Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

IMPLEMENTACIÓN DE UN SISTEMA DE DETECCIÓN DE DISPOSITVOS WIFI PARA UN ESTUDIO DE MERCADO Y SISTEMAS DE SEGURIDAD

Víctor Maestro Lara

Director: Oscar Amoros Huget
Realitzat a: Departament de
Matemàtiques i Informàtica

Barcelona, 13 de Septembre de 2020

Resumen

Desde los inicios de Internet nuestro mundo está cada vez más digitalizado e interconectado. La creación de nuevos dispositivos electrónicos que emiten información por Internet (IoT) ha ido creciendo considerablemente proporcionando nuevas fuentes de información sobre el comportamiento y presencia de las personas.

En este proyecto se presentará un diseño de una arquitectura de ingesta de información en tiempo real, procedente de los paquetes de redes WiFi para identificar la presencia de dispositivos móviles que llevan las personas.

Veremos cómo la información recopilada puede ser utilizada tanto para fines de estudio de mercado, mediante la extracción de Indicadores Clave de Rendimiento, o “KPIs” sobre nuestros datos como para sistemas de seguridad gracias a la dirección MAC, única e indistinta para cada dispositivo móvil de las personas.

Por tanto, el sistema presentado nos permite detectar la presencia de personas, por lo que puede ser muy útil en el mundo empresarial para realizar estudios de mercado dando un significado a los datos recopilados. De la misma manera, puede ser práctico para un sistema de seguridad en espacios controlados proporcionando una capa de protección en recintos.

Las limitaciones de desplazamiento producidas en el país debido a las circunstancias y al estado de alarma no han permitido seguir el plan inicial de montar el sistema en la Universidad de Barcelona por lo que no existirá una gran diversidad de los datos. No obstante, se pretende demostrar que con independencia del lugar y circunstancia este sistema puede ser utilizado para diferentes aplicaciones con estos datos.

Resum

Des de l'inici d'Internet el nostre món ha estat cada vegada més digitalitzat i interconnectat. La creació de nous dispositius electrònics que difonen informació a través d'Internet (IoT) ha anat creixent considerablement proporcionant noves fonts d'informació quan s'envia a través d'Internet.

En aquest projecte es presentarà un disseny d'una arquitectura de ingesta d'informació en temps real, a partir de paquets de xarxa WiFi per a identificar la presència de dispositius mòbils que les persones porten.

Veurem com la informació recopilada es pot utilitzar tant amb finalitats d'investigació de mercat, mitjançant l'extracció d'indicadors clau de rendiment, o "KPI" en les nostres dades com per a sistemes de seguretat gràcies a l'adreça MAC única i indistinta per a cada dispositiu mòbil de les persones.

Per tant, el sistema presentat ens permet detectar la presència de persones, per la qual cosa pot ser molt útil en el món empresarial per a dur a terme estudis de mercat donant sentit a les dades recollides. De la mateixa manera, pot ser pràctic per a un sistema de seguretat en espais controlats proporcionant una capa de protecció en recintes.

Les limitacions de viatge produïdes al país degut a les circumstàncies i l'estat d'alarma no han permès seguir el pla inicial per posar en marxa el sistema a la Universitat de Barcelona, per aquest motiu hi haurà una gran diversitat de dades. No obstant això, es pretén demostrar que independentment del lloc i circumstància aquest sistema es pot utilitzar per a diferents aplicacions amb aquestes dades.

Abstract

Since the beginning of the Internet our world is becoming much more digitalized and interconnected. The invention of new electronic devices that emit information over the Internet (IOT) has been growing considerably, providing new sources of information when it is sent over the Internet.

This paper describes a design of an architecture for ingesting information in real time from WiFi network packages to identify the presence of mobile devices that people carry.

We will see how the information collected can be used both for market research purposes, by extracting Key Performance Indicators, or "KPIs" on our data and for security systems thanks to the unique and indistinct MAC address for each mobile device of the people.

Therefore, the system presented allows us to detect the presence of people, so it can be very useful in the business world to carry out market studies giving meaning to the collected data. Similarly, it can be practical for a security system in controlled spaces by providing a layer of protection in enclosures.

The travel limitations produced in the country due to the circumstances and the state of alarm have not allowed to follow the initial plan of setting up the system at the University of Barcelona, so there will not be a great diversity of data. However, it is intended to demonstrate that regardless of the place and circumstance this system can be used for different applications with this data.

Índice

1 INTRODUCCIÓN Y ANTECEDENTES

La obtención de datos sobre las personas es una información de creciente valor en los últimos años. La digitalización de muchas empresas y sectores ha potenciado la recogida de datos proporcionando nuevas vías sobre las que invertir para generar una mejor experiencia de usuario, maximizar las ganancias o crear sistemas de seguridad más eficientes.

Podemos ver hoy en día como todos nuestros datos son cada vez más importantes y pueden ser explotados para un beneficio, desde anuncios en televisión, ofertas en las tiendas… hasta la misma propaganda de las elecciones. Las personas somos un objetivo de negocio y cualquier dato relevante sobre nosotros es susceptible de ser explotado en el mercado. De la misma forma, al tener toda persona un dispositivo móvil asociado a su identidad, podemos usar la información que emite para detectar la presencia de determinadas personas en espacios controlados.

Con dicha información presencial podemos detectar el volumen de personas en un centro comercial, el tiempo que pasa en cada sitio, y los lugares que más visitan. Todos estos datos permiten a las empresas adaptarse a las necesidades de las personas y generar un plan de comercio que les permita generar el mayor beneficio económico posible.

Incluso, podemos usar estos datos para detectar la presencia de ciertos individuos implementando un sistema de seguridad que nos permita detectar ciertas personas que queremos tener controladas.

Para poder obtener todos estos datos, centros comerciales y comercios implementan sistemas de detección de personas basados en la visión por computador con cámaras. De la misma manera, los almacenes donde se suele guardar el stock, naves industriales o recintos de alto valor suelen implementar sistemas con cámaras y alarmas para vigilar la zona y detectar la presencia de personas.

Un ejemplo bastante conocido es Amazon Go, una tienda con decenas de cámaras instaladas que mantienen el seguimiento de cada una de las personas (donde están, que productos cogen y dejan) y permite a los clientes comprar los productos sin pasar por una caja física ya que el sistema los detecta y les hace el cobro de todos sus productos en su cuenta al salir. [1]

Incluso en el tránsito y transporte público con el tan conocido Google Maps se emplean técnicas de detección de personas como por ejemplo la cantidad de dispositivos en una zona (entre otras) para determinar el tráfico por una calle y así proporcionar las rutas más rápidas. [2]

Es por tanto en este campo donde reside el interés de este trabajo que investiga la detección de dispositivos wifi de una manera económica y sencilla a partir de los datos que se pueden

obtener de las tramas de red Probe Request (Peticiones de sondeo) que se producen entre los dispositivos y los puntos de acceso.

2 MOTIVACIÓN

Como se ha comentado anteriormente existen hoy en día diferentes empresas y herramientas que nos permiten detectar personas y obtener datos sobre ellas. Existen casos como Amazon Go que permite la detección mediante el uso de cámaras y la visión por computador o casos más sofisticados como el de Google Maps.

Este tipo de tecnologías son muy potentes, pero al mismo tiempo requieren de una gran inversión que no todos los comercios o establecimientos están dispuestos a realizar, especialmente si no han tenido ninguna experiencia con estas tecnologías.

En este trabajo se pretende abordar este problema desde otra perspectiva utilizando microcontroladores ESP32 para capturar el tráfico de red WiFi que se realiza entre los dispositivos de las personas y los puntos de acceso de su alrededor. Esta tecnología se ha visto implementada principalmente en el mundo del Hacking con el objetivo de rastrear y substraer las redes WiFi que más se suelen frecuentar con el conocido "***Wardriving***". [3]

Nuestros dispositivos móviles están continuamente buscando redes WiFi a las que conectarse y envían pequeñas tramas de información que se conocen como "Probe Requets" o "Solicitudes de Sondeo".

Estas solicitudes son pequeños fragmentos de información que son enviados con independencia de si el dispositivo está o no conectado a una red WiFi y contienen entre otros campos la dirección MAC del dispositivo, la señal RSSI (la intensidad de señal logarítmica), y una lista de SSID (nombres de redes) encontrados previamente.

Debido a que cada dispositivo envía su dirección MAC única podemos aprovecharnos para recoger estos datos e identificar el tipo de dispositivo, el momento, la ubicación etc. De hecho, se abren muchos campos de investigación y aplicación, desde el rastreo de personas en recintos a tiempo real, modelos de predicción de futuros pasos del usuario, obtención de multitudes de KPIs como por ejemplo tipos de dispositivos e incluso como capa de seguridad en recintos como alarma.

En los últimos años se ha observador como los dispositivos han intentado atacar este problema intentando anonimizarse enviando una dirección MAC que no es real, no obstante, como veremos más adelante podremos identificar que direcciones MAC no son reales.

Teniendo en cuenta lo descrito anteriormente, este trabajo se centra en la parte técnica de la obtención de datos para la detección de presencia de personas. Para ello se propone lo siguiente:

- Emplear microcontroladores ESP32 para la recogida de datos con el objetivo de reducir costes, respecto a otras propuestas de usar Raspberry que son más costosas.
- Garantizar la seguridad de los datos recogidos en la comunicación entre los microcontroladores y el servidor.
- Implementar una arquitectura que sea escalable, rápida y sencilla.
- Hacer una prueba en un entorno distribuido, donde se dediquen al menos 3 microcontroladores a recoger información en una zona por 2 semanas.
- Implementar un sistema de alertas o notificaciones por correo electrónico que nos permita evaluar la capacidad de la tecnología implementada en tiempo real.

3 IMPLEMENTACIÓN

3.1 Tecnologías para utilizar

Antes de avanzar con el diseño e implementación es necesario entender la naturaleza del problema y el cómo nos adaptaremos a sus necesidades para poder lograr los objetivos planteados. Las necesidades y objetivos definirán nuestra implementación final y las decisiones realizadas tanto a nivel de hardware como de software.

Es necesario que nuestro sistema sea distribuido escalable, barato y que pueda procesar datos en tiempo real ya que queremos detectar personas y recibir notificaciones de email.

Asimismo, se requiere de un sistema sencillo, rápido y seguro permitiendo una persistencia de los datos para que luego puedan ser analizados.

3.1.1 Estandard IEEE 802.11

Para poder entender la recolecta de datos hay que entender su procedencia y estructura. El estándar sobre el cual trabajaremos es el IEEE 802.11, Wireless LAN.

Los datos que obtendremos tendrán su procedencia en los frames o “tramas” – PDU (Protocol Data Unit) de nuestra capa de enlace de la pila ISO-OSI. Cada trama a nivel MAC incluye un conjunto de campos que siguen una estructura predefinida: [4]

- Una cabecera (MAC header).
- El cuerpo de la trama (Frame Body) – De longitud variable con un máximo de (2312 Byte).

- Un FCS (Frame Check Sequence) que contiene un código de redundancia cíclica (CRC) a 32 bit.

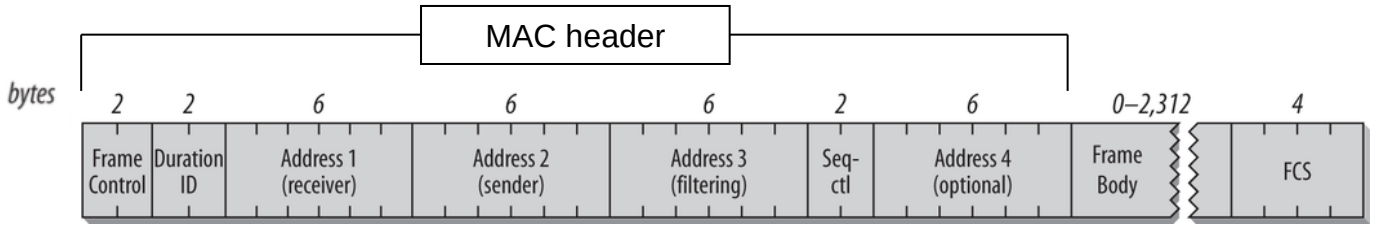


Fig. 1 Formato general de la trama MAC 802.11

Es en la cabecera MAC donde nos centraremos para obtener la mayoría de nuestros datos. Para simplificar la explicación nos centraremos en cómo podemos identificar los Probe Requests Frames, aunque a la hora de la práctica el Framework que utilizaremos nos facilitará mucho todo a la hora de filtrar.

Para identificar un Probe Request Frame hay que fijarse en los primeros 2 bytes de la cabecera MAC, en el Frame Control que está formado por los siguientes campos:

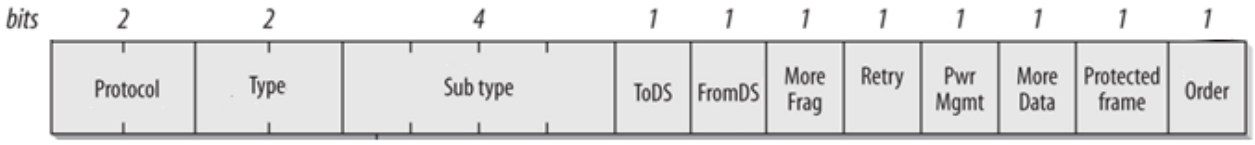


Fig. 2 Formato del campo Frame Control

El campo type nos permitirá identificar qué tipo de trama es:

- Trama de Datos (Data Frames), cuando los bits son 10.
- Trama de Control (Control Frames), cuando los bits son 01.
- Trama de Gestión (Management Frames), cuando los bits son 00.

En nuestro caso nos interesa las tramas de gestión que son utilizadas para intercambiar información sobre la gestión de la conexión. Los siguientes 4 bits de subtipo nos informarán de que subtipo de trama se trata:

Type value b3... b2	Type Description	Subtype Value B7 ... B4	Subtype Description
00	Management	0000	Association Request
00	Management	0001	Association Response
00	Management	0010	Reassociation Request
00	Management	0011	Reassociation Response
00	Management	0100	Probe Request
00	Management	0101	Probe Response
00	Management	0110	Timing Advertisement
00	Management	0111	Reserved
00	Management	1000	Beacon
00	Management	1001	ATIM
00	Management	1010	Disassociation
00	Management	1011	Authentication
00	Management	1100	Deauthenticaion
00	Management	1101	Action
00	Management	1110	Action No Ack (NACK)
00	Management	1111	Reserved

Tabla 1 Interpretación de los cambios Type y Subtype de una trama de gestión

Por tanto, cuando los bits de tipo sean 00 y los de subtipo 0100 estaremos hablando de una Probe Request Frame que es del tipo Management.

3.1.1.1 Estructura de Probe Requests Frames

La estructura de la una trama de subtipo Probe Request es la siguiente:

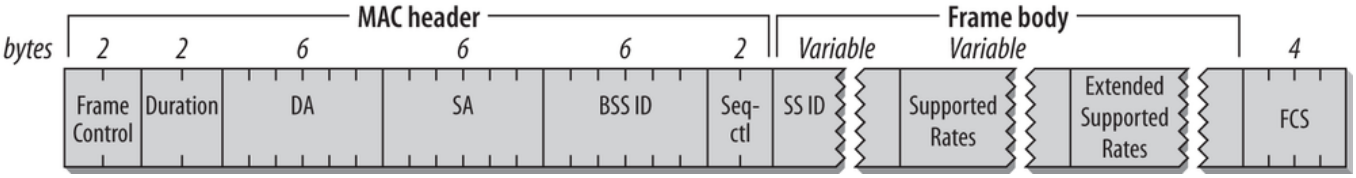


Fig. 3 Estructura general de una trama de subtipo Probe Request

Como podemos observar en la fig. 3 la estructura de una trama de subtipo Probe Request consta de los siguientes campos:

- DA (Destination Address): La dirección de destino será siempre un broadcast (FF:FF:FF:FF:FF:FF).
- SA (Source Address): Contiene la dirección MAC del dispositivo que envía la petición.
- BSSID (Basic Service Set Identifier): Indica nuevamente un broadcast.
- Sequence Control: Un campo de 16 bits que está formado por el número de fragmento y de secuencia.

- SSID (Identificador de red): El nombre público de la red WLAN que está buscando y se intenta conectar. Hay que destacar este campo puede estar en vacío, pero puede estar establecido en busca de una de las redes WiFi conocidas por el dispositivo.

- Supported Rates: Las ratios de transmisión deben de ser soportados por el punto de acceso para poder realizar la conexión.

También se incluyen otros campos de información que pueden estar presentes entre los cuales se encuentra HT Capabilities que es único para cada conexión y podría ser utilizado para diferenciar entre dispositivos que intentan ocultar su MAC real. [5]

3.1.2 Hardware empleado

Antes de seguir con la implementación es necesario explicar que hardware emplearemos para la implementación.

Necesitaremos un dispositivo que pueda cumplir los objetivos planteados. Usaremos un microcontrolador de la familia ESP32 creado por **Espressif Systems** los cuales destacan por su bajo coste y consumo de energía con capacidades WiFi y Bluetooth.

La familia ESP32 incluye microprocesadores Tensilica Xtensa LX6 de único o doble núcleo con una frecuencia de reloj de hasta 240MHz. Además, sus chips están especialmente diseñados para aplicaciones de IoT (Internet of Things) por lo que encontraremos con soporte para muchas de las necesidades.

Su conectividad inalámbrica abarca:

- WiFi: 802.11 b/g/n a 2.4GHz hasta 150 Mbit/s.
- Bluetoot: v4.2 BR/EDR y Bluetooh de bajo consumo (BLE)

Además, cuenta con soporte para aceleración hardware para encriptación entre otros AES, SHA-2, RSA etc. [6]

Emplearemos por tanto una tarjeta NodeMCU-32 modelo (ESP32-WROOM32) que está especialmente diseñada para trabajar en protoboard y podremos alimentarlo directamente por un puerto USB.

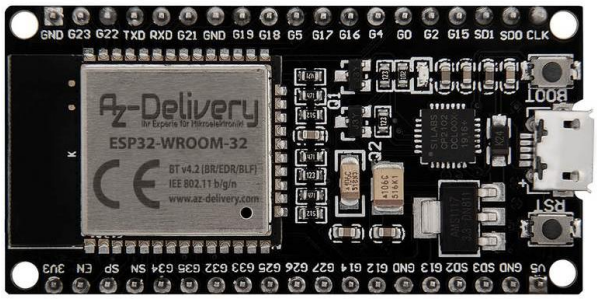


Fig. 4 NodeMCU ESP32 WROOM32

3.1.3 Protocolo de comunicación MQTT

MQTT es el acrónimo de "**Message Queuing Telemetry Transport**", es un protocolo de comunicación de mensajes ligero, que se utiliza principalmente en casos donde los clientes están conectados a redes poco fiables o con recursos de ancho de banda limitados. Es utilizado principalmente en conexiones de IOT por su sencillez y ligereza ya que muchas veces estos dispositivos tienen limitaciones de potencia, consumo y/o ancho de banda. [7]

3.1.3.1 Funcionamiento de MQTT

El protocolo emplea un patrón de publicador/suscriptor (pub-sub) con clientes y brokers. No existe una transmisión de datos continua, sino que se rige por eventos, cuando los clientes tienen información para enviar.

La información es filtrada en topics o temas. Estos son estructuras que emplean una jerarquía con carácter de "/" como delimitador como si de un árbol de directorios en un sistema de archivos se tratase. Un cliente puede publicar mensajes en un determinado topic y otros clientes pueden suscribirse, y el broker se encargará de hacer llegar los mensajes a estos.

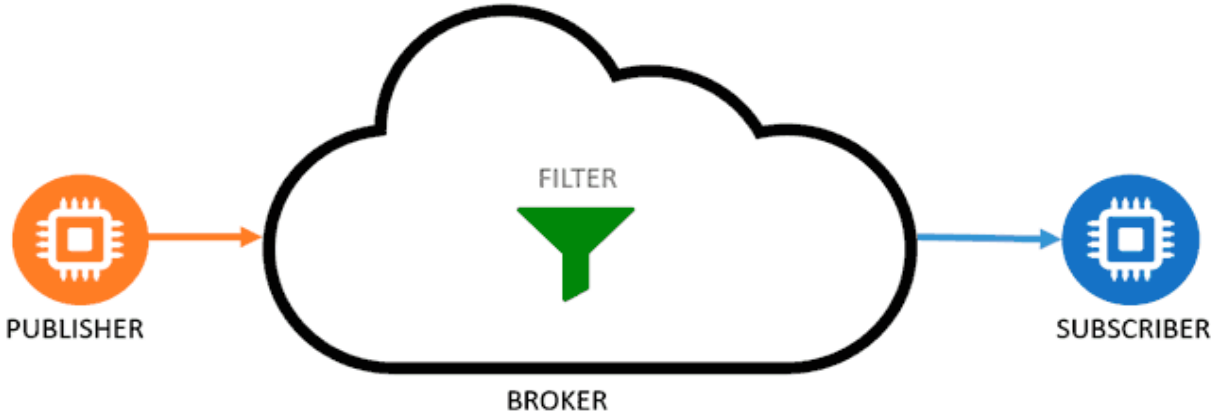


Fig. 5 Protocolo MQTT publicador y suscriptor en un topic de un broker.

Los clientes se conectan al broker mediante una conexión TCP/IP el cual mantiene un registro de los clientes conectados hasta que el cliente la finaliza. Por defecto en MQTT emplea el puerto 1883 y el 8883 en caso de que funcione sobre TLS. [8]

Para mantener el protocolo lo más sencillo solo se pueden realizar estas acciones en comunicación:

- Conectar/Desconectar
- Publicar
- Suscriberse/Desuscribirse
- Ping

Para poder establecer esta conexión el cliente envía un mensaje de CONNECT que contiene la información necesaria como (usuario, contraseña, client_id…), el broker responde con un CONNACK que informa si la conexión ha sido aceptada o rechazada.

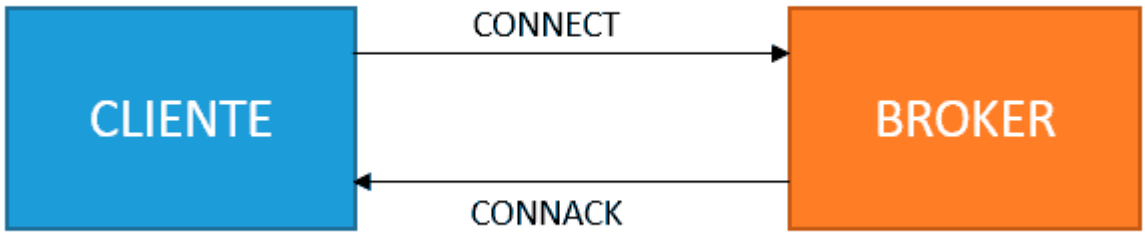


Fig. 13 Mensaje de CONNECT entre cliente y broker

Por otra parte, para enviar los mensajes el cliente emplea PUBLISH que contiene el topic y el payload con la información.

Fig. 14 Mensaje de PUBLISH entre cliente y broker

Y para suscribirse y darse de baja se emplean mensajes de SUSCRIBE y UNSUSCRIBE que el servidor responde con un SUBACK y UNSUBACK respectivamente.

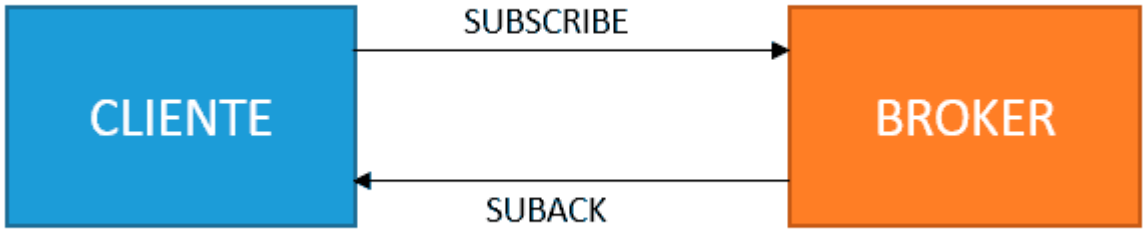
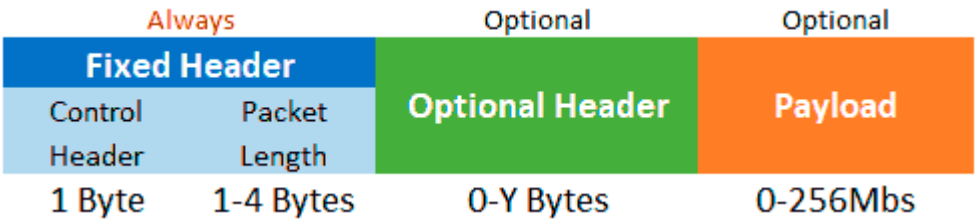


Fig. 15 Mensaje de SUBSCRIBE entre cliente y broker

Además, para asegurar que la conexión está activa los clientes manda periódicamente un mensaje PINGREQ que es respondido por el servidor con un PINGRESP y para desconectarse se emplea un mensaje de DISCONNECT.

3.1.3.2 Estructura de un mensaje de MQTT

Los mensajes del protocolo de MQTT son muy simples y tienen la siguiente estructura:



- **Cabecera Fija** que ocupa de 2 a 5 bytes y consta del código de control que identifica el tipo de mensaje y la longitud del mensaje.
- **Cabecera Variable** es opcional y contiene información adicional necesaria en ciertos mensajes.
- **Contenido o payload** Es el contenido real del mensaje que puede tener como máximo 256Mb aunque en una implementación real el máximo es de 2 a 4kB.

3.1.3.3 Calidad del Servicio (QoS)

MQTT dispone de tres niveles de calidad QoS que nos permite elegir entre minimizar la transmisión de datos y maximizar la confiabilidad:

- **QoS 0** Emplea una cantidad mínima en la transmisión de datos. El mensaje es entregado, pero no hay forma de saber si los suscriptores de ese topic recibieron el mensaje. Es decir, no se espera ningún tipo de confirmación, se asume que la entrega está completa.
- **QoS 1** Se intenta entregar el mensaje y luego de esperar una respuesta de confirmación del suscriptor si no se recibe una confirmación dentro de un periodo de tiempo configurado el mensaje se reenvía nuevamente. De esta manera, se puede enviar el mensaje más de una vez.
- **Qos 2** En este modo se utiliza un protocolo de enlace de cuatro pasos para garantizar que el mensaje se reciba y que se reciba una sola vez. Esto se suele conocer como “entregar exactamente una vez”.

Para situaciones en las que las comunicaciones son confiables pero limitadas se suele emplear QoS 0 pero en situaciones donde la comunicación no es tan confiable, pero no se dispone de recursos limitados QoS 2 sería la mejor. En el caso de QoS 1 se proporciona una mejor solución para ambos casos, pero requiere que la aplicación que recibe los datos sepa manejar datos duplicados. [9]

3.2 Diseño e implementación

3.2.1 Propuesta de arquitectura

La arquitectura presentada en este proyecto pretende garantizar los siguientes objetivos:

- La rapidez y sencillez
- La escalabilidad
- La integridad de los datos
- La seguridad en la comunicación

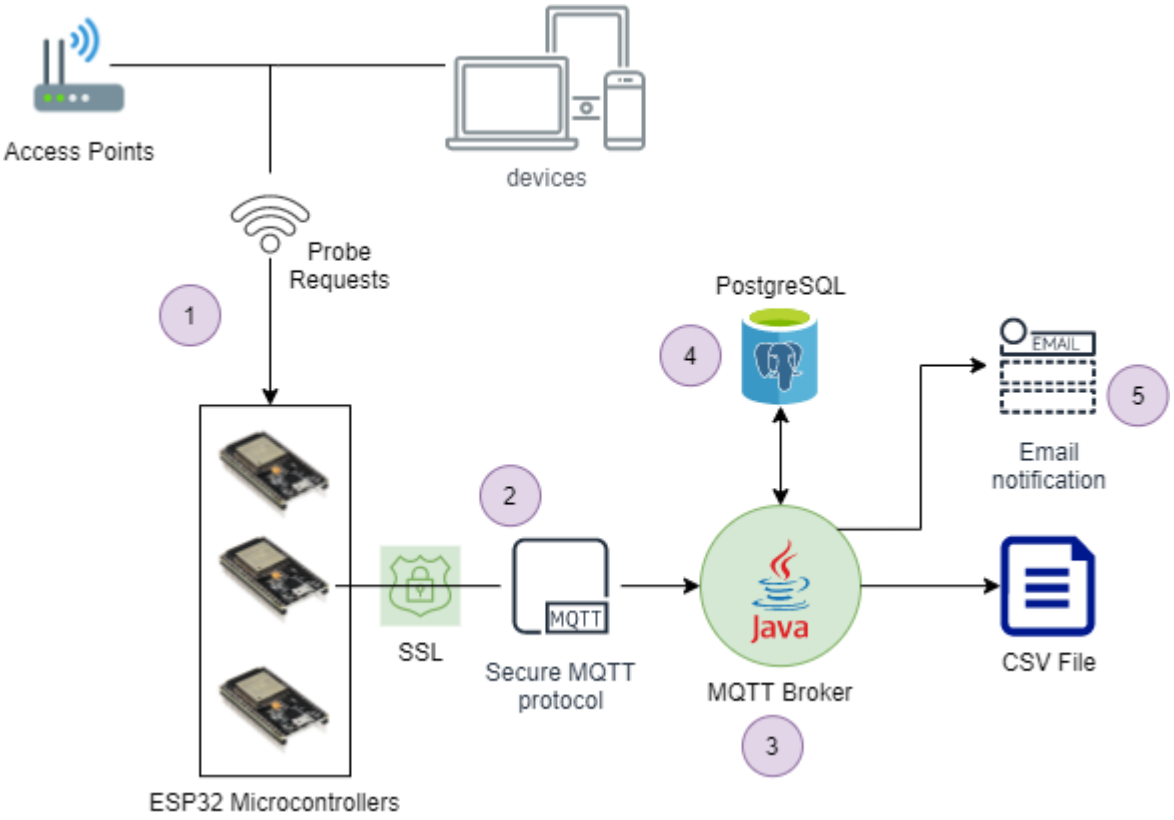


Fig. 16 Esquema general de la arquitectura presentada

Primero: *Captura del tráfico de red con los microcontroladores*

Los diferentes dispositivos móviles mandaran tramas de probe request cada cierto tiempo por los diferentes canales WiFi del 1 al 11 independientemente de que estén o no conectados a un punto de acceso. La cantidad de estas tramas se ve incrementada según si el dispositivo está en uso o no.

Los microcontroladores se encargarán de ir recopilando la información de los diferentes canales WiFi y la guardarán en memoria durante 1 minuto.

Segundo: *Envío de datos mediante el protocolo MQTT*

Los microcontroladores se encargan de transformar la información en un JSON String del siguiente formato:


```
{
    "id": "1",
    "channel": "2",
    "timestamp":"1593899819863",
    "rssi": "-94",
    "srcMac": "48:2c:a0:ac:82:12",
    "ssid": "ADAMO-MODI-Wifipublic",
    "seqNumb": "16497",
    "htCapInf": "052d"
}
```


Ilustración 1 Ejemplo de la información extraída de una trama Probe Request en formato JSON para enviarse al broker

Cada trama de Probe Request será enviada a un topic del broker. Esta conexión es segura ya que se utiliza una conexión TLS 1.2 con un cifrado RSA 2048 con el objetivo de proteger la integridad de los datos.

A continuación, los microcontroladores dejan de capturar información por un breve instante para conectarse a un punto de acceso y reestablecer la conexión con el broker el cual recibirá la información y la procesará.

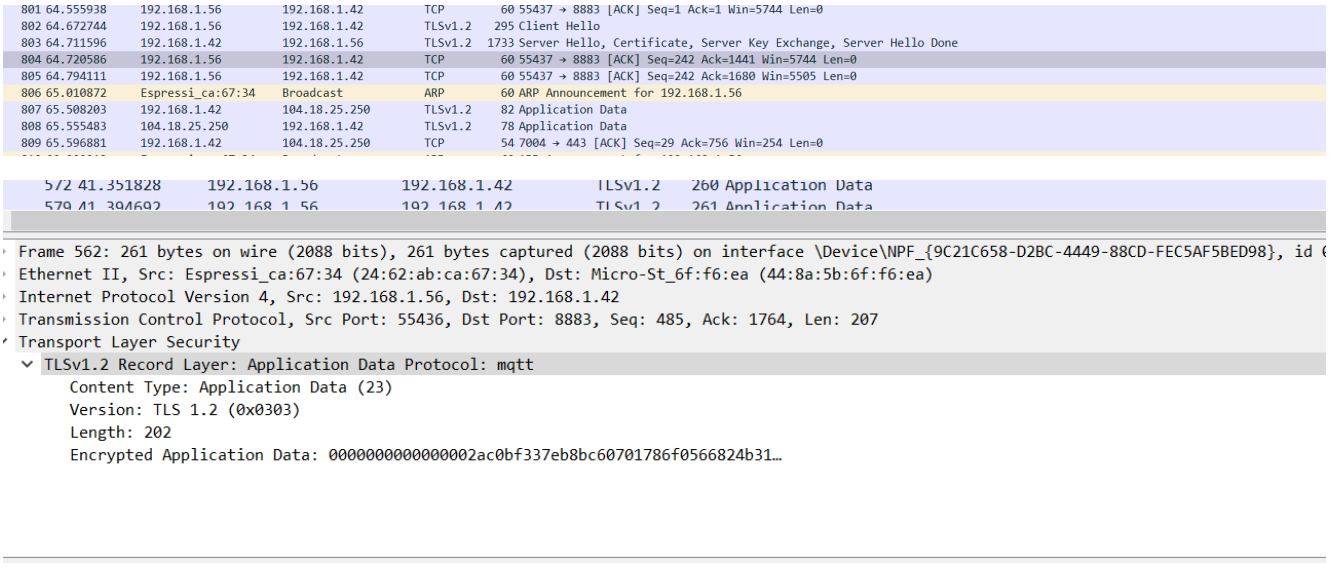


Ilustración 2 Muestra de conexión TLSv1.2 entre un microcontrolador y el broker ejecutado en un entorno local

Tercero: *Java MQTT Broker*

El broker recibirá la información de los microcontroladores y verificará que se recibe en el topic correcto. Este se encargará de transformar el objeto JSON en un objeto Java que nos servirá para realizar el procesado de la información e ingesta tanto en el Base de Datos como en el CSV.

Cuarto: *Ingesta de información en PostgreSQL y en un fichero CSV*

La información de cada trama de Probe Request es guardada en forma de fila en un fichero CSV a modo de persistencia local y por otra parte se guarda la misma información en una base de datos PostgreSQL versión 12.0.

Es necesario destacar que es en el guardado de información en el momento en el que se extrae el fabricante del dispositivo a partir del OUI de la dirección MAC. De la misma manera esta información es guardada tanto en el CSV como en base de datos.

Quinto: *Generación de alertas / notificaciones email*

En este punto se procesa la trama de Probe Request, en concreto se comprueba si para la dirección MAC del dispositivo existe alguna alerta de notificación en base de datos que satisface las condiciones de:

- Si ha pasado suficientemente tiempo desde que se activó la alerta la última vez. Cada alerta tiene configurado un tiempo mínimo entre alerta y alerta por lo que si no ha pasado ese mínimo de tiempo la notificación no se generará.
- Si la trama ha sido detectada por un ID del microcontrolador en concreto con un mínimo de distancia RSSI.

Más adelante hablaremos sobre las alertas y cómo funcionan más en profundidad.

3.2.2 Microcontroladores ESP32

3.2.2.1 Prerrequisitos de implementación

Para poder implementar el código en los microcontroladores ESP32 es necesario:

- Python 3.7
- Git
- Build Tools – CMake y Ninja para hacer un build completo de la aplicación
- Un editor como Visual Studio Code o Eclipse en nuestro caso emplearemos Visual Studio Code

- ESP-IDF que contiene esencialmente la API (Librerias y código fuente) para ESP32 y scripts para operar el Toolchain.
- Toolchain para compilar el código de ESP32
- Como emplearemos Visual Studio Code es necesario instalar las extensiones de:
    - Python
    - C/C++ IntelliSense
    - Espressif IDF

Es necesario destacar que para la instalación en Windows 10 disponemos en la documentación oficial de un ejecutable esp-idf-tools-setup-2.3.exe que nos facilita mucho la instalación de los requisitos. Únicamente deberíamos de ejecutarlo seleccionando la versión 4.0 o superior de las librerías ESP-IDF y posteriormente instalar Visual Studio Code con las extensiones especificadas. [10]

3.2.2.2 Implementación y flujo de ejecución

La implementación se realizará en el lenguaje de programación C utilizando Visual Studio Code con la documentación oficial de Espressif IDF para ESP32 en la versión actual Stable que corresponde a la versión 4.0. [11]

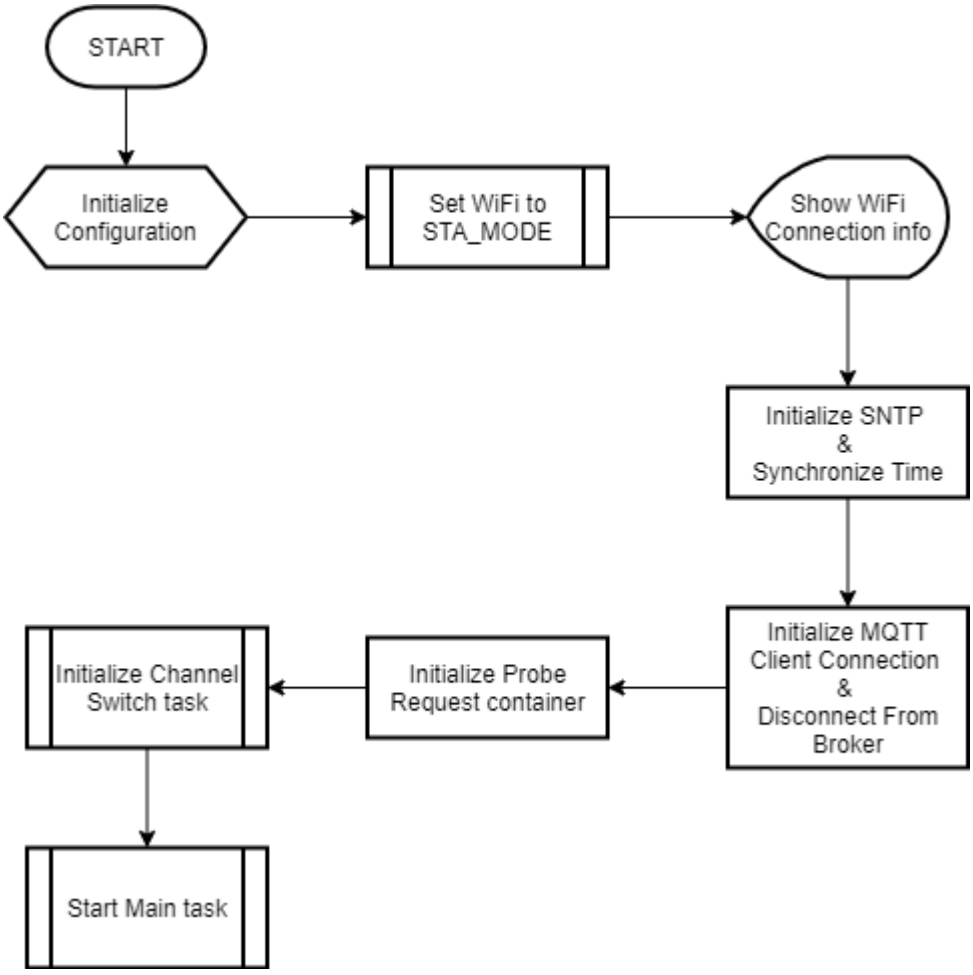


Fig. 17 Flujo de ejecución inicial

Cuando se inicia el microcontrolador primeramente se inicializa la configuración y todos los componentes necesarios para el correcto funcionamiento del dispositivo.

Debido a que no disponemos de ninguna función que nos permita saber la marca de tiempo actual en Unix (Current Unix Timestamp) nos conectaremos a un punto de acceso estableciendo el modo de funcionamiento en STA_MODE. Una vez nos hemos conectado mostramos información de la conexión:

- Dirección Mac
- Dirección IP
- La máscara de Subred
- Gateway

Seguidamente inicializamos la conexión con SNTP (Network Time Protocol Server) que es un standard del protocolo IP para sincronizar los relojes del ordenador o en este caso de la ESP32 a través de la red. Sincronizaremos nuestro reloj interno con el tiempo actual UTC de manera que podremos extraer en un futuro el timestamp sin necesidad de estar conectados a internet. [12]

Este paso es imprescindible y necesario debido a que como veremos más adelante solo disponemos de una antena WiFi y la API no soporta una conexión WiFi al mismo tiempo que estamos recogiendo las tramas de red en modo Promiscuo.

Es decir, no podemos operar en varios modos WiFi al mismo tiempo y, es más, tampoco podemos ir cambiando de canal de WiFi en el que escuchamos en modo promiscuo al mismo tiempo que estamos conectados a un punto de acceso por lo que nos sería imposible recoger el timestamp si no realizáramos esta sincronización. [13]

Estando todavía conectados a la red WiFi inicializamos el cliente de MQTT y su conexión con el broker estableciendo la configuración del cliente de MQTT. Después, nos desconectamos e inicializamos el contenedor donde se guardará la información recogida de las tramas. Este contenedor es simplemente una implementación de una estructura de datos (un SET) así evitaríamos tramas que pudiesen duplicarse.

Finalmente se inicia la tarea de cambio de canal y se ejecuta el bucle de la tarea principal hasta que se apaga el microcontrolador.

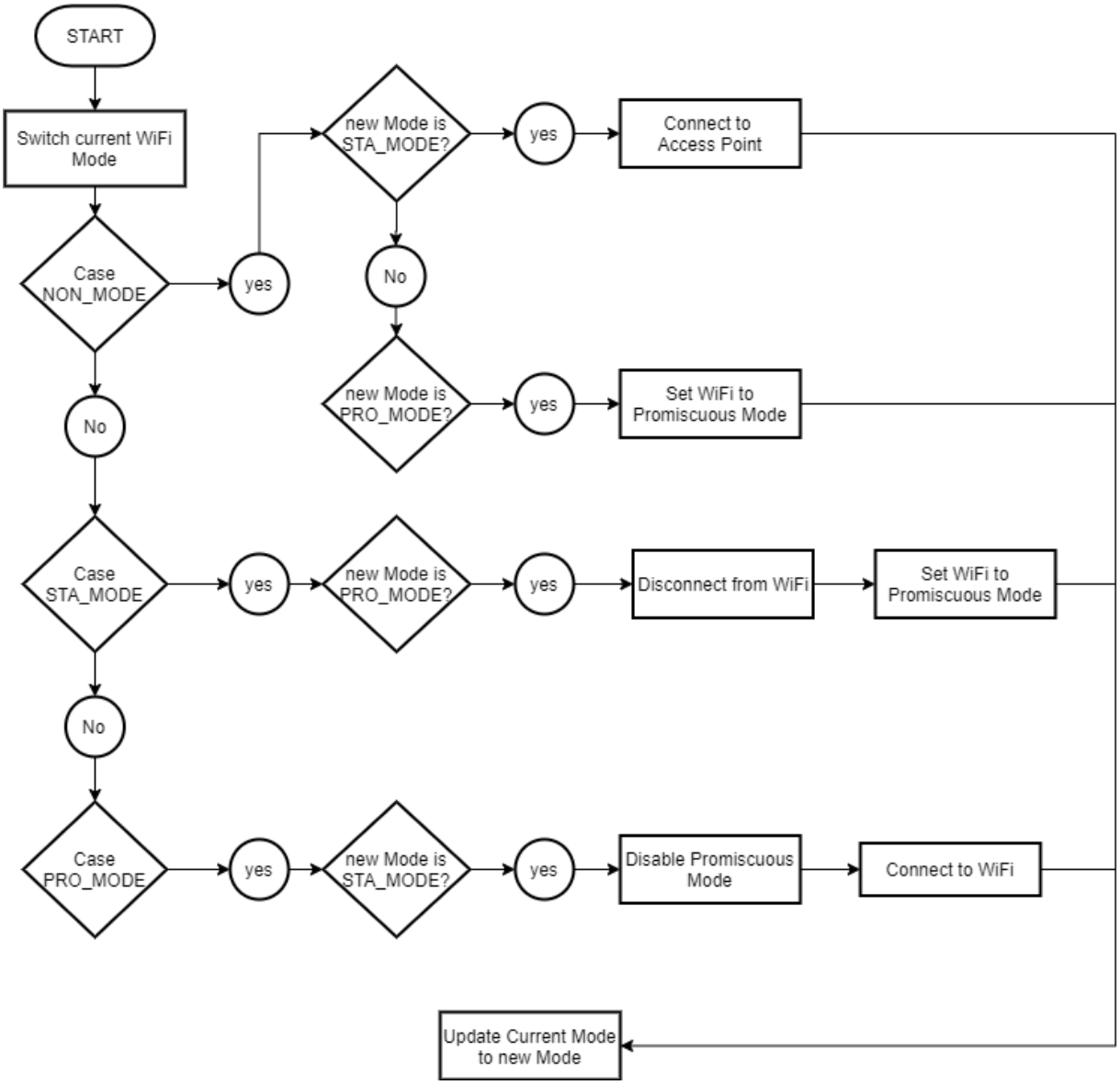


Fig. 18 Flujo de ejecución cambio de modo de operación WiFi

El microcontrolador tiene 3 estados de funcionamiento WiFi que hemos definido por los modos:

- NON_MODE: Es el modo de ejecución por defecto inicial que no hace ningún tipo de operación – ni está conectado a un punto de acceso ni tampoco recoge el tráfico de WiFi de su alrededor.

- STA_MODE: Es el modo de ejecución en el que el microcontrolador tiene desactivado las tareas relacionadas con la recogida del tráfico de red por los diferentes canales y en el que se conecta al punto de acceso configurado.

- PRO_MODE: Es el modo de ejecución en el que el microcontrolador está desconectado de la red WiFi. Se realizan las tareas de cambio de canal WiFi cada 100ms y recoge los paquetes de red de su alrededor.

El flujo de ejecución de cambio de modo es muy simple. Únicamente se comprueba el modo actual y de ejecución y el nuevo modo que se quiere establecer y se actualiza el estado actual por el nuevo.

Como podemos observar desde el estado inicial de NON_MODE nos conectamos al punto de acceso por primera vez. Las siguientes veces y desde el resto de los estados como ya tenemos la configuración cargada con un simple scan rápido nos podemos conectar y desconectar en apenas unos segundos.

Activar o desactivar el modo promiscuo es tan sencillo como ejecutar la siguiente línea de código con un true o false:

- `ESP_ERROR_CHECK (esp_wifi_set_promiscuous(false));`

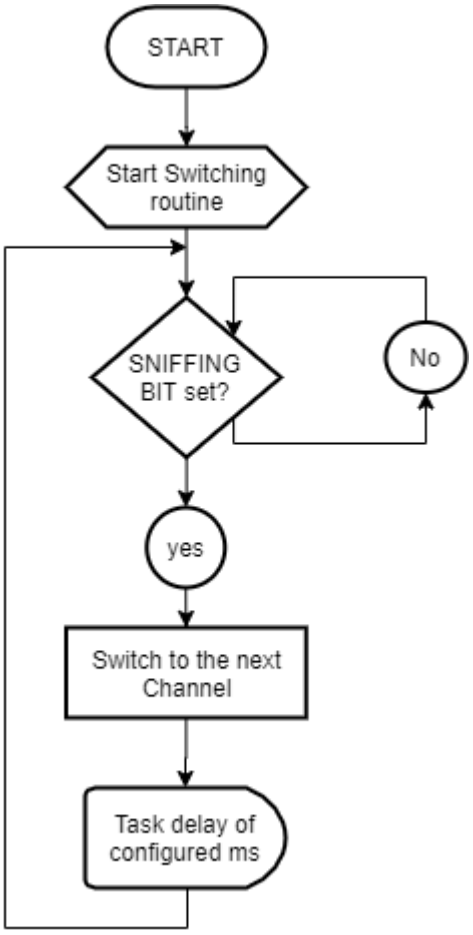


Fig. 19 Flujo de ejecución de la rutina de cambio de canal sobre el que se recoge las tramas de Probe Request

La rutina de cambio de canal WiFi se crea y ejecuta en nuestro segundo core del microprocesador. Esta tarea se inicializa con la siguiente línea de código:

- `xTaskCreatePinnedToCore (&channel_task, "channel_task", configMINIMAL_STACK_SIZE, NULL, 5, &xHandle_channel, 1);`

Este método nos permite crear una tarea &channel_task y asignarla para que se ejecute en un core en concreto el cual es especificado en el último argumento (1). Creamos esta tarea vinculada a este core para que cuando estemos realizando la recogida de datos esta no se vea afectada en rendimiento ya que queremos evitar la mayor pérdida de paquetes de red.

Como se ha mencionado anteriormente, no podemos realizar cambio de canal cuando estamos conectados a una red WiFi porque esto nos produciría un “core panic error crash”, para evitar este error y con el objetivo también de evitar ir creando, y eliminado la tarea – lo cual afectaría al rendimiento, se ha implementado un control de flujo mediante el sincronizador de tareas que nos ofrece la librería de la API “FreeRTOS Event Groups”.

El Event Group de la librería contiene una lista de Bits a los cuales se les asigna una Flag. Mediante los siguientes métodos podemos controlar y sincronizar las diferentes tareas [14]:

- `xEventGroupCreate ();`
- `xEventGroupWaitBits (EventGroup, BitsToWaitFor, ClearTheBitsOnExit, WaitForAllBits, TimeOut);`
- `xEventGroupSetBits (EventGroup, BitsToSet);`
- `xEventGroupCreate (EventGroup, BitsToClear);`

Cuando esta tarea se ejecuta lo primero que hace es comprobar si el SNIFFING_BIT el cual hemos especificado como BIT1 se encuentra activo, si este bit se encuentra activo entonces se comprueba cual es el canal actual en el que se está recogiendo los paquetes de red y se pasa al siguiente del 1 al 11 de forma cíclica; en caso contrario esperamos hasta que el BIT1 esté activo.

Para cambiar de canal usamos el método:

- `esp_wifi_set_channel(sniffing_channel, WIFI_SECOND_CHAN_NONE);`

Finalmente, para dar un tiempo a cada canal se genera un delay o espera de 100 milisegundos y se vuelve a ejecutar el bucle desde el inicio.

Para generar un delay de 100 milisegundos empleamos el método:

- `vTaskDelay(100 / portTICK_PERIOD_MS);`

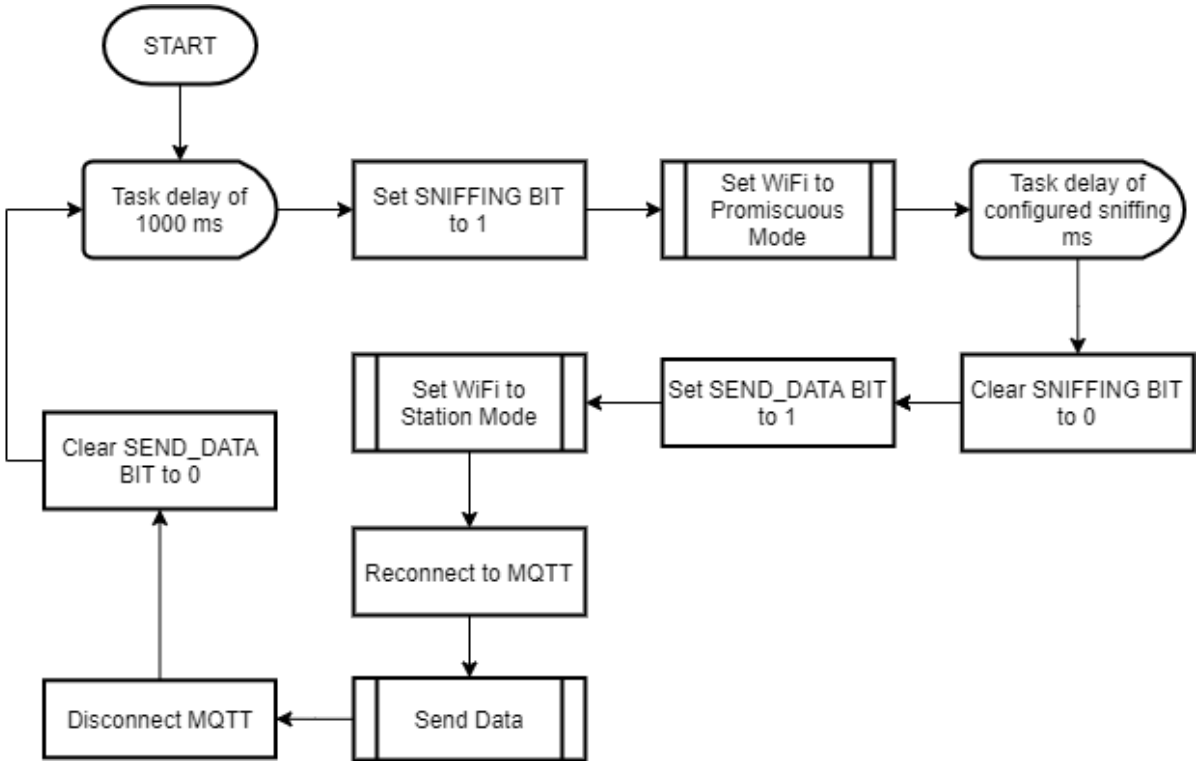


Fig. 20 Rutina principal de la aplicación

La rutina principal de la aplicación se ejecuta después de inicializar la tarea del cambio de canal como hemos podido ver en la Fig. 6. El flujo de esta rutina se basa en cambiar de modo de funcionamiento entre promiscuous y station y enviar los datos almacenados en el contenedor a nuestro broker mediante MQTT.

Primeramente, se proporciona un tiempo de delay de 1 segundo y seguidamente utilizando las funciones presentadas de Event Group establecemos el BIT1 de SNIFFING_BIT activando inmediatamente el modo promiscuo.

Seguidamente forzamos un delay de 1 minuto en el cual se irá recogiendo y guardando los paquetes recogidos de los diferentes canales ya que nuestra tarea de cambio de canal se activará al haber establecido el BIT1.

Una vez transcurrido el tiempo limpiamos el BIT1 y establecemos el BIT2 de SEND_DATA_BIT cambiando de modo de funcionamiento a station para poder reconectarnos a nuestro punto de acceso y al servidor de MQTT al cual enviamos nuestros ProbePackets en formato JSON.

Finalmente nos desconectamos de MQTT y limpiamos el BIT2 para volver a empezar la rutina nuevamente.

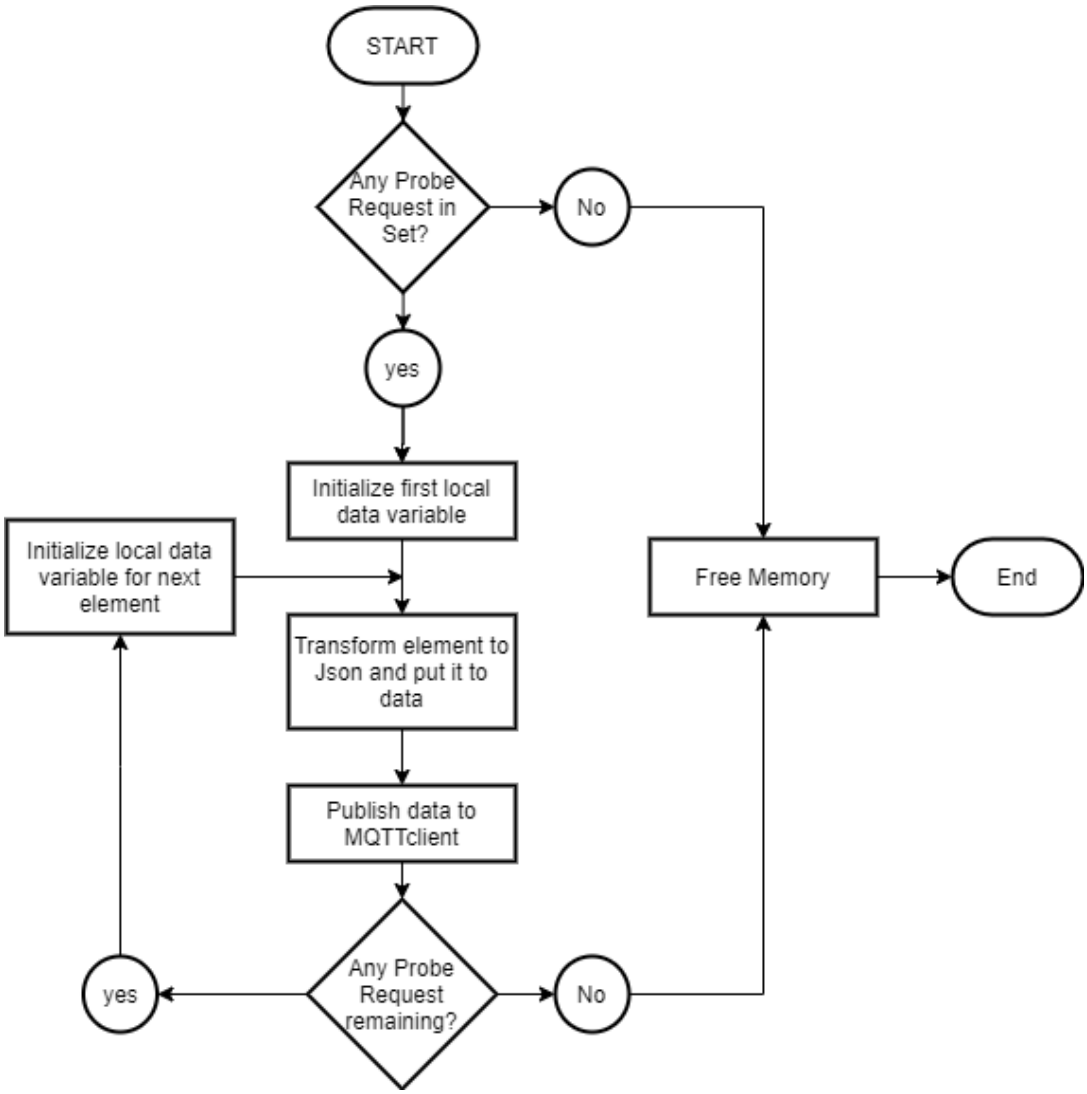


Fig. 21 Flujo de ejecución para enviar la información de los Probe Request

Para el envío de datos simplemente se itera sobre nuestro contenedor o Set de Probe Requests para cada una de ellas se inicializa una variable local (char data[512]) que será la que contendrá en formato JSON la información a enviar. Seguidamente se transforma a Json y se envía mediante MQTT publicándolo en el broker en el topic “probes”.

Esta iteración se realiza para cada elemento del contenedor y finalmente se libera la memoria reservada para evitar memory leaks ya que se trabaja con punteros.

Promiscuous callback function

Por último, hay que comentar la función que permite recoger todos estos paquetes.
Esp-idf nos proporciona un filtro automático para tan solo gestionar un cierto tipo de paquetes. Mediante el siguiente filtro y método podemos aplicarlo a la hora de instanciar el modo promiscuo.

```
const wifi_promiscuous_filter_t filter = {
.filter_mask = WIFI_PROMIS_FILTER_MASK_MGMT
};

// Set filter for only MGMT packets
ESP_ERROR_CHECK (esp_wifi_set_promiscuous_filter (&filter));
```

Con este filtro nos aseguramos de que solo capturaremos paquetes de tipo gestión y por tanto solo tenemos que filtrar aquellos que sean de subtipo probe request con los bits (0100). Asimismo, tenemos que registrar la función de callback que recibirá los eventos de 1 paquete de red:

```
// registering the callback to invoke on packet sniffed event
ESP_ERROR_CHECK (esp_wifi_set_promiscuous_rx_cb (&sniffer_callback));
```

Esta función simplemente filtra los paquetes de subtipo probe request, establece el timestamp de cada paquete y extrae la información necesaria para guardarla en el siguiente struct el cual es almacenado en el contenedor:

```
typedef struct {
    uint8_t id;
    int64_t timeStamp;
    uint8_t channel;
    int8_t rssi;
    uint8_t dest_mac [MAC_LENGTH]; /* Receiver  address    */
    uint8_t src_mac  [MAC_LENGTH]; /* Sender    address    */
    char ssid [MAX_SSID_LEN];
    uint16_t  seq_number;
    char ht_cap [5];
}ProbePacket;
```

Una de las cosas más importantes de esta función es el marcador IRAM_ATTR justo al inicio de la función de sniffer_callback:

```
IRAM_ATTR void sniffer_callback(void *buf, wifi_promiscuous_pkt_type_t type);
```

A grandes rasgos este marcador hace que el código compilado se coloque en una sección llamada “.dram.text”. El gestor de arranque ESP32, copiará esas secciones de código en la RAM real al inicio y antes de dar control a la aplicación.

A efectos prácticos, el código compilado se encuentra en la flash pero con este marcador podemos hacer que se encuentre en la RAM de manera que el código se ejecutará de una manera más rápida [15].

Este pequeño truco, nos permitirá augmentar enormemente la capacidad de captura de paquetes de red evitando perder la gran mayoría de ellos, esto se debe a que queremos que esta rutina de servicio de interrupción (ISR) se ejecute lo más rápido posible, es decir, queremos entrar y salir lo más rápido posible de ella. [16]

Por finalizar comentar que se dispone de un fichero de configuración personalizable en el que se pueden definir los siguientes argumentos para facilitar la gestión del proyecto:

- **WIFI_SSID:** Nombre del punto de acceso al que nos contactaremos.
- **WIFI_PSW:** Contraseña del punto de acceso.
- **DATA_SERVER_ENDPOINT:** Servidor al que enviaremos los datos.
- **MQTT_TOPIC:** Nombre del topic del broker.
- **MQTT_QOS:** Protocolo QoS de MQTT
- **CHANNEL_INTERVAL:** Intervalo de tiempo entre el cambio de canal.
- **ESP_ID:** Identificador del dispositivo.
- **SNIFFING_TIME:** Tiempo que se estará recopilando paquetes.
- **DEBUG_MEMORY:** Flag que permite activar un seguimiento de la memoria libre actual. Especialmente útil en el desarrollo para evitar fugas de memoria.
- **VERBOSE:** Muestra mensajes durante la ejecución en consola.

3.2.2.3 Despliegue del código en los microcontroladores

Para desplegar el código en los microcontroladores podemos hacerlo mediante comandos o aprovechar las funciones que nos da Visual Studio directamente con las cuales no tendremos que ejecutar ningún comando. No obstante, los comandos que se ejecutan son los siguientes:

- Abrir el menú de configuración: **idf.py menuconfig**
- Hacer un build del proyecto: **idf.py build**
- Flash de los binarios en la ESP32: **idf.py -p <<PORT>> [-b BAUD]** PORT debe ser reemplazado por nuestro nombre de puerto serie.
- Abrir la ventana de monitor: **idf.py -p <<PORT>> [-b BAUD] monitor**
- Limpiar el proyecto: **idf.py fullclean**

Podemos encontrar los comandos dentro del fichero tasks.json de nuestra carpeta .vscode del proyecto. También es necesario mencionar la importancia de reiniciar la NodeMCU manteniendo pulsado el botón de BOOT durante el flash.

Finalmente la desconectamos del ordenador y cuando volvamos a proporcionar energía se reiniciará automáticamente con el código que le habíamos subido.

3.2.3 Base de Datos

Como base de datos se ha decidió emplear PostgreSQL en su versión 12.0. Por la facilidad en su implementación y por el soporte que nos proporciona a la hora de trabajar con campos de tipo Json y macaddr.

El objetivo de esta base de datos es únicamente de persistencia de datos y como configurador de alertas / notificaciones para que estas puedan ser cargadas cuando el servicio está en caliente sin necesidad de tenerlo que reiniciar.

La base de datos constará de 2 tablas:

- Tabla **probes**

Columna	Tipo	Nullable
id	bigserial	Not null
espid	smallint	Not null
channel	smallint	Not null
timestamp	bigint	Not null
rssi	smallint	Not null
macsrc	macaddr	Not null
ssid	character varying (33)	
seqnumber	integer	Not null
htcapinfo	character varying (5)	
vendor	character varying (125)	

En esta tabla guardaremos los datos ya procesados de nuestro broker. Más adelante se mostrará algunas queries con las que podremos extraer KPIs.

- Tabla **alerts**

Columna	Tipo	Nullable
id	bigserial	Not null
trigger	smallint	Not null
name	character varying (33)	Not null
macs	json	Not null
email	json	Not null
condition	json	
last_trigger	bigint	Not null

Esta tabla contiene la configuración de nuestras alertas la cual se usa en el broker para comprobar si la trama recibida debe lanzar una notificación o no.

- **id** es la clave primaria, un autoincrementador.
- **trigger** indica el tiempo mínimo que tiene que pasar entre notificación y notificación.
- **name** es el nombre de la alerta.
- **macs** es una lista json que contiene todas las posibles macs que podemos detectar con la alerta.
- **email** es una lista json de los correos electrónicos a los cuales enviaremos las notificaciones de detección.
- **condition** es un objeto json que contiene un mapa de condiciones clave:valor. Donde la clave corresponde al id de la ESP32 y el valor el mínimo de RSSI que debe tener en la detección por ese microcontrolador.
- **last_trigger** contiene la última vez que se produjo una notificación con esta alerta. Si el valor es -1 significa que nunca se realizó una notificación.

Un ejemplo de configuración de alerta seria:

id	trigger	name	macs	email	condition	last_trigger
1	120	Test Alert	["a8:9c:ed:fa:1c:64"]	["victorbcn98@gmail.com","victorbcnlol@gmail.com"]	{"filters": {"0":-90, "1":-80} }	1589334921665

Más adelante veremos cómo se realiza esta comprobación y actualización.

3.2.4 MQTT Broker

Para la implementación del broker se ha decidió emplear el lenguaje de programación Java, en su versión 8. El motivo principal ha sido el conocimiento previo y soltura que se tiene sobre el lenguaje junto con la escalabilidad que nos proporciona a futuro y la gran cantidad de frameworks que podemos encontrar que facilitan nuestra implementación.

3.2.4.1 Prerrequisitos de implementación

- Java SDK 1.8 o superior
- Un entorno de desarrollo, en nuestro caso IntelliJ
- Gradle

En nuestro caso el código se ha desarrollado en Windows 10 pero es independiente del sistema operativo, solo tenemos que asegurarnos que tenemos la versión correcta de Java instalada.

3.2.4.2 Implementación

Para reducir al máximo el tiempo de desarrollo y facilitar la implementación se emplea el toolkit conocido como **vert.x**. Su librería principal define las APIs fundamentales para escribir aplicaciones asíncronas en red la cual se basa en el tan conocido **proyecto Netty**, una librería de redes asíncronas de alto rendimiento para la JVM. [17]

Uno de los puntos más fuertes de esta librería es su capacidad de manejar grandes cantidades de conexiones de red simultaneas con menos subprocesos que las API síncronas, como los servlets de java o las clases java.net de sockets.

Además, consta con un módulo dedicado a la mensajería con MQTT o RabbitMQ **(vertx-mqtt)** tanto para cliente como servidor, por lo que podemos implementar un broker de alto rendimiento con todas las funciones cubiertas y con capacidad de ser escalado a futuro, como, por ejemplo, añadir más topics o incluso añadir Apache Kafka de por medio. [18]

Por otra parte, se emplearán otras librerías como son:

- **Slf4j** Para el logging de la aplicación a diferentes niveles (Info, debug, error, trace). [19]
- **Apache commons (lang3, csv, email, io)** que nos facilitan las tareas tanto del guardado de información como para la creación de notificaciones. [20]
- **Google GSON** para gestionar el formato JSON de una manera rápida y eficiente. [21]
- **HikariCP** y **PostgreSQL** para la conexión en base de datos. [22]

Al inicio de nuestra aplicación se cargará la configuración que tenemos en nuestro fichero **configuration.properties** este fichero contiene la configuración siguiente:

- **SERVER_KEY_PATH & SERVER_CER_PATH** donde se especifica el path de la clave RSA de nuestra comunicación.
- **MQTT_BROKER_SERVICE_HOST** la IP de nuestro broker.
- **MQTT_BROKER_SERVICE_PORT** el puerto en el que ejecutaremos el broker.
- **MQTT_BROKER_TOPIC** el nombre del topic en el que recibiremos la información.
- **MAIL_SENDER** el email que enviará las notificaciones de alerta.
- **MAIL_PASSWORD** contraseña del email.

- **VERBOSE** para mostrar información extra por consola.
- **ENVIROMENT** el entorno de desarrollo que puede ser dev (desarrollo) o prod (producción).

Además, también cargamos nuestro fichero **oui.txt** que contiene los IEEE OUI públicos de la base de datos de wireshark y que usamos para posteriormente extraer el fabricante del dispositivo a través de la MAC de cada Probe Request. Más adelante hablaremos de está implementación.

Finalmente establecemos conexión con nuestra base de datos y cargamos en memoria las diferentes alertas que tenemos configuradas en base de datos. Estas configuraciones de alertas serán recargadas cada 10 minutos para facilitar su configuración cuando el servicio está en marcha.

La implementación del broker es muy sencilla gracias a la librería, únicamente necesitamos definir la IP de nuestro host y el puerto, activando su opción de SSL y estableciendo los certificados en su configuración:

```
MqttServerOptions options = new MqttServerOptions()
        .setHost(Configuration.MQTT_BROKER_SERVICE_HOST)
        .setPort(Configuration.MQTT_BROKER_SERVICE_PORT)
        .setKeyCertOptions(new PemKeyCertOptions()
                .setKeyPath(Configuration.SERVER_KEY_PATH)
                .setCertPath(Configuration.SERVER_CER_PATH))
        .setSsl(true);

MqttServer.create(Vertx.vertx(), options);
```

Seguidamente tenemos que inicializar un endpoint el cual será el encargado de escuchar, recibir las conexiones y gestionar las subscripciones en los diferentes topics. La mayoría del código ya está implementado en la documentación oficial y solo tendremos que encargarnos de crear el topic en el cual recibiremos la información, para ello creamos una clase que se llama **ProbeTopicHandler.java**

Cada mensaje que se reciba en el topic configurado será procesado de la siguiente manera:

1. Se transforma de JSON a un objeto que hemos llamado **Data** y contiene toda la información para que sea manipulada de una manera más rápida y eficiente.
2. Se incrementa nuestro contador de numero de paquetes recibido. En nuestra consola se muestra el número de paquetes recibido cada minuto con el uso de un Thread.
3. Se guarda la información de la Probe Request en formato CSV en un fichero data.csv de disco.

4. Se guarda la información de la Probe Request en la base de datos PostgreSQL.

5. Se procesa la Probe Request para comprobar si esta debe de lanzar una alerta según las alertas que tenemos configuradas en ese instante.

3.2.4.3 Extracción del fabricante del dispositivo

La dirección MAC está formada por 12 dígitos en base hexadecimal (48 Bits) que se representa mediante 6 octetos cada uno separado generalmente mediante la notación de los dos puntos ":". Estas direcciones son únicas por dispositivo y se integran en la tarjeta de red durante la fabricación por eso se conoce como dirección física de un dispositivo. [23]

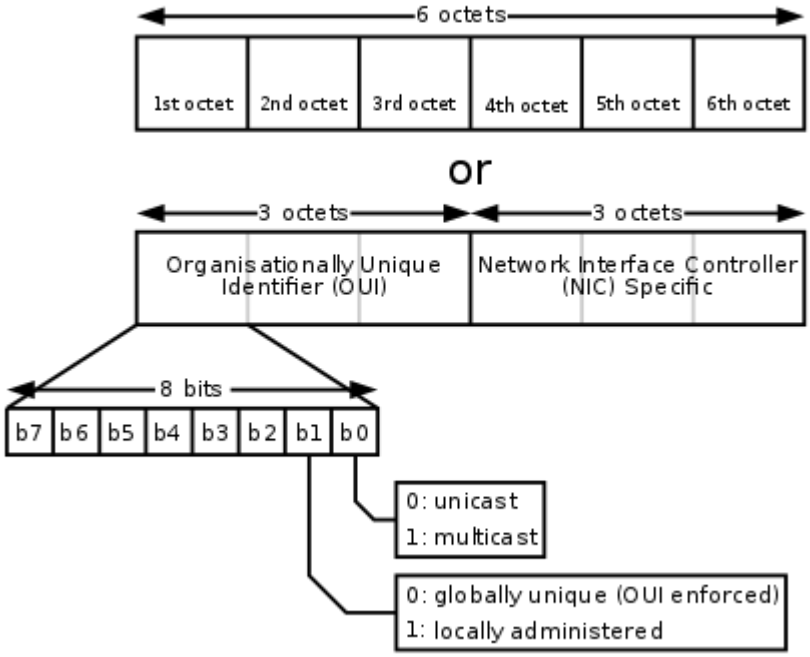


Fig. 22 Estructura de una dirección MAC

Los primeros 6 dígitos de una dirección MAC identifican al fabricante, también conocido como OUI (Identificador Único Organizacional). El comité de autoridad de Registro IEEE asigna estos prefijos MAC a sus proveedores registrados.

Por ejemplo, conocemos que el OUI **CC:46:D6** pertenece a un dispositivo **Cisco** o **3C:5A:B4** a un dispositivo de **Google, Inc.**

Por otra parte, los últimos seis dígitos representan el controlador de interfaz de red, que es asignado por el fabricante. [24]

Como se mencionó anteriormente, podemos obtener información relevante sobre los usuarios a partir de la dirección de MAC de su dispositivo. Excluyendo los intentos de las compañías en intentar conservar el anonimato mediante técnicas para ocultar su dirección

MAC real en los Probe Request podemos identificar que dispositivos nos están intentando engañar.

Esto es posible gracias a que, en dispositivos anonimizados, el bit de direccionado localmente (el segundo bit menos significativo del primer octeto ***b1***) de la dirección MAC se establece forzosamente en 1. Esto es muy bueno para nosotros ya que nos permitirá excluir esos dispositivos a la hora de analizar los datos con una simple consulta en Base de Datos.

Existen miles de herramientas online que nos permiten identificar los dispositivos a partir de su MAC de manera gratuita. No obstante, estas herramientas no abarcan la totalidad de los dispositivos o incluso pueden realizar fallos en la identificación ya que solo contemplan máscaras de red de 24 bits, de hecho, el identificador oficial de Wireshark presenta fallos en algunas direcciones MAC.

Por este motivo se ha optado por adaptar una librería de alto rendimiento de Python (**manuf by Michael Huang – coolbho3k**) en Java. Como se explica en la librería de Python es mucho más precisa y tiene soporte para las máscaras de red. [25]

Esto es esencial ya una entrada habitual como por ejemplo **BC:EE:7B** (AsustekC), el fabricante “posee” la última mitad (24 bits) de la dirección MAC libres a su asignación.

No obstante, algunas entradas en el archivo oficial aparecen también con este formato:

- 00:1B:C5:00:00:00/36 Convergi # Converging Systems Inc.
- 00:1B:C5:00:10:00/36 OpenrbCo # OpenRB.com, Direct SIA

/36 es una máscara de red, lo que significa que el fabricante “posee” solo los últimos 12 bits de la dirección MAC en lugar de los 24 bits habituales (48 bits - 36 bits = 12 bits).

Esto significa que Converging Systems solo puede asignar direcciones de 00:1B:C5:**00:00:00** a 00:1B:C5:00:0**F:FF** por lo que cualquier dirección fuera de este rango ya pertenece a otro fabricante.

La herramienta oficial de búsqueda Wireshark proporciona un fabricante de manera errónea en estos casos y devuelve solo “IEEE REGISTRATION AUHTORITY” cuando debería de devolver “Converging Systems Inc.”.

El algoritmo que se emplea para la identificación es bastante rápido con un tiempo de búsqueda promedio O(1), tiempo de carga de datos y espacio en memoria de O(n).

Simplemente se lee el archivo oui.txt en memoria. Este fichero se actualiza cada semana en el repositorio git de Wireshark y cada línea de fabricación se almacena en un diccionario con un una clave que nos sirve para iterar sobre los diferentes rangos de la MAC de cada fabricante según su máscara de red y un objeto Vendor que contiene toda la información del fabricante como valor.

Extraer el fabricante es tan sencillo como extraer el OUI y realizar una búsqueda en el diccionario iterando sobre las n mascaras de red posibles. Una explicación más detallada se puede encontrar en la documentación oficial de la implementación en Python [26].

3.2.4.4 Sistema de detección presencial y alertas

El mensaje de correo electrónico tiene como tema "**[TFG] Tracking detected**" y proporciona la información de esta última detección en formato HTML para darle estilo al mensaje recibido:

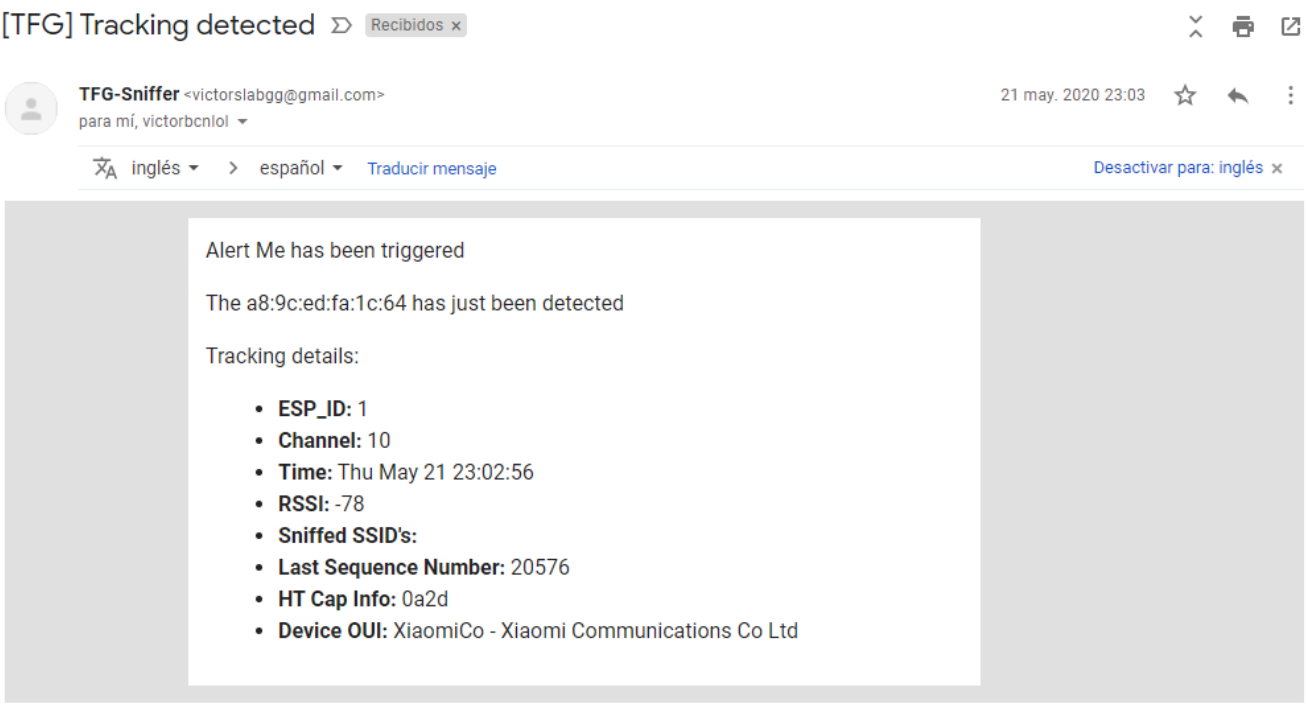


Fig. 23 Correo electrónico recibido sobre la dirección MAC a8:9c:ed:fa:1c:64

Para poder detectar la presencia de ciertos dispositivos y realizar las notificaciones es necesario realizar un procesado para cada Probe Request. Esta comprobación se realiza en paralelo y para garantizar su rapidez y evitar una comprobación por cada alerta de manera innecesaria la configuración de las alertas que están cargadas en memoria e indexadas por la dirección MAC.

De esta manera lo primero que se comprueba es si existe alguna alerta para esa dirección MAC y en caso afirmativo se realiza una comprobación por cada una ellas.
No obstante, y con el objetivo de incrementar aún más su velocidad, se comprueba primeramente en cada alerta si ha pasado el tiempo mínimo entre alerta y alerta evitando así hacer comprobaciones innecesarias.

Finalmente se comprueba si la Probe Request cumple las condiciones de la alerta: ID de la ESP32 que ha detectado el dispositivo y el mínimo de RSSI con el que lo ha detectado.

En el caso de que se cumplan todas las condiciones se establece el momento actual como el último instante en el que se lanzó la alerta, se actualiza en base de datos para garantizar la integridad del estado de la alerta y se procede a enviar una notificación por correo electrónico a todos los emails que tiene asignado la alerta.

3.3 Despliegue en un entorno real (Microsoft Azure)

El despliegue se realizo en mi lugar de residencia en un séptimo piso. Los microcontroladores fueron situados en diferentes zonas de la casa formando un triángulo entre ellos con una separación de aproximadamente 8 metros y paredes de por medio entre ellos.

- ESPID 1: Fue situado en mi habitación que esta cerca del patio interior que junta diversos bloques del edificio.
- ESPID 2: Fue situado en el comedor y es la zona que está más cerca del ascensor y de otros vecinos.
- ESPID3: Situado en la habitación de mis padres es la zona más próxima a la calle y al otro bloque de edificios desde el otro extremo.

Cada microcontrolador fue conectado a la corriente y, por tanto, encendido en el mismo instante con una diferencia de apenas unos segundos entre ellos.

3.3.1 Microsoft Azure

Aprovechando la ayuda que proporciona la Universidad de Barcelona con servicios de Microsoft Azure para estudiantes con un crédito de 170 euros se decidió contratar un servidor y una base de datos de pago por uso durante los 14 días.

3.3.1.1 PostgreSQL

Para PostgreSQL se contrató una base de datos en el Centro de Francia con las siguientes características y configuración:

- **Ubicación:** Centro de Francia
- **Versión PostgreSQL:** 12.0
- **Proceso y almacenamiento:** Basic, Gen5, 2 núcleos virtual 50GB de almacenamiento.
- **Período de retención de copia de seguridad:** 7 días
- **Redundancia de copia de seguridad:** Localmente
- **Crecimiento automático de almacenamiento:** Activado
- **Permitir que los servicios y recursos de Azure accedan al servidor:** Sí

La configuración es bastante simple y al final de todo nos especificaba el coste final por el uso de un mes entero. [27]

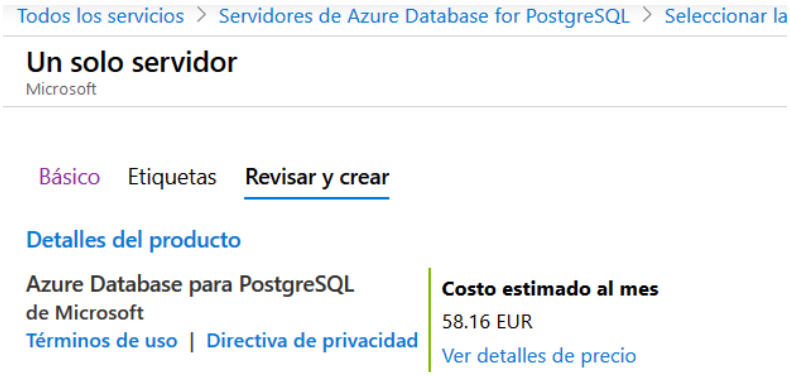


Fig. 24 Precio final de la instancia de base de datos PostgreSQL por mes.

Por otra parte, utilizando PgAdmin se conectó a la instancia de base de datos y se crearon las tablas necesarias. [28]

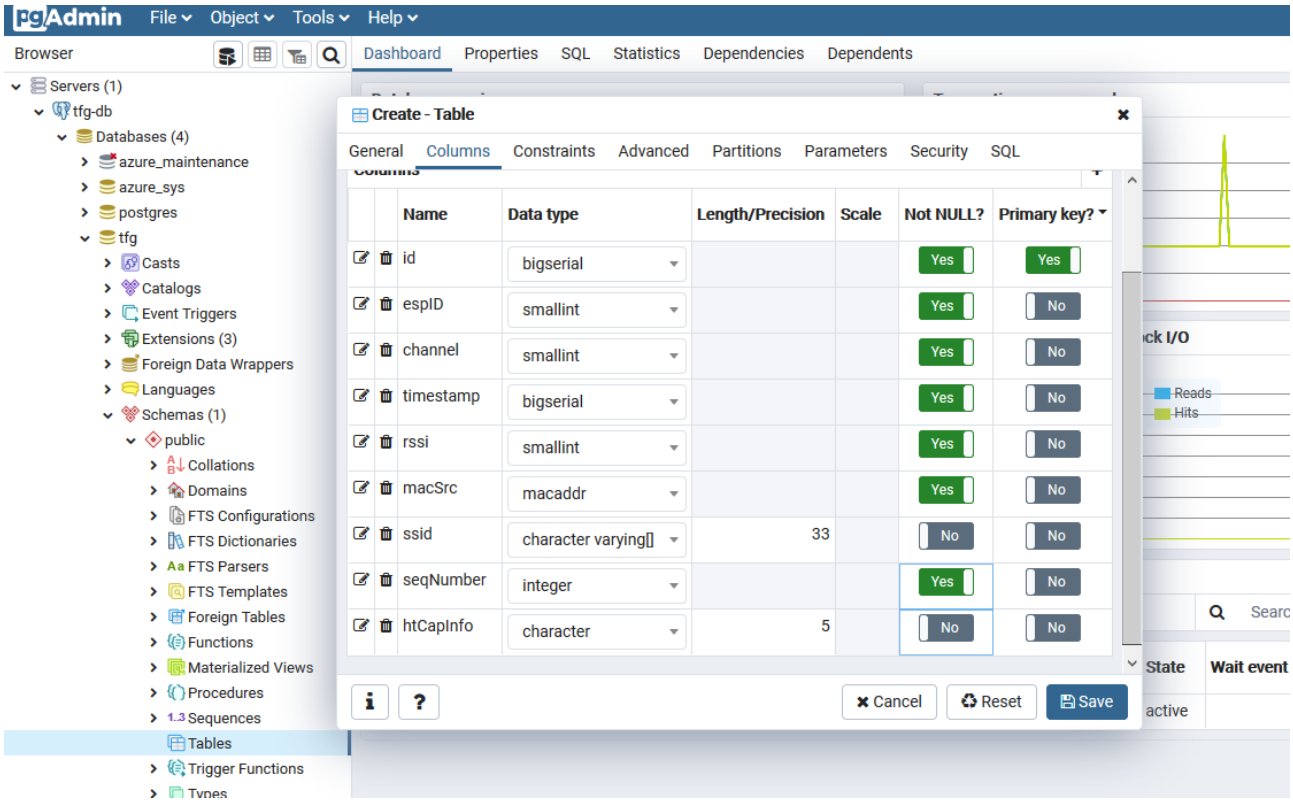


Fig. 25 Creación de la tabla probes en la base de datos de Microsoft Azure PostgreSQL contratada utilizando pgAdmin.

3.3.1.2 Servidor

En el caso del servidor se contrató nuevamente en el Centro de Francia con el objetivo de reducir las latencias al mínimo con las siguientes características y configuración:

- **Ubicación:** Centro de Francia
- **Sistema operativo:** Linux
- **Tamaño:** B4ms estándar
- **vCPU:** 4
- **RAM:** 16 GiB
- **Disco del SO:** SSD Premium 30 GiB
    - **IOPS Máximas:** 120
    - **Rendimiento máximo:** 25 Mbps
    - **Cifrado:** SSE con PMK

Esta configuración es más que suficiente para correr el broker, de hecho, podríamos haber instalado postgreSQL en el mismo servidor, pero se optó por esta configuración para aprender a utilizar la plataforma de Microsoft Azure.

En este caso al final de todo nos mostró el coste por hora de nuestra instancia. [29]

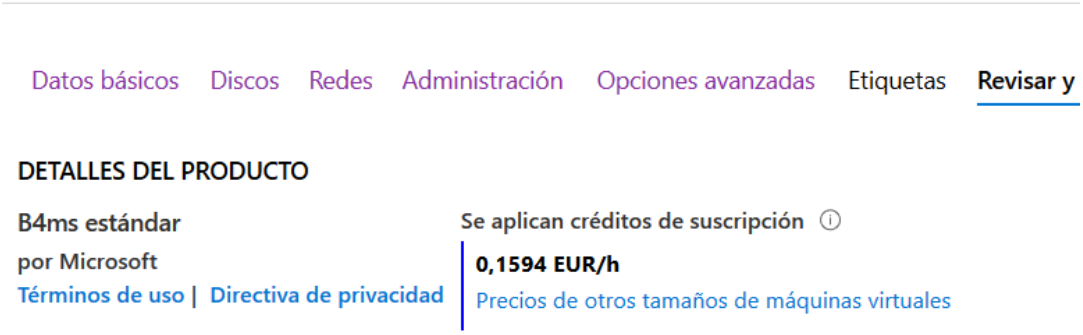


Fig. 26 Precio final de la instancia de la máquina virtual Linux por precio por hora en uso.

El coste de uso se esperaba que fuera de 14 días, es decir:

- 14D * 24h * 0,1594 Eur/h = **53,55 Euros**

Por otra parte, fue necesario configurar los puertos para así permitir recibir los datos por una parte en el puerto 8883 del broker y en el puerto 22 para la conexión via SSH.

Además, en la siguiente imagen también podemos observar que IP pública y privada tenemos que nos servirá para configurar la comunicación desde los microcontroladores al servidor.

Fig. 27 Configuración de puertos SSH y MQTT de la máquina virtual contratada donde podemos obtener también la IP Publica y privada del servidor.

4 RESULTADOS

En este apartado se presentan los resultados obtenidos tras recolectar datos durante 14 días en mi residencia personal.

4.1 Sistema de seguridad

Como se presentó en el apartado 5.4.4 cuando se detecta un dispositivo el cual tenemos marcado como alerta recibimos un mensaje por correo electrónico de que ese dispositivo ha sido detectado en la zona.

Tras realizar múltiples pruebas, pude observar que cuando el dispositivo entraba dentro del alcance de los microcontroladores con una diferencia de 1 minuto se recibía está notificación de manera inmediata y se intensificaba su eficacia si el dispositivo estaba siendo utilizado. Especialmente si al llegar a la residencia se manipulaba el teléfono realizando cualquier acción incluso con tan solo encender la pantalla.

De hecho, el número de Probe Request realizadas se ve incrementado drásticamente cuando el dispositivo esta en funcionamiento por lo que esto nos permite recopilar más información del usuario y en su defecto podría emplearse como sistema de seguimiento.

Aunque las notificaciones estaban restringidas a ciertas direcciones MAC se puede obviar cualquier filtro y se puede emplear para recibir notificación de cualquier dirección MAC de alrededor por lo que se podría usar perfectamente para tener una capa de seguridad extra dentro de un recinto en el que excluiríamos las direcciones MAC conocidas.

4.2 Análisis de datos y extracción de KPIs

Podemos hacer miles de consultas a base de datos y extraer multitudes de KPIs por lo que no se mostraran todas, únicamente las más relevantes para nosotros junto con algunos ejemplos para demostrar el potencial de estos datos.

4.2.1 Volumen de datos

Sobre el volumen de datos nos puede interesar saber qué cantidad de datos disponemos:

```
SELECT COUNT(*) FROM public.probes;
```

Esta consulta nos devuelve un único valor total de **1.305.501**. Es decir, casi se ha recopilado unas 90.000 entradas por día. Teniendo en cuenta que estamos en una residencia con 3 microcontroladores podemos hacernos a la idea de que trabajar en un entorno público incrementaría considerablemente esta cantidad.

De hecho, podemos saber cuántos datos ha recopilado cada microcontrolador realizando un simple GROUP BY espid.

```
SELECT espid, COUNT(*) FROM public.probes GROUP BY espid;
```

espid	count
1	431952
2	446783
3	426766

Podemos observar cómo los datos están casi equitativamente distribuidos siendo el segundo microcontrolador (el del comedor) el que más datos ha sido capaz de recopilar, probablemente porque se encuentra más cerca de otras residencias.

4.2.2 RSSI y Canal

Conocer que RSSI (Indicador de intensidad de señal) es el más frecuente en nuestros datos nos permite deducir si los dispositivos se encuentran muy lejos o no de nuestros microcontroladores.

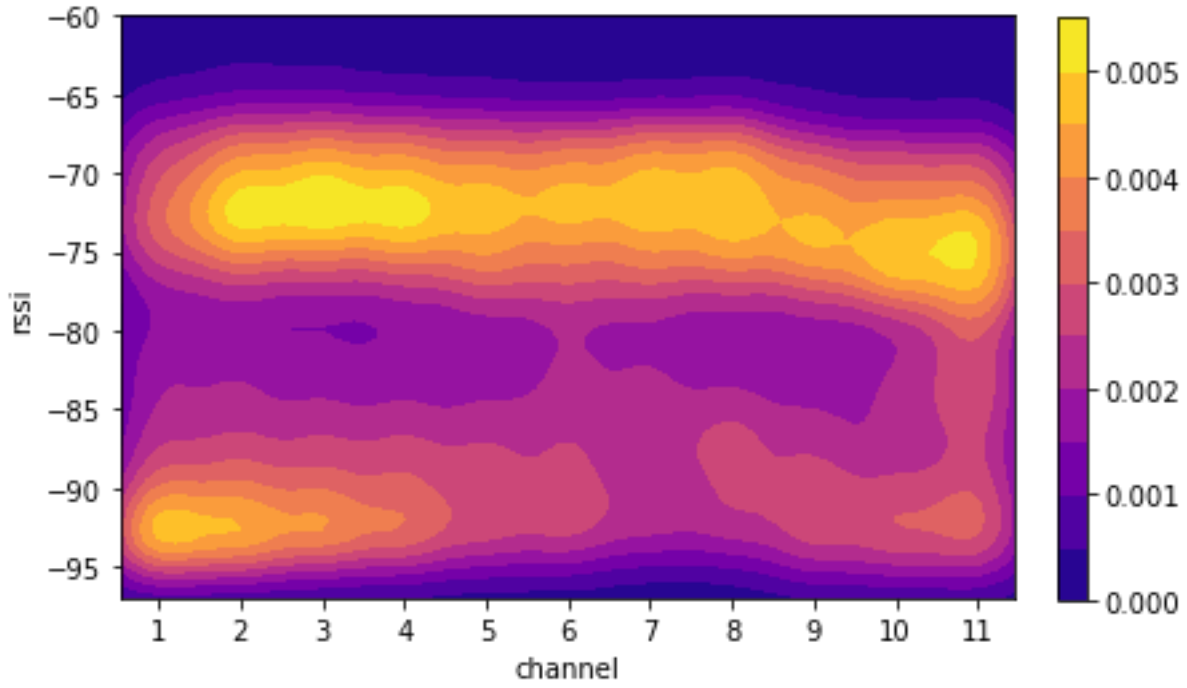
espid	moda rssi	mediana_rssi	avg_rssi
1	-72	-76	-77.94
2	-74	-79	-80.56
3	-73	-77	-79.91

Podemos observar como la mayoría de los datos se concentran en un rango de -70 dbm y -76 dmb.

Así mismo, podemos distribuirlos por los diferentes canales y así conocer sobre que canales se realizan más peticiones con relación al RSSI. Para poder observar esto de una manera visual emplearemos un gráfico 2D de densidad el cual nos mostrará la distribución de nuestros datos sobre los diferentes canales.

En el grafico las zonas más amarillas muestran mayor volumen de datos y las de colores violetas y azulados menor.

Volumen de datos recopilados en los diferentes canales agrupados por RSSI



Se puede observar cómo los canales 2, 3, 4 y 11 presentan un gran volumen de datos en RSSI de media -73 dbm. Así mismo, se puede ver como el canal 1 presenta un gran volumen de datos en la zona de los -93 dbm.

El mayor volumen de datos se encuentra entre estas franjas, aunque también podemos encontrar una pequeña cantidad entre los -75 dbm i los -90 dbm.

4.2.3 SSID

Respecto el campo de SSID, sabemos que no todas las Probe Requests nos proporcionan un SSID. A efectos prácticos, este campo nos puede interesar para saber que dispositivos o fabricantes son más susceptibles a proporcionarnos esta información.

```
SELECT vendor, count(*) as probes FROM public.probes
WHERE ssid is NOT NULL and ssid != '' or ssid != ' '
GROUP BY vendor ORDER BY probes DESC;
```

Esta query nos permite quedarnos únicamente con aquellas peticiones que reportaron un SSID y después agrupándolo por su fabricante podemos saber que fabricante de dispositivos fueron los que más SSID reportaron. Como la tabla es muy larga se mostrarán solo el top 10.

Fabricante	Probes
HonHaiPr	341295
Google	273792
SamsungE	154137
Unknown	110991
Apple	110201
IntelCor	65900
Technico	61498
XiaomiCo	35739
Guangzho	35354
Espressi	29659

Además, también se puede emplear para generar un mapa de calor sobre que ubicaciones suelen frecuentar los dispositivos de nuestra zona gracias a la herramienta gratuita de Wigle.

Con una simple query podemos extraer una lista con los nombres de los puntos de acceso.

```
SELECT DISTINCT(ssid) FROM public.probes;
```

Y utilizar alguna SSID de esta lista para buscarla en wigle.net [30]. Por ejemplo, entre toda la lista podemos encontrar el SSID **eduroam** bastante conocido por los estudiantes. Si lo buscamos en Wigle podemos obtener un mapa de calor como el siguiente:

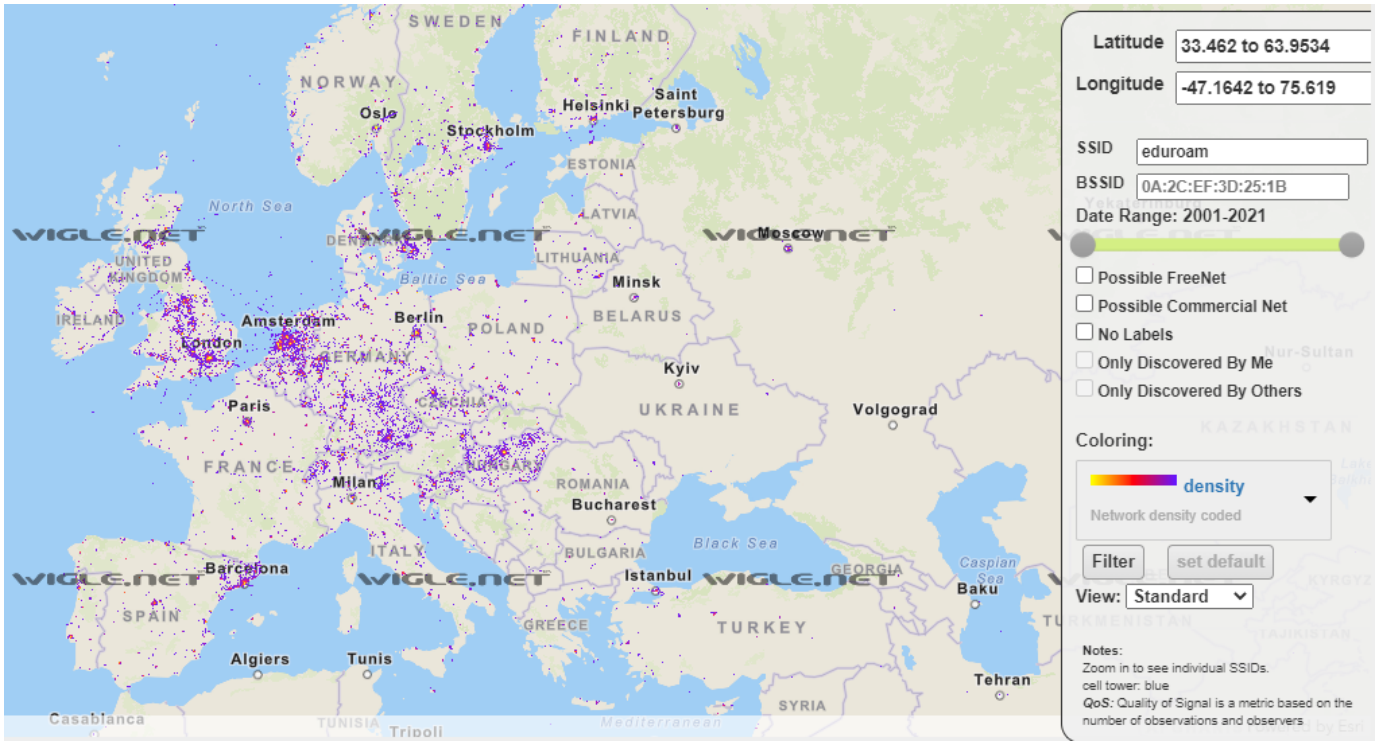


Fig. 28 Mapa de calor de la red WiFi por nombre eduroam obtenido mediante la herramienta de Wigle

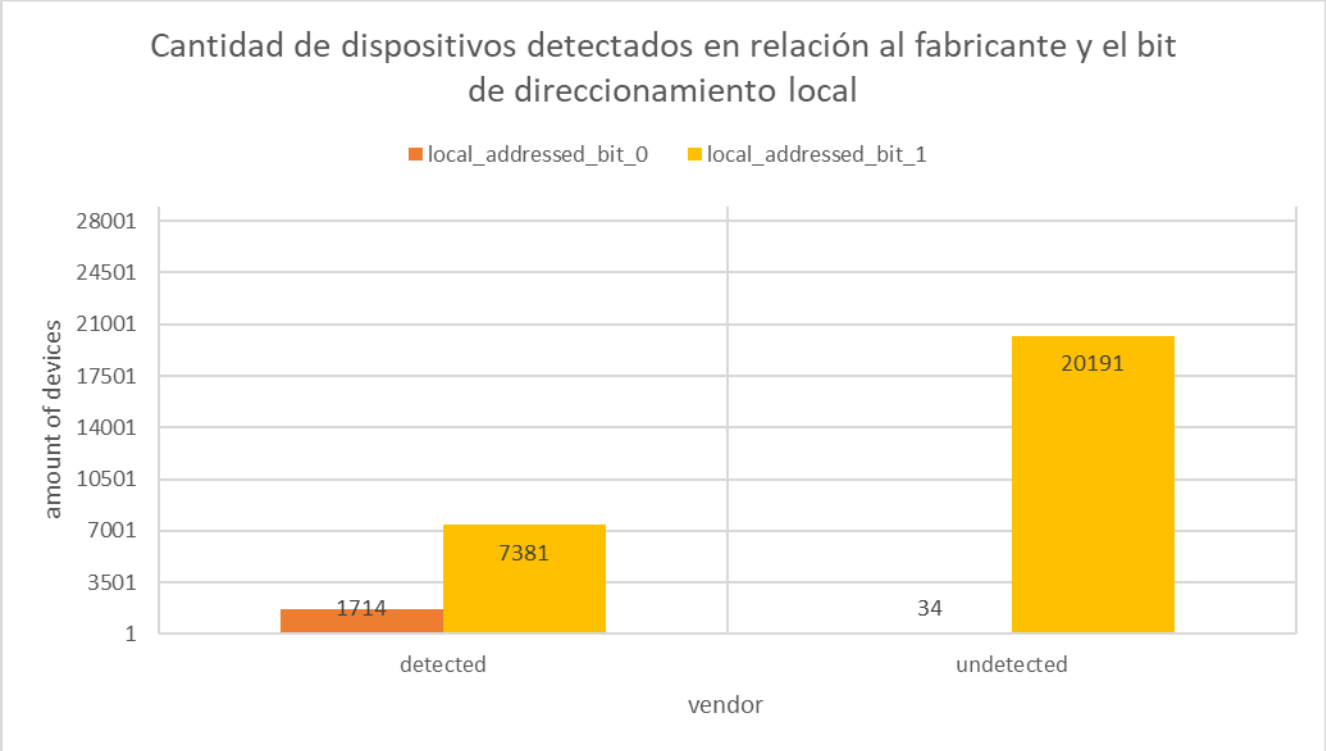
Podemos ver como esta red WiFi es bastante común y se encuentra presente por toda Europa. La propia herramienta de Wigle nos permite utilizar su API para buscar cualquier red WiFi y aunque en el caso de esta sea bastante común se puede encontrar redes WiFi de bares, centros públicos y privados etc. de todo el mundo.

4.2.4 Detección de dispositivos por MAC y Fabricante

Saber detectar cuantos dispositivos y su fabricante no debería de ser una tarea complicada. Tenemos que saber que cada dirección MAC es única por dispositivo y que únicamente aquellas direcciones MAC que presenten su **bit de direccionamiento local en 1** son dispositivos cuya MAC real es desconocida y de los cuales probablemente no conozcamos su fabricante.

	detected	undetected	Total
local_addressed_bit_0	1714	34	1748
local_addressed_bit_1	7381	20191	27572
Total	9095	20225	**29320**

Realizando una serie de filtros en base de datos podemos identificar qué dispositivos han sido detectados o no, es decir, si hemos identificado su fabricante o no. De hecho, podemos conocer cuántos dispositivos diferentes hemos detectado (**29.320**) y separarlos para saber si su dirección MAC contiene el bit de direccionamiento local en 1 o no.



Vemos claramente como no hemos identificado el fabricante de la mayoría de dispositivos, no obstante, sabemos que estos dispositivos reportan una MAC que no es real y por tanto no podemos identificar su fabricante, nada fuera de lo común.

Y apenas 34 dispositivos cuya dirección MAC es real no han sido detectados, porque no se encuentran en la base de datos que hemos empleado.

Por otra parte, se puede observar como un gran volumen de dispositivos si han sido detectados provienen de una MAC cuyo bit de direccionamiento está establecido en 1. Haciendo una consulta en base de datos se observa que de los 7381 dispositivos **7319** proceden del vendor Google con un OUI bastante identificativo: **DA:A1:19**.

Este hecho me lleva a deducir que existen ciertas compañías cuyo algoritmo para ocultar su MAC real es bastante reconocible y por tanto fácil de identificar.

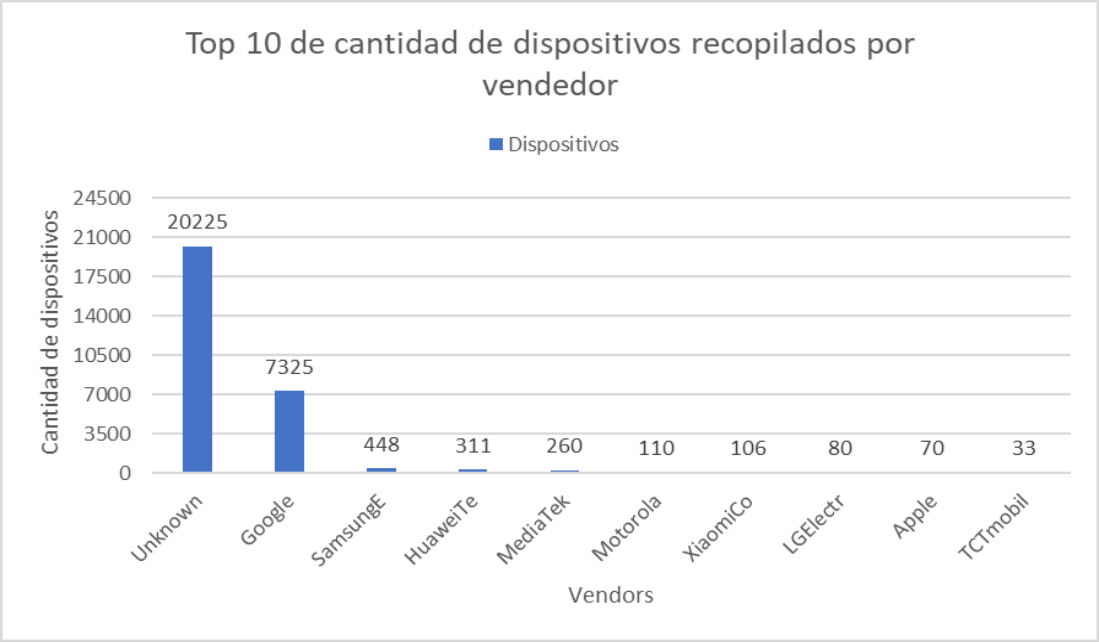
Es importante destacar que un dispositivo procura no reporta su dirección MAC real cuando no se encuentra conectado a ninguna red WiFi. No obstante, esto no siempre se aplica de hecho la mayoría de los dispositivos con el tiempo acaban reportando su MAC real ya sea porque encuentran una red pública o por que se acaban conectando a un punto de acceso.

Por otra parte, es necesario mencionar que de la totalidad de los datos recopilados apenas un **12,9%** proceden de dispositivos cuya MAC tenia el bit de direccionamiento local en 1:

local_addressed_bit_0_count	local_addressed_bit_1_count2
1137124	168377

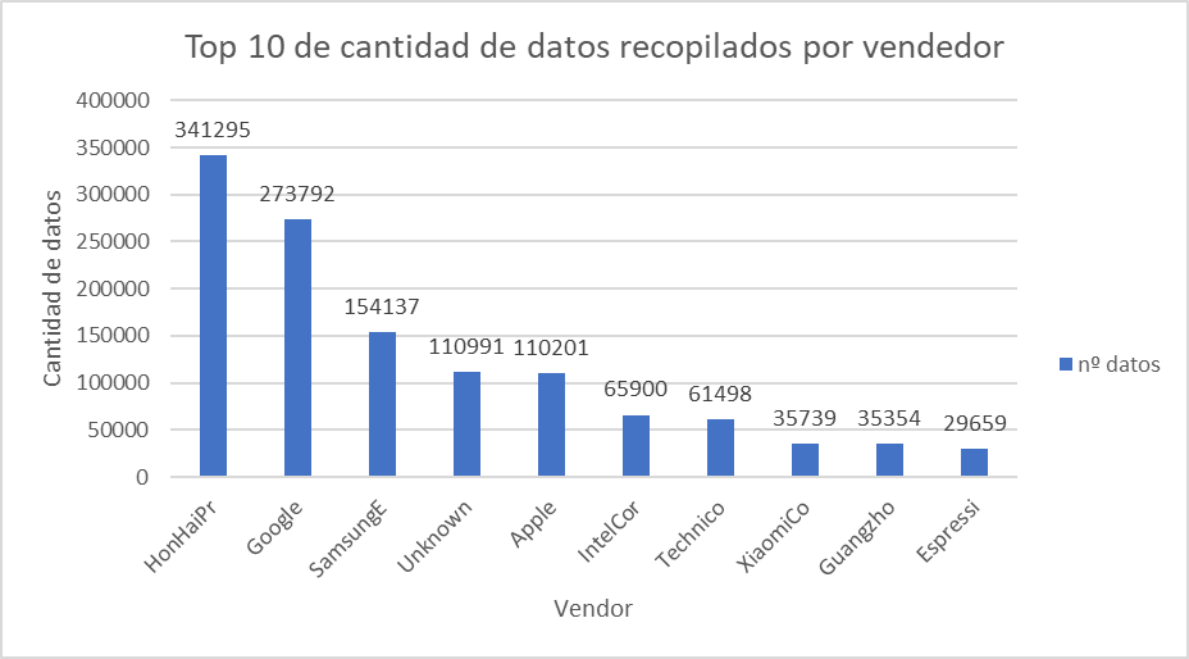
El hecho de que apenas haya un 12,9% de datos procedentes de MAC anonimizadas significa que el mayor volumen de datos es de mayor calidad o relevancia.

Así mismo en un estudio de mercado nos interesa saber cuántos dispositivos distintos hemos detectado de cada fabricante en nuestra zona. Como la tabla consta de 102 fabricantes se mostrará el top 10.



Excluyendo el fabricante “Unknown” del cual conocemos su procedencia podemos observar que claramente los dispositivos de Google son los que tienen una mayor presencia.

De la misma manera podemos extraer que volumen de datos de nuestra base de datos pertenece a cada fabricante. Nuevamente se muestra un top 10.



4.2.5 Evolución en el tiempo

Saber cuantos dispositivos hay en cada instante de tiempo o cuantas peticiones se realizaron en diferentes intervalos de tiempo no es una tarea tan sencilla.

A pesar de que disponemos de la función DATE_TRUNC en postgreSQL ha sido necesario implementar una función personalizada que nos permita agregar los tiempos para cualquier tipo de granularidad.

Esta función que se está empleando se ha obtenido de un proyecto de github que se encontraba con el mismo problema que este. [31]

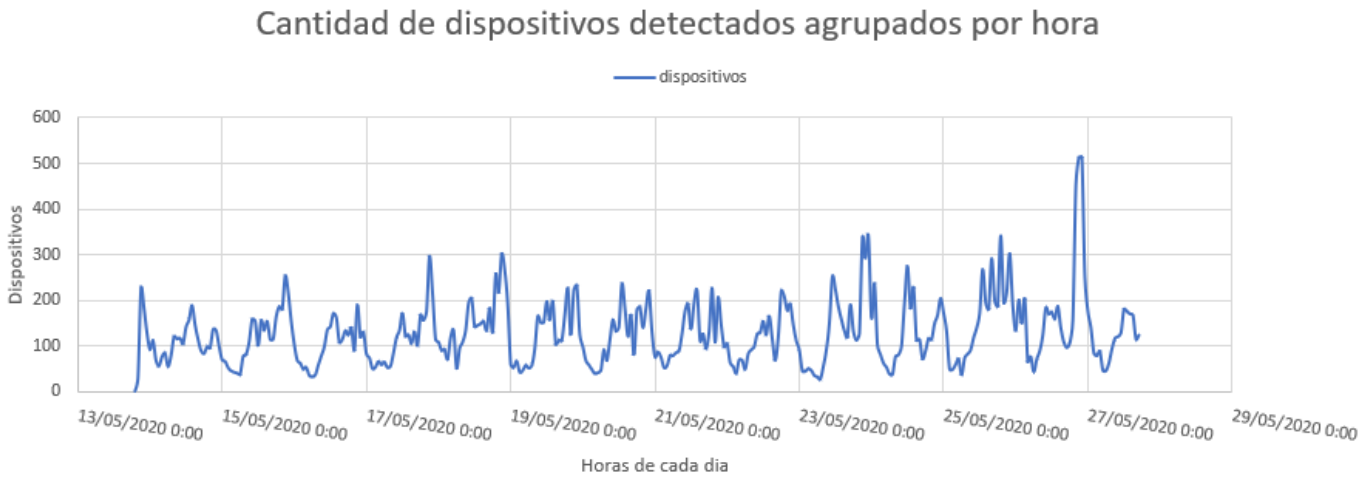
```
CREATE OR REPLACE FUNCTION date_trunc_by_interval
  (f_stamp TIMESTAMP WITHOUT TIME ZONE, f_interval INTERVAL)
    RETURNS TIMESTAMP WITHOUT TIME ZONE AS $$
DECLARE
  epoch_interval DOUBLE PRECISION : = EXTRACT('epoch' FROM f_interval);
BEGIN
  RETURN TO_TIMESTAMP(
    FLOOR(
      EXTRACT('epoch' FROM f_stamp) / epoch_interval) * epoch_interval
    ) AT TIME ZONE 'UTC';
END;
$$ LANGUAGE PLPGSQL
IMMUTABLE;
```

A efectos prácticos esta función funciona de manera similar a DATE_TRUNC pero nos permite agregar por cualquier tipo de granularidad que queramos 10 Seconds, 15 minutes, 1 month, 2 hours etc.

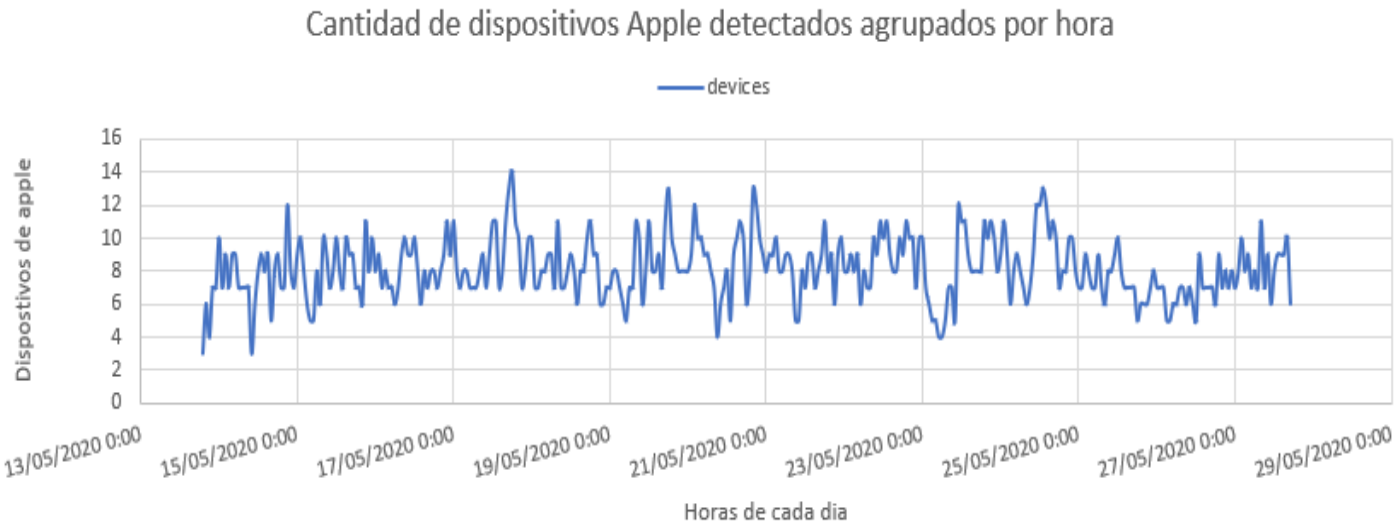
A modo de demostración se mostrará cuantos dispositivos fueron identificados cada día y hora adaptando la siguiente query según la granularidad de tiempo que queremos.

```
SELECT count(DISTINCT(macsrc)) as devices,
date_trunc_by_interval(TO_TIMESTAMP(timestamp/ 1000) AT TIME ZONE
'Europe/Madrid', '1 Day')
as times from public.probes GROUP BY times;
```

Estos gráficos nos permiten saber en qué momentos hubo más dispositivos por lo que en recintos nos permitiría inferir el volumen de gente.



De hecho, podríamos llevar esto aún mas lejos y seleccionar un fabricante en concreto por lo que podríamos saber que tantos dispositivos de una marca estaban en un instante de tiempo.



O incluso podemos filtrar por un día en concreto.

5 CONCLUSIONES

Implementar un sistema que permita recopilar todas las Probe Request a un bajo coste no es una tarea sencilla. Los microcontroladores ESP32 han demostrado tener un gran potencial de adaptación y capacidad, han presentado una buena respuesta en la recolecta de datos respecto a su bajo coste y son muy sencillos de montar y desplegar.

No obstante, también han presentado ciertas de carencias que en un principio no se esperaban y que son necesario mencionar:

- Tienen limitaciones de memoria, probablemente en un espacio donde haya miles de peticiones por segundo podría existir un problema de memoria.
- Ahora mismo la librería de esp-idf no permite realizar un cambio de canal al mismo tiempo que se está conectado a un punto de acceso, por tanto, no podemos escuchar paquetes al mismo tiempo que estamos conectados a una red WiFi la cual es necesaria para poder ingestar los datos en tiempo real.
- A parte de la pérdida de datos, por no poder recopilar datos al mismo tiempo que se envían, los microcontroladores solo tienen una antena. El cambio de canal también puede hacer que perdamos algunos datos que se habían enviado en un canal que en este momento no estábamos escuchando.
- La implementación del código es más costosa respecto a lo que sería en una raspeberry.

Por otra parte, la arquitectura del broker no era centro de desarrollo del trabajo y aunque cumple con todos los requisitos y necesidades, a nivel escalable sería necesario desacoplar

los elementos y dejar al broker únicamente como un mero intermediario entre los microcontroladores y el backend donde se guardan y procesan los datos.

Una buena práctica y solución, sería utilizar Apache Kafka para tener una retención de mensajes y luego poderlos distribuir en otros servicios donde los podríamos tratar para ser procesados y guardados.

Emplear esta técnica como sistema de seguridad es una buena idea y se ha demostrado que es eficiente, funciona y te avisa cuando algún dispositivo entra en la zona.

No obstante, no creo que este sistema pueda sustituir los sistemas de seguridad actuales debido a que no todos los intrusos tienen porque llevar un dispositivo móvil, pero si puede servir como refuerzo, especialmente en recintos donde el tránsito de gente es bajo como naves industriales.

Respecto a su viabilidad como estudio de mercado, creo que tiene un gran potencial; a presentado poder extraer datos muy relevantes especialmente en la identificación de dispositivos; a esto hay que sumarle la cantidad de KPIs que se pueden extraer y se podría utilizar para saber el volumen de gente que hay en la zona.

Por otra parte, se me ocurren multitud de aplicaciones sobre esto donde pueden surgir nuevos caminos de investigación; especialmente en el tracking de posición utilizando el RSSI e incluso si se desplegaran suficientes en diferentes zonas o tiendas de un centro comercial se podría estudiar que recorridos suelen realizar las personas.

Desde otras perspectivas podríamos emplear este sistema para montar una casa inteligente en el que cuando detectara tu dispositivo cerca por ejemplo se levantaran las persianas o se encendieran las luces.

En definitiva, puede ser un añadido como capa de seguridad en recintos y puede ser perfectamente viable para multitud de aplicaciones dentro del mercado que habría que explorar y ampliar.

6 TRABAJO FUTURO

Para seguir con este proyecto se podría implementar mejoras de diferentes tipos:

- La ESP32 no controla si no se ha podido establecer conexión con el broker por lo que los mensajes de esa iteración son eliminados y se pierden. Esto tiene implicaciones a nivel de servicio porque lanzar una actualización en el broker implica una pérdida de datos. Se podría implementar un sistema de guardado en un sistema de ficheros o añadiendo un componente extra como tarjeta SD externa de manera que pudiéramos reenviar la información en otro momento.

- Los microcontroladores desplegados no pueden actualizar el código en remoto, aunque no es un imprescindible, a nivel de gestión sería mucho más elegante y eficaz a la hora de mantener una plataforma.

- La lógica de recogida, procesado y almacenamiento de datos se encuentra incluida dentro del mismo broker. Desacoplar esta lógica e implementarla mediante un sistema de gestión de mensajes como puede ser Apache Kafka beneficiaria a la escalabilidad y rendimiento de la plataforma.

- Explorar nuevas aplicaciones sobre estos datos que permitan extraer más información y montar una plataforma online donde se pueda mostrar gráficas en tiempo real de los datos extraídos con una API.

7 REFERENCIAS

[1] Amazon Go: https://www.xataka.com/analisis/visitamos-amazon-go-asi-es-la-experiencia-de-compra-en-una-tienda-semi-automatizada-sin-cajeros

[2] Google Maps: https://computerhoy.com/noticias/tecnologia/asi-sabe-google-maps-que-hay-atasco-tu-camino-293627

[3] Wardriving: https://es.wikipedia.org/wiki/Wardriving#:~:text=Se%20llama%20wardriving%20a%20la,de%20un%20esc%C3%A1ner%20para%20radio.

[4] Estandard IEEE 802.11: https://www.oreilly.com/library/view/80211-wireless-networks/0596100523/ch04.html

[5] Estructura de un probe request: https://mrncciew.com/2014/09/29/cwap-802-11-mgmt-frame-types/

[6] Espressif Systems ESP32: http://esp32.net/

[7] Que es MQTT: https://www.paessler.com/it-explained/mqtt

[8] MQTT en IoT: https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/

[9] MQTT QoS: https://aprendiendoarduino.wordpress.com/tag/mqtt-qos/

[10] Instalación de ESP-IDF en Windows 10 : https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/windows-setup.html

[11] Documentación de la API del framework esp-idf: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/index.html

[12] Sincronización del reloj interno de la ESP32: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/system_time.html

[13] esp-idf no soporta cambiar de canal cuando está conectado a un punto de acceso: https://github.com/espressif/esp-idf/issues/4311

[14] FreeRTOS Event Groups: https://icircuit.net/esp32-task-synchronization-freertos-event-groups/1957

[15] Marcador IRAM_ATTR: https://esp32.com/viewtopic.php?t=4978

[16] Limitaciones en captura de datos: https://www.esp32.com/viewtopic.php?t=4145

[17] Librería Vertx: https://vertx.io/docs/guide-for-java-devs/#_what_is_vert_x

[18] Vertx MQTT: https://vertx.io/docs/vertx-mqtt/java/

[19] SLF4J logger: http://www.slf4j.org/manual.html

[20] Apache Commons: https://commons.apache.org/

[21] Google GSON: https://github.com/google/gson

[22] HikariCP: https://github.com/brettwooldridge/HikariCP

[23] Estructura de una dirección MAC: https://networkencyclopedia.com/mac-address/#:~:text=MAC%20Address%20is%20a%20unique,on%20an%20Ethernet%2Dbased%20network.

[24] MAC address: https://en.wikipedia.org/wiki/MAC_address

[25] Libreria Python manuf by Michael Huang – coolbho3k: https://github.com/coolbho3k/manuf

[26] Documentación libreria manuf: https://pypi.org/project/manuf/

[27] Azure PostgreSQL configuration: https://docs.microsoft.com/es-es/azure/postgresql/quickstart-create-server-database-portal

[28] PgAdmin: https://www.pgadmin.org/

[29] Azure VM linux configuration: https://docs.microsoft.com/es-es/azure/virtual-machines/linux/quick-create-portal

[30] Wigle: https://wigle.net/

[31] Funcion SQL agregación de datos por tiempo: https://github.com/lycantropos/seek-well/blob/8529a9c70c2ddf95d8e84fa201b9ab1273ca7a2f/test_scripts/functions/date_trunc_by_interval.sql