aquellos suscritos.
- Topic: nombre del mensaje. Los clientes publican, se suscriben o hacen ambas cosas a un topic.
- Cliente: dispositivo que se conecta al broker para enviar o recibir información.
- Publisher: cliente que envía información al broker para distribuirla a los clientes interesados en base al nombre del tema –topic-.
- Subscriber: cliente que se suscribe a un topic para recibir los mensajes que llegan al broker. Cuando un cliente se suscribe a un tema, cualquier mensaje publicado al broker es distribuido a los suscriptores de ese tema. Los clientes también pueden cancelar la suscripción para dejar de recibir mensajes del broker sobre ese topic.
- QoS: Calidad de Servicio. Cada conexión puede especificar una calidad de servicio al broker con un valor entero que va de 0 a 2. La calidad de servicio no afecta al manejo de las transmisiones de datos TCP, sólo entre los clientes MQTT.
    - 0 especifica una vez y sólo una vez sin requerir un ACK. A menudo esto se denomina “fire and forget” -disparar y olvidar-.
    - 1 especifica al menos una vez. El mensaje se envía varias veces hasta que se recibe un ACK. Los mensajes pueden llegar más tarde o duplicados.
    - 2 especifica exactamente una vez. Los clientes remitente y receptor se aseguran de que sólo se recibe una copia del mensaje, lo que se conoce como entrega asegurada.

El funcionamiento es el siguiente:

- El cliente envía un mensaje CONNECT que contiene información necesaria -nombre de usuario, contraseña, client-id…-. El broker responde con un mensaje CONNACK, que contiene el resultado de la conexión -aceptada, rechazada, etc-.
- Para enviar los mensajes el cliente emplea mensajes PUBLISH, que contienen el topic y el payload.
- Para suscribirse y desuscribirse se emplean mensajes SUBSCRIBE y UNSUSCRIBE, que el servidor responde con SUBACK y UNSUBACK.
- Por otro lado, para asegurar que la conexión está activa los clientes mandan periódicamente un mensaje PINGREQ que es respondido por el servidor con un PINGRESP. Finalmente, el cliente se desconecta enviando un mensaje de DISCONNECT.

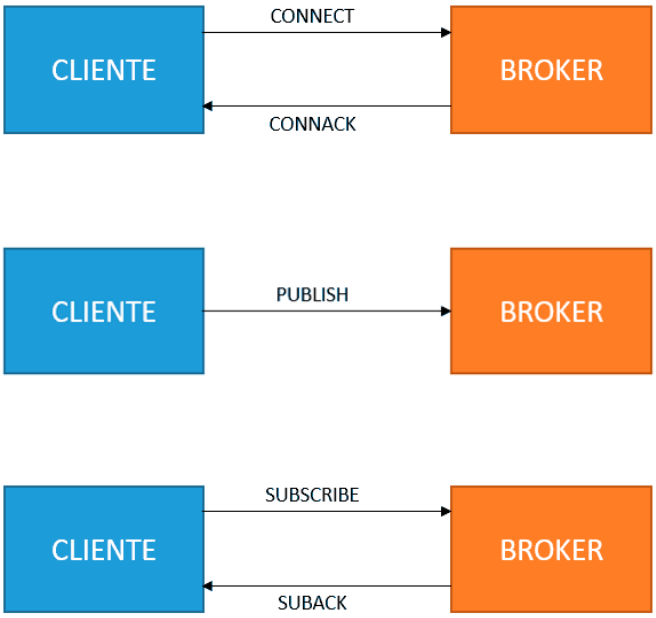


Figura 17. Cronograma funcionamiento MQTT

Estructura mensaje MQTT:

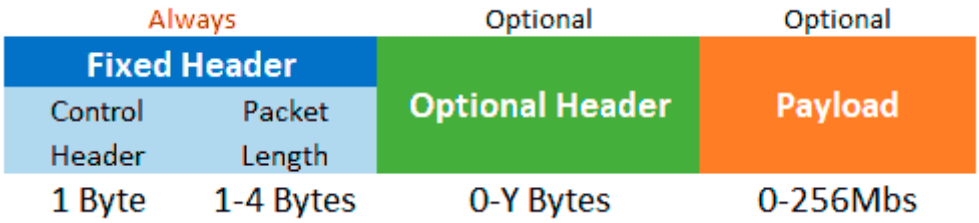


Figura 18. Estructura trama MQTT

- Cabecera fija. Ocupa de 2 a 5 bytes, obligatoria. Consta de un código de control, que identifica el tipo de mensaje enviado, y de la longitud del mensaje. La longitud se codifica en 1 a 4 bytes, de los cuales se emplean los 7 primeros bits, y el último es un bit de continuidad.
- Cabecera variable. Opcional, contiene información adicional que es necesaria en ciertos mensajes o situaciones.
- Contenido –payload-. Es el contenido real del mensaje. Puede tener un máximo de 256 Mb aunque en implementaciones reales el máximo es de 2 a 4 kB.

Los tipos de mensajes y códigos de control que se envían en el protocolo MQTT son los siguientes:

Message	Code
CONNECT	0x10
CONNACK	0x20
PUBLISH	0x30
PUBACK	0x40
PUBREC	0x50
PUBREL	0x60
PUBCOMP	0x70
SUSBSCRBE	0x80
SUBACK	0x90
UNSUSCRIBE	0xA0
UNSUBACK	0xB0
PINGREQ	0xC=
PINGRESP	0xD0
DISCONNECT	0xE0

Tabla 1. Mensajes y códigos de control MQTT

Capítulo 3. Diseño e implementación de un Gateway IoT basado en Raspberry Pi

3.1 Arquitectura y funcionalidad del Gateway IoT Raspberry Pi

Figura 19. Raspberry Pi y Dongle USB externo real

La Raspberry Pi es un ordenador de bajo coste y tamaño reducido compuesto por un SoC, CPU, memoria RAM, puertos de entrada y salida de audio y vídeo, conectividad de red, ranura SD para almacenamiento, reloj, una toma para la alimentación, conexiones para periféricos de bajo nivel, reloj, etc. El único detalle es que no tiene interruptor para encender o apagar.

Para ponerlo en marcha tenemos que conectar periféricos de entrada y salida para poder interactuar como una pantalla, un ratón y un teclado y grabar un sistema operativo para Raspberry Pi en la tarjeta SD. Ya solo queda conectarlo a la corriente y estamos listos para funcionar.

Existen infinidad de proyectos que se pueden llevar a cabo con una Raspberry, pero normalmente los usuarios la emplean de estas formas:

- Como media center, para convertir una televisión en una smart TV, con software LIBRELEC o OSMC.
- Para emular una videoconsola retro jugando a grandes clásicos con RetroPie instalado.
- Para Domótica, con Windows 10 IoT Core, lo que permite hacer de nuestra casa un espacio un poco más inteligente con proyectos como estaciones meteorológicas o hubs inteligentes.
- Como ordenador con sistema Linux, a través de distribuciones como Ubuntu, Raspbian(Debian) o Pidora (Fedora).

Este último punto describe el uso que le vamos a dar a la Raspberry.

La Raspberry está conectada a un dongle USB externo, que proporciona otra interfaz bluetooth aparte de la interna de la placa. Al tener una antena es posible que el rango de captación de tramas bluetooth sea mayor que el de la interfaz interna, esto garantiza la cobertura con todos los sensores.

El motivo de tener dos interfaces bluetooth en la Raspberry, una interna y otra externa, es para asegurar que ninguna trama enviada por lo sensores se pierde, pues el código está adaptado para captar los mensajes que vienen de ambas interfaces y procesarlos, evitando duplicados o pérdidas de mensajes. Esto proporciona redundancia de datos de sensores a filtrar, funcionalidad edge-computing, la cual estamos trabajando.

La funcionalidad de nuestra Raspberry es actuar como Gateway, recogiendo información que envían los sensores de temperatura y humedad conectados a módulos ESP32 para procesarla y posteriormente mandarla a un servidor.

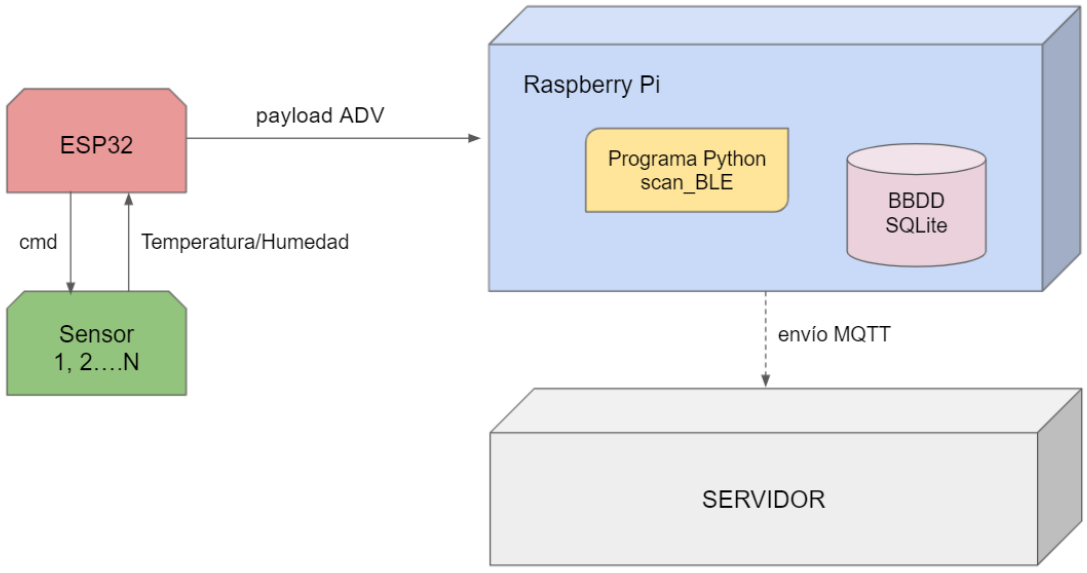


Figura 20. Diagrama de bloques del funcionamiento Raspberry Pi

Dentro de la Raspberry, tenemos un código que escanea y recoge la información de los mensajes advertisement que envían los sensores, filtra y procesa esta información y la guarda en una base de datos que actúa como backup. Además se encarga de enviar por MQTT los datos obtenidos a un servidor MQTT, que explicaremos más adelante en el siguiente punto.

Aparte de escanear y filtrar datos, el código posee una comprobación de alarmas de sensores: unas reglas básicas de chequeo para generar avisos al servidor, proporcionando así funcionalidad edge-computing. Lo explicaremos en el apartado 3 de esta misma sección.

Además, se hace uso de PM2 [15], que monitoriza el servicio generando mensajes sobre cualquier evento anómalo en el sistema, por ejemplo, si hay un corte de luz o si el programa se reinicia. Lo aclararemos mejor en el apartado 4 de esta misma sección.

El funcionamiento del programa principal es el siguiente:

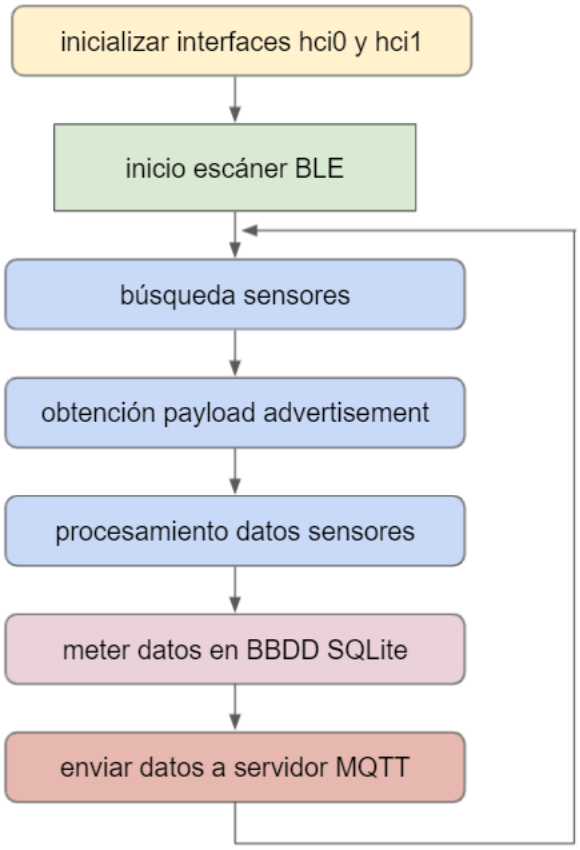


Figura 21. Diagrama de flujo programa escáner Raspberry Pi

- El cronograma comienza con la inicialización de las interfaces que vamos a utilizar, hci0 interfaz interna y hci1 interfaz externa, además de importar las librerías correspondientes para el desarrollo del código. Estas son: asyncio, bleak, datetime, json, logging, mqtt, sqlite3 y time.

- Iniciamos el escáner bluetooth, que buscará los mensajes advertisement que envíen los sensores. El formato de payload de nuestros sensores es el siguiente:

Flags			UUID								2		1	1	2	2	2	2	2	2	2
0x02	0x01	0x06	0x03	0x03	0xE2	0xFF	0x17	0x16	0xE2	0xFF	LS_ID	MS_ID	Nº Seq	Battery	T1	H1	T2	H2	T3	H3	H4	T4

Figura 22. Especificación formato payload ADV sensor

Nos fijamos a partir de los bytes del UUID:

- Empezando por el UUID, es el mecanismo para identificar, de entre todas las tramas Bluetooth del medio, aquellas que nos interesan. Debemos buscar las tramas cuyo UUID sea 0xE2FF, que es el UUID de nuestros sensores.

- Continuo al UUID comienza la trama:
  0x17 longitud de la subtrama
  0x16 tipo: datos
  0xE2FF: UUID de nuevo, debe aparecer.

- Seguido tenemos el ID -2 bytes- para identificar entre dichas tramas, qué sensor la ha enviado. Como aclaración: LS = Least Significant; MS = Most Significant.

- Continuo a este obtenemos el número de secuencia Seq -1byte-, el nivel de batería Bat -1byte-, la temperatura T -2bytes- y la humedad H -2bytes-.

- Una vez encontrado el sensor, nos dispondremos a obtener su payload para filtrar y procesar la información que nos envía, obteniendo así:

  - Id sensor
  - Número secuencia
  - Temperatura
  - Humedad
  - Batería

- Estos datos, junto con la MAC del dispositivo y una marca de tiempo, se procesan y guardan en la base de datos de la Raspberry. A continuación se explica el procedimiento para crear e insertar los datos en la base de datos:

- En el terminal escribimos `sudo apt install sqlite3`

- Dentro de la carpeta donde estamos trabajando, creamos otra carpeta para añadir ahí la base de datos.
  En mi caso: scan_BLE/sqlite3/sqlite, escribimos `nano BBDD.sqlite3`

- Una vez creada la BBDD, hacemos `cd ..` para dirigirnos a la carpeta anterior, donde crearemos la tabla para nuestra base de datos.

En mi caso: scan_BLE/sqlite3, escribimos `nano tabla-BBDD.py` y añadimos el siguiente código:

```
import sqlite3

#Conectar a la base de datos

conection = sqlite3.connect("sqlite/BBDD.sqlite3")   #ruta donde se encuentra la bbdd

#Seleccionar cursor para realizar la consulta

consulta = conection.cursor()

#Crear nueva tabla con columnas de distintos tipos

sql = """
CREATE TABLE IF NOT EXISTS datos_sensores(
timestamp  DATETIME NOT NULL,
id INTEGER NOT NULL,
mac INTEGER NOT NULL,
seq INTEGER NOT NULL,
temperatura FLOAT NOT NULL,
humedad FLOAT NOT NULL,
bateria INTEGER NOT NULL)"""

#Ejecutar consulta
if(consulta.execute(sql)): print("Tabla creada con exito") #si consulta se lleva a cabo con exito
else: print("Ha ocurrido un error al crear la tabla")

#Terminamos la consulta
consulta.close()

#Guardamos los cambios en la base de datos
conection.commit()

#Cerramos conexion a la bbdd
conection.close()
```

Figura 23. Código para crear tabla BBDD en Raspberry Pi

- Hacemos `sudo python3 tabla-BBDD.py` y aparecerá un mensaje de "Tabla creada con éxito". Una vez creada procedemos a insertar los datos en la tabla, dentro de nuestro código de SCAN BLE:

```
conection = sqlite3.connect("sqlite/BBDD.sqlite3")


date_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

consulta = conection.cursor()

argumentos = (datetime.datetime.now(), id, device.address, nseq, temp, hum, bat)

sql = """
INSERT INTO datos_sensores(timestamp, id, mac, seq, temperatura, humedad, bateria)
VALUES(?, ?, ?, ?, ?, ?, ?)
"""

if (consulta.execute(sql,argumentos)):
    print("Registro guardado en BBDD con exito")
else: print("Ha ocurrido un error al guardar el registro")

#Terminamos la consulta
consulta.close()

#Guardamos los cambios en la base de datos
conection.commit()
```

Figura 24. Código para insertar datos en tabla BBDD Raspberry Pi

Los argumentos mostrados en la figura 24 se obtienen filtrando y procesando la información obtenida de los mensajes advertisement que envían los sensores.

No cerramos la conexión a la BBDD puesto que el escáner se mantiene siempre activo e insertando datos cuando se reciben y procesan.

- Seguidamente enviamos estos mismos datos por medio de MQTT al servidor Mosquitto o broker, que estará escuchando bajo el topic 'ALBA' en nuestro servidor. Este envío se describe en el siguiente apartado.

3.2 Procesamiento de datos sensores y reenvío de datos al server mediante MQTT

3.2.1 Procesamiento de datos sensores

Para evitar duplicados y datos redundantes de los sensores según el número de secuencia, se hace uso de la funcionalidad que tiene Python [6] de los diccionarios -ver anexo para aclaración en el uso de los diccionarios-. El diagrama de flujo del programa de la Raspberry Pi usando diccionarios es el siguiente:

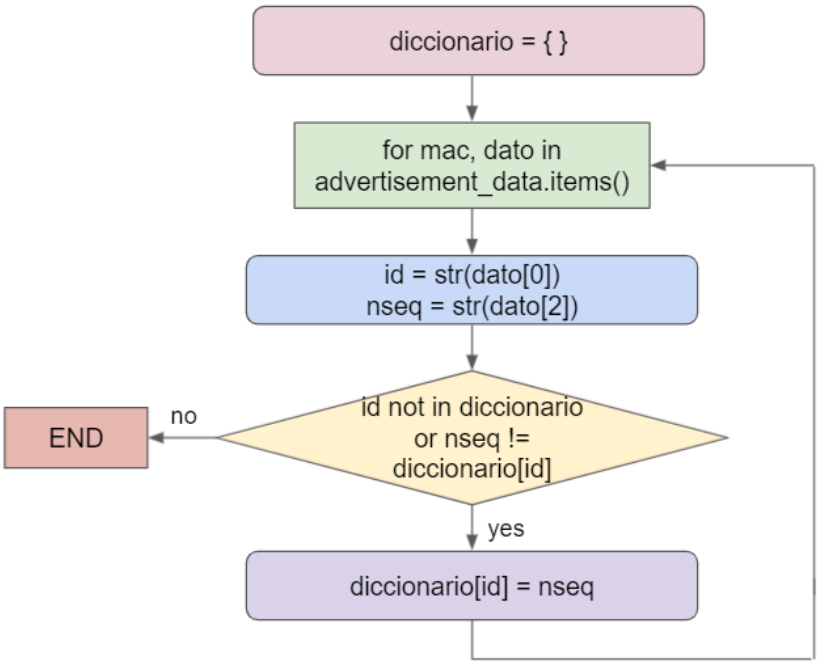


Figura 25. Diagrama de flujo utilización diccionarios en Raspberry Pi

- Declaro la variable diccionario = { } donde guardaremos los elementos que queremos comprobar.
- En mi función de escáner de advertisements, recorro una lista de elementos escaneados buscando el UUID que poseen y envían mis sensores.
- Si encuentro el UUID que busco, filtro el ID y el número de secuencia del sensor, y evalúo lo siguiente:
    - Si el ID encontrado no se encuentra en el diccionario.
    - Si el número de secuencia no está en el diccionario.
- Una vez evaluado, si se da el caso de uno de ellos, guardo en el diccionario la clave=id y el valor=número de secuencia.
- Cuando se guardan en el diccionario esos valores, y volvemos a recorrer la lista de advertisements, evaluaremos de nuevo las dos comprobaciones.

- Si llega un ID que ya está en el diccionario, pero el nuevo valor de número de secuencia no está en él, lo guardaremos. Así se irá actualizando el diccionario comprobando y evitando duplicados y datos redundantes de los sensores según el número de secuencia.

La Raspberry está conectada a un dongle USB externo, que proporciona otra interfaz bluetooth aparte de la interna de la placa. Al tener una antena es posible que el rango de captación de tramas bluetooth sea mayor que el de la interfaz interna, esto garantiza la cobertura con todos los sensores.

Además de la funcionalidad que utilizo en Python de los diccionarios para el procesamiento de datos, el motivo de tener dos interfaces bluetooth en la Raspberry, una interna y otra externa, es para asegurar que ninguna trama enviada por lo sensores se pierde, pues el código está adaptado para captar los mensajes que vienen de ambas interfaces y procesarlos, evitando duplicados o pérdidas de mensajes. Esto proporciona redundancia de datos de sensores a filtrar, funcionalidad edge-computing, la cual estamos trabajando.

3.2.2 Reenvío de datos al server mediante MQTT

Respecto al envío de datos al server mediante MQTT [2], primero debemos instalar Mosquitto Hub en nuestra Raspberry [13]. Esto se realiza mediante los comandos:

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

Para comprobar su correcto funcionamiento, abrimos dos ventanas de comandos:

- 1º ventana de comandos: nos suscribimos al topic "ALBA"

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

```
pi@raspberrypi:~ $ mosquitto_sub -d -h localhost -p 1883 -t "ALBA"
Client mosqsub|1266-raspberryp sending CONNECT
Client mosqsub|1266-raspberryp received CONNACK (0)
Client mosqsub|1266-raspberryp sending SUBSCRIBE (Mid: 1, Topic: ALBA, QoS: 0)
Client mosqsub|1266-raspberryp received SUBACK
Subscribed (mid: 1): 0
```

- 2º ventana de comandos: publicamos un mensaje 'Hola Mundo'

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

```
pi@raspberrypi:~ $ mosquitto_pub -d -h localhost -p 1883 -t "ALBA" -m "Hola Mundo"
Client mosqpub|1267-raspberryp sending CONNECT
Client mosqpub|1267-raspberryp received CONNACK (0)
Client mosqpub|1267-raspberryp sending PUBLISH (d0, q0, r0, m1, 'ALBA', ... (10 bytes))
Client mosqpub|1267-raspberryp sending DISCONNECT
```

- Obtenemos, en la primera ventana de comandos -subscriber-:

```
pi@raspberrypi:~ $ mosquitto_sub -d -h localhost -p 1883 -t "ALBA"
Client mosqsub|1266-raspberryp sending CONNECT
Client mosqsub|1266-raspberryp received CONNACK (0)
Client mosqsub|1266-raspberryp sending SUBSCRIBE (Mid: 1, Topic: ALBA, QoS: 0)
Client mosqsub|1266-raspberryp received SUBACK
Subscribed (mid: 1): 0
Client mosqsub|1266-raspberryp received PUBLISH (d0, q0, r0, m0, 'ALBA', ... (10 bytes))
Hola Mundo
```

Una vez instalado y comprobado su funcionamiento, pasamos a usarlo en el código de SCAN BLE:

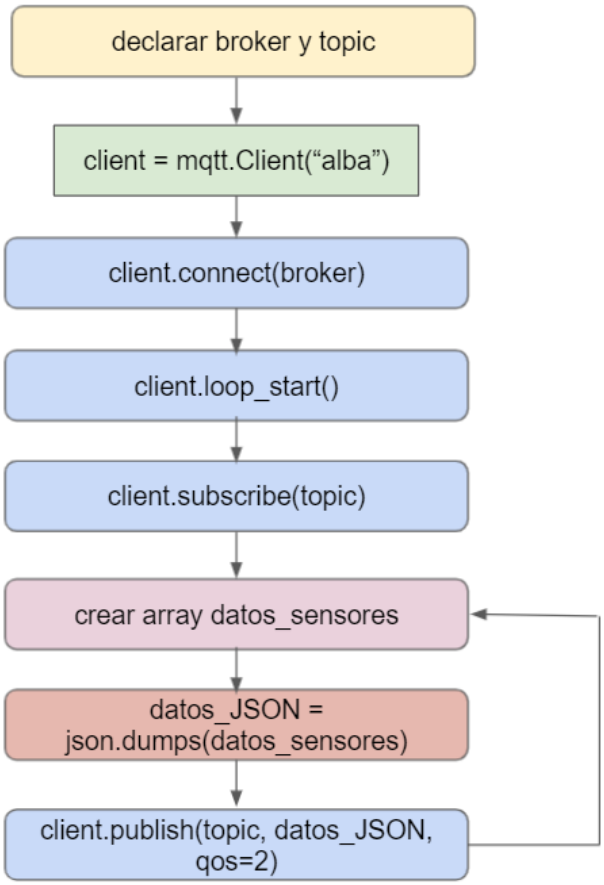


Figura 26. Diagrama de flujo envío datos por MQTT en Raspberry Pi

- Declaro el broker, que será la IP donde está alojado el servidor Mosquitto, y el topic, en mi caso topic = "ALBA".
- Creamos un cliente.
- Hacemos que nuestro cliente se conecte al broker y se suscriba al topic que queremos.

- Creamos un array de datos_sensores donde guardamos: timestamp, id sensor, mac BLE, número de secuencia, temperatura, humedad y batería del sensor que estamos escaneando y obteniendo información en ese momento.
- Estos datos deben ser convertidos a formato JSON para poder enviarse mediante el protocolo MQTT, por lo que hacemos uso del método json.dumps()
- Una vez convertidos a formato JSON, publicamos los datos en el topic declarado, y con QoS=2, donde los clientes remitente y receptor se aseguran de que sólo se recibe una copia del mensaje, lo que se conoce como entrega asegurada.
- Este funcionamiento y envío por MQTT se repetirá tantas veces como mensajes de sensores sean escaneados. La declaración del broker y topic, así como el cliente, su conexión y suscripción sólo se realizan al principio del programa, pues siempre tendremos esos valores.

3.3 Gestión de alarmas de sensores en el gateway

Nuestro Gateway aporta la funcionalidad edge-computing a nuestro sistema: recopila datos de numerosos sensores de temperatura y humedad y hace un preprocesamiento -procesamiento con los datos brutos para transformarlos en datos que tengan formatos que sean más fáciles de utilizar- y tratamiento de dichos datos minimizando el envío de datos y la carga computacional del servidor, pues lo libera de hacer operaciones simples como detectar alarmas. Con edge-computing es posible lograr un mayor ahorro de energía y reducción del ancho de banda empleado en comunicaciones.

Al código SCAN BLE se le ha añadido la funcionalidad de detectar cuatro tipos de alarmas:

- **AlarmaP**. Alarma principal: cada 5 minutos se mirará si algún sensor ha dejado de transmitir. Se chequea si no se reciben datos de un sensor en un tiempo establecido de 30 segundos, y si es así, se creará “Alarma SENSOR” con el mensaje de aviso:

  {Timestamp} -> Alarma SENSOR: Han pasado X segundos desde que id Y ha dejado de transmitir.

- **Alarma1**. Si se pierde el número de secuencia de algún sensor, por ejemplo, si se espera recibir el número de secuencia 25 y llega en su lugar el 26, se creará “Alarma NSEQ” con el mensaje de aviso:

  {Timestamp} -> Alarma NSEQ: Número de secuencia fuera de orden, se esperaba en id X: Y y llega Z.

- **Alarma2**: Si el nivel batería es bajo, por ejemplo, si batería < 20%, se creará “Alarma BATERÍA” con el mensaje de aviso:

  {Timestamp} -> Alarma BATERÍA: Nivel de batería en id X bajo: Y% , considere cambiar batería.

- **Alarma3**. Si se reciben valores anómalos. Para ello, establezco ciertas reglas para comprobar si el valor obtenido por el sensor es normal, mediante la media móvil exponencial [16]:

  En estadística es un cálculo para analizar puntos de datos mediante la creación de una serie de promedios de diferentes subconjuntos del conjunto de datos completo.
  Dada una serie de números y un tamaño de subconjunto fijo, el primer elemento de la media móvil se obtiene tomando el promedio del subconjunto fijo inicial de la serie de números. Luego, el subconjunto se modifica "desplazándose hacia adelante", es decir, excluir el primer número de la serie e incluir el siguiente valor en el subconjunto.
  Visto de forma simple, se puede considerar que suaviza los datos.

El funcionamiento es el siguiente:

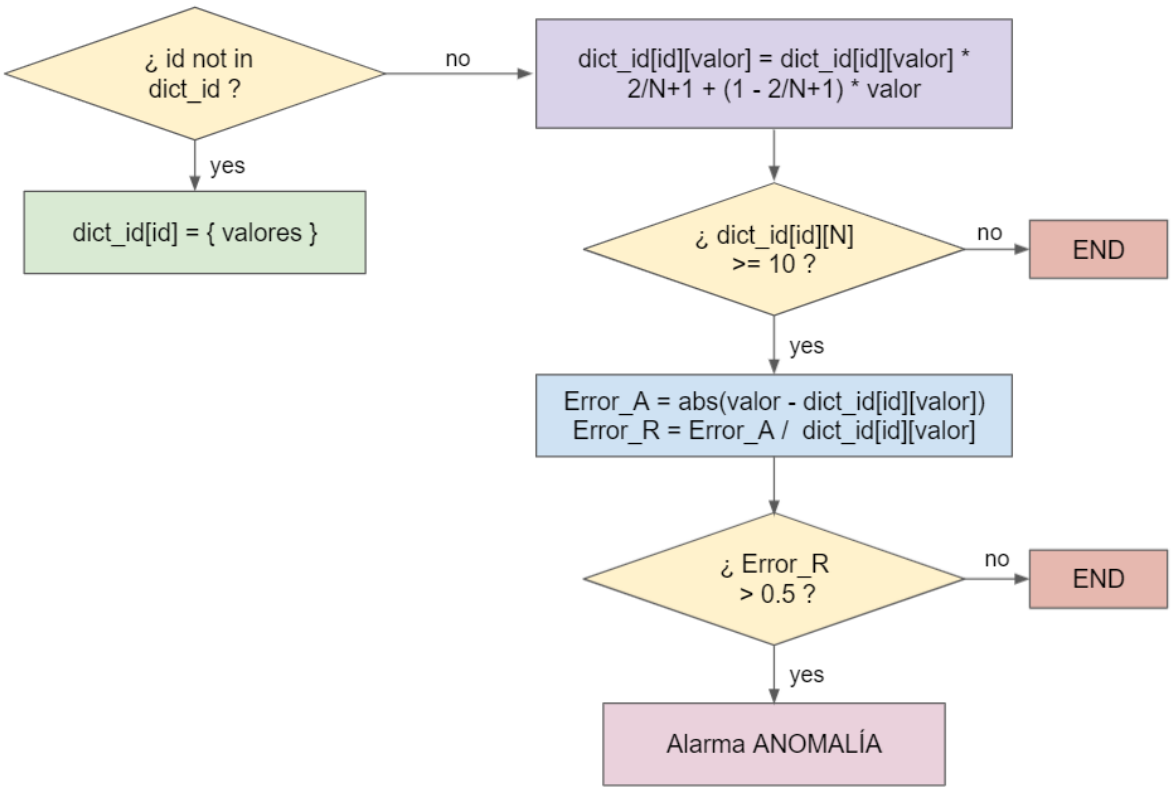


Figura 27. Diagrama de flujo funcionamiento Alarma Anomalía Raspberry Pi

- Si la ID del sensor no se encuentra en un diccionario de ids, se guardan los valores obtenidos de temperatura, humedad y batería en él.
- Si ya existe esa ID en el diccionario de ids, se calcula la media móvil exponencial, siendo de esta forma: $dict = dict * \alpha + (1 - \alpha) * valor$. Donde:

  $dict =$ diccionario de ids donde se va actualizando la media.
  $valor =$ valor de temperatura, humedad o batería, dependiendo del caso. $\alpha = 2 / (N + 1)$. N = número de muestras, en nuestro caso 10.
  Esta fórmula se repite N veces.
- Cuando se han medido N = 10 muestras, comprobamos:

  $$Error\ absoluto = abs(valor - dict)$$
  $$Error\ relativo = error\ absoluto\ /\ dict$$
- Si el error relativo es mayor del 50% (si diferencia entre valor medido y media es ±50% del valor de la media) quiere decir que se ha producido un valor anómalo, y se creará "Alarma ANOMALÍA" con el mensaje de aviso:

  {Timestamp} -> Alarma ANOMALÍA: (Temperatura/Humedad/Batería) anómala en id X: para una media de $dict$ se ha recibido $valor$, error del Y%;

Todas las alarmas vienen junto con el timestamp del momento el cual se produjeron y se envían mediante el protocolo MQTT bajo el topic 'ALBA/ALARM' al servidor para posteriormente ser visualizadas en la página del usuario.

3.4 Gestión log y monitorización servicio con PM2

PM2 [15] es un administrador de procesos de NodeJS que nos ayuda a administrar y mantener aplicaciones en línea. Se ofrece como una CLI -interfaz de línea de comandos- simple e intuitiva, que se puede instalar a través de npm. Algunas características clave de PM2 son el equilibrio automático de carga de aplicaciones, la configuración de aplicaciones declarativas, el sistema de implementación y la supervisión.

PM2 permite mantener siempre activas las aplicaciones y volver a cargarlas evitando los tiempos de inactividad, a la vez que facilita tareas comunes de administrador del sistema.

Esto proporciona robustez en nuestro código, pues PM2 se encarga de monitorizar y levantar el servicio. Gracias a PM2, el programa principal se ejecutará con tan solo alimentar la Raspberry, así como si por ejemplo se produce un corte de alimentación, la Raspberry debe inicializarse sola y ejecutar el programa.

- Primero se debe instalar npm mediante los siguientes comandos:

```
$ curl -o-
https://raw.githubusercontent.com/creationix/nvm/v0.35.3/install.
sh | bash
$ nvm list
$ nvm ls-remote
$ nvm install 14.5.0
$ nvm use 14.5.0
```

  Se mostrará el mensaje de `Now using node v14.5.0 (npm v6.14.5)`

```
$ nvm alias default 14.5.0
$ node -v
$ npm install -g npm
$ npm -v
```

  Se mostrará el mensaje de `7.20.5`

```
$ echo fs.inotify.max_user_watches=524288 | sudo tee -a
/etc/sysctl.conf && sudo sysctl -p
```

- Una vez instalado npm procedemos a instalar PM2:

```
npm install pm2@latest -g
```

- Para inicializar la aplicación con python3:

```
pm2 start <aplicación> --interpreter python3
```

- Ejecutamos después: `pm2 startup`
  Nos aparecerá un mensake pidiéndonos que copiemos y peguemos cierto comando:

```
pi@raspberrypi:~ $ pm2 startup
[PM2] Init System found: systemd
[PM2] To setup the Startup Script, copy/paste the following command:
sudo env PATH=$PATH:/home/pi/.config/nvm/versions/node/v14.5.0/bin /home/pi/.config/nvm/versions/node/v14.5.0/lib/n
ode_modules/pm2/bin/pm2 startup systemd -u pi --hp /home/pi
```

- Una vez copiado y pegado, lo guardamos mediante `pm2 save`, obteniendo:

```
[PM2] Init System found: systemd
Platform systemd
Template
[Unit]
Description=PM2 process manager
Documentation=https://pm2.keymetrics.io/
After=network.target

[Service]
Type=forking
User=pi
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity
Environment=PATH=/home/pi/.config/nvm/versions/node/v14.5.0/bin:/home/pi/.local/bin:/usr/local/sbin:/usr/local/bin:
/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games:/snap/bin:/home/pi/.config/nvm/versions/node/v14.5.0/bin:
/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
Environment=PM2_HOME=/home/pi/.pm2
PIDFile=/home/pi/.pm2/pm2.pid
Restart=on-failure

ExecStart=/home/pi/.config/nvm/versions/node/v14.5.0/lib/node_modules/pm2/bin/pm2 resurrect
ExecReload=/home/pi/.config/nvm/versions/node/v14.5.0/lib/node_modules/pm2/bin/pm2 reload all
ExecStop=/home/pi/.config/nvm/versions/node/v14.5.0/lib/node_modules/pm2/bin/pm2 kill

[Install]
WantedBy=multi-user.target

Target path
/etc/systemd/system/pm2-pi.service
Command list
[ 'systemctl enable pm2-pi' ]
[PM2] Writing init configuration in /etc/systemd/system/pm2-pi.service
[PM2] Making script booting at startup...
[PM2] [-] Executing: systemctl enable pm2-pi...
[PM2] [v] Command successfully executed.
+---------------------------------------+
[PM2] Freeze a process list on reboot via:
$ pm2 save

[PM2] Remove init script via:
$ pm2 unstartup systemd
pi@raspberrypi:~/scan_BLE/sqlite3 $
pi@raspberrypi:~/scan_BLE/sqlite3 $ pm2 save
[PM2] Saving current process list...
[PM2] Successfully saved in /home/pi/.pm2/dump.pm2
```

- De esta forma ya tendremos el código monitorizado por PM2. Si ocurre cualquier tipo de problema o reinicio, PM2 se encargará de mantener nuestro programa activo.

- Para comprobar que se generan logs de depuración escribimos:

  `pm2 logs --timestamp`

  De esta manera veremos a qué hora se ha inicializado de nuevo el código, junto con cualquier otro error que se pueda generar.

Capítulo 4. Diseño e implementación de la arquitectura del server

Para el desarrollo de esta arquitectura se han utilizado como elementos principales:

- Gateway IoT Raspberry Pi, descrita en el capítulo anterior.
- Server con los siguientes módulos principales:
    - Programa Mosquitto Hub, broker al que la Raspberry envía los mensajes mediante protocolo MQTT.
    - Back-end servidor, desarrollado con el framework de Javascript NodeJS [4].
    - BBDD SQLite para almacenamiento de los datos.
    - Front-end de visualización de datos para el usuario.
    - Programa de alta y configuración de sensores ESP32.

En el anexo se describen los pasos de instalación de los módulos software que conforman el server. La funcionalidad principal del Server es la recepción y almacenamiento de los datos de los sensores ESP32 reenviados por el Gateway IoT. A su vez, el server gestiona la visualización web y acceso a los datos de los sensores para un usuario de forma sencilla mediante un navegador web. Cabe destacar que hay un programa de alta y configuración de los sensores ESP32; para un caso de uso donde el Server se instala en local en el mismo escenario que los sensores, se considera que dicho programa de alta y configuración corre en la misma máquina que el Server. A continuación se detalla la arquitectura e implementación del Server.

4.1 Arquitectura global del Server

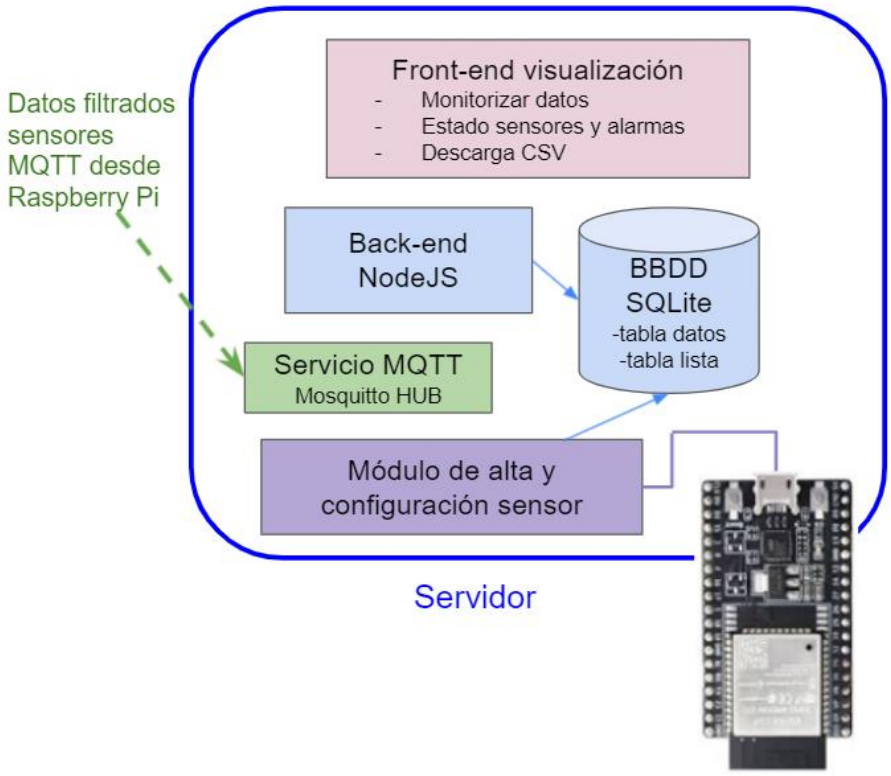


Figura 28. Módulo servidor

El sistema posee los siguientes elementos principales:

- **Gateway IoT Raspberry Pi**. Recoge información de los sensores de temperatura y humedad conectados a varios ESP32. Su funcionalidad es escuchar al medio y escanear los mensajes advertisements que envían los sensores, filtra y procesa esa información, la guarda en su base de datos y la envía a un servidor mediante MQTT.

Server de recepción y visualización de datos de los sensores formado por:

- **Mosquitto Hub**. Servidor MQTT con el que se comunica la Raspberry Pi. Broker al que la Raspberry envía los mensajes mediante protocolo MQTT, cuya función es escuchar a los publishers y reenviar los datos a los subscribers.
- **Back-end servidor**. Éste se encarga de suscribirse al Mosquitto Hub bajo el mismo topic donde la Raspberry está volcando su información. Esta información la guardará en su base de datos SQLite.

También aloja un servicio REST, una interfaz para conectar varios sistemas basados en el protocolo HTTP que nos sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como XML y JSON. Este servicio espera peticiones HTML de usuarios que solicitan acceso a una página gestionada por el servidor. Profundizaremos en esto más adelante.

- **Front-end visualización**. Es la parte que se ejecuta en el dispositivo del cliente, que contiene el código tanto para registrarse en la página como para una vez registrado, visualizar la información que la Raspberry Pi le envía al servidor, en concreto una tabla de datos sensores, con posibilidad de descarga seleccionando fecha y hora, y una tabla de aviso de alarmas que puedan ocasionarse.

En la figura 29 se muestra la arquitectura global del sistema:

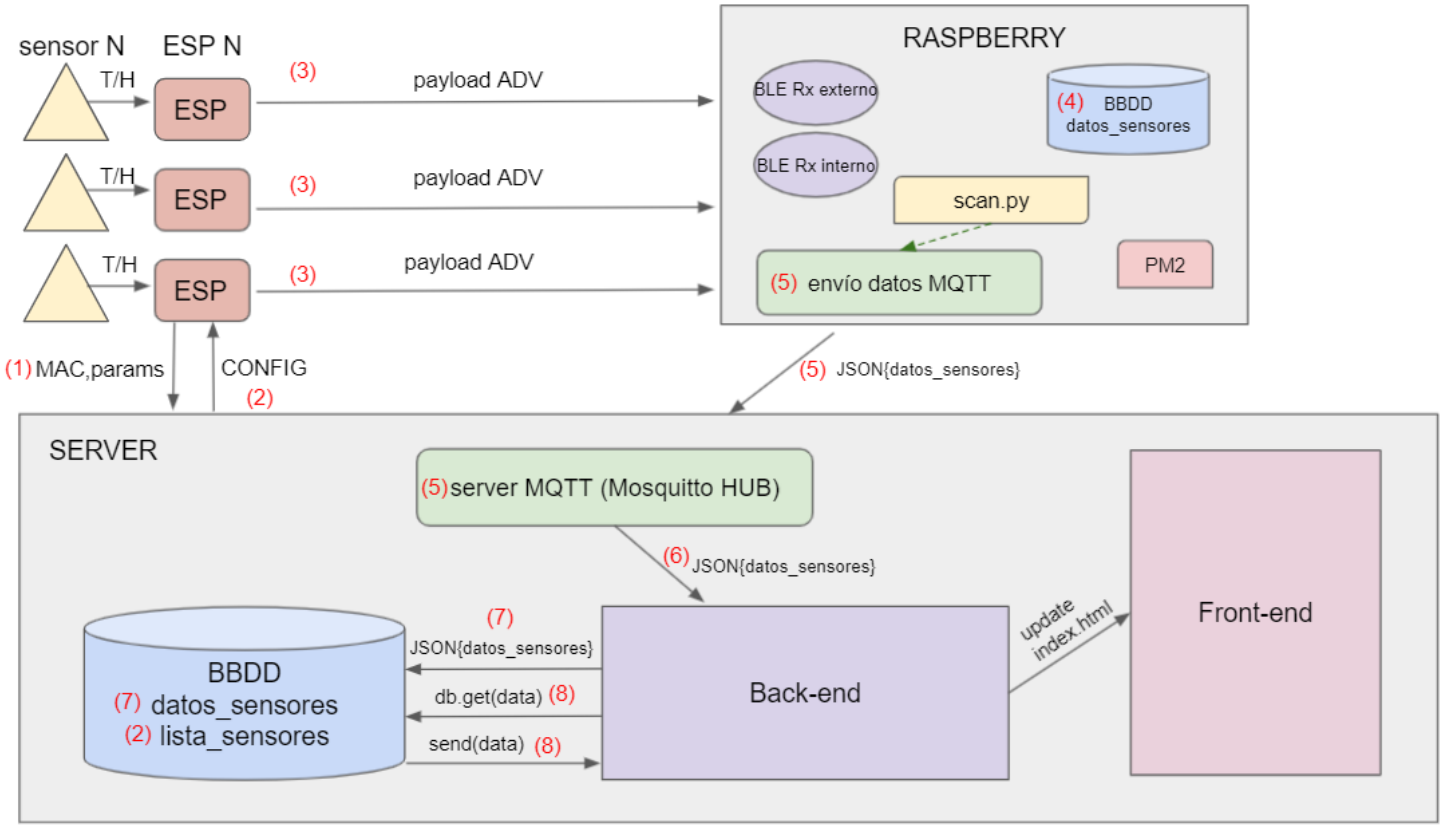


Figura 29. Arquitectura del sistema implementado

El funcionamiento del sistema implementado mostrado en la figura 29 es el siguiente:

1. Sensor ESP32 envía MAC y parámetros configuración por puerto serie al servidor. -se explica en la siguiente página qué parámetros son-

2. Servidor graba configuración en ESP32 mediante programa Python lista_sensores.py y añade en la BBDD, tabla lista_sensores: codigo sensor, id, descripción, mac y estado del sensor.

3. Pasado 1 minuto, ESP32 comienza a transmitir payload ADV gracias al sensor de temperatura/humedad a través de BLE al programa Raspberry scan.py.

4. Programa Raspberry tiene 2 módulos: scan interno y USB scan externo, que reciben los datos BLE que envía el ESP, una vez recibidos scan.py los filtra y procesa para guardarlos en su BBDD datos_sensores.

5. Los datos recibidos son codificados en formato JSON y enviados por MQTT al server Mosquitto.

6. Mosquitto HUB -broker- recibe las tramas bajo un topic y las pasa al Back-end, el cual está suscrito a ese topic.

7. El Back-end inserta las tramas en su Base de Datos, tabla datos_sensores del servidor.

8. La BBDD almacena información y queda accesible para otros usos.
   Back-end puede pedir información a la BBDD y actualizar el Front-end conforme a ella.

Función del programa sensor ESP32:

- Cuando se alimenta por primera vez, durante 30 segundos envía por puerto serie su MAC y los parámetros de configuración configurados.
- Pasado ese tiempo escuchará durante otros 30 segundos para que se le pase otros parámetros, en caso de no recibir nada utilizará los que ya tenía configurados. Los parámetros se envían siguiendo el siguiente cronograma:

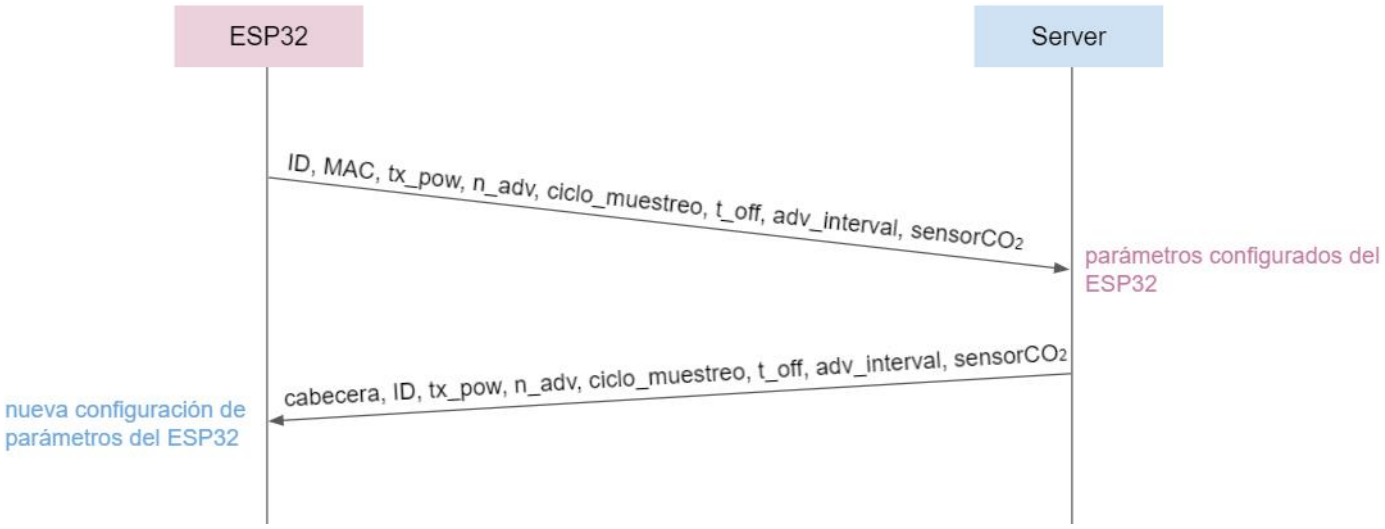


Figura 30. Cronograma parámetros configuración ESP32

- Ciclo de trabajo del sensor ESP32: a continuación el sensor seguirá un ciclo de trabajo de sueño, para el ahorro de batería, y activación para el muestreo del sensor y envío de datos por mensajes BLE. El tiempo de sueño es un parámetro configurable y puede ser de 1 a 5 minutos para una aplicación final. Es decir, que se deben recibir datos de muestreo de los sensores de temperatura y humedad periódicamente cada 1 o 5 minutos.

En el servidor corre un programa en local de alta y configuración de sensores ESP32.

Una vez que los sensores ESP32 se dan de alta y se instalan en un punto físico del entorno, están periódicamente enviando datos al Gateway IoT y este a su vez al Server. El programa de alta y configuración de sensores se explica detalladamente en el apartado 4.5 de este capítulo.

4.2 Diseño Back-end NodeJS y BBDD del servidor

El Back-end es la parte que se conecta con la base de datos y el servidor, no es directamente accesible por los usuarios. Contiene la lógica de la aplicación.

A continuación se muestra el diagrama de bloques del funcionamiento del Back-end, que interactúa con el Mosquitto Hub y la Raspberry Pi:

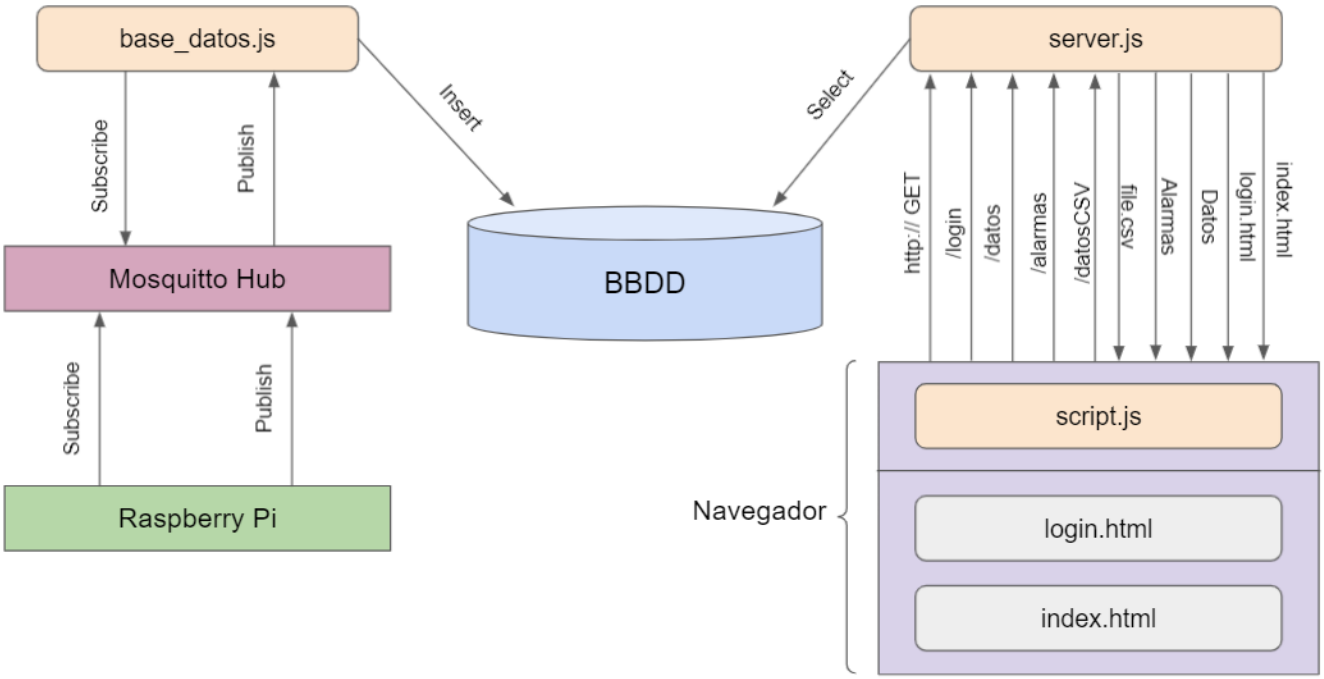


Figura 31. Diagrama de bloques e interconexión módulos del servidor

- La Raspberry Pi se conecta y suscribe al broker Mosquitto y por medio del protocolo MQTT manda tramas publish bajo el topic principal 'ALBA' o el subtopic de alarmas 'ALBA/ALARM', que contienen la información procesada de los mensajes advertisement que se han escaneado en el medio o el tipo de alarma detectada en el código de scan BLE respectivamente.

- Mosquitto Hub recibe la información publicada por la Raspberry y la manda a los dispositivos suscritos al mismo topic, en este caso al Back-end del servidor.

- El Back-end contiene el script **base_datos.js**, cuyo código se conecta al servidor Mosquitto, se suscribe al topic principal y escucha los mensajes que se le envían.
  Además, se conecta a la base de datos del servidor y cuando recibe los mensajes los inserta en la base de datos de SQLite en su correspondiente tabla.

- El Back-end también posee el script **server.js**, el cual escucha peticiones del Front-end que pueden venir de los usuarios que desean registrarse en la página para visualizar el contenido o para actualizar la página con nuevos datos de los sensores. Cuando escucha las peticiones devolverá la información solicitada en la petición.

- Si server.js recibe una petición tipo **/login**, quiere decir que un usuario se está intentando conectar a la página. Éste devolverá un archivo **login.html** con un script que se encarga de darle el contenido al html y que el usuario pueda registrarse.

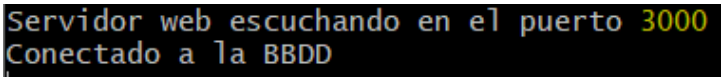
- Una vez registrado el usuario correctamente mediante login.html, server.js devolverá el archivo **index.html** que posee el script de visualización de la página principal de datos sensores. En ella se pueden realizar varias funciones que explicaremos a continuación. Si la autenticación es incorrecta devolverá de nuevo el archivo login.html.

- Si server.js recibe una petición tipo **/datos**, dicha petición deberá llevar hora y fecha del intervalo que se desea recibir y visualizar en la página. El servidor inicia una búsqueda SQL en la base de datos para encontrar las tramas demandadas y las devuelve al Front-end para su visualización en la tabla de datos sensores de la página.

- Cuando server.js recibe una petición tipo **/alarmas**, iniciará otra búsqueda SQL en la base de datos del servidor donde habrá una tabla de las alarmas establecidas y detectadas por el scan BLE de la Raspberry. Devolverá su búsqueda al Front-end para visualizarse en la tabla alarmas de la página.

- Si el servidor recibe una petición tipo **/datosCSV**, realizará una consulta en su base de datos con el intervalo de fecha y hora indicado en la petición, y creará un archivo .csv de los datos de los sensores solicitados que podrá ser descargado por el usuario.

Para inicializar el servidor, abrimos Git Bash en la carpeta donde se ubican todos los archivos del servidor y ejecutamos el comando `node server.js` y aparecerá el siguiente mensaje:



Inicializamos también el server Mosquitto –aunque este siempre está activo-, ejecutando PowerShell como administrador y dirigiéndonos a la carpeta donde está ubicado, en mi caso D:/Mosquitto y escribimos:

```
.\mosquitto_sub -d -h "192.168.1.106" -p 1883 -t "ALBA"
```

para subscribirnos y escuchar las tramas que le enviará la Raspberry.

Una vez puestos en marcha el servidor Mosquitto y nuestro servidor, base_datos.js estará conectada al broker, suscrita al topic y escuchando los mensajes que envíe el server Mosquitto.

Cuando base_datos.js reciba mensajes, los guardará en la base de datos del servidor para que server.js pueda obtenerlos y mandarlos al Front-end para su visualización en el momento que sean solicitados por el usuario.

4.3 Diseño del front-end e interfaz de usuario

En este apartado se explica cómo funciona el Front-end y la interfaz del usuario.

El Front-end es la parte que se ejecuta en el dispositivo del cliente, que contiene el código tanto para registrarse en la página como para una vez registrado, visualizar la información que la Raspberry Pi le envía al servidor, en concreto una tabla de datos sensores, con posibilidad de descarga seleccionando fecha y hora, y una tabla de aviso de alarmas que puedan ocasionarse.

A continuación se muestra el diagrama de flujo simplificado del Front-end:

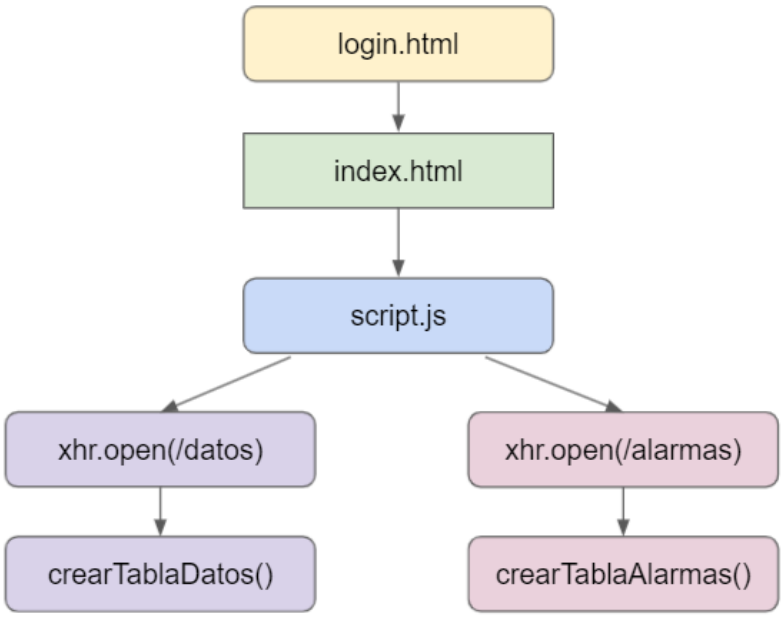


Figura 32. Diagrama de flujo Front-end

- El navegador ejecuta el código **login.html** que contiene el esqueleto de la página del registro de usuario, ahí el usuario introduce nombre y contraseña, que explicaremos en el siguiente apartado, y una vez verificado se ejecuta el código index.html.

- El código **index.html** es el que incluye el esqueleto de la página principal, el cual contiene un script interno de javascript script.js que se encarga de comprobar las peticiones del usuario. Index.html también proporciona la funcionalidad de generar un archivo CSV con la fecha y hora establecida por el usuario, y descargarlo mediante un botón.

- **Script.js** contiene el código que se comunica con el servidor para obtener la información sobre los datos de los sensores y las alarmas ocasionadas, y crear así dos tablas respectivamente que serán visualizadas en la página por el usuario.

La página principal se verá de esta forma:

Figura 33. Visualización página principal

Como podemos observar, la funcionalidad para generar el archivo CSV permite seleccionar la fecha y hora de inicio de búsqueda de archivos, y la fecha y hora de final de la búsqueda.

Una vez seleccionado y pulsado el botón de "Generar CSV" se mostrará otro botón de descarga del archivo:

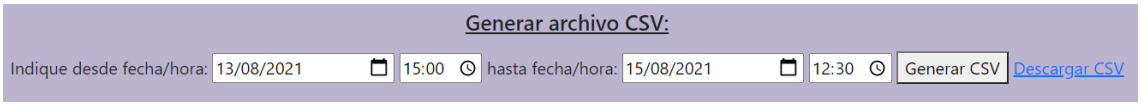


Figura 34. Generación y descarga de archivo CSV

Una vez pulsado el botón de descarga, se descargará el archivo CSV con las fechas y horas estipuladas en el ordenador del usuario.

Además, el script.js, que se ejecuta en el ordenador del usuario, proporciona la funcionalidad de recarga automática de la página con los nuevos datos obtenidos de los sensores cada 5 minutos, pudiendo visualizar los datos más recientes en la tabla.

4.4 Sistema de login de usuarios

En nuestra base de datos tenemos una lista de diez usuarios con el nombre de usuario y la contraseña cifrada con el método hash md5:

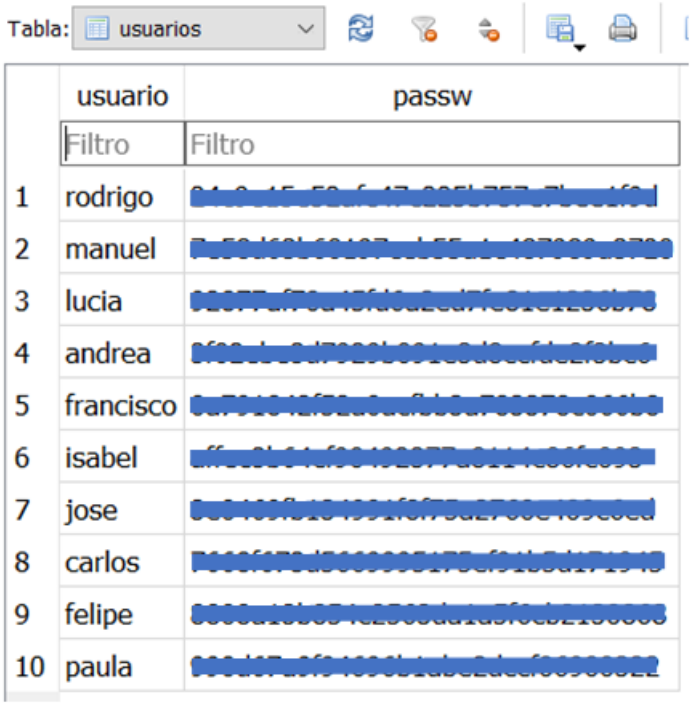


Figura 35. Lista de usuarios en base de datos del servidor

La visualización del login es la siguiente:

Figura 36. Visualización página de registro de usuario

El usuario deberá introducir su nombre y contraseña proporcionada para así acceder a la página principal index.html con la tabla datos sensores.

El código de server.js, que utilizamos para obtener los datos que son visualizados en la tabla de index.html, y obtener los datos para generar y descargar el archivo .csv, además utiliza Passport [17] para poder autenticar el usuario y contraseña.
Passport es un middleware de autenticación para Node.js, extremadamente flexible y modular. Passport puede incorporarse discretamente a cualquier aplicación web basada en Express. Un conjunto completo de estrategias admite la autenticación mediante un nombre de usuario y una contraseña.

- En el script de server.js se utilizan sesiones para establecer una cookie para cada usuario autenticado. Con Passport se puede serializar y deserializar el usuario.
- Una vez inicializado Passport y creada la sesión, se llamará a LocalStrategy con una función para comprobar que el usuario y contraseña introducidos son los correctos, primero cifrando la contraseña introducida mediante el método hash md5, un algoritmo de reducción criptográfico de 128 bits ampliamente usado, y comprobándolo posteriormente en la tabla de usuarios de la base de datos del servidor.
- La verificación de la llamada -callback- para la autenticación local acepta argumentos de nombre de usuario y contraseña, que se envían a la aplicación a través de un formulario de inicio de sesión, en nuestro caso login.html.
- De forma predeterminada, LocalStrategy espera encontrar credenciales en parámetros denominados nombre de usuario y contraseña.
- El formulario de inicio de sesión se envía al servidor a través del método POST. El uso de authenticate() con la estrategia local -LocalStrategy- manejará la solicitud de inicio de sesión.
- Si la autenticación es correcta se redirigirá a la página index.html, de lo contrario seguirá en login.html.

Para mayor robustez y seguridad en el login, si un usuario no autorizado -intruso- intenta acceder a index.html aparecerá esta página:

Figura 37. Visualización index.html para usuario no autorizado

Esta página es index.html, pero el usuario no autorizado no puede ver la tabla de datos sensores, y si intenta generar el archivo CSV y descargarlo, se le redirigirá a la página login.html.

4.5 Modulo de configuración y alta de sensores en BBDD

Se ha creado un código lista_sensores.py en el servidor, el cual se encarga de, cuando se conecte o instale por primera vez un módulo ESP32 en el ordenador del servidor, se configuren ciertos parámetros y se guarden en la base de datos, en la tabla lista_sensores junto con la tabla datos_sensores para su posterior visualización en la página.

El funcionamiento del código lista_sensores.py se muestra a continuación:

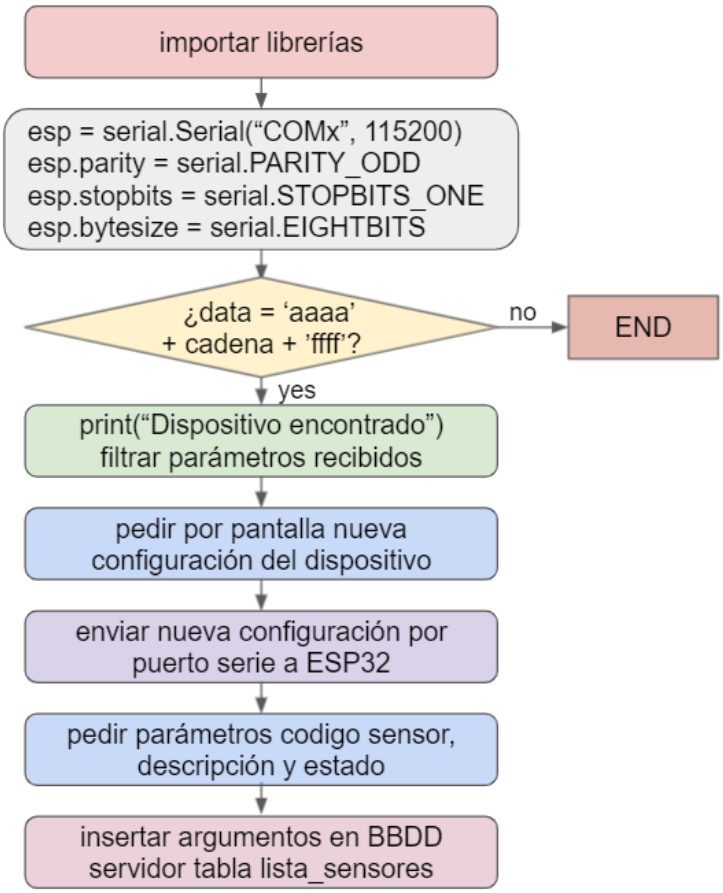


Figura 38. Diagrama de flujo funcionamiento lista_sensores.py

- Primero debemos importar las librerías serial, time, sqlite3 y numpy para el desarrollo del código.
- Declaramos el puerto serie al que se ha conectado el módulo ESP32 -COMx-, así como al baudrate, y los bits de paridad, si hay bits de stop y la longitud de los bits -8 o 9-.
- Inicializamos todas las variables que vamos a tomar del ESP32 y nos conectamos a la base de datos.

- El ESP32 una vez conectado a la corriente enviará durante 30 segundos por puerto serie su MAC y los parámetros de configuración configurados. Pasado ese tiempo escuchará durante otros 30 segundos para que se le pase otros parámetros: id, potencia de transmisión, número de advertisement, número de ciclos de muestreo, tiempo de sueño, intervalo advetisement y sensor de CO_2 (4 en nuestro caso). En caso de no recibir nada utilizará los que ya tenía configurados.
- El código de Python se encarga de recibir la MAC y los parámetros que envía el ESP32, filtrarlos y pedir por consola los nuevos parámetros a configurar.
- Una vez seleccionada la configuración que queremos introducir en el ESP32, se enviará por puerto serie al ESP32 y se esperará un intervalo de tiempo para que se configure.
- Cuando se realice la configuración con éxito se pedirá por pantalla además el código del sensor, descripción y estado -activo-.
- Los parámetros código del sensor, id, descripción, mac y estado se guardarán en la base de datos del servidor en la tabla lista_sensores.
- Posteriormente cuando el usuario acceda a la página, se seleccionarán el código, id y estado de lista_sensores, junto con temperatura, humedad y timestamp de datos_sensores para así añadirlo a la tabla que se visualizará en el Front-end.

4.6 Acceso remoto al servicio

Para poder acceder al servicio de visualización y descarga de datos de forma remota desde cualquier ordenador con un navegador con conexión a internet se ha implementado un DNS dinámico, basado en el servicio gratuíto No-IP, y en la redirección de puerto PNAT en el router. A continuación se describe la configuración del DNS dinámico con No-IP.

- Para ello debemos registrarnos en su página web oficial utilizando nuestro email, una vez registrado creamos el dominio:

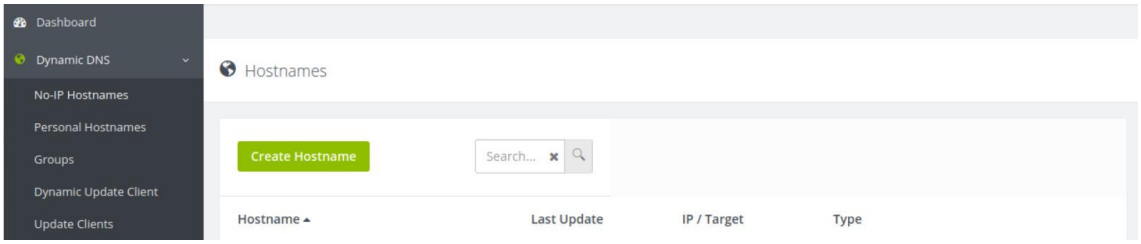


Figura 39. Visualización página No-IP

- Aparecerá esta ventana, donde crearemos nuestro host:

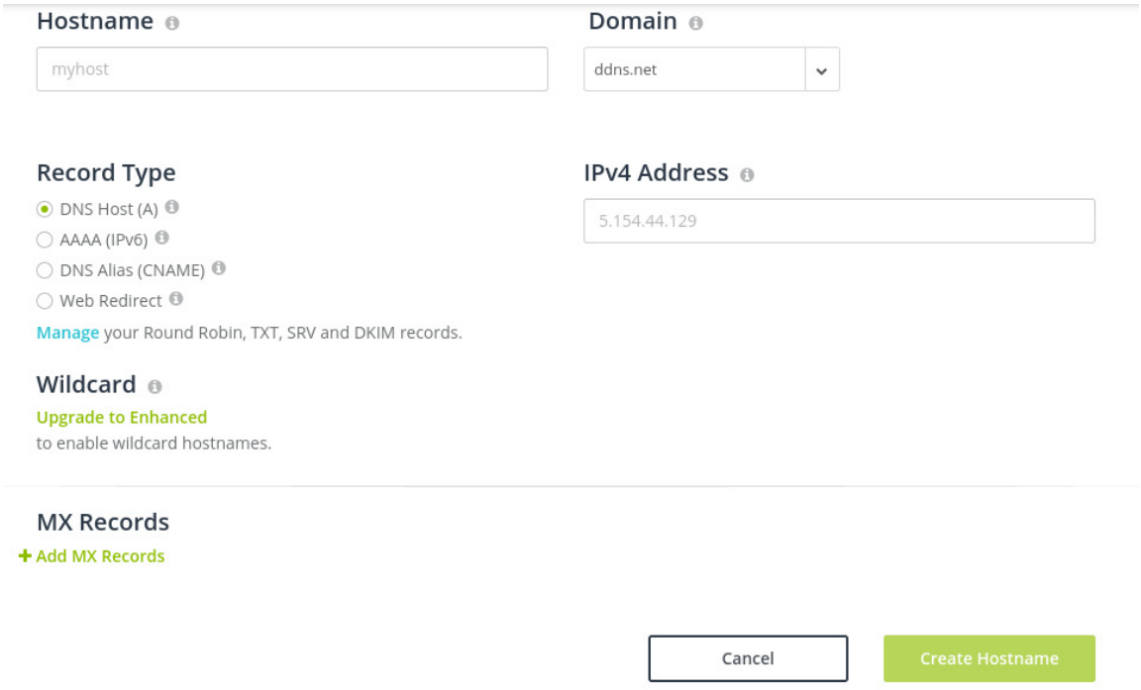


Figura 40. Creación host DNS

- Lo llamaremos ‘rsensortyh’ con dominio ddns.net

- Siendo un servicio gratuito, el nombre del dominio expirará en 30 días.

Figura 41. Expiración del host DNS

- Una vez creado el host, nos dirigimos a la interfaz de configuración del router de la red donde se encuentra el servidor, para ello se debe ingresar la dirección IP del router en el navegador.

    1) Entramos en la dirección de nuestro router

Figura 42. Entrada en el router

2) Nos dirigimos a la pestaña 'Avanzado'

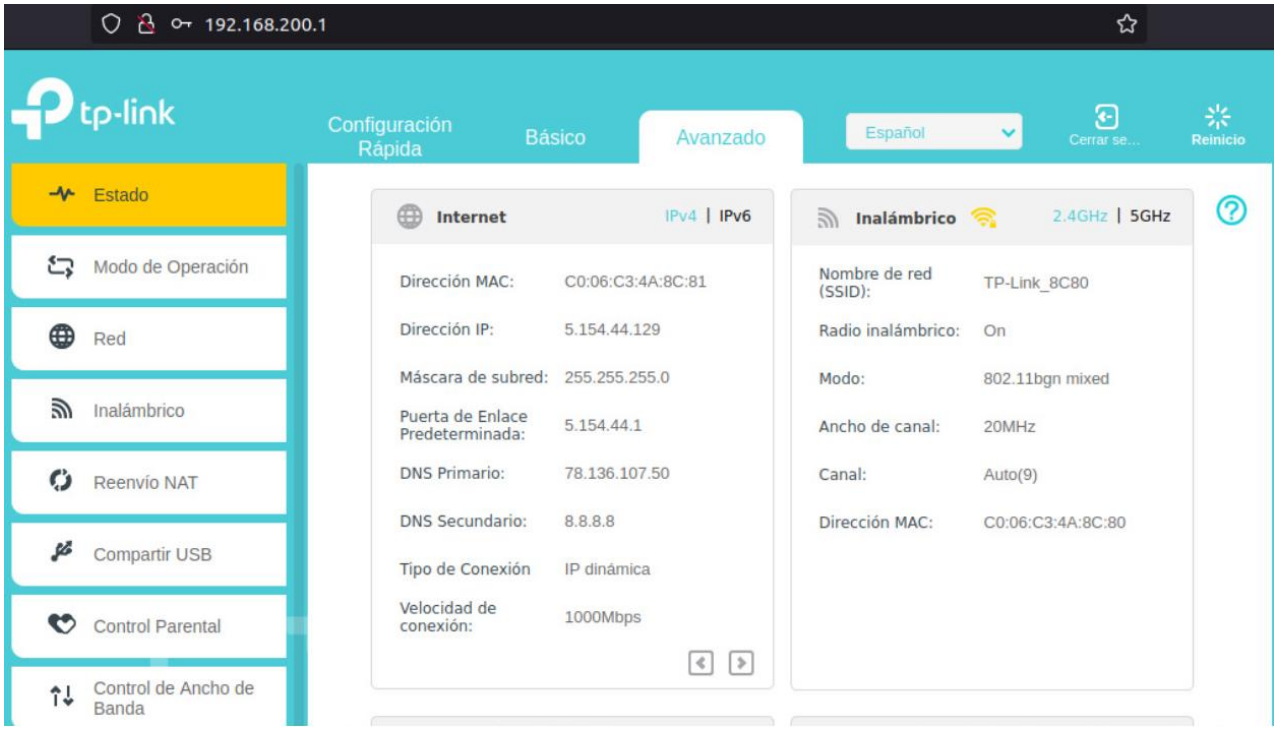


Figura 43. Pestaña avanzado del router

3) Vamos a Red -> DNS Dinámico y le damos a proveedor de servicio NO-IP, ponemos nuestro nombre de usuario creado y el nombre de dominio creado:

Figura 44. Configuración DNS en router

4) Le damos al botón 'Iniciar sesión', una vez pulsado aparecerá 'Éxito'

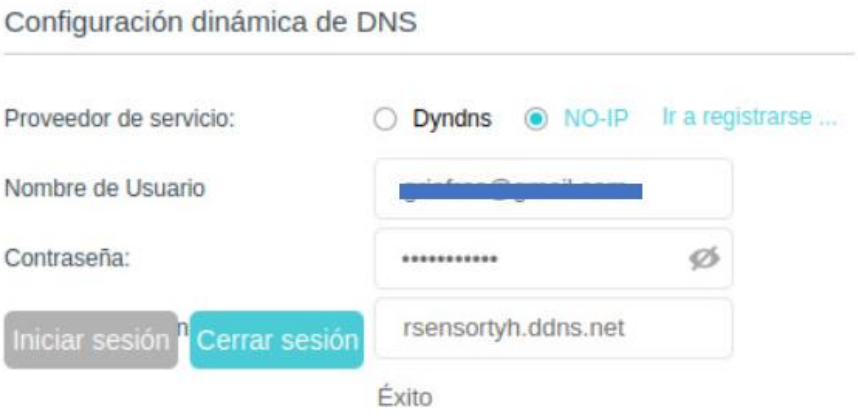


Figura 45. Configuración DNS en router - éxito

5) El router ya tiene el servicio DNS configurado, ahora vamos a configurar NAT para redirigir puertos. Nos vamos a Reenvío NAT -> Servidores Virtuales

Figura 46. Servidores virtuales router

6) Añadimos un servidor virtual.
Escribimos el nombre de la interfaz, el tipo de servicio HTTP, el puerto externo –por defecto 80 para las páginas web-, la IP interna del dispositivo de la red interna donde se encuentra alojado el servidor, el puerto interno –puerto en el que está escuchando el dispositivo en cuestión- y el protocolo TCP:

Figura 47. Añadir servidor virtual en router

7) Una vez guardado aparecerá la siguiente tabla:

Figura 48. Servidor virtual en router añadido

Una vez realizado esto, tendremos la dirección 'http://rsensortyh.ddns.net' lista para visualizar los datos de los sensores captados por la Raspberry.

Capítulo 5. Pruebas y resultados

Para la ejecución de las pruebas de validación y testeo ha sido necesaria la Raspberry Pi que actúa como Gateway, un ordenador cuya función es servidor, y tres módulos ESP32 con sus respectivos sensores de temperatura y humedad. Las pruebas se han diseñado para responder a las siguientes cuestiones:

- Evaluación empírica de la distancia máxima en la que pueden estar los sensores ESP32 del Gateway IoT, para poder planificar la ubicación de sensores ESP32 en un despliegue real.
- Validar que el sistema en conjunto funciona según los requisitos y el flujo de datos es correcto y adecuado.
- Comprobar que el sistema es robusto emulando una alta carga de datos y es robusto con el paso del tiempo cuando esté funcionando días o semanas.
- Identificar puntos débiles para mejorar en próximas versiones incrementales.

5.1 Pruebas de cobertura y alcance

Para la realización de estas pruebas se han utilizado tres módulos ESP32, cada uno conectado a un sensor con su respectiva ID grabada en el programa del ESP32, el Gateway –rol desempeñado por la Raspberry- con el código de SCAN BLE para escanear los mensajes advertisement que envía el ESP32 y un ordenador que actúa como servidor para monitorizar las tramas enviadas a la base de datos.

Cada uno de los ESP32, encargados de enviar los datos de temperatura y humedad que captan los sensores, estarán conectados a una batería y posicionados en distintos lugares de la casa, para así poder medir la cobertura y alcance del escáner de tramas de la Raspberry, la cual estará ubicada en una de las habitaciones donde posicionaremos dos módulos.

Para esta prueba se han mantenido los dispositivos fijos hasta la finalización de esta. Se han configurado las IDs de cada ESP32 y se ha seleccionado el lugar de ubicación de cada módulo de la siguiente forma:

- ESP sensor ID1: Sobre una escalera elevada, a 5.2m de distancia de la Raspberry.
- ESP sensor ID2: En la habitación contigua al salón, a la derecha, con una pared de por medio. A 4m de distancia.
- ESP sensor ID3: Mesa baja cerca de la Raspberry, a una distancia de 3m entre ellos.

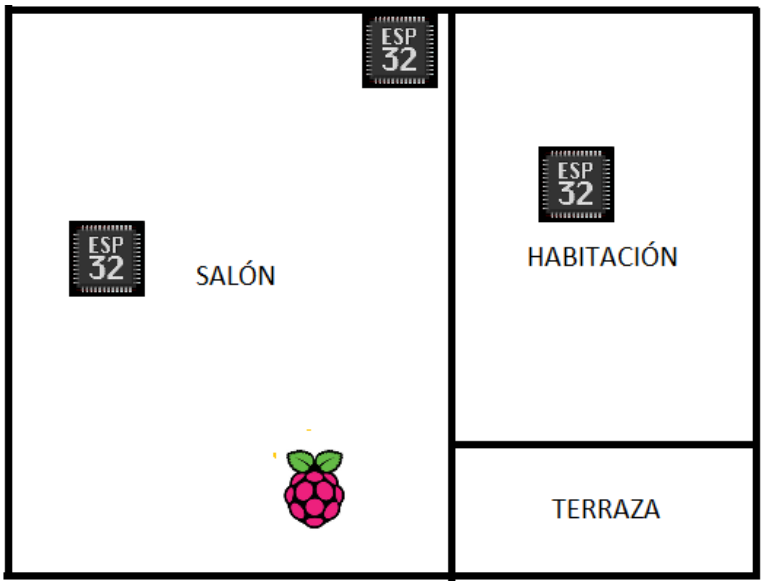


Figura 49. Plano distribución sensores ESP32 y Raspberry

Sus posiciones y cómo están conectados los módulos los podemos observar en las siguientes figuras:

Figura 50. Escenario de pruebas de cobertura

En el salón están ubicados dos de los módulos ESP32 junto con la Raspberry. En el lado derecho de la foto, en otra habitación se ubica otro módulo ESP32.

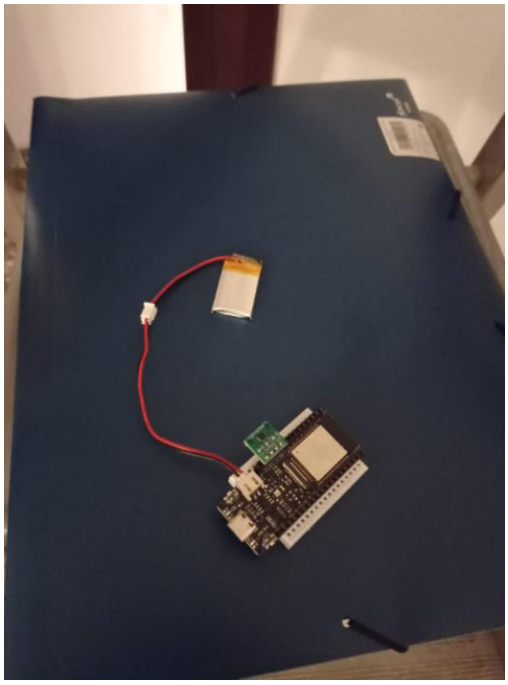


Figura 51. Ubicación ESP32 con sensor ID1

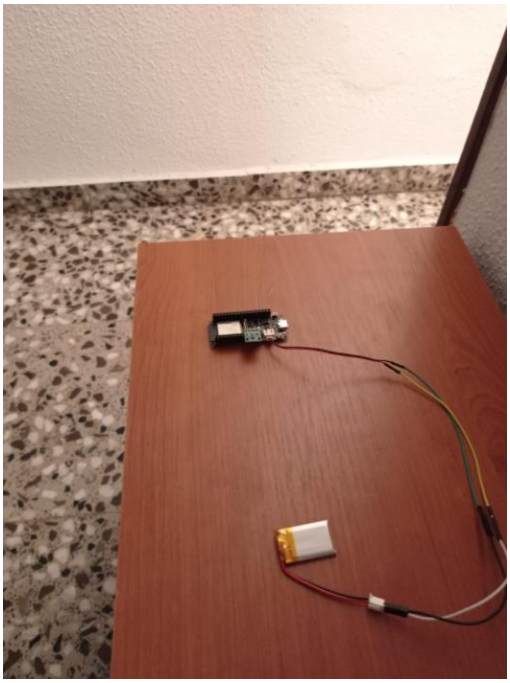


Figura 52. Ubicación ESP32 con sensor ID2

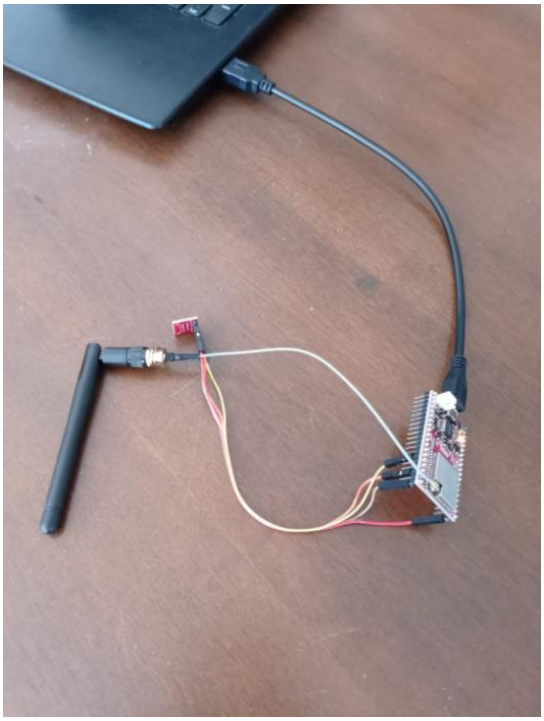


Figura 53. Ubicación ESP32 con sensor ID3

En el siguiente cronograma se muestra el funcionamiento:

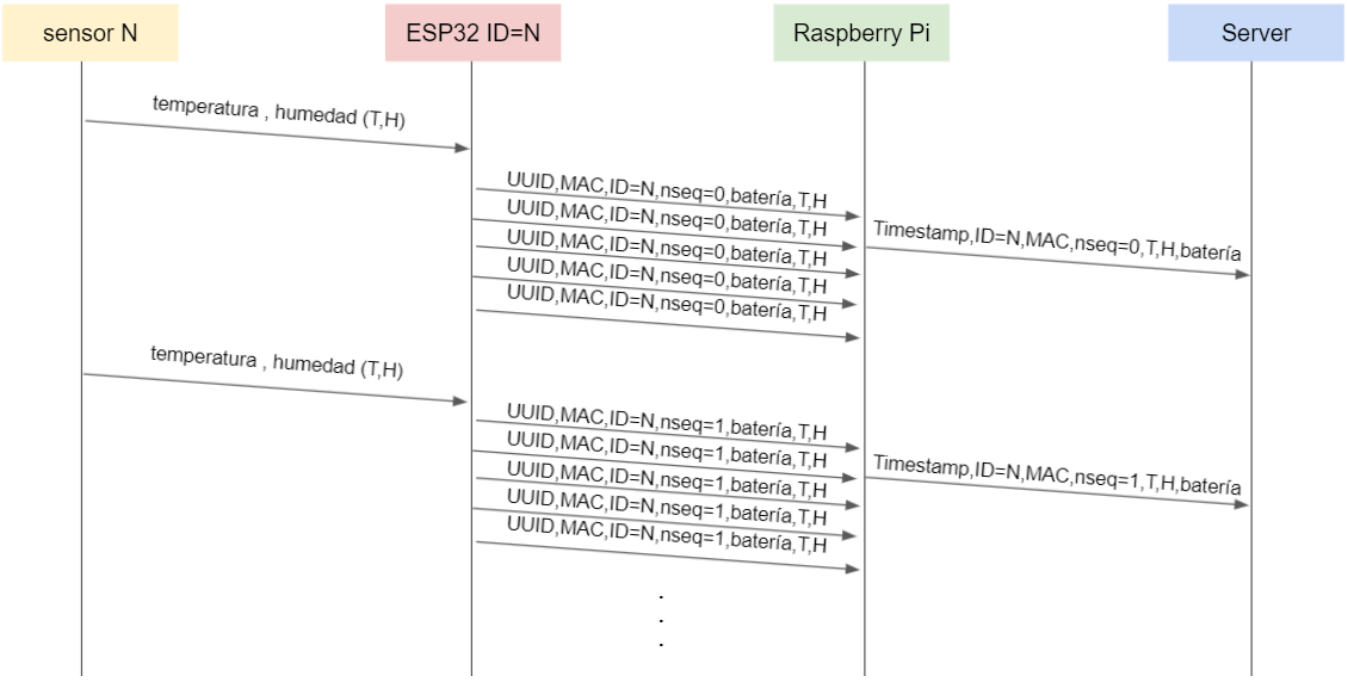


Figura 54. Cronograma envío de tramas ESP32-Raspberry-servidor, prueba cobertura 1

En esta prueba, cada sensor tiene un **tiempo de sueño de 20 segundos**, pasado ese tiempo despertará y enviará **5 tramas advertisements iguales durante un segundo** -con mismo número de secuencia- a 0,1 segundos cada trama. Una vez enviadas las 5 tramas iguales durante 1 segundo, dormirá otros 20 segundos para volver a transmitir incrementando el número de secuencia.

Estas tramas enviadas por los módulos ESP32 conectados a los sensores, son escaneadas por la Raspberry, procesadas para sólo coger una trama de las 5 iguales recibidas y enviadas al servidor, donde guardará la información en su base de datos.

Se realizaron varias mediciones de 20 minutos cada una.

- Medición realizada desde las 17:24:30 hasta las 17:45:35

Nos dirigimos a DB Browser y realizamos la siguiente consulta SQL:

SELECT * FROM datos_sensores WHERE timestamp BETWEEN "2021-09-03 17:24:30" AND "2021-09-03 17:45:35" ORDER BY id_sensor

Esta consulta nos permite ver hasta cuántos números de secuencia distintos ha enviado cada sensor en el periodo de 20 minutos. En esta medición son 58 números de secuencia enviados.
Tenemos 3 IDs, y cada ID ha enviado hasta 58 números de secuencia.
Teóricamente se deben recibir:
$3IDs\ x\ 58nseq\ =\ 174\ números\ de\ secuencia\ totales.$

Realizamos la consulta anterior cambiando * por COUNT(num_sec) para visualizar cuántos números de secuencia se han recibido en total. Obtenemos:

$$COUNT(num\_sec) = 137\ números\ de\ secuencia\ recibidos\ en\ la\ prueba.$$

Calculamos el porcentaje de pérdidas en la tercera medición:

$$174\ números\ de\ secuencia\ totales - 137\ números\ de\ secuencia\ recibidos = 37\ números\ de\ secuencia\ perdidos.$$

Perdemos 37 números de secuencia de 174 totales → $21\%\ de\ pérdidas.$

→Realizamos esta misma prueba por la noche. El funcionamiento es exactamente el mismo, sólo hemos variado la hora de medición.

- Medición realizada desde las 22:20:00 hasta las 22:40:04

Nos dirigimos a DB Browser y realizamos la siguiente consulta SQL:

SELECT * FROM datos_sensores WHERE timestamp BETWEEN "2021-09-03 22:20:00" AND "2021-09-03 22:40:04" ORDER BY id_sensor

Esta consulta nos permite ver hasta cuántos números de secuencia distintos ha enviado cada sensor en el periodo de 20 minutos. En esta medición son 55 números de secuencia enviados.
Tenemos 3 IDs, y cada ID ha enviado hasta 55 números de secuencia.
Teóricamente se deben recibir:

$$3IDs\ x\ 55nseq = 165\ números\ de\ secuencia\ totales.$$

Realizamos la consulta anterior cambiando * por COUNT(num_sec) para visualizar cuántos números de secuencia se han recibido en total. Obtenemos:

$$COUNT(num\_sec) = 141\ números\ de\ secuencia\ recibidos\ en\ la\ prueba.$$

Calculamos el porcentaje de pérdidas en la cuarta medición:

$$165\ números\ de\ secuencia\ totales - 141\ números\ de\ secuencia\ recibidos = 24\ números\ de\ secuencia\ perdidos.$$

Perdemos 24 números de secuencia de 165 totales → $14.5\%\ de\ pérdidas.$

Como podemos observar, obtenemos de un 21% a un 14.5% de pérdida de tramas.

Estas pérdidas pueden deberse en parte al gran número de dispositivos Bluetooth que se promocionan al medio, pues nuestro escáner recibe tramas de todos los dispositivos BLE a su alcance, pero sólo procesa y guarda los que tenemos definidos, cuyo UUID posee los bytes FFE2.

Al recibir tales cantidades de tramas al mismo tiempo, el buffer interno de la Raspberry no consigue obtener todas las tramas que le hemos enviado, pues pueden solaparse con alguna trama desconocida que debemos descartar.

Digamos que, enviando 5 tramas advertisement iguales a 0,1 segundo cada una durante 1 segundo, la Raspberry puede llegar a descartar tramas las cuales nos interesan, ya que al recibirse junto con tramas desconocidas el buffer se llena produciendo solapamiento.

Probablemente si aumentamos el número de tramas iguales que se envían en un segundo, la Raspberry sea capaz de obtener al menos una de ellas.

→Realizamos el cambio de número de tramas a enviar por el ESP32, en vez de 5 advertisements iguales, enviaremos **10** tramas advertisement iguales a 0,1 segundos cada una durante 1 segundo. Después el ESP32 dormirá durante 20 segundos y una vez despierte incrementará el número de secuencia. Esta prueba la realizamos para comprobar si se pierden menos tramas que en las anteriores mediciones incrementando el número de advertisements enviados. El funcionamiento es similar a las pruebas anteriores y se muestra en el siguiente cronograma:

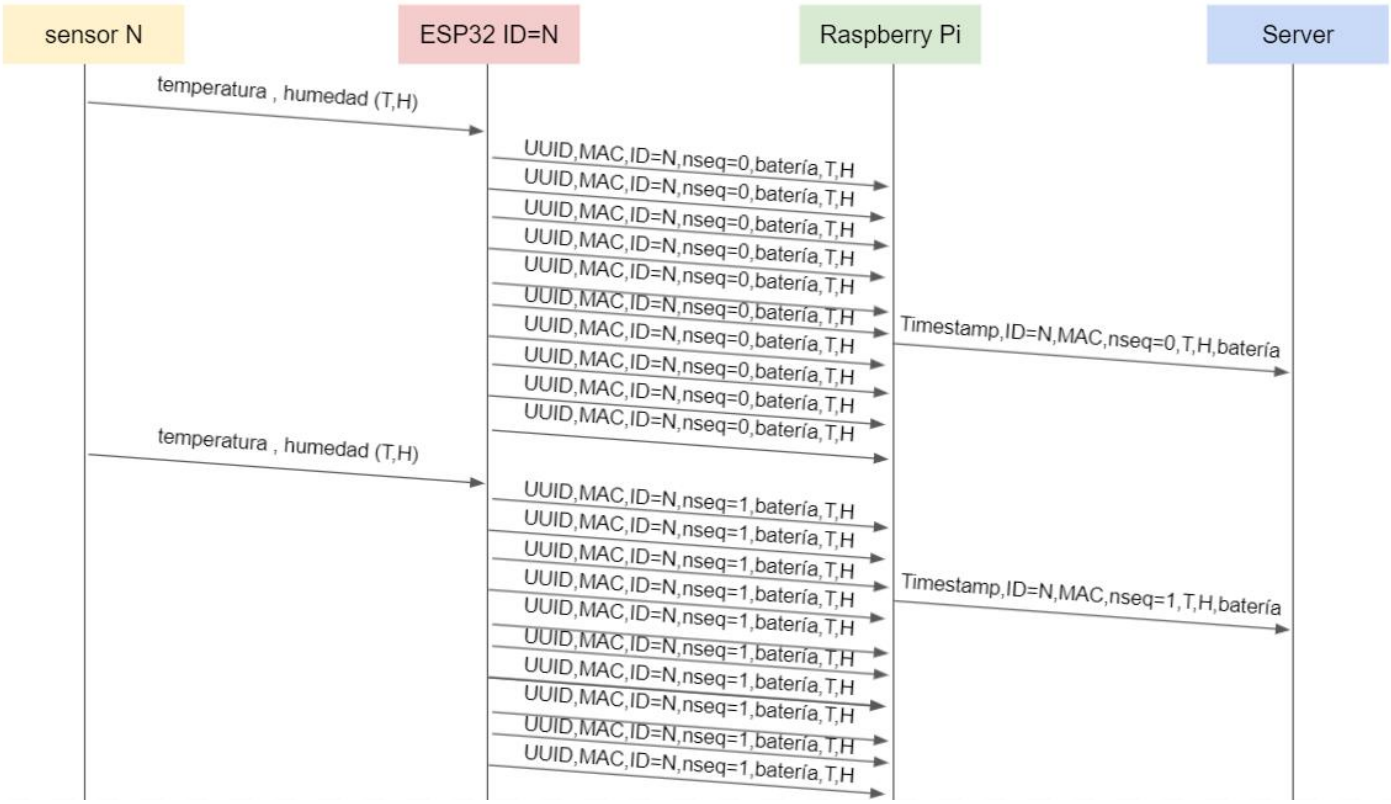


Figura 55. Cronograma envío de tramas ESP32-Raspberry-servidor, prueba cobertura 2

En esta prueba, cada sensor tiene un **tiempo de sueño de 20 segundos**, pasado ese tiempo despertará y enviará **10 tramas advertisements iguales durante un segundo** –con mismo número de secuencia- a 0,1 segundos cada trama. Una vez enviadas las 10 tramas iguales durante 1 segundo, dormirá otros 20 segundos para volver a transmitir incrementando el número de secuencia.

Estas tramas enviadas por los módulos ESP32 conectados a los sensores, son escaneadas por la Raspberry, procesadas para sólo coger una trama de las 10 iguales recibidas y enviadas al servidor, donde guardará la información en su base de datos.

Prueba de cobertura cambiando a 10 tramas advertisements iguales por segundo:

- Medición realizada desde las 22:49:50 hasta las 23:09:22

Nos dirigimos a DB Browser y realizamos la siguiente consulta SQL:

SELECT * FROM datos_sensores WHERE timestamp BETWEEN "2021-09-03 22:49:50" AND "2021-09-03 23:09:22" ORDER BY timestamp

Esta consulta nos permite ver hasta cuántos números de secuencia distintos ha enviado cada sensor en el periodo de 20 minutos.
Al principio de la prueba nos dimos cuenta de que el sensor con ID1 no estaba bien conectado a su batería, lo conectamos de nuevo y este empezó a transmitir a las 22:53:11, por lo tanto para esta medición la base de datos recibió hasta 53 números de secuencia de los IDs 2 y 3 respectivamente, y hasta 44 números de secuencia de ID1.

Teóricamente se deben recibir:
$2 IDs\ x\ 53 nseq\ =\ 106\ números\ de\ secuencia\ entre\ ID2\ e\ ID3.$
Sumados a los 44 números de secuencia recibidos por ID1:

$$106\ números\ de\ secuencia\ ID2\ e\ ID3 + 44\ números\ de\ secuencia\ ID1 = 150\ números\ de\ secuencia\ totales$$

Realizamos la consulta anterior cambiando * por COUNT(num_sec) para visualizar cuántos números de secuencia se han recibido en total. Obtenemos:

$$COUNT(num\_sec)\ =\ 149\ números\ de\ secuencia\ recibidos\ en\ la\ prueba.$$

Calculamos el porcentaje de pérdidas de esta medición:

$$150\ números\ de\ secuencia\ totales - 149\ números\ de\ secuencia\ recibidos = 1\ número\ de\ secuencia\ perdido.$$

Perdemos 1 número de secuencia de 150 totales → $\boxed{0.66\%\ de\ pérdidas.}$

Como podemos observar, se han reducido considerablemente las pérdidas gracias a un mayor envío de tramas iguales -10adv-, que han podido ser captadas por la Raspberry y procesadas.

→Se ha realizado además una prueba de cobertura en la que **se varía la distancia** entre el transmisor -sensor conectado a ESP32- y el receptor -Raspberry- para comprobar a qué distancia máxima se puede ubicar el sensor respecto a la Raspberry para que haya suficiente cobertura entre ellos.

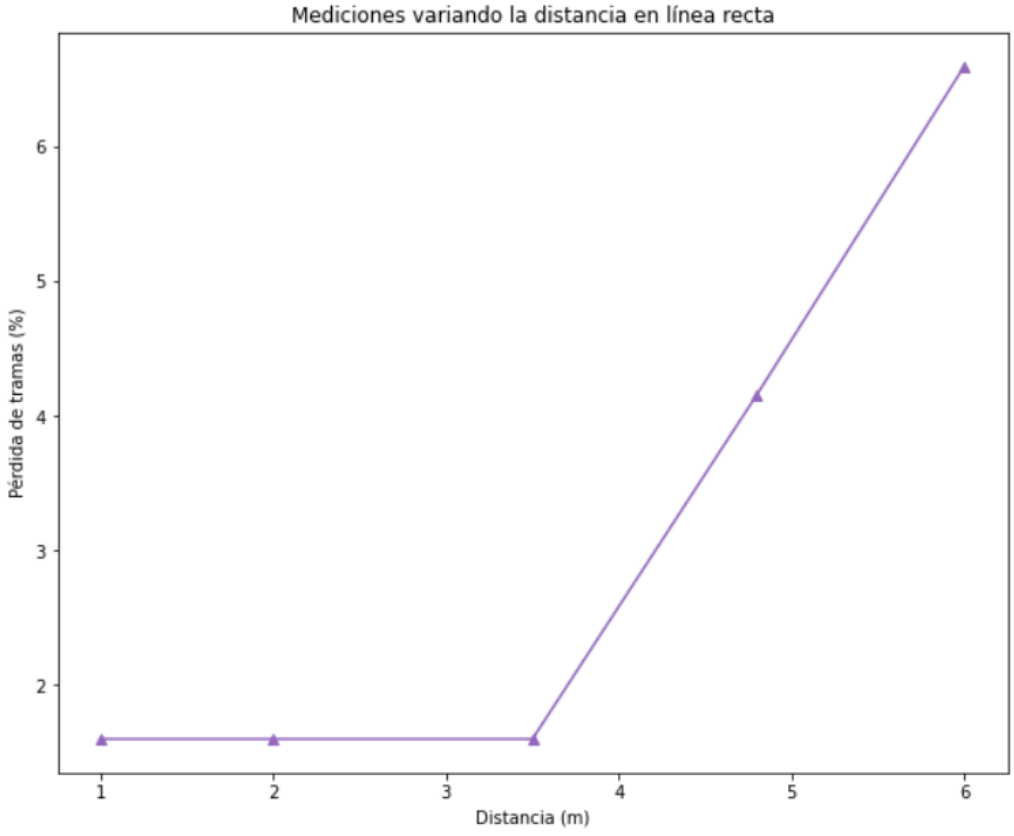
El cronograma del funcionamiento es el mismo que el de la anterior figura 55.

En esta prueba utilizaremos un módulo ESP32 trucado emulando **20 IDs que transmite constantemente -sin tiempo de sueño- cada 0,1 segundos, 10 tramas advertisements iguales durante un segundo.**

Cada ID enviará 10 tramas iguales en un segundo, pasado ese tiempo se incrementará el número de secuencia de cada ID y seguirá transmitiendo.

Se harán varias pruebas de 2 minutos cada una variando la distancia entre emisor y receptor. La antena de la Raspberry se encuentra orientada apuntando al ordenador que sostiene el módulo ESP32 trucado.

Se realizaron 5 pruebas variando la distancia y este es el resultado obtenido:



Gráfica 1. Mediciones variando la distancia en línea recta

1) **Distancia de 1m** → $1.6\%\ de\ pérdidas$.
2) **Distancia de 2m** → $1.6\%\ de\ pérdidas$.
3) **Distancia de 3'5m** → $1.6\%\ de\ pérdidas$.
4) **Distancia de 4'8m** → $4.16\%\ de\ pérdidas$.
5) **Distancia de 6m** → $6.6\%\ de\ pérdidas$.

Como podemos observar en la gráfica 1, si colocamos un sensor a una distancia máxima de 3'5 metros, la pérdida de tramas es minúscula -1.6% de pérdida en 2 minutos- pudiendo perder hasta 2 tramas de 120 enviadas.

Además debemos tener en cuenta que esta prueba se ha realizado con un módulo que emula 20 sensores -IDs- sin tiempo de sueño, enviando constantemente cada segundo, por lo tanto es probable que algún número de secuencia -o trama- se haya intentado enviar al mismo tiempo que cualquier otro dispositivo que no estamos midiendo y se haya causado un solapamiento, pues el escáner de la Raspberry detecta cualquier trama de dispositivos BLE.

Es probable que el buffer interno de la Raspberry no haya conseguido obtener todas las tramas, pues algunas pueden solaparse con las de otros dispositivos BLE, aparte de que, una vez la Raspberry recibe una trama de un ID con un número de secuencia, lo guarda en su buffer y descarta todas las demás que le lleguen iguales, por ello si dentro del buffer se produce solapamiento, la Raspberry igualmente ya habría recibido la trama y no va a volver a guardarla si es la misma.

Podemos observar también que, a distancias superiores a 5 metros, las pérdidas de tramas pueden llegar a ser del 6.6%. Esto se puede deber a las interferencias que ocurren en el medio, así como la gran cantidad de dispositivos que pueden estar promocionándose en el medio y la Raspberry debe descartar, pudiendo así perder alguna trama que realmente necesitemos, pues pueden solaparse las tramas 'intrusas' con nuestras tramas y descartarse.

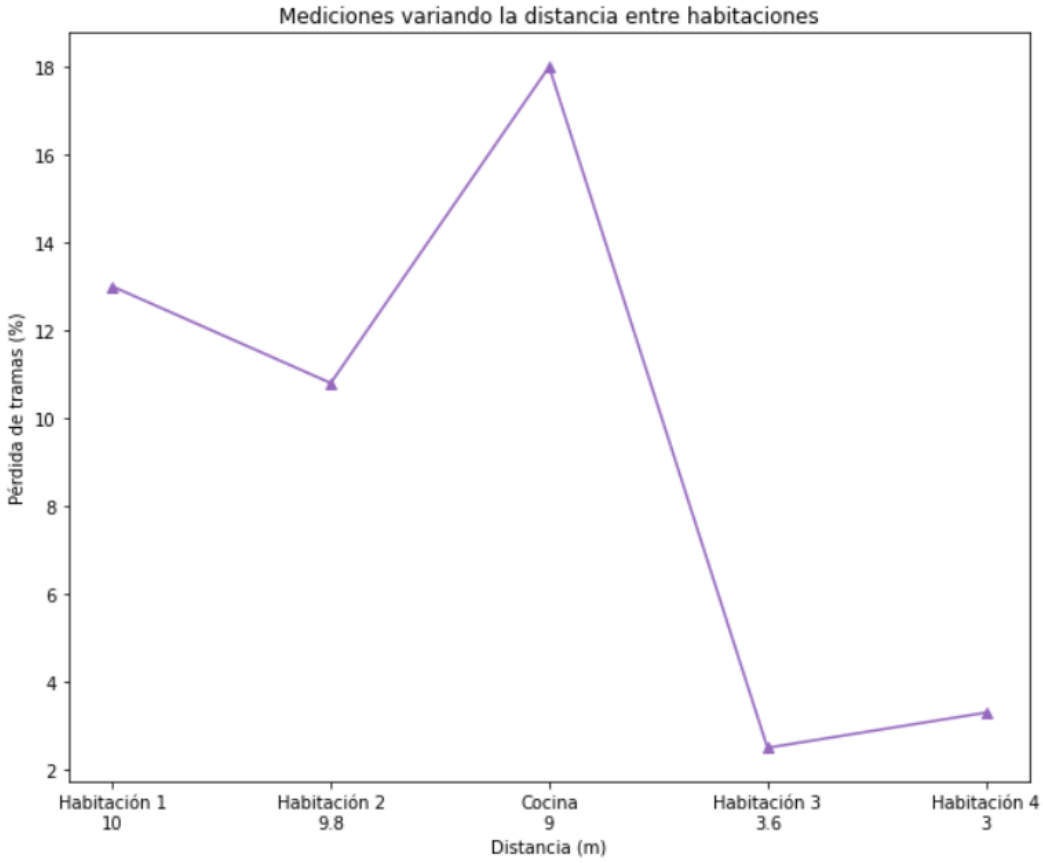
→Realizamos ahora varias mediciones **variando la distancia entre emisor y receptor** ubicando los sensores en **distintas habitaciones con paredes** de por medio entre ellos.

Estas mediciones siguen el mismo patrón que las anteriores mencionadas: se envían cada segundo 10 tramas iguales por ID y pasado ese segundo se incrementa el número de secuencia de cada ID. Mediciones de duración 2 minutos.

1) **En una habitación a 10m** → $13\%\ de\ pérdidas.$
2) **En una habitación a 9.8m** →$10.8\%\ de\ pérdidas.$
3) **Cocina, a 9m** → $18\%\ de\ pérdidas.$
4) **En una habitación a 3.6m** →$18\%\ de\ pérdidas.$ *
   Reubicando antena → $2.5\%\ de\ pérdidas.$
5) **En una habitación a 3m** → $20\%\ de\ pérdidas.$ *
   Reubicando antena →$3.3\%\ de\ pérdidas.$

*Este gran porcentaje de pérdidas se debe a que la antena está orientada a las habitaciones las cuales hemos medido anteriormente -de frente-, sin embargo esta habitación se encuentra en paralelo a la antena.
Reubicamos la antena de la Raspberry de modo que esté orientada hacia la habitación que medimos en este momento.

Obtenemos la siguiente gráfica:



Gráfica 2. Mediciones variando la distancia entre habitaciones

Como observamos en la gráfica 2, en estas dos últimas habitaciones, si reorientamos la antena hacia ellas, obtendremos mejores resultados y menores pérdidas.

Cabe destacar además que la atenuación de la cocina puede deberse a que las paredes están decoradas con azulejo, los cuales son capaces de refractar la luz, por lo que es posible que también sean capaces de que la señal proveniente de la cocina se refracte y refleje, provocando una atenuación mayor que dificulte la captación de las tramas que se envían desde allí. Se tiene en dicha cocina un horno-microondas que funciona a una frecuencia de 2.4GHz -la misma que utiliza bluetooth- por lo que también es un atenuante en el nivel de señal.

Metales, azulejos y baldosas hacen que la señal sea menor.

5.2 Pruebas de validación del funcionamiento del sistema

Se realizan varias pruebas de validación del funcionamiento del sistema desarrollando un programa en ESP que simula ciertos sensores que envían **variando su ID cada segundo y sin tiempo de sueño, durante 10 minutos.**

Para este programa hemos utilizado tres módulos ESP32 con IDs:

ESP1: IDs 1-20
ESP2: IDs 21-40
ESP3: IDs 41-60

En total tenemos 3 módulos ESP32 emulando 60 sensores -20 IDs cada uno-.

Estos sensores se encuentran a 1m de distancia de la Raspberry. El cronograma es el siguiente:

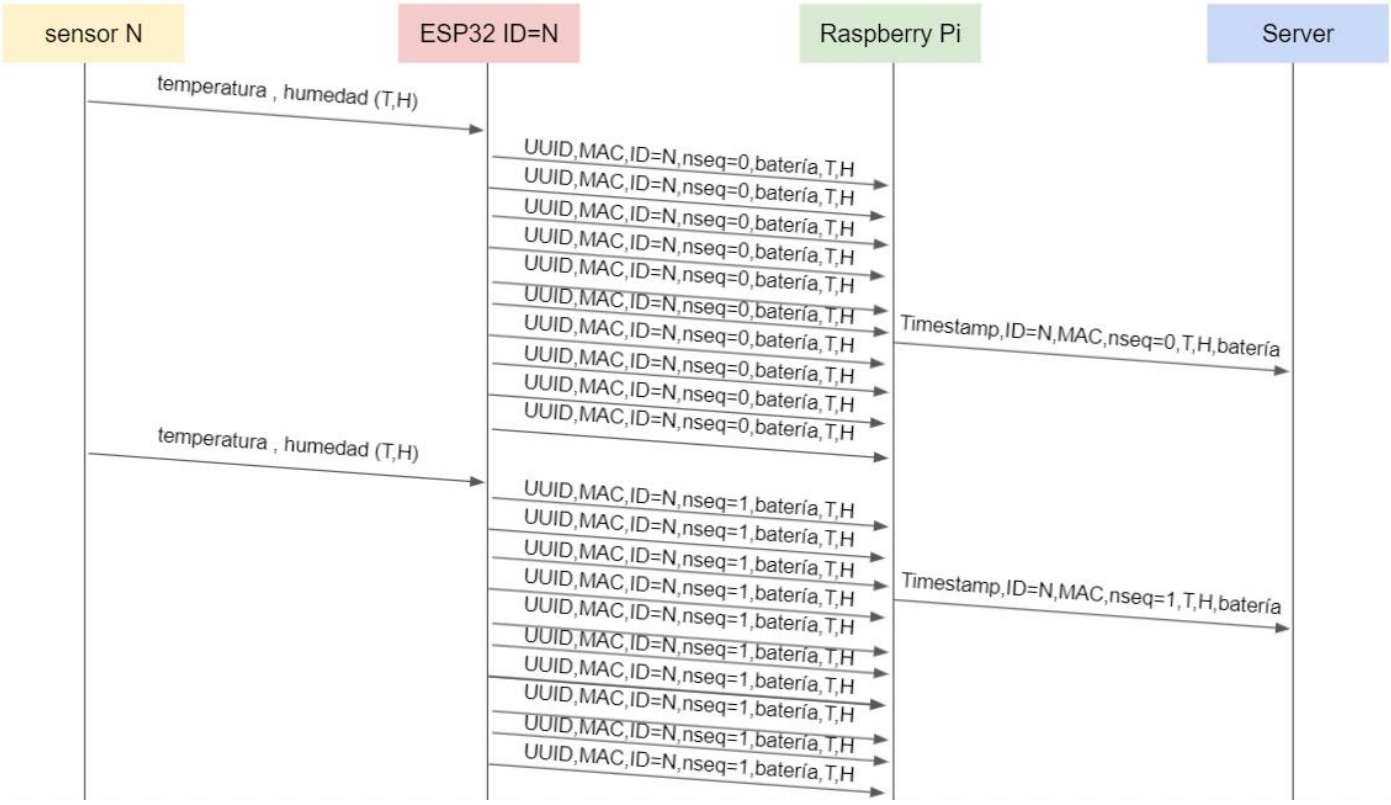


Figura 56. Cronograma envío de tramas ESP32-Raspberry-servidor, pruebas validación

Como vemos en el cronograma, **cada ID envía 10 advertisements iguales durante 1 segundo**, a 0,1 segundos cada uno.

Estas tramas enviadas por los módulos ESP32 conectados a los sensores, son escaneadas por la Raspberry, procesadas para sólo coger una trama de las 10 iguales recibidas y enviadas al servidor, donde guardará la información en su base de datos.

- Medición realizada desde las 13:10:00 hasta las 13:21:04

Nos dirigimos a DB Browser y realizamos la siguiente consulta SQL:

SELECT * FROM datos_sensores WHERE timestamp BETWEEN "2021-09-01 13:10:00" AND "2021-09-01 13:21:04" ORDER BY id_sensor

Esta consulta nos permite ver hasta cuántos números de secuencia distintos ha enviado cada sensor en el periodo de 11 minutos. En esta medición son 30 números de secuencia enviados.
Tenemos 60 IDs, y cada ID ha enviado hasta 30 números de secuencia.
Teóricamente se deben recibir:

$$60IDs\ x\ 30nseq\ =\ 1800\ números\ de\ secuencia\ totales.$$

Realizamos la consulta anterior cambiando * por COUNT(num_sec) para visualizar cuántos números de secuencia se han recibido en total. Obtenemos:

$$COUNT(num\_sec)\ =\ 1783\ números\ de\ secuencia\ recibidos\ en\ la\ prueba.$$

Calculamos el porcentaje de pérdidas en esta medición:

$$1800\ números\ de\ secuencia\ totales\ -\ 1783\ números\ de\ secuencia\ recibidos\ =\ 17\ números\ de\ secuencia\ perdidos.$$

Perdemos 17 números de secuencia de 1800 totales → $\boxed{0.94\%\ de\ pérdidas.}$

Vamos a realizar otra medición. El funcionamiento es exactamente el mismo, sólo hemos variado la hora de medición.

- Medición realizada desde las 20:30:21 hasta las 20:40:51

Nos dirigimos a DB Browser y realizamos la siguiente consulta SQL:

SELECT * FROM datos_sensores WHERE timestamp BETWEEN "2021-09-06 20:30:21" AND "2021-09-06 20:40:51" ORDER BY id_sensor

Esta consulta nos permite ver hasta cuántos números de secuencia distintos ha enviado cada sensor en el periodo de 10 minutos. En esta medición son 30 números de secuencia enviados.
Tenemos 60 IDs, y cada ID ha enviado hasta 30 números de secuencia.
Teóricamente se deben recibir:

$$60IDs\ x\ 30nseq\ =\ 1800\ números\ de\ secuencia\ totales.$$

Realizamos la consulta anterior cambiando * por COUNT(num_sec) para visualizar cuántos números de secuencia se han recibido en total. Obtenemos:

$$COUNT(num\_sec)\ =\ 1789\ números\ de\ secuencia\ recibidos\ en\ la\ prueba.$$

Calculamos el porcentaje de pérdidas en esta medición:

$$1800\ números\ de\ secuencia\ totales\ -\ 1789\ números\ de\ secuencia\ recibidos\ =\ 11\ números\ de\ secuencia\ perdidos.$$

Perdemos 11 números de secuencia de 1800 totales → $\boxed{0.61\%\ de\ pérdidas.}$

Obtenemos un porcentaje de pérdidas muy bajo, el cual quiere decir que el sistema funciona correctamente.

→Realizamos una prueba de 5 minutos de duración en la que se intenta sobrecargar el sistema. Para ello utilizamos un módulo ESP32 que emula **10 IDs que envían 10 tramas iguales con el mismo número de secuencia cada segundo y sin tiempo de sueño**.

El cronograma de envío de mensajes es el mismo que el de la anterior figura 49.

- Medición realizada desde las 13:36:12 hasta las 13:42:08

Nos dirigimos a DB Browser y realizamos la siguiente consulta SQL:

SELECT * FROM datos_sensores WHERE timestamp BETWEEN "2021-09-03 13:36:12" AND "2021-09-03 13:42:08" ORDER BY id_sensor

Esta consulta nos permite ver hasta cuántos números de secuencia distintos ha enviado cada sensor en el periodo de 5 minutos. En esta medición son 35 números de secuencia enviados.
Tenemos 10 IDs, y cada ID ha enviado hasta 35 números de secuencia.
Teóricamente se deben recibir:

$$10IDs\ x\ 35nseq\ = 350\ números\ de\ secuencia\ totales.$$

Realizamos la consulta anterior cambiando * por COUNT(num_sec) para visualizar cuántos números de secuencia se han recibido en total. Obtenemos:

$$COUNT(num\_sec)\ =\ 347\ números\ de\ secuencia\ recibidos\ en\ la\ prueba.$$

Calculamos el porcentaje de pérdidas en la primera medición:

$$350\ números\ de\ secuencia\ totales - 347\ números\ de\ secuencia\ recibidos = 3\ números\ de\ secuencia\ perdidos.$$

Perdemos 3 números de secuencia de 350 totales → $\boxed{0.85\%\ de\ pérdidas.}$

5.3 Pruebas de larga duración

Se realiza una prueba conjunta de larga duración, intento de sobrecarga del sistema y cobertura, la cual consiste en posicionar tres módulos ESP32, cada uno conectado a un sensor con varias IDs grabadas en el programa del ESP32, en distintas zonas de la casa para probar su alcance.

Cada uno de los ESP32, encargados de enviar los datos de temperatura y humedad que captan los sensores, estarán conectados a una batería y posicionados en distintos lugares de la casa, para así poder medir la cobertura y alcance del escáner de tramas de la Raspberry, la cual estará ubicada en una de las habitaciones donde posicionaremos dos módulos.

Para esta prueba se han mantenido los dispositivos fijos hasta la finalización de esta. Se han configurado las IDs de cada ESP32 y se ha seleccionado el lugar de ubicación de cada módulo de la siguiente forma:

- ESP sensor IDs 1-20: Sobre una escalera elevada, a 5,2m de distancia de la Raspberry.
- ESP sensor IDs 21-40: En la habitación contigua al salón, a la derecha, con una pared de por medio. A 4m de distancia.
- ESP sensor IDs 41-60: Mesa baja cerca de la Raspberry, a una distancia de 3m entre ellos.

La ubicación es la misma descrita en el apartado 5.1 de esta sección.

Aquí se muestra la distribución sobre un plano simplificado de la vivienda:

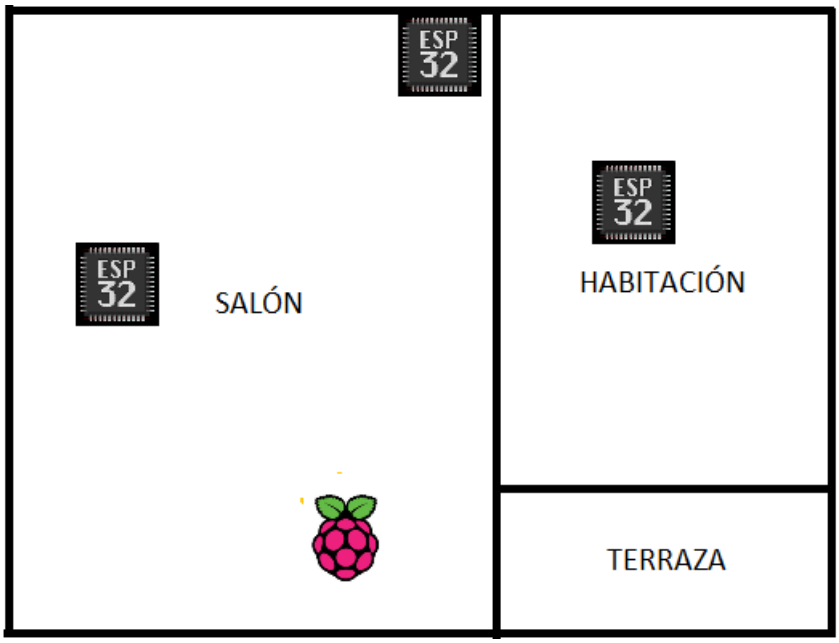


Figura 57. Plano distribución sensores ESP32 y Raspberry

El cronograma del funcionamiento de esta prueba es el siguiente:

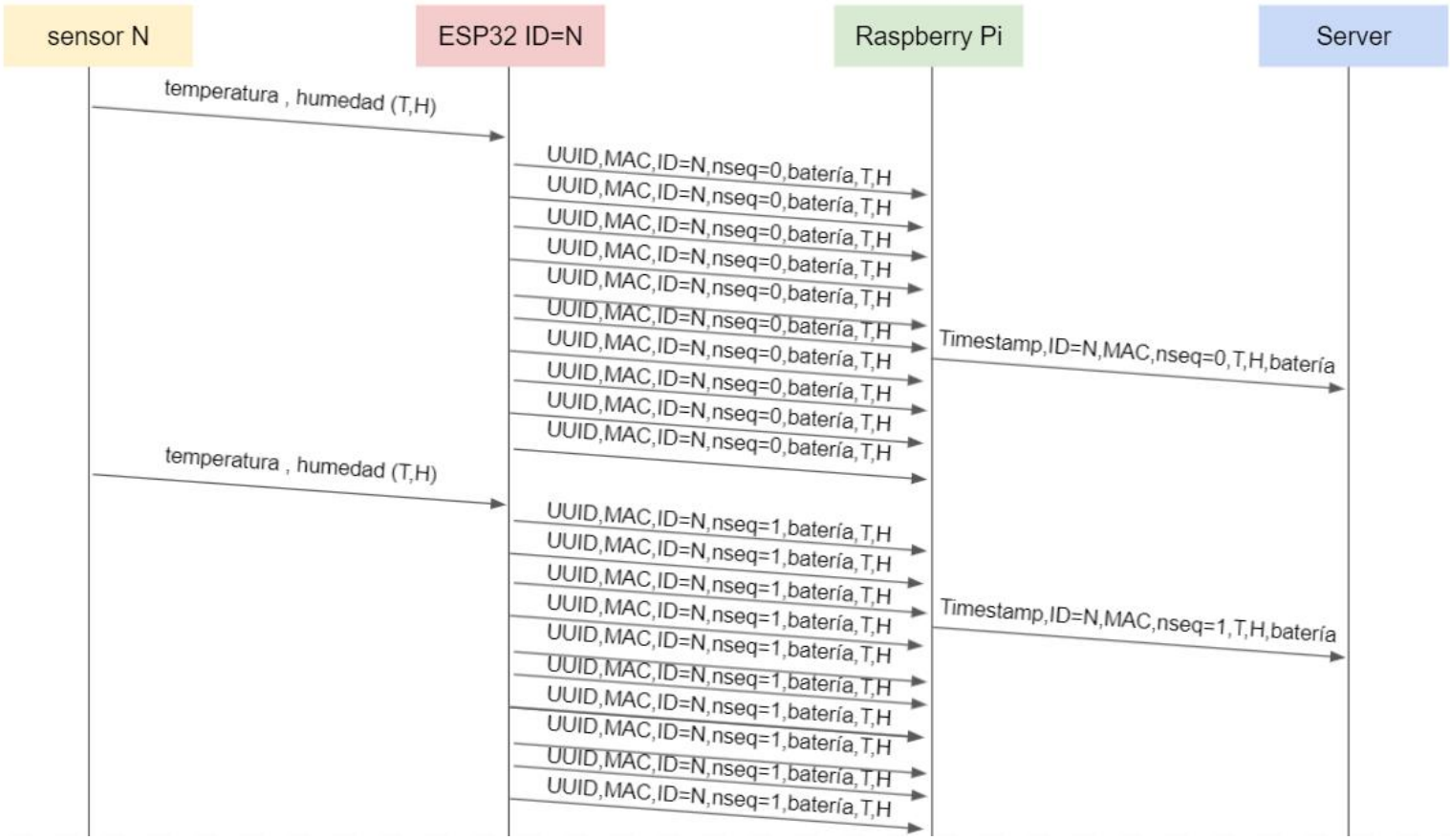


Figura 58. Cronograma envío de tramas ESP32-Raspberry-servidor, pruebas larga duración

Para esta prueba, cada sensor enviará **10 tramas advertisements iguales durante un segundo** –con mismo número de secuencia- a 0,1 segundos cada trama, sin tiempo de sueño. Una vez enviadas las 10 tramas iguales durante 1 segundo, incrementará el número de secuencia y seguirá enviando.

Estas tramas enviadas por los módulos ESP32 conectados a los sensores, son escaneadas por la Raspberry, procesadas para sólo coger una trama de las 10 iguales recibidas y enviadas al servidor, donde guardará la información en su base de datos.

- Medición realizada desde las 23:23:53 hasta las 03:00:47

Nos dirigimos a DB Browser y realizamos la siguiente consulta SQL:

SELECT * FROM datos_sensores WHERE timestamp BETWEEN "2021-09-01 23:23:53" AND "2021-09-01 03:00:47" ORDER BY id_sensor

Esta consulta nos permite ver hasta cuántos números de secuencia distintos ha enviado cada sensor en el periodo de 4 horas y 27 minutos. En esta medición son 256+256+123 = 635 números de secuencia enviados.
Tenemos 60 IDs, y cada ID ha enviado hasta 635 números de secuencia.
Teóricamente se deben recibir:

$$60IDs\ x\ 635nseq\ =\ 38100\ números\ de\ secuencia\ totales.$$

Realizamos la consulta anterior cambiando * por COUNT(num_sec) para visualizar cuántos números de secuencia se han recibido en total. Obtenemos:

$$COUNT(num\_sec)\ =\ 36826\ números\ de\ secuencia\ recibidos\ en\ la\ prueba.$$

Calculamos el porcentaje de pérdidas en la primera medición:

$$38100\ números\ de\ secuencia\ totales\ -\ 36826\ números\ de\ secuencia\ recibidos\ =\ 1274\ números\ de\ secuencia\ perdidos.$$

Perdemos 1274 números de secuencia de 38100 totales → $3.3\%\ de\ pérdidas.$

Estas pérdidas pueden deberse a las interferencias del edificio donde estamos realizando las pruebas, pues la Raspberry está captando tramas de distintos dispositivos los cuales no necesitamos, pero estas se almacenan en un buffer interno, y si se reciben demasiadas le es imposible escanear realmente todas las que enviamos nosotros.

Capítulo 6. Conclusiones y trabajos futuros

6.1 Conclusiones

Para este trabajo se ha utilizado mayoritariamente la Raspberry Pi para llevar a cabo la función de Gateway edge-computing con SCAN BLE.

La Raspberry es un aparato muy versátil que puede realizar las funciones de un ordenador a un coste menor, ocupando menos recursos los cuales requiere un ordenador. Esto nos permite tener un servidor centralizado y varias Raspberrys que recojan los datos por él.

Teniendo una antena externa junto con la suya interna, puede recoger información en una vivienda sin apenas errores. Pudiéndose escanear dispositivos ubicados hasta un máximo de 10 metros perdiendo un 15% de las tramas generadas. Sin embargo a distancias de hasta 4 metros el porcentaje de tramas perdidas es menor del 2%. Cabe destacar que en la zona de la cocina, donde hay azulejos y microondas, se produce una atenuación mayor en la señal.

Las pérdidas pueden deberse al diagrama de radiación de la antena –bidireccional, omnidireccional, unidireccional- pues como hemos visto en las pruebas de cobertura, al reorientar la antena hacia las habitaciones donde se ubicaban los sensores las pérdidas han sido mucho menores.

También se ha podido observar que el hecho de aumentar el envío de tramas advertisement iguales –de 5 a 10- ha favorecido en que la tasa de pérdidas disminuya, pues como existen muchos dispositivos BLE promocionándose en el medio al mismo tiempo, las tramas que buscamos pueden solaparse o ser interferidas por estas y la Raspberry puede estar escuchando estas antes que las nuestras. Al enviarse 10 advertisements iguales por segundo hay más probabilidades de que por lo menos una trama se reciba.

Puede ser factible el uso de la Raspberry como elemento de edge-computing, pero no ideal, pues el hecho de enviar 10 tramas advertisements iguales para tener un porcentaje de pérdidas aceptable será un cuello de botella en el caso en el que usemos muchos sensores emitiendo a la vez, pues solo existen 3 canales BLE.

A su vez, el hecho de tener que reorientar la antena hacia donde están posicionados los sensores limita su distribución en el espacio.

Aun con las pérdidas, gracias al menor coste de una Raspberry respecto un ordenador normal, estas pérdidas debidas a la distancia se pueden compensar añadiendo más dispositivos Raspberry en el hogar para poder así captar todos los sensores que se necesiten.

6.2 Grado de consecución de los objetivos y formación adquirida

Resumen de los objetivos propuestos y alcanzados

Los objetivos logrados en este trabajo han sido los siguientes:

- Estudio y uso de Bluetooth Low Energy para comunicación con sensores IoT.
- Desarrollo de un programa Python que sirve para configurar ID y parámetros en el sensor y dar de alta en base de datos.
- Estudio y pruebas de envío de información a través del protocolo MQTT.
- Implementación de Gateway IoT basado en Raspberry Pi y Python usando Bluetooth Low Energy que recibe y procesa los datos de sensores ESP32, elimina redundancias, genera alarmas de alto nivel y reenvía a un server mediante MQTT.
- Diseño y desarrollo de un Back-end y Front-end para consulta y obtención de los datos obtenidos por el Gateway IoT que serán visualizados en una página web previo registro de usuario.
- Realización de pruebas de cobertura, rendimiento y larga duración para comprobar su correcto funcionamiento.

Curva de aprendizaje y principales dificultades encontradas

La mayor dificultad en este proyecto ha sido trabajar y entender las librerías BLE de Python.

He descubierto que existen muy pocas librerías de scan BLE en Python que capten las tramas advertisements del medio, pues la gran mayoría de librerías se encargan de mostrar la MAC del dispositivo BLE y el nivel de señal RSSI.

Aparte, estuve trabajando sobre una librería -adafruit- durante la mayor parte del tiempo, pero en pruebas de larga duración aparecía un error de 'Too many open files' -demasiados archivos abiertos- y se paraba el código. Después de investigar me percaté de que cada vez que se llamaba a la función escáner, esta abría archivos los cuales después no era capaz de cerrar. Intenté poner un límite superior al número de archivos abiertos, así como un script que funcionase como un bucle, para que, si saltaba el error de nuevo y se cerraba el código, este script lo volviera a ejecutar. Pero estas soluciones no eran lo suficientemente robustas para llevar a cabo el correcto funcionamiento del programa SCAN BLE.

Como el fallo tenía su origen en la implementación de la librería y no en mi propio código, no tuve más remedio que cambiar de librería y el código del escáner por completo.

Por lo tanto, terminé trabajando sobre una librería que resultó ser bastante más concreta y sencilla de implementar, bleak, en la cual utilicé al principio sólo la antena interna de la Raspberry, pero cuando debí utilizar tanto la antena interna como la externa del dongle para las pruebas reales me percaté de que esa librería en cuestión no funciona con hilos -threads-, por lo que se hizo uso de la librería asyncio junto con esta, que es una alternativa basada en eventos, por lo que no fue necesario el uso de hilos y el problema fue solventado.

Reflexión sobre la formación adquirida en conocimientos, metodología y competencias

Para la realización de este trabajo ha sido necesario entender bien cómo funciona la arquitectura Bluetooth Low Energy, enfocándonos en cómo funcionan las tramas advertisement BLE para su posterior escaneo y procesamiento.

También se han aprendido conceptos sobre el protocolo de envío de mensajes MQTT y las funcionalidades que los módulos ESP32 nos proporcionan.

Se ha familiarizado más sobre cómo funcionan las librerías de Python, así como su uso en una Raspberry para escanear tramas enviadas por módulos ESP32 y procesarlas para su posterior envío mediante MQTT al servidor.

Por último, se han utilizado los conocimientos obtenidos en varias asignaturas del grado cursado para desarrollar un Back-end y Front-end en NodeJS para la visualización del contenido en una página, así como adquirir nuevas metodologías para su desarrollo.

6.3 Trabajos futuros

El objetivo del TFG es la implementación del prototipo en un entorno real de una casa y se prevé realizar las siguientes mejoras:

- Instalación y puesta en marcha del sistema en un entorno real de una casa con una previsión de 30-40 sensores.
- Mejora de la tasa de éxito de recepción de datos de sensores (que no se pierdan datos de muestreo).
- Planificación de la ubicación de sensores ESP32 y estimación del número y ubicación de los Gateways IoT necesarios en una casa real.
- Revisión de la documentación del código fuente y realización del control de versiones en Git.

Anexos

Instalación entorno desarrollo para la Raspberry Pi

- Paso 1) Instalación del sistema operativo en la Raspberry [11] a través de una tarjeta microSD:

  Descargar el programa y configurar en la ventana emergente la red Wi-Fi a la que se va a conectar nuestra Raspberry Pi.

- Paso 2) Entrar dentro de la Raspberry:

  Conectar cable hdmi, un extremo a un monitor y el otro extremo a la Raspberry Pi. Aparecerá un setup, cambiamos la contraseña por defecto e instalamos.
  En mi caso → Usuario: pi ; Contraseña: alba.

  Otra forma: mediante cable ethernet conectado al router.

- Paso 3) Obtención de IP de la Raspberry:

  Desde el terminal de la Raspberry, introducir ifconfig para ver su IP.

- Paso 4) Uso de la Raspberry desde un ordenador Windows:
  - Mediante el comando ssh: Abrir aplicación PowerShell en el ordenador donde se trabaje e introducir `ssh pi@IPraspberry`.
  - Mediante VNC: si no está habilitado, nos conectamos por medio de ssh, introducimos `sudo raspi-config` y habilitamos VNC.
    Una vez habilitado, abrir aplicación VNC Viewer e introducir @IPraspberry.
    De esta forma podemos trabajar desde un escritorio remoto.

- Paso 5) Instalación de librerías y programas necesarios:

  A continuación se muestran los comandos utilizados para ello.

```
sudo apt install snapd
sudo snap install bluez
sudo apt install python3-pip
sudo apt install libbluetooth-dev
sudo pip3 install pybluez
sudo apt-get install python3-dev libbluetooth-dev libcap2-bin
sudo apt install pkg-config libboost-python-dev libboost-thread-
dev libbluetooth-dev libglib2.0-dev python-dev
sudo pip3 install gattlib
sudo apt install libglib2.0-dev
sudo pip3 install pygatt
sudo pip install bluepy
sudo pip3 install adafruit-circuitpython-ble
sudo apt-get install bluez-hcidump
```

Explicación ejemplo de uso de diccionarios en Python

Un diccionario es una estructura de datos y un tipo de dato en Python con características especiales que nos permite almacenar cualquier tipo de valor como enteros, cadenas, listas e incluso otras funciones. Estos diccionarios nos permiten además identificar cada elemento por una clave -Key-. Se constituyen de la siguiente forma: diccionario = { 'clave' : 'valor' }
A continuación se expondrán resumidamente ejemplos de cómo se trabaja con diccionarios.

- Para definir un diccionario, se encierra el listado de elementos entre llaves. Las parejas de clave y valor se separan con comas, y la clave y el valor se separan con dos puntos:

```
diccionario = {'nombre' : 'Carlos', 'edad' : 22, 'cursos': ['Python','Django','JavaScript'] }
```

- Podemos acceder al elemento de un Diccionario mediante la clave de este elemento:

```
print diccionario['nombre'] #Carlos
print diccionario['edad']#22
print diccionario['cursos'] #['Python','Django','JavaScript']
```

- También es posible insertar una lista dentro de un diccionario. Por ejemplo, para acceder a cada uno de los 'cursos' usamos los índices:

```
print diccionario['cursos'][0]#Python
print diccionario['cursos'][1]#Django
print diccionario['cursos'][2]#JavaScript
```

Los diccionarios poseen varios métodos, nos centraremos en los siguientes:

- Items(). Devuelve una lista de tuplas, cada tupla se compone de dos elementos: el primero será la clave y el segundo su valor:

```
dic =   {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}
items = dic.items()

items → [('a',1),('b',2),('c',3),('d',4)]
```

- Keys(). Retorna una lista de elementos, los cuales serán las claves de nuestro diccionario:

```
dic =  {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}
keys= dic.keys()

keys→ ['a','b','c','d']
```

- Values(). Retorna una lista de elementos, que serán los valores de nuestro diccionario:

```
dic =  {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}
values= dic.values()

values→ [1,2,3,4]
```

Instalación y puesta en marcha de módulos software del servidor en Windows

- Paso 1) Descargar NodeJS y GIT, además de Visual Studio Code para realizar el código.
- Paso 2) Crear una carpeta donde incluiremos todos los archivos.
- Paso 3) Nos posicionamos en la ubicación de la carpeta creada.
  Botón derecho → Git Bash Here para abrir una ventana de comandos.
- Paso 4) Inicializar NodeJS: `npm init`

  La línea de comandos nos pedirá algunos datos, una vez puestos se creará un package.json dentro de nuestra carpeta con la descripción parecida a esta:

```
"name": "nodealba",
"version": "1.0.0",
"description": "primer server web",
"main": "server.js",
▷ Debug
"scripts": {
  "test": "echo \"Hola mundo\""
},
"author": "Alba Martinez Meroño",
"license": "ISC",
```

- Paso 5) Descarga e instalación de Mosquitto en Windows [13].

  Una vez instalado, ejecutamos dos consolas PowerShell como administrador, en la ruta donde se encuentra instalado para comprobar el funcionamiento de Mosquitto:

  - 1º ventana de comandos: nos suscribimos al topic "ALBA" mediante .\mosquitto_sub

```
PS D:\Mosquitto> .\mosquitto_sub -d -h localhost -p 1883 -t "ALBA"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending SUBSCRIBE (Mid: 1, Topic: ALBA, QoS: 0, Options: 0x00)
Client (null) received SUBACK
Subscribed (mid: 1): 0
```

*localhost puede sustituirse por la IP donde está alojado Mosquitto

- 2º ventana de comandos: publicamos un mensaje 'Hola Mundo' mediante .\mosquitto_pub

```
PS D:\Mosquitto> .\mosquitto_pub -d -h localhost -p 1883 -t "ALBA" -m "Hola Mundo"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q0, r0, m1, 'ALBA', ... (10 bytes))
Client (null) sending DISCONNECT
```

- Obtenemos, en la primera ventana de comandos (subscriber):

```
PS D:\Mosquitto> .\mosquitto_sub -d -h localhost -p 1883 -t "ALBA"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending SUBSCRIBE (Mid: 1, Topic: ALBA, QoS: 0, Options: 0x00)
Client (null) received SUBACK
Subscribed (mid: 1): 0
Client (null) received PUBLISH (d0, q0, r0, m0, 'ALBA', ... (10 bytes))
Hola Mundo
```

- Paso 6) Una vez instalado Mosquitto procedemos a instalar MQTT.js

```
npm install express --save
npm install mqtt -g
```

Para tener la dependencia de mqtt en package.json: `npm i mqtt`

Comprobamos que funciona creando un script prueba.js que se conecta al Mosquitto, se subscribe y está escuchando:

```
var mqtt = require('mqtt')
var client  = mqtt.connect('mqtt://192.168.1.106') //nos conectamos al servidor mosquitto

function EventoConectar() { //callback de cuando se conecta
    client.subscribe('ALBA') //me subscribo
}

function EventoMensaje(topic, message) { //callback de cuando llega mensaje
    // message is Buffer
    console.log(topic + " - " + message.toString()) //leo msj y lo transformo a string
}

client.on('connect', EventoConectar)
client.on('message', EventoMensaje)
```

Figura 59. Script de prueba funcionamiento Mosquitto en servidor

- Abrimos como administrador PowerShell y nos dirigimos a la carpeta donde está ubicado Mosquitto y escribimos el comando:

  ```
  .\mosquitto_sub -d -h "192.168.1.106" -p 1883 -t "ALBA"
  ```

  para subscribirnos y escuchar.

- En Git Bash escribimos node +nombre del script a ejecutar:

  ```
  node prueba.js
  ```

- Para publicar, en otra ventana de Git Bash escribimos:

  ```
  mqtt pub -t "ALBA" -h "192.168.1.106" -m "Hola mundo"
  ```

  ```
  mqtt pub -t "ALBA" -h "192.168.1.106" -m "Funciona
  correctamente"
  ```

```
Alba@AlbaPC MINGW64 ~/Desktop/TFG/NODE/VS
$ mqtt pub -t "ALBA" -h "192.168.1.106" -m "Hola mundo"

Alba@AlbaPC MINGW64 ~/Desktop/TFG/NODE/VS
$ mqtt pub -t "ALBA" -h "192.168.1.106" -m "Funciona correctamente"
```

- Como resultado obtenemos en la ventana de PowerShell:

```
PS D:\Mosquitto> .\mosquitto_sub -d -h "192.168.1.106" -p 1883 -t "ALBA"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending SUBSCRIBE (Mid: 1, Topic: ALBA, QoS: 0, Options: 0x00)
Client (null) received SUBACK
Subscribed (mid: 1): 0
Client (null) received PUBLISH (d0, q0, r0, m0, 'ALBA', ... (10 bytes))
Hola mundo
Client (null) received PUBLISH (d0, q0, r0, m0, 'ALBA', ... (22 bytes))
Funciona correctamente
```

- Paso 7) Creación de base de datos con SQLite.

  Nos instalamos DB Browser por SQLite [14]. Una vez instalada abrimos la aplicación y le damos al botón “Crear nueva base de datos”, elegimos la ubicación de la BBDD en la carpeta creada donde incluiremos nuestro código.

Botón “Crear tabla”, le damos nombre a nuestra tabla y añadimos las columnas necesarias:

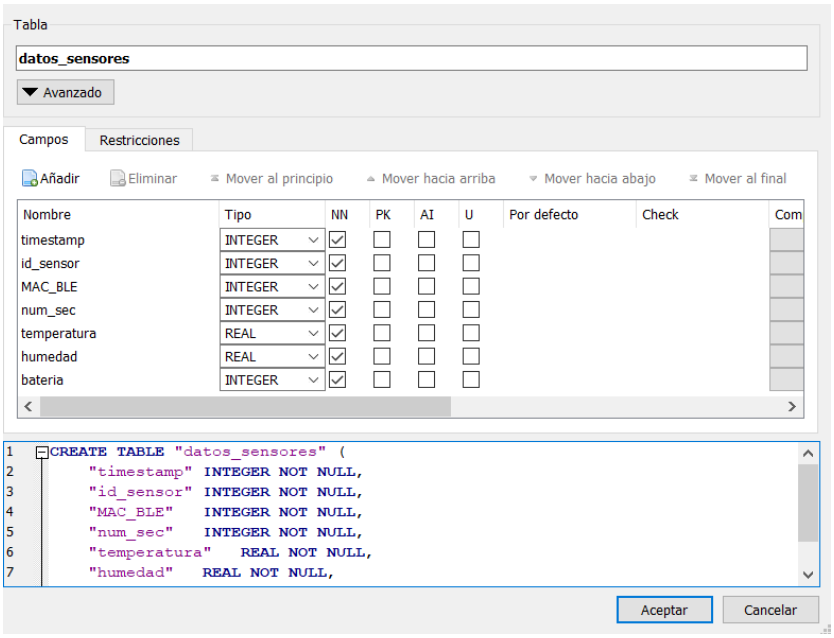


Figura 60. Creación de tabla datos_sensores en BBDD servidor

- Paso 8) Añadir sqlite3 a dependencies de package.json.

```
npm install --save sqlite3
```

Bibliografía y referencias

[1] Dispositivo sensor ESP32: www.espresiff.com

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

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

[4] NodeJS: https://nodejs.org/en/

[5] App nRFConnect para android: https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp

[6] Python: https://www.python.org/

[7] Librería PyBluez: https://pybluez.readthedocs.io/en/latest/install.html

[8] Librería Gattlib: https://pypi.org/project/gattlib/#python-pip

[9] Información sobre RaspberryPi con BLE: https://elinux.org/RPi_Bluetooth_LE

[10] Información sobre Beacons BLE: https://www.ti.com/lit/an/swra475a/swra475a.pdf?ts=1624910368319&ref_url=https%253A%252F%252Fwww.google.com%252F

[11] Sistema operativo Raspberry Pi: https://www.raspberrypi.org/software/

[12] Instalación de Anaconda: https://www.anaconda.com/products/individual

[13] Instalación Mosquitto: https://www.luisllamas.es/como-instalar-mosquitto-el-broker-mqtt/

[14] DB Browser for SQLite: https://sqlitebrowser.org/

[15] PM2: https://pm2.keymetrics.io/docs/usage/quick-start/

[16] Media móvil en muestreo señales: https://en.wikipedia.org/wiki/Moving_average

[17] Passport: http://www.passportjs.org/

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