Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

MÁSTER EN INGENIERÍA INFORMÁTICA

ÁREA DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

TRABAJO FIN DE MÁSTER

DIMENSIONAMIENTO DE REDES DE SENSORES PARA APLICACIONES IOT

D. Luis Magadán Cobo
TUTOR: D. Francisco José Suárez Alonso

FECHA: Enero 2021

Índice de contenidos

Índice de figuras

Índice de listados

Índice de tablas

1. Objetivos y alcance

El Internet de las Cosas (IoT) [1] actualmente es una de las tecnologías más importantes y con mayor proyección de futuro. Su principio básico se enfoca en la interconexión de objetos cotidianos (televisores, luces, persianas) [2] y no tan cotidianos (motores eléctricos, sistemas agrarios) [3] con Internet con el fin de permitir recoger información acerca del estado de estos dispositivos como puede ser su estado de funcionamiento o bien hacer que estos dispositivos realicen una operación que el usuario indique como por ejemplo el encender y apagar las luces en el caso de bombillas inteligentes.

Los ámbitos de aplicación de la tecnología IoT pueden dividirse en tres grandes grupos que a su vez pueden dividirse en múltiples subgrupos [4]. Estos grupos principales son el ámbito industrial o también conocido Industrial Internet of Things (IIoT), el ámbito sanitario y el ámbito de las Smart Cities. En la Ilustración 1 se muestra un gráfico en el que se recogen todos los ámbitos de aplicación del IoT.

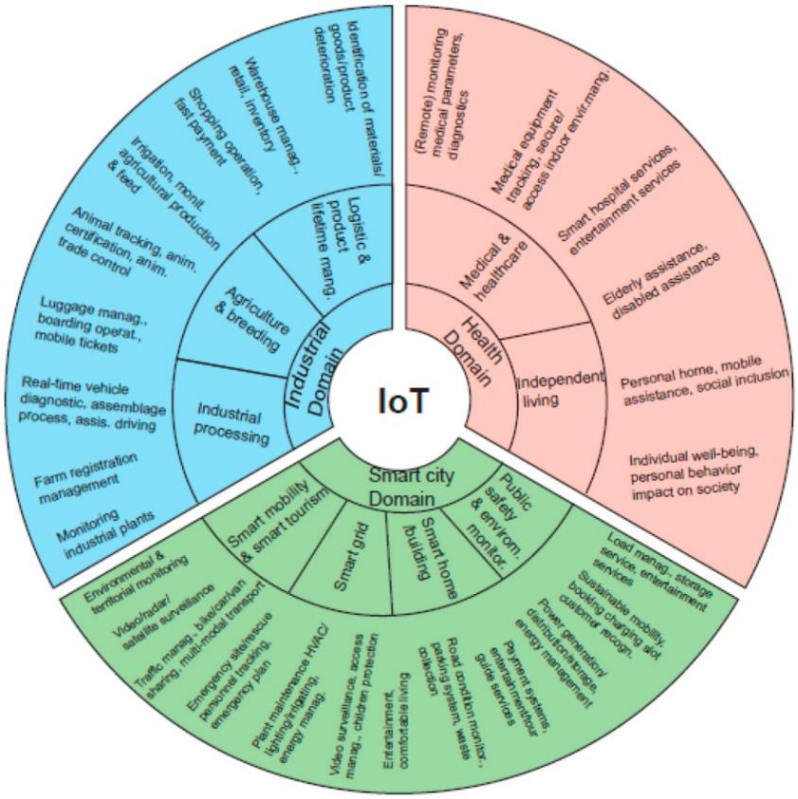

Ilustración 1: Ámbitos de aplicación del IoT

Su funcionamiento es similar en todos sus dominios de aplicación y consiste principalmente en la interconexión de dispositivos inteligentes que recopilan, envían e intercambian información y datos de su entorno. En toda arquitectura IoT participan tres grupos diferentes de dispositivos:

- Dispositivos IoT: dispositivos que permiten ser conectados mediante cable o inalámbricamente a una red más amplia.
- Gateway: puertas de enlace encargadas de interconectar los dispositivos IoT a la nube.
- Nube: servidores ya sean locales o remotos donde se almacenan y procesan los datos recogidos por los dispositivos IoT.

En cuanto a la computación en las aplicaciones IoT, como se puede observar en la Ilustración 2, existen tres tipos diferentes [5]. En primer lugar, el Cloud Computing, aquel en el que las operaciones de cómputo se llevan a cabo en la nube; en segundo lugar, el Edge Computing que se basa en realizar las transformaciones y operaciones necesarias sobre los datos capturados en el propio dispositivo IoT y finalmente, el Fog Computing, que lleva a cabo el cómputo en el Gateway antes de enviar la información a la nube.

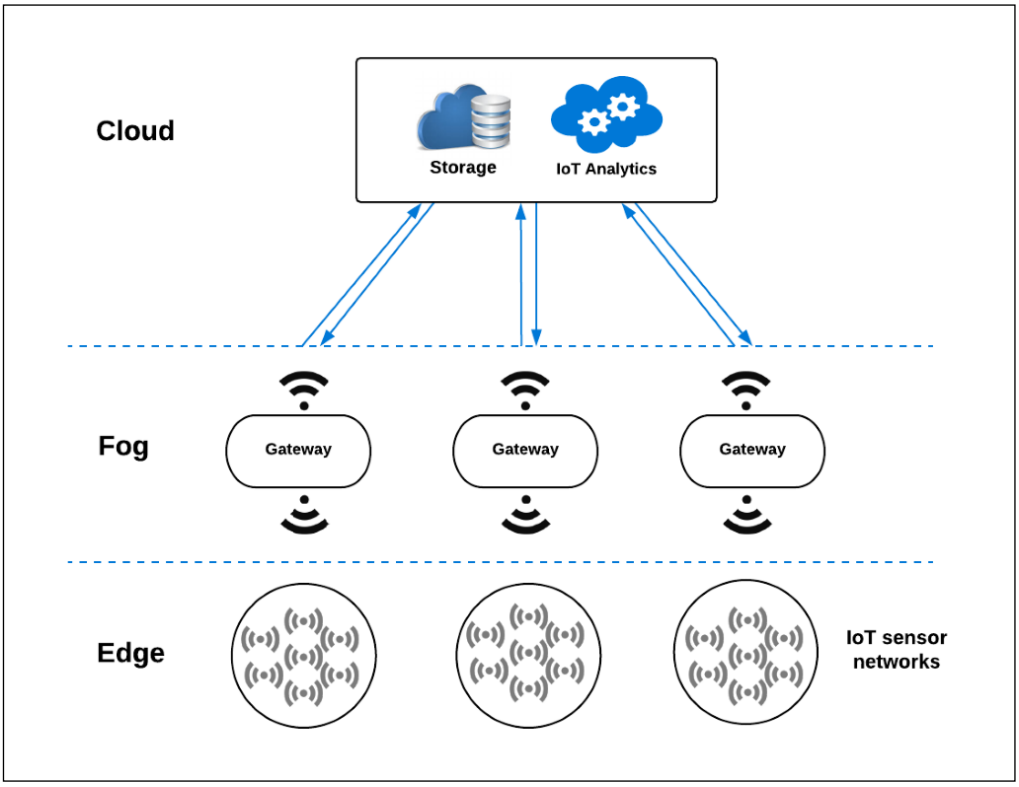

Ilustración 2: Tipos de computación en IoT

Como se puede apreciar, en toda aplicación IoT hay un elemento indispensable que es la conectividad a la red, ya sea inalámbricamente (p.e. mediante Wifi) o bien conectado directamente a través de un cable Ethernet. Sin embargo, hay ámbitos de aplicación donde por motivos de infraestructura no existen puntos de acceso cercanos a donde se encuentra posicionado el Gateway. Esto puede suceder en entornos de monitorización de edificios inteligentes [6] donde únicamente exista un punto de acceso en la planta baja del edificio o en plantas industriales donde se pretenda monitorizar motores eléctricos para llevar a cabo el mantenimiento predictivo de los mismos [7]. En este caso, puede suceder que en plantas de grandes dimensiones el punto de acceso más cercano se encuentre a más de 100 metros del elemento que se desea monitorizar, por lo que sería necesario llevar a cabo un cableado en planta, lo que supondría una modificación importante de las instalaciones y consecuentemente un coste adicional elevado.

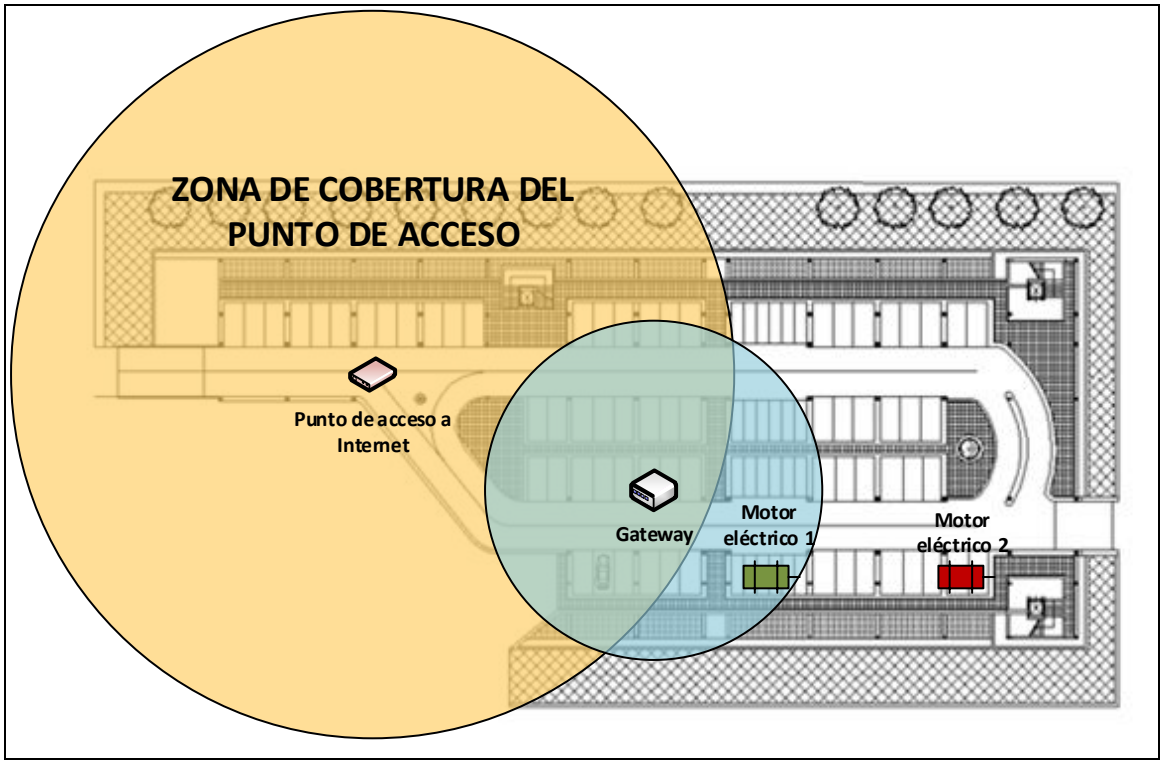

Ilustración 3: Problema de cobertura en planta industrial

En la Ilustración 3 puede apreciarse un claro ejemplo de este problema. En ella se puede observar el plano de una planta industrial en la que se encuentran dos motores eléctricos los cuales interesa monitorizar con el fin de prever fallos. Sin embargo, solo hay un punto de conexión ubicado en

el centro de la planta que da cobertura al Gateway solamente. Al emplearse tecnología inalámbrica de bajo consumo (p.e. Bluetooth Low Energy) entre el Gateway y los sensores, la distancia entre ellos debe ser bastante corta, por ello el Gateway da cobertura únicamente al motor eléctrico 1, haciendo que solo sea posible monitorizar dicho motor. Sin embargo, el motor eléctrico 2 al no disponer de conexión con el Gateway debido a la distancia respecto a este, no podría ser monitorizado y por tanto sería necesario modificar la infraestructura de telecomunicaciones de la planta para que dicho motor pudiera estar conectado a un Gateway con conexión a Internet.

Este problema se repite en múltiples ocasiones en lugares donde el despliegue de prototipos IoT requiere una modificación de la infraestructura de red existente. Por ello, se procederá a realizar un estudio en el que se analice una topología de red mallada de doble capa, cuya primera capa estará formada por un conjunto de Gateways interconectados entre sí mediante el uso de protocolos Ad-Hoc de comunicación inalámbrica de largo alcance, y la segunda capa estará compuesta por una red de sensores conectada mediante un protocolo de comunicación inalámbrico de corto alcance, con el fin de minimizar el consumo eléctrico de dichos dispositivos. Los objetivos propuestos para llevar a cabo este trabajo son los listados a continuación:

- Estudio de los modelos de redes existentes, los protocolos Ad-Hoc inalámbricos existentes para llevar a cabo la interconexión de los Gateways, así como la selección del más adecuado y un análisis del funcionamiento de este.

- Estudio de los protocolos de comunicación inalámbrica de corto alcance a emplear por los sensores, así como la selección del más adecuado y un análisis del funcionamiento de este.

- Implementación de un prototipo demostrador sobre el que se realizarán los análisis necesarios para dimensionar de forma adecuada la red en el ámbito de la monitorización de edificios con el fin de extenderlo en un futuro al ámbito IIoT. Se pretende desarrollar un prototipo de red de dos capas, la primera de ellas formada por Gateways y Repetidores interconectados entre sí mediante un protocolo Ad-Hoc inalámbrico y la segunda formada por un conjunto de sensores heterogéneos que se comuniquen con los Gateways/Repetidores mediante un protocolo de comunicaciones de bajo consumo.

2. Estado del arte

Como se ha comentado en el apartado anterior, la dificultad para desplegar aplicaciones IoT en algunas instalaciones se debe a los problemas que estas tienen para ofrecer una cobertura total de la instalación. Para el desarrollo del prototipo de este trabajo es necesario analizar los diferentes protocolos de red inalámbrica existentes, así como los protocolos y tecnologías de comunicación de bajo consumo.

2.1. Estudio de las redes inalámbricas existentes

Para solventar este problema existen varios tipos de red inalámbrica también conocidos como Wireless Networks, en los que los nodos se encuentran conectados inalámbricamente y tienen la facilidad de poder cambiarse de ubicación dentro de un área de cobertura sin perder la conectividad entre ellos.

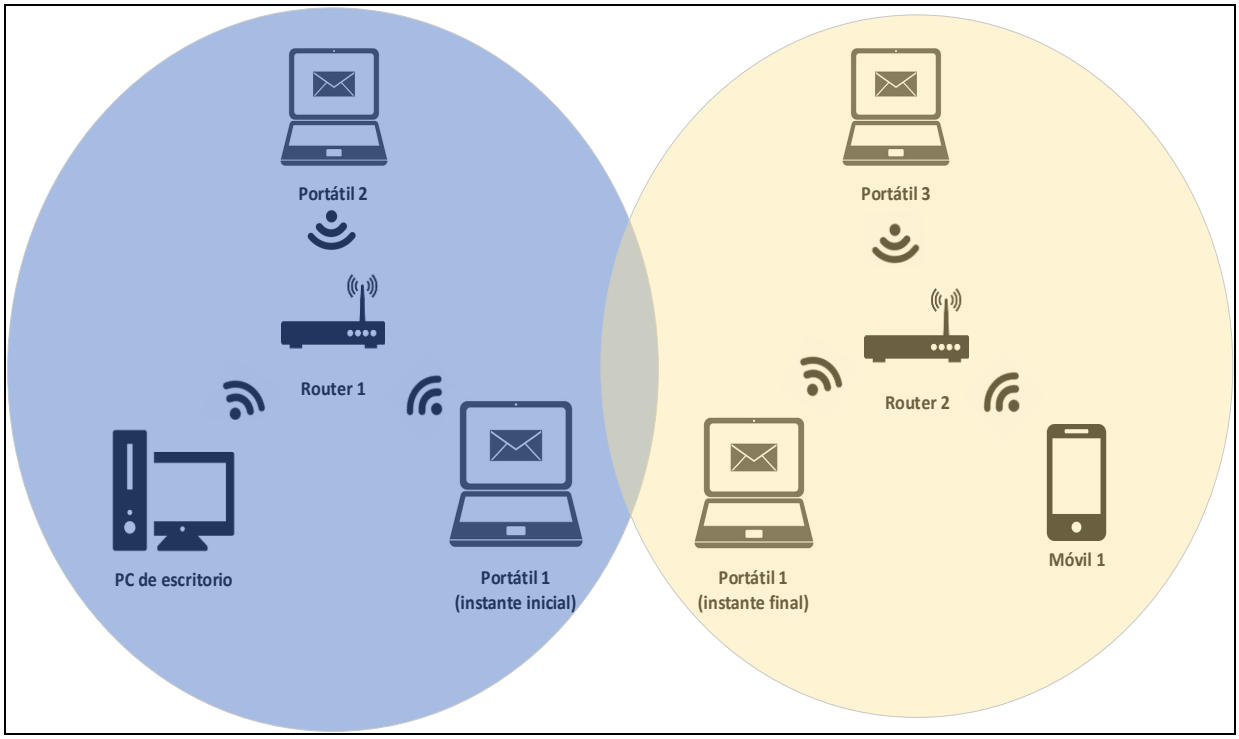

Ilustración 4: Topología Wireless Mesh Network

Actualmente, son dos los tipos de redes inalámbricas dominantes. En primer lugar, las redes inalámbricas malladas o Wireless Mesh Networks (WMNs) caracterizadas por mezclar la topología Ad-Hoc y la topología infraestructura. En ellas no todos los dispositivos actúan como routers de la red, basándose su funcionamiento en que cuando un dispositivo sale del alcance de un router, entra en el rango de otro router, sin perder de esta forma la conectividad [8].

En la Ilustración 4 se muestra una topología WMN en la que inicialmente el Portátil 1 se encuentra conectado inalámbricamente al Router 1 y posteriormente se desplaza y se conecta al Router 2 sin perder la conectividad con la red.

La otra topología de red inalámbrica dominante es la red Ad-Hoc móvil o Mobile ad hoc Networks (MANETs) [9]. En este caso todos los dispositivos actúan como routers, redirigiendo el tráfico a través de los nodos de la red, formando una red inalámbrica creada, administrada y organizada de forma totalmente autónoma, sin necesidad de llevar a cabo configuraciones adicionales cada vez que se produce un cambio en ella. Un ejemplo de topología de esta red puede observarse en la Ilustración 5.

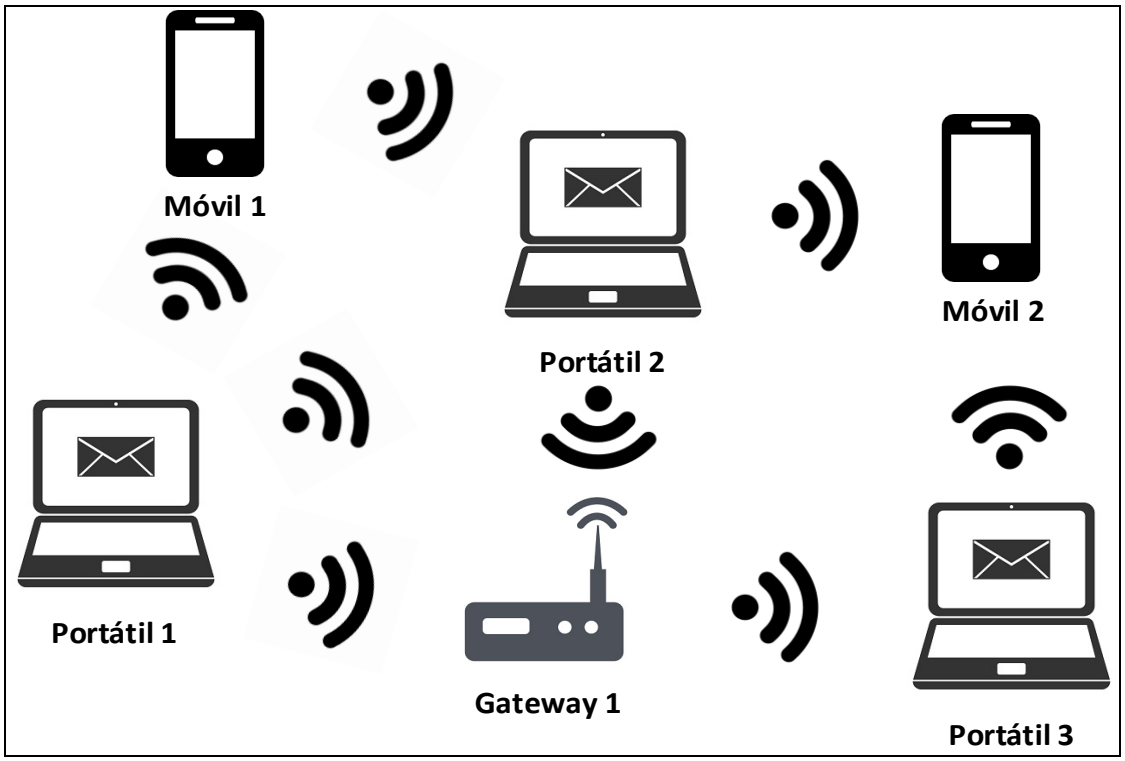

Ilustración 5: Topología de red MANETs

Como se comentó antes, todos los nodos presentan la posibilidad de trabajar como routers de la red actualizando su tabla de rutas con el camino óptimo para reenviar un paquete en función del protocolo de enrutamiento empleado. Además, al igual que en el caso de las WMNs, todos los dispositivos presentan la posibilidad de poder desplazarse gracias a la gestión autónoma que presentan dichos nodos, de forma que cuando un nodo pierde la conectividad con la red, automáticamente esta actualiza todos los caminos existentes a la hora de reenviar un paquete. Esto mismo sucede cuando aparece un nuevo nodo en la red. Estas características favorecen a generar una red fácilmente adaptable de bajo coste, si se escogen los dispositivos adecuados, escalable y tolerante a fallos gracias a la redundancia de nodos.

No todas las redes inalámbricas funcionan de manera similar, el enrutado de paquetes varía en función del protocolo de enrutamiento empleado en ellas. Dicho protocolo determina el algoritmo a emplear a la hora de redirigir un paquete de un nodo destino a un nodo fin en base a diferentes criterios, además de mantener el tamaño de la tabla de enrutamiento reducido, ya que una tabla de enrutamiento grande puede afectar al correcto funcionamiento de la red. En la actualidad los protocolos de enrutamiento pueden clasificarse en tres tipos diferentes [10] tal y como puede apreciarse en la Ilustración 6.

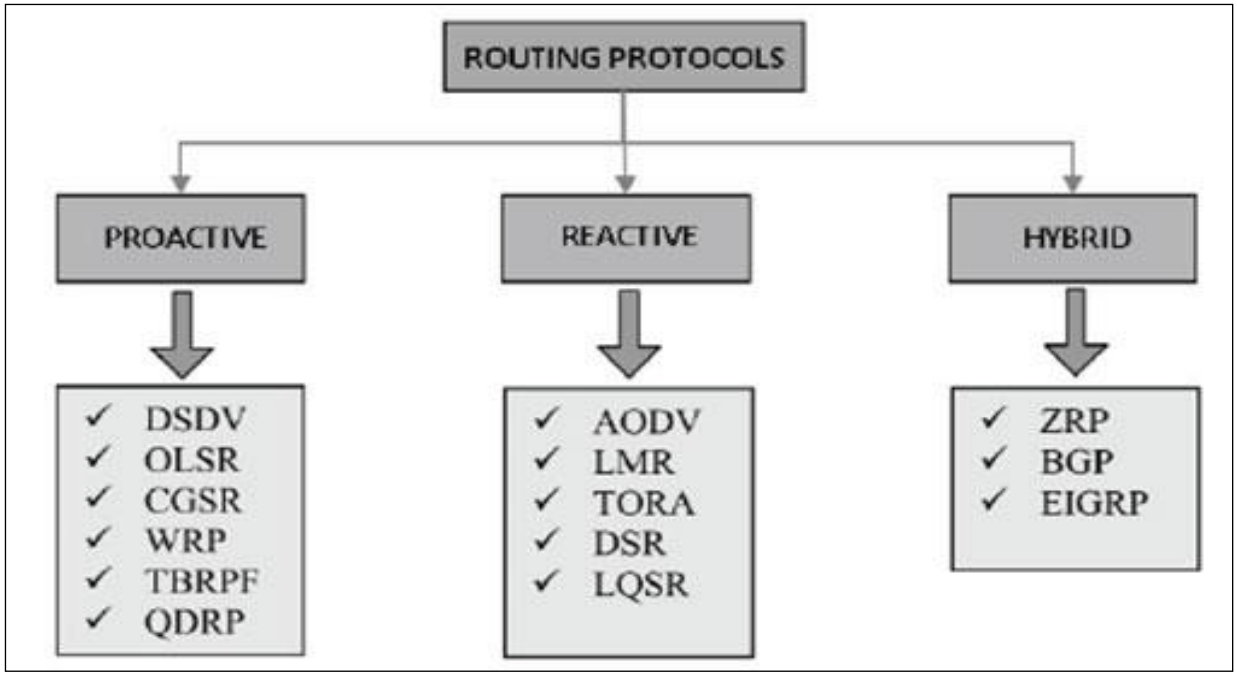

Ilustración 6: Tipos de protocolos de enrutamiento

En el caso de los protocolos proactivos, se caracterizan por disponer de una tabla de enrutamiento actualizada y completa antes de que esta sea necesaria, es decir, antes de que un nodo envíe un paquete a otro nodo de la red utilizando algoritmos de enrutamiento basados en el estado del enlace. Cada nodo debe poseer al menos una tabla de rutas con la información de sus nodos vecinos [11] y esta información debe propagarse entre todos los nodos de la red manteniéndose siempre actualizados [12], llevándose a cabo esta actualización cada cierto periodo de tiempo o siempre que se produzca algún cambio en la topología de la red. El principal inconveniente de este tipo de protocolo es la lenta reacción a cambios o fallos en la red y la elevada cantidad de información a mantener. Además, emplean de forma ininterrumpida una parte importante de la capacidad de la red cuando se producen cambios constantes en ella, lo que conlleva a que sean inadecuados para ser empleados en redes móviles ad-hoc reconfigurables periódicamente [13].

En cuanto a los protocolos reactivos, reducen los gastos generales de consumo de red ya que en lugar de disponer de tablas de enrutamiento en cada nodo antes de que estas sean necesarias para poder enviar un paquete de un nodo a otro de la red, como pasa en el caso de los protocolos proactivos, poseen tablas individuales generadas bajo demanda cuando los nodos que desean enviar paquetes no conocen la topología de red existente empleando algoritmos de enrutamientos vector distancia. De esta forma, solo cuando un nodo pretende enviar datos a otro nodo y solicita el descubrimiento de la ruta, el protocolo se encarga de buscar las rutas óptimas bajo demanda actualizando su tabla de enrutamiento y seleccionando la ruta óptima [14]. Este tipo de protocolo no es recomendable para aplicaciones IoT ya que es difícilmente escalable y presenta una latencia elevada.

El último tipo de protocolo de enrutamiento es el híbrido, como su propio nombre indica se trata de una combinación de protocolos reactivos y proactivos. Tiene como objetivo reducir la sobrecarga de los protocolos de enrutamiento proactivos y también la latencia presente en los protocolos reactivos [15]. Para ello se realiza una combinación de algoritmos de enrutamiento vector distancia y estado del enlace en función de la posición del nodo en la red. Está diseñado para ser empleado en redes de gran tamaño por lo que, en este contexto, al tratarse de redes no muy grandes con un escaso número de nodos, se ha decidido no analizarlos ya que se descarta su posible uso.

	Proactivo	**Reactivo**
Latencia	Baja	Alta
Escalable	Sí, hasta en torno a 150 nodos	Sí
Disponibilidad de la ruta	Siempre disponible	Bajo demanda
Requisito de ancho de banda	Alto	Bajo
Requisito de energía	Alto	Bajo

Tabla 1: Comparativa protocolos proactivos vs reactivos

Los protocolos de enrutamiento más usados en las redes MANET son *Destination-Sequenced Distance Vector Routing (DSDV)*, *Optimized Link State Routing (OLSR)*, *Dynamic Source Routing (DSR)* y *Ad Hoc On-Demand Distance Vector Routing (AODV)* entre otros [16].

DSDV es un protocolo proactivo, que mantiene su tabla de rutas desde el instante inicial actualizada y que emplea el algoritmo de Bellmand-Ford para calcular el salto óptimo [17]. Evita el problema de los bucles mediante la adición a cada entrada de la tabla de rutas del d*estination sequence number (DSC)*, el cual se emplea para distinguir entre las rutas nuevas y las rutas antiguas. Además, permite llevar a cabo actualizaciones de las rutas de dos formas diferentes, mediante un evento, como puede ser la aparición de un nuevo nodo en la red o mediante un temporizador, actualizándose todas las rutas cada cierto tiempo. Cuando un nodo recibe varias actualizaciones sobre una misma ruta por parte de diferentes vecinos, el nodo agregará aquellas que no tenga y en caso de que si las tenga comprobará el *DSC*. Aquella ruta que posea el *DSC* más nuevo será la utilizada. En caso de que el *DSC* sea el mismo, la ruta óptima será la seleccionada. Dicho protocolo es adecuado para la creación de redes ad-hoc con un número limitado de nodos, y resulta inadecuado para redes que presenten gran dinamismo, ya que cada vez que se produjese un cambio en la red requeriría una actualización de todas las tablas de rutas de los nodos.

El otro protocolo proactivo es *OLSR*, este protocolo funciona correctamente en el caso de redes con un gran número de nodos y con una topología que permita cambios en la red, como puede ser la aparición o eliminación de un nodo o el cambio de posición de uno de los ya existentes [18]. Su funcionamiento se basa en el intercambio constante de paquetes de red entre los diferentes nodos para conocer el estado de todos los nodos que conforman la red, lo que supone

un elevado consumo de ancho de banda. Para evitarlo, utiliza la técnica de los *Multi Point Relay (MPR)* [19], que son aquellos nodos encargados de la retransmisión de los paquetes broadcast y que son siempre inferiores al número total de nodos de la red, reduciéndose así el número de paquetes broadcast. Al tratarse de un protocolo proactivo, se conoce desde el momento inicial la tabla de rutas de toda la topología, además de la ruta óptima gracias al uso del algoritmo de Dijkstra, lo que produce una reducción en el tiempo de envío de los paquetes al no tener que calcularla. Sin embargo, el mantener dicha tabla actualizada constantemente ocasiona que se produzca una sobrecarga de la red.

En cuanto al protocolo *DSR*, se trata de un protocolo reactivo que funciona bajo demanda, es decir, actualiza sus rutas cuando es necesario enviar información a un nodo, causando una elevada latencia en las primeras comunicaciones llevadas a cabo entre los nodos de la red [20]. Además, detecta instantáneamente cambios en la arquitectura de la red y actualiza las rutas que se pueden ver afectadas por dichos cambios, lo que supone que sea óptimo para trabajar en entornos donde haya mucha movilidad de los dispositivos. Entre sus características destacan la capacidad de almacenar varias rutas para un mismo destino, facilitando de esta forma el balanceo de carga e incrementando la robustez de la red. Otra característica destacable es que los nodos a la hora de enviar los paquetes envían en la cabecera de estos los diferentes identificadores de cada uno de los nodos por los que ha de dirigirse dicho paquete, lo que provoca que este tipo de protocolo no sea óptimo para redes escalables, pues a partir de un número determinado de nodos en la red, las cabeceras de los paquetes crecen demasiado y producen fallos a la hora de redirigirlos [21]. En cuanto a los beneficios que se obtienen de emplear este protocolo destaca la reducción de la carga de red que se produce en el caso de los protocolos proactivos.

Finalmente, el protocolo *AODV* es un protocolo reactivo basado en los protocolos *DSR* y *DSDV*, el cual determina las rutas bajo demanda, es decir, únicamente cuando estas son necesarias para enviar un paquete entre nodos en el caso de no haberse determinado con anterioridad [22]. Esto causa una reducción del consumo de ancho de banda y de CPU, pero a su vez incrementa la latencia en las primeras comunicaciones entre nodos de la red mientras se establece la ruta. Cabe destacar, que no existe ningún nodo en toda la red que tenga una tabla de rutas al completo, sino que almacenan únicamente hacia quien debe enviar el primer salto para alcanzar al nodo destino y la distancia a la que se encuentra el nodo destino. Además, incluye en cada paquete una “hora

lógica" (identificador del nodo como bien puede ser la IP y un número de secuencia que se corresponde con el número de paquete enviado y que va autoincrementándose), permitiendo saber que paquete es el más reciente. Con el fin de reducir el almacenamiento de memoria requerido por esta clase de protocolos, *AODV* elimina la información almacenada cuando esta caduca [23].

Además de estos protocolos clásicos también existen otros muchos entre los que cabe destacar el protocolo *Better Approach To Mobile Adhoc Networking (B.A.T.M.A.N.)*. Se trata de un protocolo de enrutamiento proactivo diseñado especialmente para ser empleado en redes móviles ad-hoc multisalto. Su funcionamiento se basa en la división del conocimiento sobre los mejores caminos posibles de extremo a extremo entre los diferentes nodos que conforman la malla. Cada uno de esos nodos mantiene exclusivamente la información sobre el mejor salto posible hacia el resto de los nodos, evitando así la necesidad de control sobre los cambios producidos en la red [24]. Además, envía mensajes broadcast (*OGMs*) al resto de nodos vecinos para informarles de su existencia y estos nodos vecinos a su vez reenvían los *OGMs* recibidos inicialmente al resto de nodos de la red, basándose siempre en unas reglas específicas.

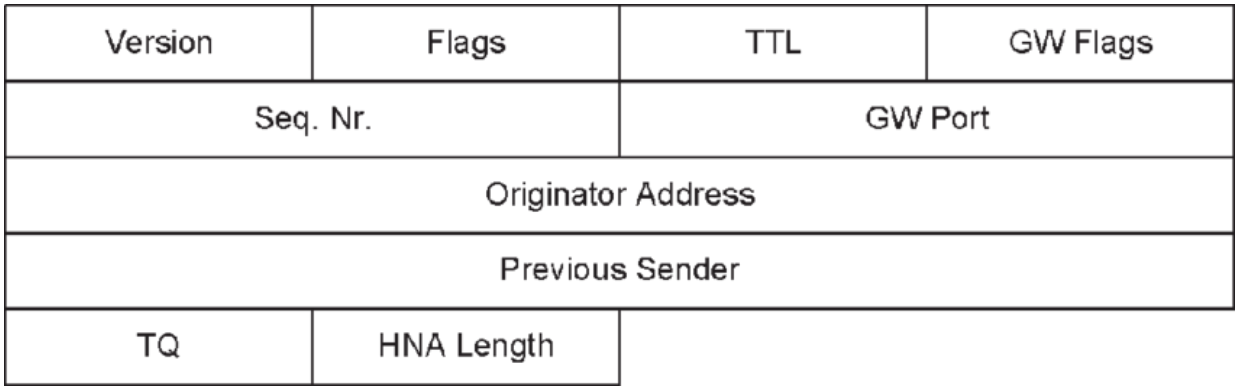

Ilustración 7: Formato típico de los mensajes OGM

En la Ilustración 7 se puede apreciar el formato típico de los mensajes *OGMs*, se trata de un paquete de 52 bytes en el que se incluye la dirección IP y la cabecera UDP. Dicho paquete contiene al menos la dirección origen, la dirección del nodo que transmite el paquete, un TTL y un número de secuencia. El número de secuencia se emplea para constatar que dicho paquete ha sido recibido una vez o más de una vez. Cabe destacar que cada nodo retransmite únicamente aquellos *OGMs* recibidos del vecino que ha sido identificado como el mejor próximo salto y solamente lo realiza una vez [25]. Este protocolo ha tenido múltiples versiones, de las cuales

únicamente se encuentran disponibles *B.A.T.M.A.N. III*, *B.A.T.M.A.N. IV* y *B.A.T.M.A.N. V*, aunque la versión *B.A.T.M.A.N. III* se encuentra en desuso.

En *B.A.T.M.A.N. III* cada nodo de una red emite un mensaje broadcast informando a sus vecinos cercanos acerca de su existencia, dichos vecinos reenvían esos mensajes al resto de la red y así sucesivamente de acuerdo con unas reglas de reenvío establecidas por el protocolo. Estos mensajes son reenviados como broadcasts UDP hasta que sucede una de las siguientes acciones:

- Cada nodo haya recibido al menos una vez el mensaje.
- Se pierdan por fallos de comunicación.
- Su tiempo de vida (TTL) haya expirado.

A la hora del despliegue, la pérdida de paquetes *OGMs* debido a colisiones, interferencias o congestión en el tráfico de red es algo muy habitual, por lo tanto, se emplea como medida de calidad de la ruta el número de *OGMs* recibidos por un nodo, ya sea de un salto (single-hop) o múltiples saltos (multi-hop). Por tanto, con el fin de encontrar la mejor ruta hacia un determinado nodo, se contabilizan el número de *OGMs* recibidos y se registra que vecino los retransmitió. El principal problema que presenta esta versión y que es causa de su desuso es el hacer frente a la gran cantidad de enlaces asimétricos, que provocan un incremento en el número de paquetes perdidos [26]. Para afrontar este problema se desarrolló la versión *B.A.T.M.A.N. IV*.

La versión *B.A.T.M.A.N. IV*, presenta una funcionalidad similar a la anterior salvo que, en este caso, para evitar el problema de los enlaces asimétricos se divide la calidad del enlace en dos partes: la calidad de recepción del enlace (*RQ*), que representa la probabilidad de éxito a la hora de recibir un paquete, y la calidad de transmisión del enlace (*TQ*), que representa la probabilidad de éxito a la hora de enviar un paquete a un nodo vecino [27]. Se conocen tanto *RQ* a partir del recuento de paquetes de sus vecinos, como la calidad eco del enlace (*EQ*), a partir del recuento de reenvíos de sus propios paquetes por parte de sus vecinos. Sin embargo, se desconoce por completo *TQ*, pero a partir de estos valores, se permite calcular *TQ* mediante la siguiente fórmula:

$$TQ = \frac{EQ}{RQ}$$

B.A.T.M.A.N. V es la última versión disponible del protocolo. Se desarrolló debido a que la pérdida de paquetes en las interfaces inalámbricas era bastante elevada, y se le daba más importancia a propagar cambios en la calidad de los enlaces de la red que a detectar cambios locales en los propios enlaces [28]. Por ello, esta versión emplea el protocolo *Echo Location (ELP)* [29] para localizar a los nodos vecinos y mensajes *OGM* para informar acerca de la calidad de los enlaces. En resumen, separa las tareas de descubrimiento de vecinos y de enrutamiento a través de la red, logrando de esta forma una reducción de la sobrecarga en la red y permitiendo gestionar el descubrimiento de nodos vecinos y la recolección de información sobre los enlaces, empleando tanto técnicas como intervalos diferentes para cada una de esas tareas.

2.2. Estudio de tecnologías y protocolos de comunicación de bajo consumo

En cuanto a las tecnologías de bajo consumo empleadas por los sensores para comunicarse con el Gateway, son muchas las alternativas existentes. Entre ellas, destacan las siguientes: *Bluetooth*, *Bluetooth Low Energy (BLE)*, *Bluetooth Low Energy Mesh (BLE Mesh)*, *LoRaWAN*, *NFC*, *Sigfox*, *6LoWPAN*, *Zigbee*, *WiFi HaLow* y *Z-Wave*.

Bluetooth es una tecnología inalámbrica de corto alcance, cuya primera versión fue lanzada en 1999. Opera en el canal de 2,4 GHz, canal dentro de la banda de frecuencia para el área industrial, científica y médica (ISM), por lo que se garantiza que cualquier dispositivo que lo emplee funcione de manera correcta independientemente del lugar donde se encuentre [30]. Permite una velocidad de transmisión de hasta 3 Mbps y un alcance máximo comprendido entre los 10 y los 100 metros, variando en función de los dispositivos empleados [31]. Además, cada una de las aplicaciones que emplee la tecnología Bluetooth tiene un perfil propio generalmente definido por el fabricante del dispositivo.

Con el transcurso de los años han sido muchas las versiones desarrolladas del protocolo Bluetooth, pero hay una que destaca por encima del resto, se trata de *Bluetooth Low Energy (BLE)*, una versión de bajo consumo de energía especificada en la versión 4.0 [32] y publicada en el año 2009. Como su nombre indica, el consumo energético es muy inferior al empleado por la tecnología Bluetooth. Sin embargo, la velocidad de transmisión de los datos está limitada entre 125 kbps y 2 Mbps [33]. Así mismo, la distancia máxima soportada por esta tecnología se

mantiene entre los 10 y los 100 metros, pudiendo ser incluso mayor en algunas ocasiones, dependiendo del tipo de dispositivos empleados [34].

Posteriormente, en 2017, se desarrolló una nueva tecnología basada en Bluetooth, se trata de *Bluetooth Mesh*, una tecnología que permite interconectar miles de dispositivos bluetooth (con un máximo teórico de 32767) formando una red inalámbrica mallada y un máximo de hasta 128 saltos [35]. De esta forma, cuando un nodo recibe un mensaje lo reenvía al resto de nodos de la red que se encuentran dentro de un rango excepto a aquel que se lo ha enviado. Todos los nodos de la red funcionan tanto de transmisores como de receptores. Actualmente es soportada por pocos dispositivos, pues se encuentra en desarrollo y presenta un gran problema de seguridad, ya que un fallo en la seguridad de un dispositivo no concierne únicamente al propio dispositivo como pasa en el caso de Bluetooth o BLE, sino que pone en riesgo a toda la red mallada [36].

Otra alternativa existente es el protocolo de comunicaciones inalámbrico *LoRaWAN*, que utiliza la tecnología *LoRa*, una tecnología inalámbrica de largo alcance que emplea un canal de comunicaciones dentro de la banda de frecuencia para el área industrial, científica y médica (ISM). Usa las bandas ISM 433 MHz, 868 MHz o 915 MHz en función de la zona en la que se encuentre [37]. Sus principales ventajas son que asegura comunicaciones a decenas de kilómetros de distancia y ofrece una duración de batería próxima a 10 años en condiciones óptimas. Sin embargo, este protocolo presenta una latencia elevada y una velocidad de transferencia de datos baja, en torno a 50 kbps, lo que supone que este protocolo no sea apto para su uso en aplicaciones que requieran una transferencia de datos elevada y continua [38].

Al igual que las tecnologías anteriores, *Near Field Communication (NFC)*, es una tecnología inalámbrica de corto alcance que opera en la frecuencia 13.56 MHz, y que permite una velocidad de transferencia de 424 kbps a una distancia máxima de en torno a 10 cm [39]. Por ello, esta tecnología no se considera útil para aplicaciones en el ámbito IIoT ni para la monitorización de edificios.

Sigfox es un protocolo de comunicaciones diseñado para operar en las mismas bandas de frecuencia que el protocolo *LoRaWAN* (433 MHz en Asia, 868 MHz en Europa y 915 MHz en Norte América) [40]. Su velocidad de transmisión se encuentra limitada a 100 bps y permite realizar comunicaciones a una distancia máxima comprendida entre los 10 y los 50 kilómetros

[41]. Se estima que la duración de una batería de 2400 mAh de un dispositivo que emplee este protocolo puede ser de 1.5 años enviando mensajes de 1 byte a una velocidad de 100 bps cada 10 minutos e incluso alcanzar los 14.6 años a medida que la velocidad de transmisión se ve reducida [42].

Otra opción existente es *6LoWPAN*, una capa de adaptación, que permite realizar comunicaciones IPv6 en redes inalámbricas formadas por módulos inalámbricos de baja potencia [43]. Soporta comunicaciones a distancias comprendidas entre 10 y 30 metros y ofrece una velocidad de transmisión de 20 kbps a 240 kbps [44].

Zigbee es un protocolo de red que utiliza los servicios de transporte de la especificación IEEE 802.15.4. El consumo de batería de los dispositivos que emplean este protocolo es muy bajo, pero, a pesar de ello, logra un alcance de 150 metros. Utiliza los canales de frecuencia de 868 MHz, 915 MHz y 2.4 GHz ofreciendo una velocidad de transmisión de 20 kbps, 40 kbps y 250 kbps respectivamente [45].

Otra tecnología de comunicación empleada es *WiFi HaLow*, tecnología de bajo consumo WiFi óptima para ser empleada en IoT. Permite soportar hasta 8191 dispositivos y llevar a cabo comunicaciones a una distancia de hasta 1 kilómetro en exteriores. La velocidad de transmisión empleando esta tecnología está comprendida entre los 150 kbps y los 86.7 Mbps [46].

El último de estos protocolos es Z-Wave, un protocolo de comunicaciones inalámbricas empleado por lo general en el ámbito de la domótica. Opera en las frecuencias 868.42 MHz (Europa) y 908.42 MHz (Estados Unidos). La velocidad de transmisión es de 9.6 kbps y 40 kbps respectivamente [47] logrando llevar a cabo comunicaciones a una distancia de hasta 150 metros.

Otras tecnologías también existentes son NB-IoT, 2G, 3G o LTE Cat 0/1. No se han decidido analizar ya que su coste es elevado y no sería aplicable en el contexto de este trabajo.

En la Tabla 2, se resumen todas las tecnologías y protocolos de bajo consumo antes mencionados. Como se puede apreciar en el caso de los protocolos *LoRaWAN, Sigfox, ZigBee, WiFi HaLow* y *Z-Wave*, a medida que aumenta el alcance para realizar comunicaciones, el coste de adquisición de los dispositivos aumenta también, ofreciendo también una velocidad de transmisión de kbps, salvo en el caso de *WiFi HaLow* donde la velocidad de transmisión puede alcanzar los 86.7 Mbps.

En cuanto a la tecnología *NFC*, al tener un alcance máximo de 10 cm, no sería aplicable en este trabajo. La tecnología *Bluetooth* ofrece un alcance óptimo para este trabajo, así como un coste de adquisición y una velocidad de transmisión adecuada, sin embargo, presenta un elevado consumo de batería que supondría un grave problema ya que reduciría drásticamente la vida útil de las pilas empleadas por los módulos multisensor. El uso de la capa de adaptación *6LoWPAN*, ofrece una velocidad de transmisión, rango de alcance, consumo y coste adecuados para este trabajo, pero presenta el problema de que no todos los módulos multisensor permiten su utilización. Finalmente, el protocolo *Bluetooth Low Energy*, presenta una velocidad de transmisión adecuada, un consumo de batería muy bajo y un coste de adquisición bajo. En cuanto al alcance logrado por este protocolo, teóricamente alcanza los 100 metros, aunque a más de 10 metros la comunicación comienza a ser ineficiente y la pérdida de paquetes es elevada.

	Velocidad de transmisión máxima	**Alcance máximo**	**Consumo de batería**	**Coste**
Bluetooth	1-3 Mbps	10 - 100 m	Medio	Bajo
BLE	125 kbps – 2 Mbps	10 - 100 m	Muy bajo	Bajo
LoRaWAN	50 kbps	10 - 20 km	Bajo	Medio
NFC	424 kbps	10 cm	Bajo	Bajo
Sigfox	100 bps	10 - 50 km	Bajo	Medio
6LoWPAN	20 kbps – 240 kbps	10 - 30 m	Bajo	Bajo
ZigBee	20 kbps – 250 kbps	150 m	Bajo	Medio
WiFi HaLow	150 kbps – 86.7 Mbps	1 km	Bajo	Medio
Z-Wave	9.6 kbps – 40 kbps	150 m	Bajo	Medio

Tabla 2: Comparativa tecnologías de bajo consumo

3. Tecnologías empleadas

Durante el desarrollo del trabajo, son múltiples las tecnologías empleadas para llevar a cabo el prototipo desarrollado. A continuación, se resumirán y explicarán cada una de ellas.

3.1. Raspberry Pi 3 Model B +

Se trata de un microordenador de un precio bastante reducido (~30€) que permite ser usado como Gateway o Repetidor en la topología desarrollada, encargándose de recoger los datos enviados por los módulos multisensor, del filtrado y preprocesamiento de estos y posteriormente de su enrutamiento a través de la red.

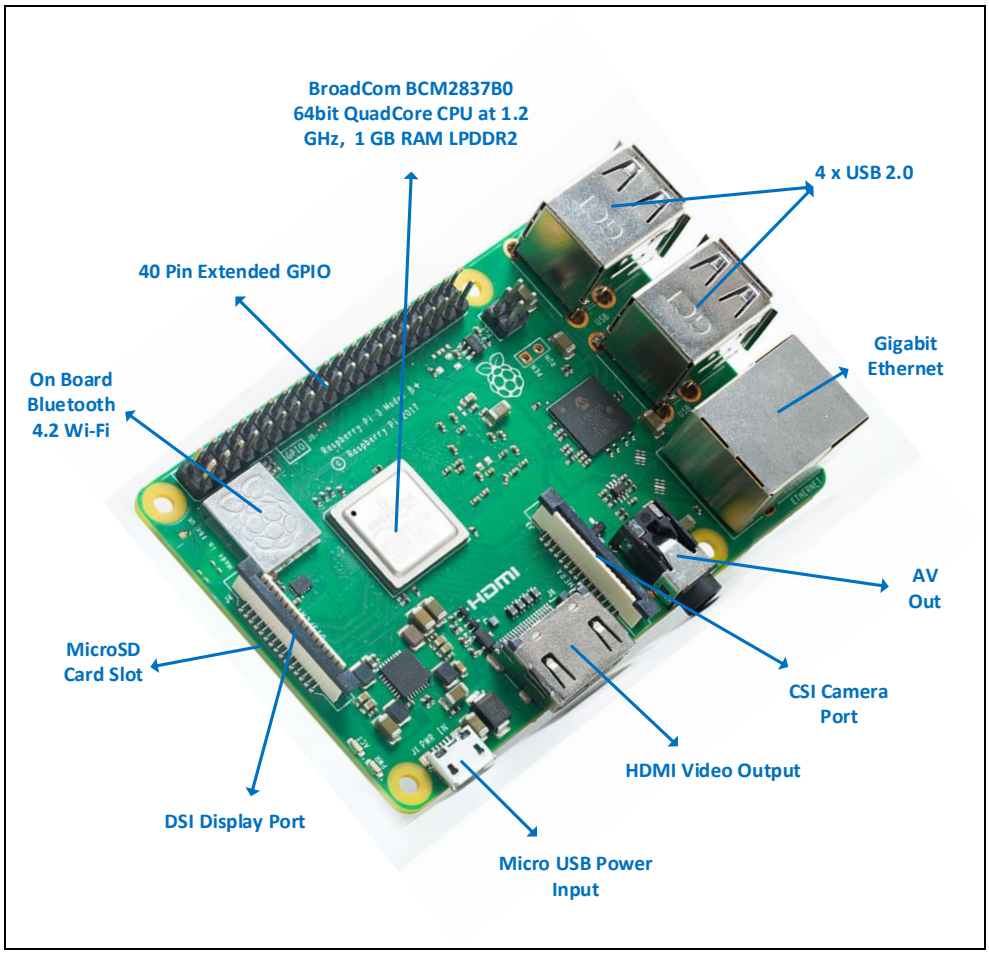

Ilustración 8: Raspberry Pi 3 Model B+

Tal y como se puede apreciar en la Ilustración 8, las prestaciones de este microordenador son las listadas a continuación:

- CPU + GPU: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @1.4GHz
- Dispone de WiFi y Bluetooth 4.2, admitiendo Bluetooth Low Energy.
- Memoria RAM de 1 GB LPDDR2
- 1 puerto Gigabit Ethernet sobre USB 2.0 que alcanza una velocidad máxima de 300 Mbps.
- 1 puerto HDMI, 4 puertos USB 2.0 y puerto CSI y DSI.

3.2. Raspberry Pi 4

La siguiente versión a la Raspberry PI 3 Model B+, que también se empleará como Gateway y Repetidor en la topología de red desarrollada, desempeñando las funciones de recolección, filtrado, computación, enrutamiento y envío de los datos a la nube. Este dispositivo es ligeramente más caro que su anterior versión (~45€).

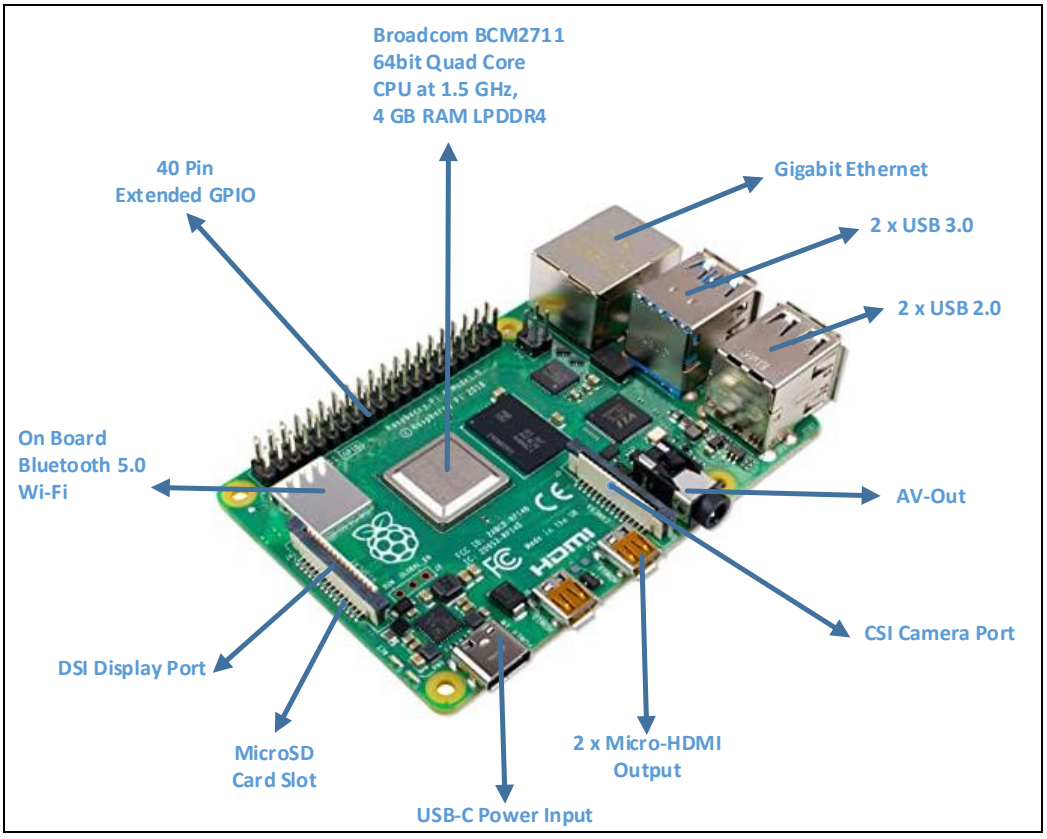

Ilustración 9: Raspberry Pi 4

A la vista de la Ilustración 9, los cambios más destacables son en la RAM, tanto a nivel de tamaño, ya que pasa a ser de 4 GB como a nivel de velocidad, ya que pasa a ser LPDDR4. También aparecen mejoras notables en la conexión Ethernet ya que esta no se encuentra limitada a ninguna velocidad. Además, en la Raspberry Pi 4 se emplea Bluetooth 5.0 mientras que en la Raspberry Pi 3 Model B+ se utiliza Bluetooth 4.2. En ambos modelos de Raspberry se ha empleado *Raspberry Pi OS (32-bit) with desktop and recommended software* como sistema operativo.

3.3. SensorTag CC2650

Se trata de un módulo multisensor desarrollado por Texas Instruments. Está formado por el microcontrolador CC2650, que posee un procesador ARM Cortex M3, así como 28 kB de memoria SRAM, de la cual 8 kB están destinados a cache y 128 kB de memoria Flash programable. Además de un total de cinco sensores de bajo consumo distribuidos tal y como se muestra en la Ilustración 10.

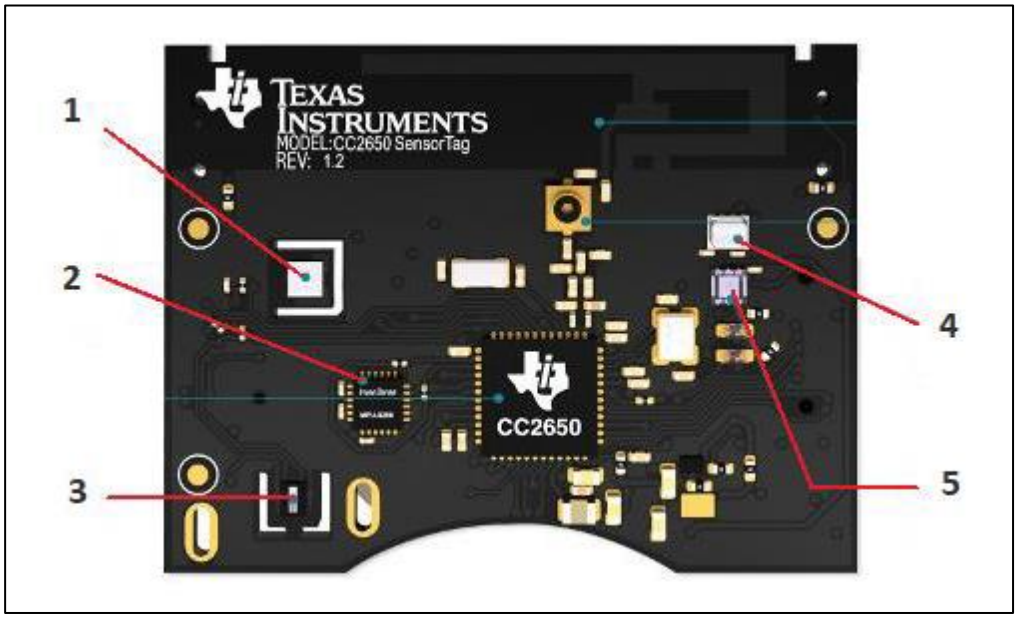

Ilustración 10: SensorTag CC2650

La función de los sensores mostrados en la Ilustración 10 es la siguiente:

1. Sensor de temperatura por infrarrojos (*TMP007*) que captura tanto la temperatura local del dispositivo como la temperatura ambiente. Sin embargo, este sensor ha dejado de

producirse en junio de 2017 aunque los dispositivos empleados en este trabajo aún lo contienen.

2. Sensor de movimiento (*MPU9250*) que dispone de acelerómetro, que permite capturar las aceleraciones en cada uno de los tres ejes del sensor (G), giroscopio, que mide la orientación en el espacio (deg/s) de cada eje y magnetómetro, que permite medir la fuerza del campo magnético en cada uno de los ejes (μT).
3. Higrómetro (*HDC1000*) encargado de capturar tanto la temperatura ambiente (ºC) como el porcentaje de humedad relativa del entorno.
4. Barómetro (*BMP280*) que mide la presión ejercida sobre el sensor (mBar) y la temperatura ambiente (ºC).
5. Luxómetro (*OPT3001*) que mide la cantidad de luz proyectada (lx).

A la hora de programar este microcontrolador es necesario emplear el SimpleLink SensorTag Debugger DevPack, mostrado en la Ilustración 11, un elemento que al incorporarlo al SensorTag CC2650, habilita sus opciones de depuración y desarrollo permitiendo depurar y grabar en la memoria Flash del dispositivo el firmware desarrollado.

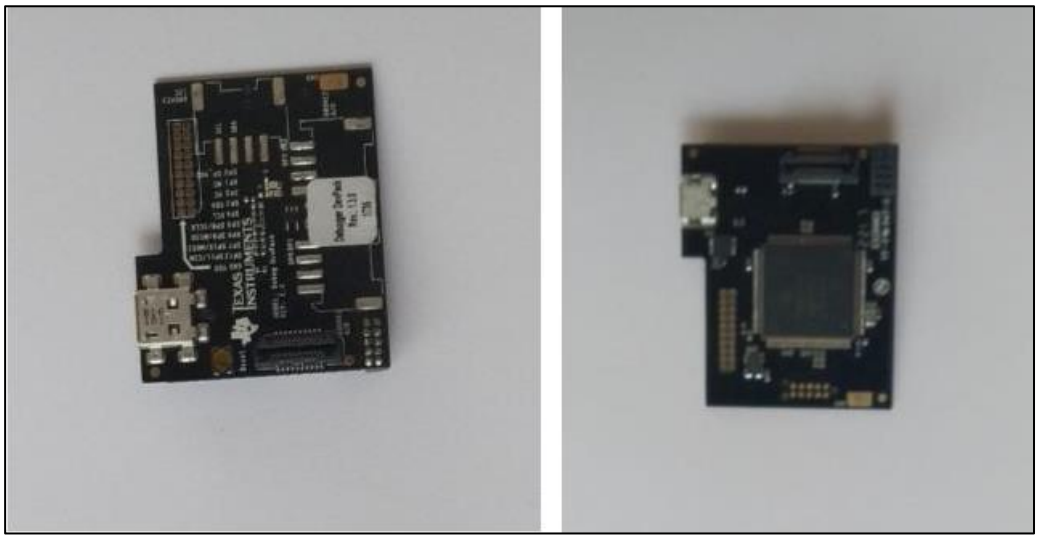

Ilustración 11: SimpleLink SensorTag Debugger DevPack

Para el desarrollo del firmware de este módulo multisensor se parte de un proyecto ya desarrollado por el fabricante y disponible a través de su página web [48]. Una vez descargado ese proyecto es necesario emplear el IDE de desarrollo Code Composer Studio (CCS) que facilita

el desarrollo de firmware para procesadores de Texas Instruments. Una vez se haya generado el firmware deseado, para grabarlo en la memoria Flash del SensorTag CC2650, hay que emplear el programa SmartRF Flash Programmer v2, siendo necesario en primer lugar la conexión del SensorTag CC2650 al DevPack y la de este último mediante un cable USB al ordenador. Un ejemplo más detallado del proceso de programación de la memoria Flash del SensorTag CC2650 puede verse en el **Anexo 2. Actualización del Firmware de los módulos multisensor.**

3.4. Smartbond DA14585 IoT

El Smartbond DA14585 IoT es un módulo multisensor, fabricado por la compañía Dialog Semiconductor, que está compuesto por el microcontrolador DA14585, que posee un procesador ARM Cortex M0 y 2 MB de memoria Flash externa donde se grabará el firmware desarrollado para que este se ejecute de forma correcta al encender el módulo multisensor, así como 64 kB de memoria OTP (One-Time-Programmable), 96 kB de memoria SRAM y 128 kB de memoria ROM.

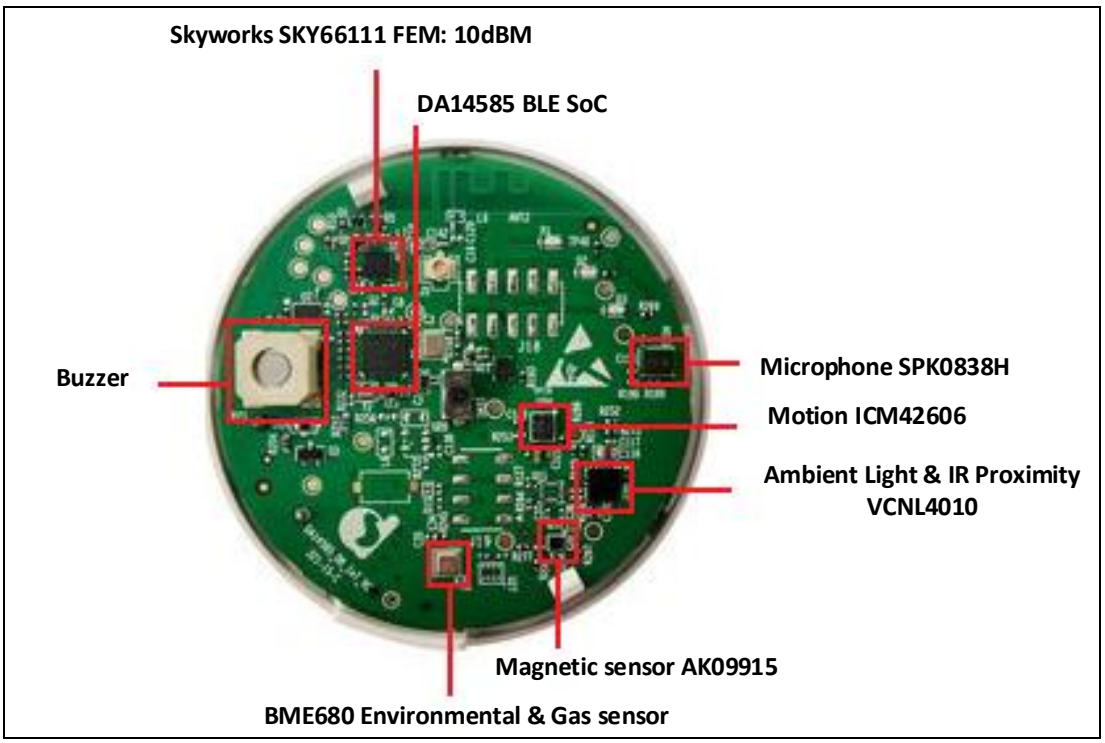

Ilustración 12: Smartbond DA14585

Como puede apreciarse en la Ilustración 12, el Smartbond DA14585 IoT dispone de cinco sensores. El sensor BME680 es un sensor medioambiental encargado de medir la temperatura ambiente, la humedad relativa, la presión y el gas. Otro sensor que puede resultar interesante para este trabajo es el VCNL4010, que permite medir la luz ambiental y detectar proximidad. Además de estos dos sensores, también dispone de un sensor de movimiento, el ICM42606, que actúa como acelerómetro y giroscopio, un magnetómetro AK09915, y un micrófono que permite monitorizar el ruido del entorno.

Para llevar a cabo una actualización del firmware del dispositivo es necesario emplear la placa D-SCPINTERFACEBRD mostrada en la Ilustración 13, la cual una vez conectada al módulo multisensor habilita sus funciones de desarrollo y depuración.

Ilustración 13: D-SCPINTERFACEBRD

Durante este trabajo se ha desarrollado un firmware para este módulo multisensor partiendo de un proyecto proporcionado por Dialog Semiconductor [49]. Para realizar las modificaciones necesarias se ha empleado el entorno de desarrollo SmartSnippets Studio V2.0.10. Tras llevar a cabo las modificaciones pertinentes, es necesario compilar el proyecto para generar el nuevo firmware del sensor, que posteriormente será grabado en la memoria Flash del módulo multisensor. Para ello, se emplea el programa SmartSnippets Toolbox V5.0.10, que graba en la memoria Flash el firmware que se desee una vez se haya conectado el SmartBond DA14585 IoT a la placa D-SCPINTERFACEBRD y esta a su vez mediante un cable USB al ordenador desde

el que se vaya a grabar el firmware. Un ejemplo más detallado del proceso a seguir a la hora de grabar nuevo firmware en el módulo multisensor se muestra en el **Anexo 2. Actualización del Firmware de los módulos multisensor.**

3.5. Sensortile.box (STEVAL-MKSBOX1v1)

Otro módulo multisensor empleado en este trabajo es el Sensortile.box o STEVAL-MKSBOX1v1. Está equipado con un microcontrolador ARM Cortex-M4 STM32L4R9 que dispone de 2 MB de memoria Flash programable, así como del módulo bluetooth 4.2 SPBTLE-1S de bajo consumo.

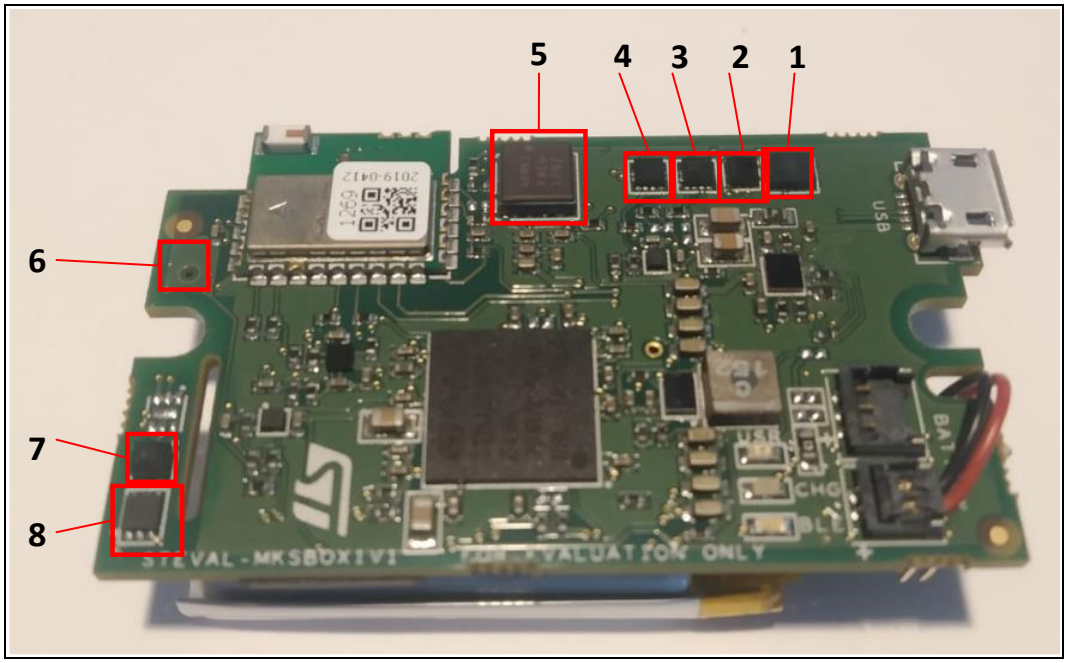

Ilustración 14: Sensortile.box

Además, integra ocho sensores de bajo consumo que son los mostrados en la Ilustración 14. Estos sensores son los siguientes:

1. El sensor de movimiento LM6DSOX, que actúa tanto de acelerómetro como giroscopio permitiendo medir las aceleraciones y la orientación en el espacio de cada uno de sus ejes.
2. El sensor de presión LPS22HH que permite medir la presión a la que es sometida el sensor.

3. El magnetómetro LIS2MDL, encargado de captar la fuerza del campo magnético de cada uno de los ejes del sensor.
4. El acelerómetro LIS2DW12, que captura las aceleraciones en cada uno de los tres ejes del sensor.
5. El acelerómetro LIS3DHH, que al igual que el acelerómetro LIS2DW12, mide las aceleraciones en cada uno de los ejes del sensor.
6. Un micrófono de bajo consumo, el MP23ABS1, encargado de capturar el sonido del entorno.
7. El sensor de temperatura STTS751, el cual capta la temperatura del entorno.
8. El higrómetro HTS221, encargado de captar el porcentaje relativo de humedad y la temperatura del entorno.

Para llevar a cabo la modificación del firmware de este módulo multisensor, se parte de un proyecto de partida proporcionado por ST Microelectronics [50]. Sobre este proyecto se realizarán las modificaciones necesarias, entre ellas, la selección de los sensores a emplear, la frecuencia de captura de datos y de envío de estos al Gateway, etc. Estas modificaciones se realizarán empleando el entorno de desarrollo STM32CubeIDE. Una vez generado el firmware, es necesario cargarlo en la memoria Flash del módulo multisensor.

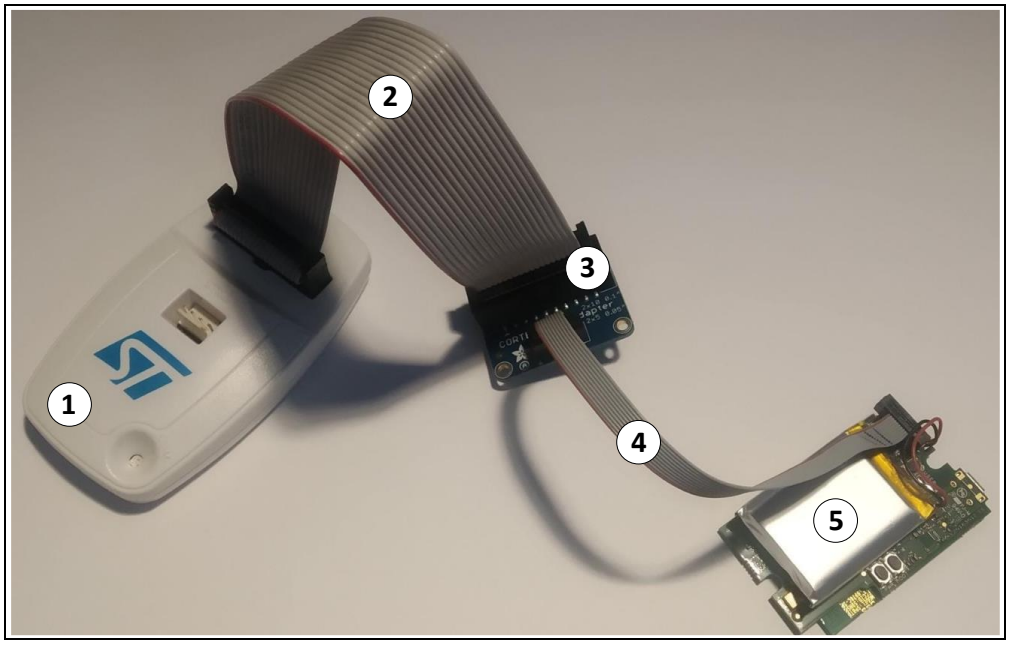

Ilustración 15: Montaje ST LINK V2

Como se muestra en la Ilustración 15, el ST LINK V2 (1) es un depurador y programador que conectado mediante un cable de cinta con conector de 20 pines (2) al adaptador JTAG 20-10 pines (3) y este a su vez conectado mediante un cable de cinta 14-10 pines (4) al Sensortile.box (5) permite habilitar las funcionalidades de programación y depuración del módulo multisensor. Para programar la memoria Flash del módulo multisensor, es necesario realizar el montaje mostrado en la Ilustración 15 conectando el ST LINK V2 mediante un cable USB al ordenador, y emplear el programa STM32CubeProgrammer, desde el que se seleccionará la versión del firmware deseada. Un ejemplo más detallado del proceso seguido para actualizar el firmware del Sensortile.box se incluye en el **Anexo 2. Actualización del Firmware de los módulos multisensor.**

3.6. BlueTile (STEVAL-BCN002V1)

El BlueTile o STEVAL-BCN002V1 es un módulo multisensor desarrollado por ST Microelectronics basado en el SoC BlueNRG-2, el cual funciona con un núcleo ARM Cortex M0 y posee una memoria Flash Programable de 256 kB.

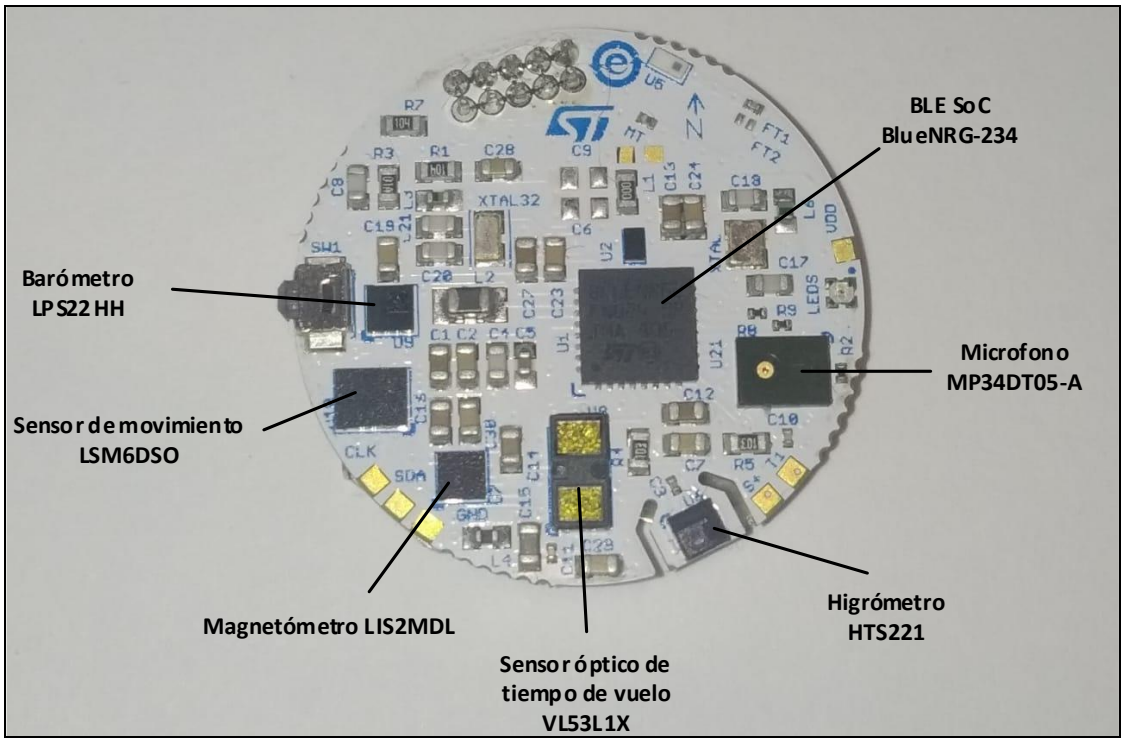

Ilustración 16: BlueTile

En la Ilustración 16, se puede observar los seis sensores de los que dispone el BlueTile. Emplea varios sensores utilizados también en el Sensortile.box, como el sensor de movimiento LSM6DSO, el barómetro LPS22HH, el magnetómetro LIS2MDL y el higrómetro HTS221. Además, incluye un micrófono MP34DT05-A y el sensor VL53L1X, un sensor óptico de tiempo de vuelo de largo alcance, que permite detectar objetos a larga distancia.

Para llevar a cabo el desarrollo del firmware empleado durante este trabajo, se ha partido de un proyecto de partida suministrado por el propio fabricante [51]. Sobre este proyecto se han llevado a cabo múltiples modificaciones para lograr seleccionar aquellos sensores que se deseaba emplear, la frecuencia de captura y envío de los datos, así como el formato de envío de estos. Esto se ha realizado empleando el entorno de desarrollo Atollic TrueSTUDIO. Tras generar el firmware es necesario grabarlo en el módulo multisensor, y para ello es necesario habilitar las opciones de depuración y programación del BlueTile mediante el uso de la placa de programación STEVAL-BCN002V1D mostrada en la Ilustración 17.

Ilustración 17: STEVAL-BCN002V1D parte delantera (1) y trasera (2)

Una vez conectado el BlueTile a la placa de programación y esta a su vez al ordenador, se emplea el programa BlueNRG-X Flasher Utility que permite seleccionar el firmware deseado y grabarlo en el módulo multisensor. En el **Anexo 2. Actualización del Firmware de los módulos multisensor** se incluye un ejemplo detallado de cómo llevar a cabo la modificación del firmware del módulo multisensor BlueTile.

3.7. Thingsboard

Se trata de una plataforma IoT open Source y nube privada que ofrece al usuario la capacidad de recolectar y almacenar datos, procesarlos, visualizarlos y gestionar los diferentes dispositivos conectados a ella. En resumen, permite gestionar de forma sencilla las redes IoT, así como los elementos que las forman y los datos recogidos por cada uno de ellos. Dentro de Thingsboard existen tres tipos diferentes de roles, cada uno con unos permisos determinados:

- **Administrador del sistema**: se encarga de llevar a cabo la configuración del sistema tanto a nivel general como de seguridad. Permitiendo crear "Organizaciones", las cuales representan a un colectivo (individuo, empresa, etc) que son los propietarios de un conjunto de dispositivos.
- **Administrador de la organización**: puede crear y gestionar dispositivos dentro de la Organización creada, así como activos, clientes, etc. También permite realizar configuraciones utilizando Rule Engine, que se trata de un motor de reglas que permite definir acciones a realizar cuando ocurren determinados eventos, como puede ser, por ejemplo, que cuando el valor de la batería de uno de los dispositivos sea menor del 20% enviar una notificación al cliente indicándolo, o cuando se pierde la conectividad con un dispositivo realizar exactamente lo mismo.
- **Cliente:** usuario el cual puede acceder exclusivamente a los dispositivos o gráficos a los que el administrador de la organización le haya concedido permiso.

Para albergar Thingsboard se emplea una máquina virtual ejecutada en un servidor de la Universidad de Oviedo.

4. Metodología y plan de trabajo

Durante la investigación, se han llevado a cabo varias fases diferentes con el fin de cumplir todos los objetivos previamente detallados, así como asegurar que una vez finalizada cada una de las fases realizadas se hayan obtenido resultados coherentes y correctos.

A continuación, se listan las fases por las que se ha ido pasando, así como una breve descripción de estas.

1. Definición de la topología de red a desarrollar.
2. Protocolos de comunicación empleados, donde se realiza un análisis y selección de los protocolos de comunicación de bajo consumo para llevar a cabo las comunicaciones entre módulo multisensor y Gateway/Repetidor, así como de los protocolos de comunicación ad-hoc inalámbricos a emplear por los Gateway/Repetidores.
3. Configuración de los Gateways y los Repetidores, realizando toda la configuración necesaria para dar soporte al protocolo de comunicaciones ad-hoc inalámbrico empleado, así como la realización de múltiples pruebas que demuestren su correcto funcionamiento.
4. Configuración de los módulos multisensor, desarrollando el firmware específico para cada uno de los cuatro modelos de módulos multisensor a emplear en el prototipo de demostración, así como una prueba de su correcto funcionamiento.
5. Configuración de la nube, realizando la instalación de la nube a emplear en una máquina virtual de la Universidad de Oviedo y adaptando los scripts de captura desarrollados para que envíen la información recolectada a la nube empleando un bróker de mensajes MQTT.
6. Análisis de la comunicación entre Gateways y Repetidores, realizando varios estudios en los que se analiza la variación del RSSI en función de la distancia entre dispositivos,

así como un análisis del rendimiento, la latencia y el jitter según varía el número de saltos realizados en la red.

7. Análisis de la distancia máxima entre Gateway/Repetidor y módulos multisensor, realizando un estudio de la variación del RSSI en función de la distancia entre Gateway/Repetidor y módulo multisensor.

Estas fases se explican más detalladamente en el apartado **Desarrollo del trabajo realizado**.

Una vez finalizadas todas esas fases, se detalla el **Despliegue del prototipo demostrador**, donde se comprueba el correcto funcionamiento del trabajo desarrollado en el edificio del Departamento de Informática en el Campus de Gijón. Sobre esta instalación se repite el análisis previamente desarrollado para comprobar los niveles RSSI entre dispositivos, así como un análisis del throughput, latencia y jitter entre los Repetidores y el Gateway.

En el **Anexo 1. Planificación temporal** se incluye una planificación en la que se muestran cada una de las fases previamente mencionadas junto al tiempo de duración y el diagrama Gantt correspondiente.

5. Desarrollo del trabajo realizado

En esta sección, se describirá con más detalle el trabajo realizado en cada una de las fases comentadas en el apartado **Metodología y plan de trabajo**, justificando en todo momento el porqué de su realización.

5.1. Definición de la topología de red a desarrollar

Existen muchos tipos de topologías de red como se puede apreciar en la Ilustración 18, las más conocidas son:

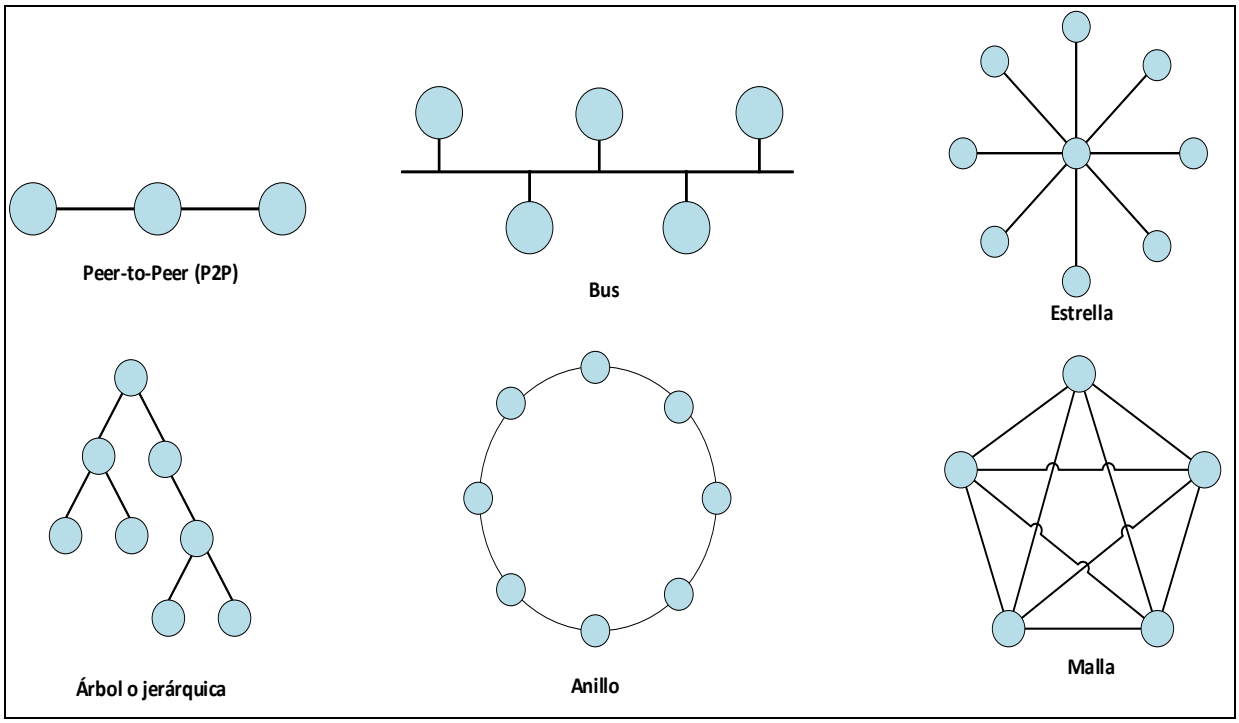

Ilustración 18: Topologías de red

- *Peer-to-Peer (P2P),* se caracteriza por ser la más simple, está compuesta por un enlace permanente entre dos dispositivos finales.
- *Topología en bus*, todos los nodos de la red comparten un mismo bus por el que viaja la información de un nodo a otro.
- *Topología en estrella,* en ella todos los nodos se encuentran conectados a un nodo central, generalmente un router, y todas las comunicaciones pasan a través de ese nodo central.

- *Topología en árbol*, también conocida como topología jerárquica, en ella los nodos se encuentran dispuestos en forma de árbol. Se asimila a un conjunto de topologías en estrella salvo que en este caso no existe un nodo central.
- *Topología en anillo*, recibe este nombre ya que cada nodo tiene una única conexión de entrada y una única de salida, formando de esta forma un anillo entre los nodos que la forman.
- *Topología en malla*, caracterizada porque cada nodo está conectado al resto de nodos de la red permitiendo de esta forma llevar a cabo las comunicaciones por diferentes caminos.
- *Topología híbrida*, está compuesta por una combinación de otras topologías existentes, como bien puede ser una topología en estrella y una topología en anillo.

Para este trabajo se ha decidido desarrollar una red siguiendo una topología híbrida formada por una primera capa de Gateways heterogéneos que sigan una topología mallada mediante el uso de algún protocolo Ad-Hoc inalámbrico, de forma que garantice la conectividad de todos los dispositivos a la red. La segunda capa, estará compuesta por los módulos multisensor empleados para monitorizar el entorno y seguirán una topología de red en estrella, siendo el nodo central un Gateway o Repetidor encargado de gestionar un conjunto de módulos multisensor mediante comunicaciones realizadas a través de un protocolo de comunicación de corto alcance a determinar.

Como se puede apreciar en la Ilustración 19, son tres los diferentes tipos de dispositivos empleados. Estos dispositivos son:

- Repetidores: son los dispositivos que no tienen conexión directa con la red, encargados de reenviar los mensajes a través de la red mallada hasta que estos lleguen al Gateway, con el fin de comunicarse también con la plataforma IoT o nube. Estos dispositivos están a cargo de un conjunto de Endpoints, y realizan las transformaciones y operaciones necesarias sobre los datos recibidos. La comunicación entre Gateways y Repetidores se realiza empleando un protocolo de enrutamiento inalámbrico Ad-Hoc. En este trabajo, los dispositivos que llevarán a cabo la función de Repetidores son la Raspberry Pi 3 Model B+ y la Raspberry Pi 4.

- Gateways: son aquellos dispositivos conectados directamente a la red, encargados de realizar el envío de los datos recogidos por la red de sensores a la plataforma IoT o nube donde se deseen almacenar. Además, pueden hacerse cargo de forma análoga a los Repetidores de un grupo de Endpoints, recolectando los datos emitidos por los sensores y aplicando sobre ellos las transformaciones y operaciones necesarias. Al igual que en el caso de los Gateways, los dispositivos encargados de llevar a cabo esta funcionalidad serán la Raspberry Pi 3 Model B+ y la Raspberry Pi 4.

- Los Endpoints son una red de módulos multisensor heterogéneos, encargados de capturar los datos del entorno tales como humedad, temperatura, luminosidad y otros datos de carácter relevante. Estos dispositivos están conectados siguiendo una topología de estrella, mediante un protocolo o tecnología de corto alcance, con los Repetidores o Gateways de la red mallada con el fin de que esos datos sean almacenados de forma correcta en la plataforma IoT o nube. Estarán compuestos por los módulos multisensor SensorTag CC2650, Smartbond DA14585 IoT, Sensortile.box y BlueTile.

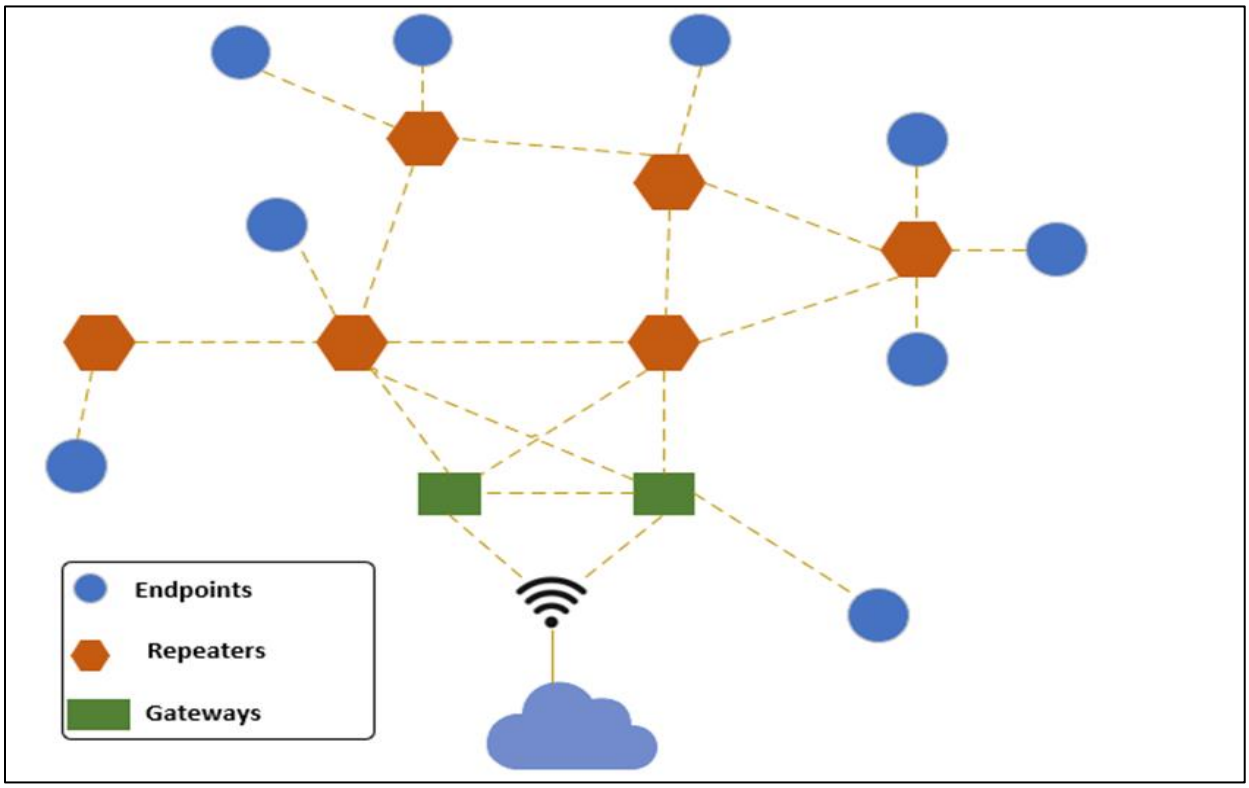

Ilustración 19: Topología seleccionada

5.2. Protocolos de comunicación empleados

Una vez definida la topología de red a seguir durante este trabajo y seleccionadas las tecnologías empleadas para llevarlo a cabo, faltaría determinar que protocolos de comunicación serán los empleados para realizar las comunicaciones entre los módulos multisensor y los Repetidores o Gateways de la red mallada, así como en las comunicaciones entre los Repetidores y Gateways y finalmente entre Gateway y nube.

Debido al bajo coste de los módulos multisensor, el número de protocolos soportado para llevar a cabo la comunicación con el Repetidor o Gateway es muy reducido. El SensorTag CC2650 soporta ZigBee y BLE, aunque para el uso del protocolo ZigBee en este dispositivo es necesaria la utilización del CC2531 como coordinador. El Smartbond DA14585 IoT únicamente soporta el protocolo de comunicación BLE. Sin embargo, en el caso de los módulos multisensor Sensortile.box y BlueTile soportan BLE Mesh además de BLE. Inicialmente se pretendía emplear BLE Mesh en este tipo de dispositivos, pero tras varias semanas de investigación, se determinó que el protocolo BLE Mesh de estos dispositivos funciona exclusivamente de forma correcta con la aplicación móvil ST BLE Mesh. Esto se debe a que se trata de una adaptación de la especificación del protocolo de comunicaciones inalámbrica BLE Mesh desarrollada por el fabricante ST Microelectronics y, que consecuentemente, no es capaz de emplear las Raspberry como proveedores de la red mallada de forma correcta, ya que no resuelve correctamente los servicios ofertados por estos dispositivos al emplear BLE Mesh.

Para confirmar que la versión del protocolo BLE Mesh empleada en estos módulos multisensor no es compatible con la versión BLE Mesh soportada por la Raspberry, se instaló la herramienta *meshctl*, herramienta recomendada por Bluetooth para que una Raspberry ejerza de proveedor de la red mallada. Para ello se siguieron los pasos de las guías de configuración proporcionadas por la página oficial de Bluetooth [52][53]. Tras ello, se trató de proveer la red mesh obteniendo los resultados mostrados en la Ilustración 20, en la que se puede apreciar cómo no se resuelven correctamente los servicios proporcionados por el BlueTile y consecuentemente no se permite conectar a ellos empleando este protocolo. En primer lugar, se pensó que esto se debía a la configuración que tenía por defecto la herramienta *meshctl*, pero tras probar modificando los diferentes parámetros de configuración de dicha herramienta, los resultados no variaron. Por ello, se determinó que meshctl solo funciona correctamente empleando sensores que utilicen los

proyectos BLE Mesh proporcionados por Zephyr [54]. Esto mismo sucede al emplear el módulo multisensor Sensortile.box. Además, se han realizado varias consultas con el soporte ofertado por ST Microelectronics y no han proporcionado ninguna alternativa existente a ello, dando a entender en todo momento que únicamente aseguran su funcionamiento con la aplicación antes comentada.

Ilustración 20: Fallo al proveer la red mesh utilizando BLE Mesh

Por tanto, se ha optado por emplear el protocolo Bluetooth Low Energy para realizar las comunicaciones entre módulos multisensor y Repetidores o Gateways. Para comunicarse con los módulos multisensor desde las Raspberry se han empleado scripts de Python. Con el fin de que esto fuese posible ha sido necesario instalar bluez en las Raspberry, una colección de módulos de

extensión para Python que permite realizar conexiones con dispositivos externos utilizando Bluetooth, y la librería bluepy que proporciona una interfaz Python para dispositivos que emplean la tecnología BLE.

En cuanto al protocolo empleado para llevar a cabo una red inalámbrica mallada en la que se interconecten los Repetidores y Gateways, se ha decidido emplear el protocolo *Better Approach To Mobile Adhoc Networking (B.A.T.M.A.N.)*. Este protocolo ofrece una implementación en forma de un módulo Kernel de Linux que funciona enteramente en capa 2 y que no solo transporta la información en paquetes Ethernet, sino que también gestiona el tráfico de redes. Este módulo Kernel recibe el nombre de *B.A.T.M.A.N. advanced* o *batman-adv*. De esta forma, todos los nodos simulan estar enlazados localmente teniendo un desconocimiento completo de la topología de la red y no viéndose afectados por posibles cambios en ella. *Batman-adv* proporciona una serie de ventajas muy útiles en el ámbito del Internet de las Cosas:

- Permite utilizar los protocolos que se deseen por encima de batman-adv (IPv4, IPv6, DHCP, etc)
- No es necesario que los Repetidores se encuentren conectados directamente a la red, mediante un cable Ethernet o a través de WiFi, para tener conexión a Internet.
- Fácil incorporación de nuevos nodos a la red sin necesidad de realizar ningún cambio en la configuración de los dispositivos. Es decir, una vez se haya implementado la versión inicial, para añadir un nuevo nodo a la red únicamente habrá que configurar ese nodo.
- Posibilidad de ser tolerante a fallos en la red mediante el uso simultaneo de varios Gateways.
- Tolerancia a fallos en la comunicación entre nodos gracias a la redundancia de Repetidores.
- Facilidad a la hora de gestionar y testear la red mallada mediante el uso de la herramienta *batctl* proporcionada al instalar el Kernel.

Finalmente, para realizar la comunicación entre Gateway y Nube, existen muchas posibilidades diferentes. Como por ejemplo el uso de un cliente MQTT que envíe un mensaje a un determinado tópico, o el uso de un cliente CoAP o HTTP que hagan una petición POST a una determinada URL. Para este trabajo, se empleará el cliente MQTT empleando el bróker de mensajes de código abierto Mosquitto.

5.3. Configuración de los Gateways y los Repetidores

Como se ha comentado en el apartado **Protocolos de comunicación empleados**, se ha decidido utilizar el protocolo B.A.T.M.A.N. para generar la red mallada entre Repetidores y Gateways. En primer lugar, es necesario descargar e instalar en la tarjeta SD de cada una de las Raspberry el Sistema Operativo a utilizar, en este caso, *Raspberry Pi OS (32-bit) with desktop and recommended software*. Tras su instalación se recomienda habilitar la conexión SSH para permitir acceder a la Raspberry desde el cliente SSH Putty. Una vez habilitado, se lleva a cabo una actualización del sistema mediante el comando:

```
sudo apt-get update && sudo apt-get upgrade -y
```

Posteriormente, es necesario instalar la herramienta batctl, que se empleará para configurar la red mesh, ejecutando el comando:

```
sudo apt-get install -y batctl
```

Finalizada la instalación, es necesario reiniciar el equipo con el fin de hacer efectivos todos los cambios, actualizaciones e instalaciones. Una vez hecho esto, se procede a generar un fichero de nombre batman-adv-conf.sh que contenga la configuración del protocolo B.A.T.M.A.N. El contenido de este fichero puede apreciarse en el Listado 1.

```
#!/bin/bash
# Seleccion de la interfaz a usar con el protocolo batman-adv
sudo batctl if add wlan0
sudo ifconfig bat0 mtu 1468

# Indicar que el dispositivo ejercera de nodo en la red Mesh
sudo batctl gw_mode client

# Habilitar y activar las interfaces empleadas por el protocolo batman-adv
sudo ifconfig wlan0 up
sudo ifconfig bat0 up

# Asignar una IP estatica al nodo
# No es necesario pero recomendable para probar la red mallada y sus interconexionesa
sudo ifconfig bat0 192.168.30.2/24
```

Listado 1: Código del fichero batman-adv-conf.sh para los Repetidores

Una vez generado el fichero, se debe hacer que este sea ejecutable mediante el comando:

```
chmod +x ~/batman-adv-conf.sh
```

También es necesario definir la interfaz wlan0, añadiendo las líneas de código mostradas en el Listado 2 al fichero /etc/network/interfaces.d/wlan0. El Wireless-essid se corresponde con el identificador de la red mallada creada, y en este caso se llamará protoype-mesh. Además, en el atributo Wireless-channel se selecciona el número de canal a emplear, siendo válidos en Europa aquellos comprendidos entre 1 y 13 [55]. Estos parámetros deberán ser iguales para todos los dispositivos que formen la red.

```
1 auto wlan0
2 iface wlan0 inet manual
3         wireless-channel 1
4         wireless-essid prototype-mesh
5         wireless-mode ad-hoc
```

Listado 2: Definición de wireless-essid y wireless-channel

Una vez realizado todos estos cambios, lo restante es hacer que el script batman-adv-conf.sh sea ejecutado durante el arranque de la Raspberry. Para ello basta con modificar el contenido del fichero /etc/rc.local de forma que sea igual al mostrado en el Listado 3.

```
 1 #!/bin/sh -e
 2 #
 3 # rc.local
 4 #
 5 # This script is executed at the end of each multiuser runlevel.
 6 # Make sure that the script will "exit 0" on success or any other
 7 # value on error.
 8 #
 9 # In order to enable or disable this script just change the execution
10 # bits.
11 #
12 # By default this script does nothing.
13
14 # Print the IP address
15 _IP=$(hostname -I) || true
16 if [ "$_IP" ]; then
17   printf "My IP address is %s\n" "$_IP"
18 fi
19
20 # Run the batman-adv-conf script when the Raspberry is booting
21 /home/pi/batman-adv-conf.sh &
22
23 exit 0
```

Listado 3: Fichero /etc/rc.local

Tras realizar las configuraciones anteriores, los Repetidores de la red mallada ya estarían listos para funcionar adecuadamente. Sin embargo, los Gateways aún no se encuentran listos para funcionar, ya que es necesario que redirijan todo el tráfico de la red hacia el exterior (la nube) y todo el tráfico proveniente del exterior a través de la red mallada. Lo primero que se debe hacer es determinar la red que va a usar la red mallada durante este trabajo. Los parámetros de la red utilizada pueden verse en la Tabla 3.

Red	192.168.30.x
Máscara	255.255.255.0
Dirección IP del Gateway	192.168.30.1

Tabla 3: Parámetros de la red mallada

En este caso, el Gateway deberá actuar como servidor DHCP de la red, de forma que proporcione una configuración de red válida para todos los dispositivos que formen parte de ella. Para ello, es necesario instalar el software DHCP en la Raspberry mediante el comando:

```
sudo apt-get install -y dnsmasq
```

Una vez instalado, se ha de modificar el fichero /etc/dnsmasq.conf y añadir al final de este el rango de direcciones válidas que se pueden asignar a los dispositivos, así como la interfaz que se empleará, tal y como se muestra en el Listado 4.

```
685 interface=bat0
686 dhcp-range=192.168.30.2,192.168.30.220,255.255.255.0,12h
```

Listado 4: Líneas a añadir al fichero /etc/dnsmasq.conf

Como se puede apreciar en el Listado 4, el rango válido de direcciones IP está comprendido entre la dirección 192.168.30.2 y la dirección 192.168.30.220 empleando una máscara de red 255.255.255.0. Todo ello sobre la interfaz bat0. Realizado esto, el servicio DHCP ya estaría en funcionamiento y solamente sería necesario modificar el fichero batman-adv-conf.sh de manera que el tráfico sea redirigido a través de la red de forma correcta, y estableciendo su modo de funcionamiento como Gateway en lugar de nodo, como sucede en el caso de los Repetidores.

También es necesario añadir una dirección IP estática al Gateway para que esta sea siempre la misma. En el caso de este trabajo, la dirección empleada en el Gateway es la 192.168.30.1/24. Toda esta configuración se puede observar en el Listado 5.

```
#!/bin/bash
# Seleccion de la interfaz a usar con el protocolo batman-adv
sudo batctl if add wlan0
sudo ifconfig bat0 mtu 1468

# Indicar que el dispositivo ejercera de Gateway de la red Mesh
sudo batctl gw_mode server

# Habilitar el reenvío de puertos
sudo sysctl -w net.ipv4.ip_forward=1
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo iptables -A FORWARD -i eth0 -o bat0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i bat0 -o eth0 -j ACCEPT

# Habilitar y activar las interfaces empleadas por el protocolo batman-adv
sudo ifconfig wlan0 up
sudo ifconfig bat0 up

# Asignar la IP estática 192.168.30.1 al Gateway
sudo ifconfig bat0 192.168.30.1/24
```

Listado 5: Código del fichero batman-adv-conf.sh para los Gateway

Para validar que el Gateway esta correctamente configurado y su funcionamiento es el deseado, basta con realizar dos comprobaciones, una mediante la llamada al comando **iwconfig** que permite ver la configuración de las interfaces inalámbricas del dispositivo, y otra mediante la llamada al comando **ifconfig** que permite ver la configuración de todas las interfaces de red del Gateway. En la Ilustración 21, se puede apreciar como la interfaz wlan0 tiene como ESSID asignado el nombre de la red mesh (protoype-mesh), además de observar que el modo de funcionamiento en el que se encuentra es Ad-Hoc.

```
pi@raspberrypi:~ $ iwconfig
bat0      no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11  ESSID:"prototype-mesh"
          Mode:Ad-Hoc  Frequency:2.412 GHz  Cell: EA:54:5A:A3:2A:E5
          Tx-Power=31 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:on

lo        no wireless extensions.
```

Ilustración 21: Resultado del comando iwconfig

El resultado obtenido tras la ejecución del comando **ifconfig**, se muestra en la Ilustración 22 donde se puede apreciar como la interfaz bat0 tiene asignada la IP estática correspondiente al Gateway, concretamente la 192.168.30.1, así como que la Raspberry se encuentra conectada a Internet mediante la interfaz Ethernet y que la interfaz wlan0 no dispone de ninguna IP.

```
pi@raspberrypi:~ $ ifconfig
bat0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1468
        inet 192.168.30.1  netmask 255.255.255.0  broadcast 192.168.30.255
        inet6 fe80::9b89:3e64:3c0:d8f8  prefixlen 64  scopeid 0x20<link>
        ether ee:58:b3:f5:d2:90  txqueuelen 1000  (Ethernet)
        RX packets 1392  bytes 98967 (96.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 450  bytes 44849 (43.7 KiB)
        TX errors 0  dropped 231 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.20  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::993f:817b:4436:fcdd  prefixlen 64  scopeid 0x20<link>
        ether dc:a6:32:60:1c:bb  txqueuelen 1000  (Ethernet)
        RX packets 9737  bytes 2500084 (2.3 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1035  bytes 109059 (106.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 5  bytes 504 (504.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5  bytes 504 (504.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::dea6:32ff:fe60:1cbd  prefixlen 64  scopeid 0x20<link>
        ether dc:a6:32:60:1c:bd  txqueuelen 1000  (Ethernet)
        RX packets 47439  bytes 2945333 (2.8 MiB)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 18110  bytes 1823529 (1.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Ilustración 22: Resultado del comando ifconfig

Finalizada la configuración de todos los dispositivos, es necesario comprobar el correcto funcionamiento de la red mallada. Para ello, se han configurado cinco Raspberry, de las cuales una actuará de Gateway y las otras cuatro serán nodos de la red mallada. En la Tabla 4 se muestra

cada uno de estos dispositivos con la dirección IP asignada y con la dirección MAC de su interfaz de red, la cual más adelante será de gran utilidad.

Modelo	Función en la red	Dirección IP	Dirección MAC
Raspberry Pi 4 Model B	Gateway	192.168.30.1	dc:a6:32:60:1c:bd
Raspberry Pi 4 Model B	Nodo 1	192.168.30.2	dc:a6:32:60:1c:ca
Raspberry Pi 4 Model B	Nodo 2	192.168.30.3	dc:a6:32:60:1b:59
Raspberry Pi 3 Model B +	Nodo 3	192.168.30.4	b8:27:eb:12:3c:7e
Raspberry Pi 4 Model B	Nodo 4	192.168.30.5	dc:a6:32:60:1b:01

Tabla 4: Dispositivos de la red mesh

Una vez configuradas, se han conectado todos los dispositivos formando una topología mallada que consta de un Gateway, conectado a la red a través de un cable Ethernet y varios nodos Repetidores, interconectados entre sí y con el Gateway a través de sus interfaces inalámbricas, formando una red mesh inalámbrica gestionada por el protocolo de enrutamiento inalámbrico B.A.T.M.A.N. Este prototipo de prueba se muestra en la Ilustración 23.

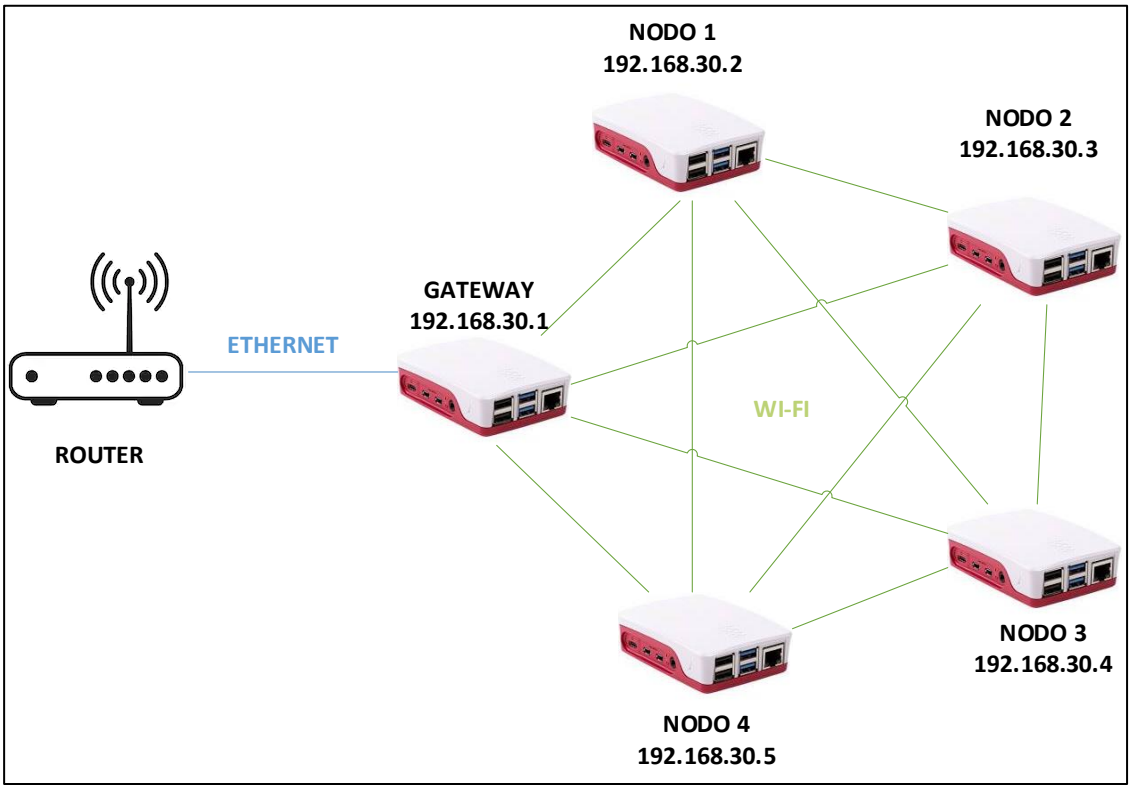

Ilustración 23: Prototipo de validación de funcionamiento de la red mesh

Para comprobar que todos los dispositivos son accesibles entre sí, el módulo Kernel batman-adv instalado en los dispositivos proporciona la herramienta de gestión batctl, que permite ejecutar una serie de comandos para manejar la red y controlar las interconexiones, así como llevar a cabo pruebas de ping a través de la capa 2 entre los dispositivos que conforman la red mallada.

En primer lugar, se emplea el comando **sudo batctl n**, que muestra todos los dispositivos vecinos al dispositivo desde el que se ejecuta el comando. Para esta prueba, se ha ejecutado el comando desde el Gateway, obteniendo el resultado mostrado en la Ilustración 24.

```
pi@raspberrypi:~ $ sudo batctl n
[B.A.T.M.A.N. adv 2018.3, MainIF/MAC: wlan0/dc:a6:32:60:1c:bd (bat0/ee:58:b3:f5:d2:90 BATMAN_IV)]
IF             Neighbor              last-seen
        wlan0     dc:a6:32:60:1b:01    0.220s
        wlan0     dc:a6:32:60:1b:59    0.550s
        wlan0     dc:a6:32:60:1c:ca    0.200s
        wlan0     b8:27:eb:12:3c:7e    0.460s
```

Ilustración 24: Resultado del comando sudo batctl n

Como se puede apreciar, son cuatro los vecinos detectados, los cuales se corresponden con los cuatro nodos que forman la red. Además, sus direcciones MAC aparecen acompañadas de el tiempo transcurrido desde que se ha detectado por última vez el dispositivo, es decir, cuando fue la última vez en la que se recibió el paquete OGM en el cual se informa al resto de nodos de su presencia.

Para comprobar la conexión con ellos, batctl proporciona también un comando que permite hacer ping a todos los vecinos a partir de su dirección MAC. Mediante su ejecución se puede asegurar que los dispositivos se conectan con el Gateway sin ningún problema.

```
pi@raspberrypi:~ $ sudo batctl ping dc:a6:32:60:1b:01
PING dc:a6:32:60:1b:01 (dc:a6:32:60:1b:01) 20(48) bytes of data
20 bytes from dc:a6:32:60:1b:01 icmp_seq=1 ttl=50 time=1.53 ms
20 bytes from dc:a6:32:60:1b:01 icmp_seq=2 ttl=50 time=1.66 ms
^C--- dc:a6:32:60:1b:01 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss
rtt min/avg/max/mdev = 1.532/1.595/1.658/0.063 ms
pi@raspberrypi:~ $ sudo batctl ping dc:a6:32:60:1b:59
PING dc:a6:32:60:1b:59 (dc:a6:32:60:1b:59) 20(48) bytes of data
20 bytes from dc:a6:32:60:1b:59 icmp_seq=1 ttl=50 time=2.18 ms
20 bytes from dc:a6:32:60:1b:59 icmp_seq=2 ttl=50 time=1.52 ms
^C--- dc:a6:32:60:1b:59 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss
rtt min/avg/max/mdev = 1.523/1.854/2.185/0.331 ms
```

Ilustración 25: Resultado del comando sudo batctl ping

En la Ilustración 25, se muestran los resultados obtenidos tras realizar ping desde el Gateway a los nodos con las direcciones MAC dc:a6:32:60:1b:01 y dc:a6:32:60:1b:59 que se corresponden con el nodo 4 y el nodo 2 respectivamente. Se puede apreciar como en ambos casos se realiza ping de forma exitosa con un tiempo medio de conexión próximo a los 1.5 ms.

Finalmente, faltaría comprobar la conexión de cada uno de los nodos Repetidores con el exterior a través del Gateway. Para ello, al disponer todos los nodos de una IP estática ya conocida, se puede acceder a cada uno vía SSH desde el Gateway, y posteriormente llevar a cabo un ping a Internet, por ejemplo, a la dirección IP 8.8.8.8. En la Ilustración 26, se muestran los resultados obtenidos al realizar ping a dicha dirección IP desde el nodo 1.

```
pi@raspberrypi:~ $ ssh 192.168.30.2
pi@192.168.30.2's password:
Linux raspberrypi 4.19.118-v7l+ #1311 SMP Mon Apr 27 14:26:42 BST 2020 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Aug 27 15:06:05 2020
-bash: warning: setlocale: LC_ALL: cannot change locale (es_ES.UTF-8)

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.

pi@raspberrypi:~ $ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=16.9 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=16.10 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=16.5 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=118 time=15.8 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 15.813/16.538/16.955/0.484 ms
```

Ilustración 26: Prueba de conexión a Internet desde el nodo 1

Como se puede apreciar, primero se accede a través de SSH a la dirección IP del nodo 1 desde el Gateway y posteriormente se ejecuta el comando ping a la dirección 8.8.8.8. En base a los resultados obtenidos, se puede asegurar que el nodo 1 dispone de conexión a Internet. Este proceso se ha repetido para cada uno de los nodos obteniendo resultados similares a los mostrados en la Ilustración 26 y consecuentemente asegurando la conectividad de todos ellos a Internet.

5.4. Configuración de los módulos multisensor

Uno de los objetivos iniciales de este trabajo, era el desarrollo de un prototipo demostrativo de la topología de red desarrollada. Se ha decidido aplicarlo al ámbito de la monitorización de edificios, concretamente del Edificio Departamental de Informática del Campus de la Escuela Politécnica de Ingeniería de Gijón. Al tratarse de este ámbito, los sensores que resultan más interesantes son los de temperatura, presión, humedad y ópticos. No todos los módulos multisensor empleados en este trabajo disponen de todos los sensores mencionados, en la Tabla 5 se muestran los sensores empleados por cada módulo multisensor. Además, cabe destacar que todos ellos salvo el Smartbond DA14585 ofrecen la posibilidad de saber el porcentaje de batería del que disponen.

	Termómetro	Barómetro	Higrómetro	Luxómetro
Sensortag CC2650	X	X	X	X
Smartbond DA14585	X	X	X	X
Sensortile.box	X	X	X	
BlueTile	X	X	X	

Tabla 5: Sensores disponibles en módulos multisensor

Para llevar a cabo las comunicaciones entre los módulos multisensor y los Repetidores o Gateways es necesario, en primer lugar, modificar el firmware de los módulos multisensor para que se adapte a lo requerido durante este trabajo y, en segundo lugar, generar un script de Python que se ejecute en los Repetidores o Gateways encargado de conectarse con los módulos multisensor y de recolectar los datos emitidos por estos últimos. En este apartado, se explicarán las modificaciones realizadas y se generará un script de Python que demuestre su correcto funcionamiento para cada uno de ellos.

5.4.1. Sensortag CC2650

En primer lugar, es necesario importar el proyecto sensortag_cc2650stk_app proporcionado por el fabricante. Tras ello, lo siguiente que se realizó fue seleccionar exclusivamente aquellos sensores que fuesen a ser empleados en el prototipo desarrollado. Para esto, hay que definir variables en los símbolos predefinidos, accediendo a las propiedades del proyecto y, una vez ahí, accedemos a la ruta ***Build > ARM Compiler > Advanced Options > Predefined Symbols*** donde se añadirán las variables EXCLUDE_TMP (que excluye el sensor de temperatura, ya que para

medir la temperatura se empleará, bien el higrómetro o el barómetro debido a que Texas Instruments ha dejado de producir los Sensortag CC2650 con dicho sensor y por tanto facilitar su posible escalabilidad en un futuro), EXCLUDE_KEYS (que elimina la posibilidad de apagar, resetear o dormir el sensor mediante la pulsación de alguno de sus botones) y EXCLUDE_MOV (que excluye acelerómetro, magnetómetro y giroscopio). Todo esto puede apreciarse en la Ilustración 27.

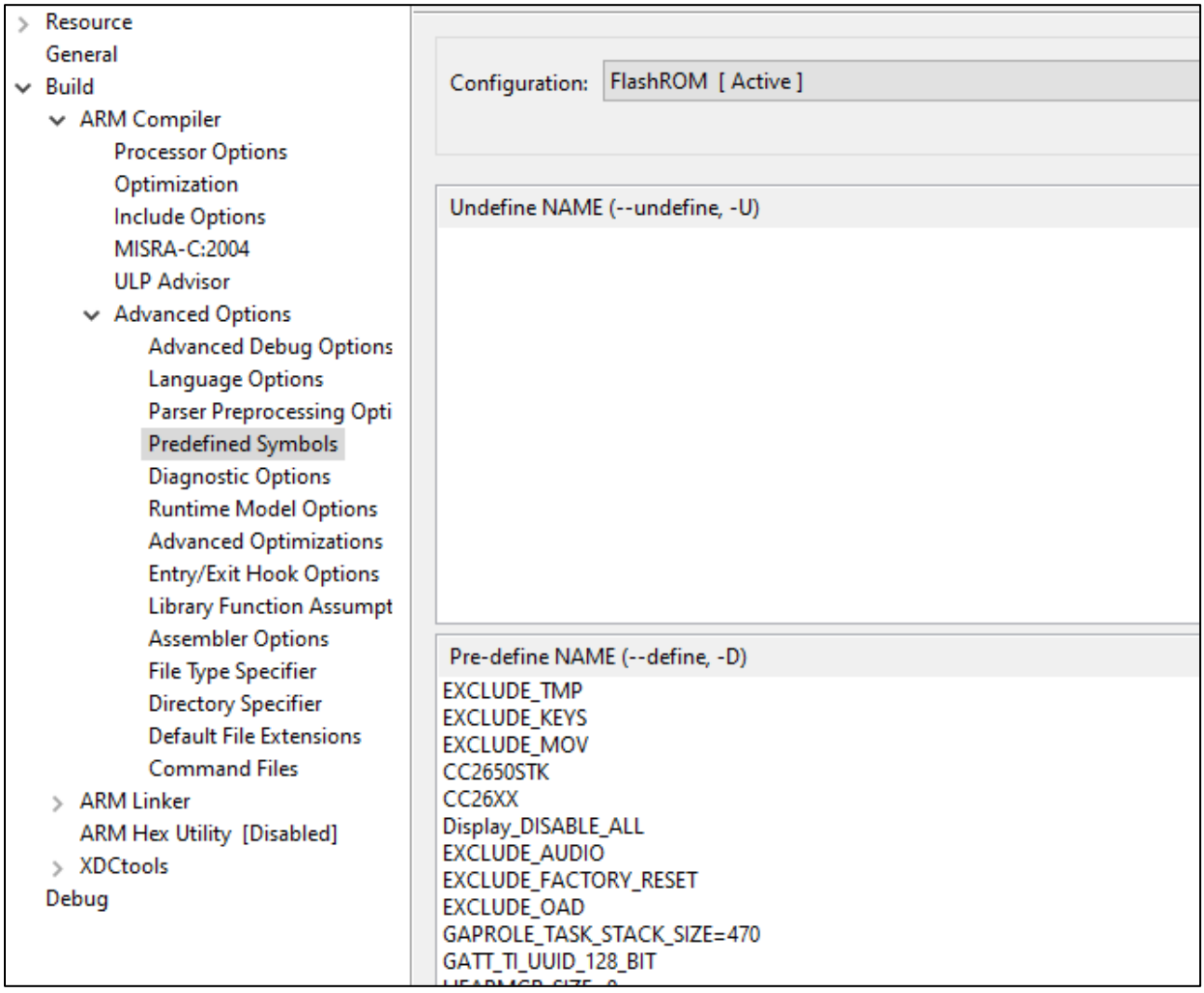

Ilustración 27: Exclusión de sensores en el Sensortag CC2650

Una vez seleccionados solo el higrómetro, barómetro y luxómetro, se definieron los periodos de lectura de cada uno de estos sensores. Se ha decidido realizar medidas cada 6 minutos. Para lograr esto ha sido necesario modificar cuatro ficheros diferentes que se corresponden con:

- **sensortag_hum.c**: concretamente la línea 77 que se corresponde con la definición del periodo de lectura por defecto del sensor de humedad en milisegundos, por lo que se ha modificado su valor inicial por 360000 ms que es equivalente a 6 minutos.

```
#define SENSOR_DEFAULT_PERIOD   360000
```

- **sensortag_batt.c**: concretamente la línea 74 que se corresponde, al igual que en el caso anterior, con la definición del periodo de lectura por defecto de la situación actual de la batería. Por ello, de forma análoga, se modifica el valor a 360000

```
#define BATT_PERIOD   360000
```

- **sensortag_bar.c**: concretamente la línea 76 que se corresponde con la definición del periodo de lectura por defecto del barómetro. Por ello, de forma análoga, se modifica el valor a 360000.

```
#define SENSOR_DEFAULT_PERIOD   360000
```

- **sensortag_opt.c**: en este caso, se modifica la línea 76 que se corresponde con la definición del periodo de lectura por defecto del luxómetro. Sin embargo, para que el sensor funcione correctamente el máximo valor permitido es 6000.

```
#define SENSOR_DEFAULT_PERIOD   6000
```

Con estos cambios no basta para tener el módulo multisensor configurado. Falta un cambio muy importante y es que inicialmente el sensor está preparado para dormirse tras varios minutos sin establecer ninguna conexión con otro dispositivo. Para evitar esto, es necesario modificar el fichero sensortag.c, cambiando el contenido de la línea 116 por el siguiente:

```
#define DEFAULT_DISCOVERABLE_MODE             GAP_ADTYPE_FLAGS_GENERAL
```

De esta forma, se logra que el Sensortag CC2650 no entre nunca en reposo y este constantemente anunciándose y aceptando conexiones por parte del Gateway o Repetidor correspondiente. Tras todas estas modificaciones se actualiza el módulo multisensor el cual ya estaría listo para su uso.

La parte restante se corresponde con la generación de un script Python el cual será utilizado en el Gateway para inicializar la conexión con el módulo multisensor y posteriormente recibir la temperatura, humedad, presión y luminosidad del entorno en el que se encuentra. Para inicializar

la conexión desde el Gateway con el módulo multisensor se emplea la función del módulo bluepy para Python, **btle.Peripheral("Dirección MAC")**, la cual recibiendo como parámetro la dirección MAC del dispositivo al que desea conectarse inicializa la conexión con él. Tras ello, la conexión al dispositivo ya estaría realizada y solo restaría leer las variables del entorno que resulten interesantes. Como se comentó antes, se emplearán los sensores de presión, humedad y luminosidad, además de que el propio sensor ofrece la posibilidad de emitir el estado actual de la batería. Por ello, se han desarrollado cuatro funciones en Python que leen los datos correspondientes a cada uno de esos sensores.

```
13 def ReadHumidity(p):
14     global hum
15     services = p.getServiceByUUID("f000aa20-0451-4000-b000-000000000000")
16     charact = services.getCharacteristics()
17     charactCtrl = charact[1]
18     charactData = charact[0]
19     charactCtrl.write(ON, withResponse=True)
20     time.sleep(5)
21     (temp, hum) = struct.unpack('<HH',charactData.read())
22     hum = 100.0 *(hum/65536.0)
23     charactCtrl.write(OFF, withResponse=False)
```

Listado 6: Función ReadHumidity(p) del Sensortag CC2650

En el Listado 6, se muestra la función encargada de obtener los datos capturados por el higrómetro. En primer lugar, se obtiene el servicio relativo al sensor de humedad que es el correspondiente al UUID **f000aa20-0451-4000-b000-000000000000**. Ese servicio a su vez dispone de 3 características, de las cuales solo resultan interesantes dos de ellas, la primera correspondiente a los datos obtenidos por el sensor y la segunda relativa al control del sensor, permitiendo encenderlo y apagarlo a voluntad del usuario. En las líneas 19 y 23, se puede apreciar como escribiendo sobre la característica relacionada con el control del sensor se enciende y apaga respectivamente (ON corresponde a los bytes 0x01 y OFF a 0x00). La lectura de los datos se realiza en la línea 21, donde se puede apreciar cómo se desempaquetan dos unsigned short en formato little-endian correspondientes a la temperatura y humedad respectivamente. Inicialmente se pretendía emplear el higrómetro para capturar la temperatura del entorno además de la humedad, pero durante el trabajo se observó la existencia de un fallo del propio sensor que provocaba que la primera lectura realizada sobre él emitiese una temperatura fuera de rango e

incorrecta, por lo que al final no se usó. De ahí, que solo se realice la transformación de la humedad (véase la línea 22 del Listado 6).

La función que permite leer el nivel de batería del módulo multisensor es mucho más simple que la anterior, pues en este caso no es necesario activar y desactivar el sensor. El contenido de esta función se muestra en el Listado 7, donde se puede apreciar cómo se obtiene el servicio con el UUID **0000180f-0000-1000-8000-00805f9b34fb**, correspondiente al servicio de batería. Dicho servicio solo dispone de una característica que es la relacionada con el nivel de batería. Para llevar a cabo la transformación de los datos obtenidos se emplea la función Python ord() la cual recibe un carácter y retorna el número Unicode que representa dicho carácter, y el cuál se corresponde con el nivel de batería del módulo multisensor (véase la línea 29 del Listado 7).

```
25 def ReadBattery(p):
26     global bat
27     services = p.getServiceByUUID("0000180f-0000-1000-8000-00805f9b34fb")
28     charact = services.getCharacteristics()[0]
29     bat = ord(charact.read())
30
31 def ReadBarometer(p):
32     global temp, pres
33     services = p.getServiceByUUID("f000aa40-0451-4000-b000-000000000000")
34     charact = services.getCharacteristics()
35     charactCtrl = charact[1]
36     charactData = charact[0]
37     charactCtrl.write(ON, withResponse=True)
38     time.sleep(5)
39     (tempL, tempM, tempH, presL, presM, presH) = struct.unpack('<BBBBBB',charactData.read())
40     temp = (tempH*65536 + tempM*256 + tempL)/100.0
41     pres = (presH*65536 + presM*256 + presL)/100.0
42     charactCtrl.write(OFF, withResponse=True)
```

Listado 7: Función ReadBattery(p) y ReadBarometer(p) del Sensortag CC2650

Además, en el Listado 7, también se puede apreciar la función que permite llevar a cabo la lectura de la presión y temperatura del sensor. Es una función muy similar a la de lectura del higrómetro, cambiando únicamente el UUID del servicio, que es **f000aa40-0451-4000-b000-000000000000** y la lectura de los datos y transformación de estos, que se puede apreciar en las líneas 39, 40 y 41. Además, cabe destacar, que en este caso al igual que en el del higrómetro si es necesario activar y desactivar el sensor para realizar las lecturas. También comentar, que durante este trabajo se empleará el barómetro para medir también la temperatura del entorno, ya que midiéndola con el barómetro no se producen los fallos apreciados al medir la temperatura con el higrómetro.

Finalmente, la última función definida es la que permite leer la luminosidad del entorno capturada por el luxómetro. El servicio del luxómetro es el **f000aa40-0451-4000-b000-000000000000.** El contenido de esta función se muestra en el Listado 8, y al igual que en el caso del higrómetro y del barómetro, es necesario activar y desactivar el sensor para obtener los datos. En las líneas 52, 53, 54 y 55 se realiza la lectura de los datos y su posterior conversión. Como se puede apreciar se desempaqueta un short siguiendo un formato little-endian.

```
44 def ReadOptical(p):
45     global opt
46     services = p.getServiceByUUID("f000aa70-0451-4000-b000-000000000000")
47     charact = services.getCharacteristics()
48     charactCtrl = charact[1]
49     charactData = charact[0]
50     charactCtrl.write(ON, withResponse=True)
51     time.sleep(5)
52     raw = struct.unpack('<h',charactData.read())[0]
53     m = raw & 0xFFF;
54     e = (raw&0xF000) >> 12;
55     opt = 0.01 * (m<<e)
56     charactCtrl.write(OFF, withResponse=False)
```

Listado 8: Función ReadOptical(p) del Sensortag CC2650

Para comprobar el correcto funcionamiento, se creó una función en la cual se inicializaba la conexión con el sensor y posteriormente se leían los datos capturados por el sensor de presión, humedad y luz, además del estado actual de la batería. Posteriormente todos esos valores se muestran en pantalla y se procede a la desconexión del módulo multisensor. Más adelante, esos valores en lugar de ser mostrados en pantalla serán enviados a la nube donde serán almacenados. Los resultados obtenidos durante la comprobación de funcionamiento se muestran en la Ilustración 28.

```
pi@raspberrypi:~ $ python3 sensortagcc2650.py

Conectando a 24:71:89:E9:52:83
Sensor conectado
Presión: 997.98mBar -  Temp: 18.89°C - Hum: 65.618896484375% - Luz: 9.16lux - Bateria: 80%
Sensor desconectado


Conectando a 24:71:89:E9:52:83
Sensor conectado
Presión: 997.95mBar -  Temp: 18.87°C - Hum: 65.52734375% - Luz: 9.24lux - Bateria: 80%
Sensor desconectado
```

Ilustración 28: Comprobación conectividad Sensortag CC2650 – Gateway

5.4.2. Smartbond DA14585 IoT

Para configurar y adaptar el firmware del módulo multisensor Smartbond DA14585 IoT lo primero que hay que hacer es abrir el proyecto iot_585 proporcionado por el fabricante. Al igual que en el caso del Sensortag CC2650 es necesario eliminar aquellos sensores que no se vayan a necesitar, es decir, dejando exclusivamente el sensor óptico, para medir la luminosidad del entorno y el sensor ambiental, encargado de medir tanto presión, temperatura y humedad del entorno. Para ello, en el fichero da1458x_config_basic.h se definen exclusivamente estos dos sensores dejando sin definir el resto, tal y como puede apreciarse a continuación.

```
// Habilitar sensor óptico
#define VCNL4010_OPTO_SENSOR_AVAILABLE
// Inhabilitar sensores de movimiento
#undef AK099XX_MAGNETO_SENSOR_AVAILABLE
#undef ACCEL_SENSOR_AVAILABLE
#ifdef ACCEL_SENSOR_AVAILABLE
    #define ICM4XX_ACCEL_SENSOR_AVAILABLE
    #undef BMI160_ACCEL_SENSOR_AVAILABLE
    #ifdef BMI160_ACCEL_SENSOR_AVAILABLE
        #undef BMI_ON_EXTERNAL_HEADER
    #endif
#endif

// Habilitar sensor ambiental
#define BME680_ENVIRONM_SENSOR_AVAILABLE
#ifdef BME680_ENVIRONM_SENSOR_AVAILABLE
    #undef IAQ_ENABLED
    #ifdef IAQ_ENABLED
        #define IAQ_RESTORE_STATUS_ENABLE
    #else
         // No interesa habilitar la lectura de gas
        #undef RAW_GAS_ENABLED
    #endif
#endif
```

Una vez excluidos los sensores que no se van a emplear, es necesario modificar la frecuencia de captura del sensor óptico, editando el fichero user_sensor_config.h e incrementando el tiempo transcurrido entre diferentes capturas al máximo permitido, siendo en el caso del sensor óptico de 0,2 Hz.

```
#define DEFAULT_OPTO_RATE                OPTO_RATE_02HZ
```

En el caso del sensor ambiental, no es necesario realizar ninguna modificación, pues la frecuencia de captura ya se encuentra establecida al mínimo posible, que se corresponde con 0,33 Hz.

De forma similar a lo sucedido en el caso del Sensortag CC2650, el sensor, pasado un tiempo, entra en modo reposo y deja de permitir entablar conexión con él. Para evitarlo, es necesario definir en el fichero user_app_iot_config.h la variable ALWAYS_ADVERTISE.

```
#define ALWAYS_ADVERTISE    // Siempre anunciandose
```

Una vez realizados estos cambios se actualizó el firmware del Smartbond DA14585 IoT y se evaluó su funcionamiento, pero se apreció que el sensor óptico producía en ocasiones lecturas incorrectas en las que no actualizaba el valor de la luminosidad del entorno, observado al ver que al encender y apagar las luces de una habitación a oscuras el valor permanecía sin modificarse. Esto se debía a que el sensor óptico además de medir la luminosidad también mide la proximidad de objetos, y cuando se actualizaba el valor de la proximidad no se actualizaba el valor de la luminosidad y viceversa. Para evitarlo, al no ser de interés el detectar la proximidad de objetos, se optó por desactivar la opción de funcionamiento que permitía detectarlos. Para ello, se realizaron modificaciones en el fichero user_sensors.c donde se cambió la función user_sensor_opto_init() para que no habilitase la detección de objetos próximos. El contenido de esta función puede verse en el Listado 9.

```
770 void user_sensor_opto_init(void)
771 {
772     si_config_t si_opto_config;
773     memset((&si_opto_config), 0, sizeof (si_config_t));
774 
775     { // DEVICE_SPECIFIC for VCNL4010 Optical Sensor
776     user_opto_setup.led_current = 2;
777     user_opto_setup.proxim_enabled = 0;
778     user_opto_setup.ambl_enabled = SENS_CONF_AMBL_ENABLE;
779     user_opto_setup.high_threshold = 2300;
780     user_opto_setup.low_threshold = 0;
781     }
782     // SI settings for OPTO
783     si_opto_config.sensor_id = OPTICAL;
784     si_opto_config.operation_mode = TIMER;
785     si_opto_config.set_sensor_config = (si_configuration_fptr_t)optical_setup;
786     si_opto_config.force_read_fn = (force_read_fptr_t)optical_set_force_read;
787     si_opto_config.read_fn = (read_data_buf_fptr_t)optical_read;
788     si_opto_config.user_app_cb = (read_data_buf_fptr_t)user_opto_data_cb;
789     si_opto_config.data_size = sizeof(struct vcnl4010_data);
790 
791     si_opto_config.read_delay = 50;
792     si_opto_config.periodic_read_interval = user_translate_opto_rate_to_num(SENS_CONF_PROX_AMBL_RATE);
793 
794     si_register_sensor(&si_opto_config);
795 }
```

Listado 9: Función user_sensor_opto_init() del Smartbond DA14585 IoT

Como puede apreciarse en el Listado 9, se modificó la línea 777 para que el valor del parámetro user_opto_setup.proxim_enabled fuese siempre 0, lo que se corresponde con que esté siempre desactivado. Tras esta modificación se actualizó el firmware del módulo multisensor y se observó como los datos obtenidos eran correctos. Una vez programado el módulo multisensor, es necesario desarrollar un script Python capaz de conectarse y recolectar los datos capturados por dicho módulo, que son temperatura, humedad y presión, recogida por el sensor de ambiente y luminosidad del entorno captada por el luxómetro.

El UUID del servicio del módulo multisensor es **2ea78970-7d44-44bb-b097-26183f402400** el cual posee dos características interesantes, que son la de control, que permite activar y desactivar los sensores, cuyo UUID es **2ea78970-7d44-44bb-b097-26183f402409**, y la de datos, cuyo UUID es **2ea78970-7d44-44bb-b097-26183f402410**. Al igual que en el caso del Sensortag CC2650, en primer lugar, se inicializa la conexión con el módulo multisensor mediante la función **btle.Peripheral("MAC")**, que recibe como parámetro la dirección MAC del dispositivo al que se pretende conectar. Tras entablar conexión, se asigna a ese módulo multisensor un manejador de notificaciones, de forma que cada vez que se actualicen los valores leídos (presión, temperatura, humedad y luminosidad), se reciban los datos sin falta de solicitarlos, siempre y cuando se hayan habilitado las notificaciones antes desde el script.

```
73          print("Conectando a " + mac)
74          D = btle.Peripheral(MAC_ADDRESS)
75          print("Sensor conectado")
76          D.setDelegate(MyDelegate())
77          EnableNotifications(D)
78          EnableSensors(D)
79          while (l==-1 or p==-1 or h==-1 or t==-1):
80              D.waitForNotifications(5)
81          DisableSensors(D)
82          DisableNotifications(D)
83          D.disconnect()
84          print("Sensor desconectado")
```

Listado 10: Proceso para obtener los datos del Smartbond DA14585 IoT

En el Listado 10, se puede ver el proceso seguido para leer los datos del Smartbond DA14585. Algo muy importante es el orden de ejecución de estas operaciones, en primer lugar, se inicializa la conexión con el sensor (línea 74) y posteriormente se asigna el manejador de notificaciones (línea 76), tras ello se habilitan las notificaciones (línea 77) y finalmente se encienden los sensores (línea 78) y se espera a haber recibido todos los datos relevantes. Una vez se han obtenido, se

apagan los sensores (línea 81) y se deshabilitan las notificaciones (línea 82). Finalmente se desconecta del módulo multisensor (línea 83).

Para llevar a cabo todo este proceso, se definieron cuatro funciones que se encargan de activar y desactivar las notificaciones, así como de encender y apagar los sensores, las cuales pueden apreciarse en el Listado 11.

```
13 GENERAL_SERVICE_UUID = "2ea78970-7d44-44bb-b097-26183f402400"
14 CONTROL_UUID = "2ea78970-7d44-44bb-b097-26183f402409"
15 DATA_UUID = "2ea78970-7d44-44bb-b097-26183f402410"
16
17 def EnableNotifications(D):
18     service = D.getServiceByUUID(GENERAL_SERVICE_UUID)
19     charactData = service.getCharacteristics(DATA_UUID)[0]
20     charactData_handle = charactData.getHandle()
21     D.writeCharacteristic(charactData_handle + 1,b"\x01\00", True)
22
23 def DisableNotifications(D):
24     service = D.getServiceByUUID(GENERAL_SERVICE_UUID)
25     charactData = service.getCharacteristics(DATA_UUID)[0]
26     charactData_handle = charactData.getHandle()
27     D.writeCharacteristic(charactData_handle + 1,b"\x00\00", True)
28
29 def EnableSensors(D):
30     service = D.getServiceByUUID(GENERAL_SERVICE_UUID)
31     charactControl = service.getCharacteristics(CONTROL_UUID)[0]
32     charactControl.write(b"\x01", True)
33
34 def DisableSensors(D):
35     service = D.getServiceByUUID(GENERAL_SERVICE_UUID)
36     charactControl = service.getCharacteristics(CONTROL_UUID)[0]
37     charactControl.write(b"\x00", True)
```

Listado 11: Funciones auxiliares Smartbond DA14585 IoT

Tanto la función EnableNotifications(), como la función DisableNotifications() acceden a la característica relativa a los datos del sensor y escriben en la característica correspondiente a las notificaciones el byte 0x0001 para activarlas y 0x0000 para desactivarlas. En el caso de las funciones EnableSensors() y DisableSensors() lo que realizan es una modificación en el valor de la característica asociada al control del módulo multisensor, escribiendo los bytes 0x01 y 0x00 para encender y apagar los sensores respectivamente.

Sin embargo, la parte más importante de este script se corresponde con la clase MyDelegate(), que es la encargada de manejar las notificaciones recibidas por parte de los sensores y de realizar las transformaciones pertinentes sobre los datos recibidos. En base a las especificaciones del módulo multisensor, se observó que cuando los datos recibidos se correspondían con la

temperatura, dichos datos contenían la cadena **"060203"**; lo mismo sucede en el caso de la presión y la humedad, que ambas tienen asociada la cadena **"040203"** y la luminosidad del entorno, con la cadena asociada **"090203"**. Gracias a estas cadenas se puede saber a qué información corresponde cada notificación.

```
class MyDelegate(DefaultDelegate):
    def handleNotification(self, cHandle, data):
        global h, p, t, l
        raw_data = binascii.hexlify(data)
        raw_data = str(raw_data)[2:-1]
        sensor_report = raw_data[4:]
        if sensor_report.find("060203")==0:
            temp_data = sensor_report[6:14]
            aux = temp_data[6:]+temp_data[4:6]+temp_data[2:4]+temp_data[:2]
            t = int(aux,16)
            t/= 100.0
        if sensor_report.find("040203")==0:
            pres_data = sensor_report[6:14]
            aux = pres_data[6:]+pres_data[4:6]+pres_data[2:4]+pres_data[:2]
            p = int(aux,16)
            p/= 100.0
            hum_data = sensor_report[20:28]
            aux = hum_data[6:]+hum_data[4:6]+hum_data[2:4]+hum_data[:2]
            h = int(aux,16)
            h/=1024.0
        if sensor_report.find("090203")==0:
            lux_data = sensor_report[6:14]
            aux = lux_data[6:]+lux_data[4:6]+lux_data[2:4]+lux_data[:2]
            l = int(aux,16)
            l/=4.0
```

Listado 12: Clase MyDelegate() Smartbond DA14585 IoT

En el Listado 12 se puede apreciar el contenido de la clase MyDelegate() así como la gestión de las notificaciones recibidas gracias a la función handleNotification(). Para comprobar el correcto funcionamiento se realizó una prueba cuyos resultados se muestran en la Ilustración 29, en la que se conecta al módulo multisensor y se obtienen y muestran en pantalla los datos relacionados con la presión, temperatura, humedad y luz del entorno.

```
pi@raspberrypi:~ $ python3 smartbond.py

Conectando a 80:EA:CA:70:B0:56
Sensor conectado
Presión: 1015.03mBar - Temp: 19.42°C - Hum: 56.7509765625% - Luz: 21.75lux
Sensor desconectado

Conectando a 80:EA:CA:70:B0:56
Sensor conectado
Presión: 1014.99mBar - Temp: 19.4°C - Hum: 56.3427734375% - Luz: 3416.5lux
Sensor desconectado
```

Ilustración 29: Comprobación conectividad Smartbond DA14585 IoT – Gateway

5.4.3. Sensortile.box

Al igual que en los casos anteriores, se realizan las modificaciones del firmware del módulo multisensor. Para ello se emplea el proyecto SensorTileBox-BLESensors. En este caso los datos que resultan relevantes los emite el sensor ambiental y el medidor del nivel de batería. Inicialmente, se pretendía recibir primero la notificación relativa al estado actual del nivel de batería del módulo multisensor y posteriormente la relacionada con el sensor ambiental. Sin embargo, para reducir el número de notificaciones recibidas, se realizaron cambios en el firmware de forma que se modificasen los datos enviados a través de la notificación de batería para que además del nivel de batería, también se enviará la temperatura, presión y humedad del entorno. En la Ilustración 30 se muestra cómo eran los datos enviados inicialmente por la notificación de batería (a) y cómo son una vez se han realizado las modificaciones pertinentes en el firmware del módulo multisensor (b).

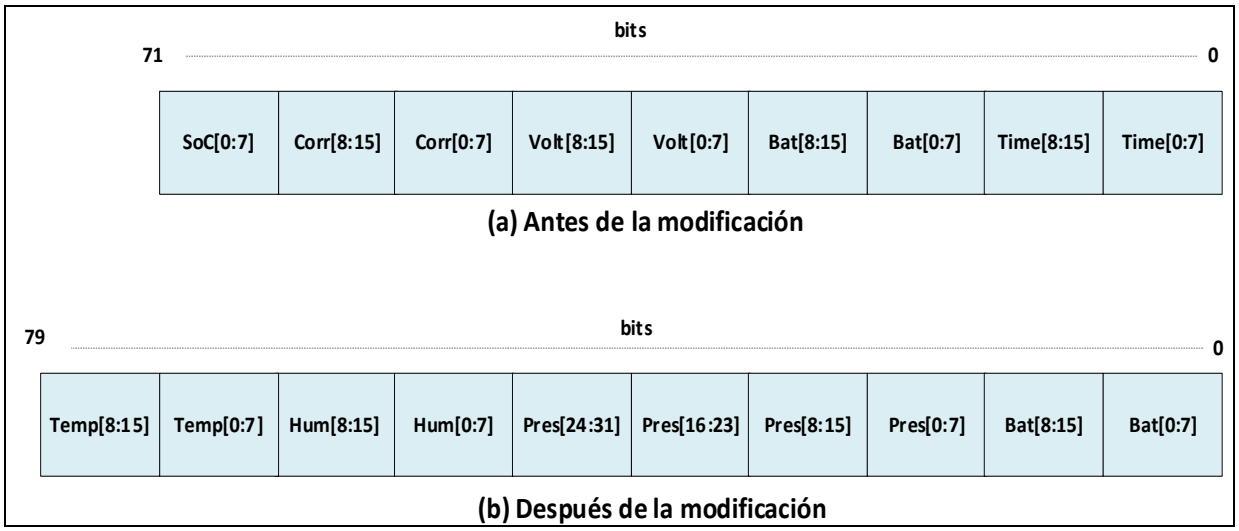

Ilustración 30: Formato de los datos antes (a) y después (b) de las modificaciones

Para lograrlo, en el fichero main.c se modificó la función main() de tal manera que esta solo enviase las notificaciones relacionadas con la batería, eliminando todos los envíos por parte de los otros sensores. Además, como se ve en el Listado 13, también se modificó la función InitMEM1_Sensors() del fichero TargetPlatform.c para que inicializase exclusivamente los sensores de presión, temperatura y humedad.

```
111 static void Init_MEM1_Sensors(void)
112 {
113   /* Humedad  */
114   if(BSP_ENV_SENSOR_Init(HTS221_0, ENV_HUMIDITY)==BSP_ERROR_NONE){
115     STBOX1_PRINTF("OK Humidity Sensor\n\r");
116   } else {
117     STBOX1_PRINTF("Error Humidity Sensor\n\r");
118     Error_Handler();
119   }
120
121   /* Temperatura */
122   if(BSP_ENV_SENSOR_Init(STTS751_0, ENV_TEMPERATURE)==BSP_ERROR_NONE){
123     STBOX1_PRINTF("OK Temperature Sensor\n\r");
124   } else {
125     STBOX1_PRINTF("Error Temperature Sensor\n\r");
126     Error_Handler();
127   }
128
129   /* Presión */
130   if(BSP_ENV_SENSOR_Init(LPS22HH_0, ENV_PRESSURE)==BSP_ERROR_NONE){
131     STBOX1_PRINTF("OK Pressure Sensor\n\r");
132   } else {
133     STBOX1_PRINTF("Error Pressure Sensor\n\r");
134     Error_Handler();
135   }
136 }
```

Listado 13: Función Init_MEM1_Sensors() del Sensortile.box

Tras estos cambios, también fue necesario modificar la función SendBatteryInfoData() del fichero main.c, mostrada en el Listado 14, de forma que además de obtener los valores relacionados con el estado de la batería del Sensortile.box, también lea los datos del sensor ambiental y finalmente llame a la función BatteryUpdate().

```
297 static void SendBatteryInfoData(void)
298 {
299   uint32_t BatteryLevel;
300   uint32_t Voltage;
301   int32_t PressToSend;
302   uint16_t HumToSend;
303   int16_t TempToSend;
304
305   ReadEnvironmentalData(&PressToSend,&HumToSend, &TempToSend, &Voltage, &BatteryLevel);
306   BSP_BC_GetVoltageAndLevel(&Voltage,&BatteryLevel);
307
308   Battery_Update(BatteryLevel,PressToSend,HumToSend,TempToSend);
309 }
```

Listado 14: Función SendBatteryInfoData() del Sensortile.box

Otra función que también es necesario modificar es BatteryUpdate() del fichero sensor_service.c encargada de crear el buffer, cargarlo con los datos relevantes y actualizarlo enviando una

notificación al Gateway. Inicialmente esta función enviaba un buffer de 9 bytes (4 short y 1 char), que se correspondían con el timestamp, el nivel de batería, el voltaje, la corriente y el estado de la batería. Tras los cambios realizados, en lugar de enviar un buffer de 9 bytes, se envía uno de 10 bytes que contiene un dato de tipo short, otro dato de tipo int y otros dos datos de tipo short, que se corresponden con el nivel de batería, la presión, la humedad y la temperatura del entorno respectivamente. En el Listado 15 se muestra el contenido de la función Battery_Update().

```
467 tBleStatus Battery_Update(uint32_t BatteryLevel, int32_t Press,uint16_t Hum,int16_t Temp)
468 {
469   tBleStatus ret;
470
471   uint8_t buff[2+4+2+2];
472
473   STORE_LE_16(buff  ,BatteryLevel*10);
474   STORE_LE_32(buff+2, Press);
475   STORE_LE_16(buff+6,Hum);
476   STORE_LE_16(buff+8,Temp); /* No info for Current */
477
478   ret = ACI_GATT_UPDATE_CHAR_VALUE(HWServW2STHandle, BatteryFeaturesCharHandle, 0, 2+2+2+2+2,buff);
479
480   if (ret != BLE_STATUS_SUCCESS){
481     if(W2ST_CHECK_CONNECTION(W2ST_CONNECT_STD_ERR)){
482       BytesToWrite = sprintf((char *)BufferToWrite, "Error Updating Bat Char\n");
483       Stderr_Update(BufferToWrite,BytesToWrite);
484     } else {
485       STBOX1_PRINTF("Error Updating Bat Char\r\n");
486     }
487   }
488   return ret;
489 }
```

Listado 15: Función Battery_Update() del Sensortile.box

Además, para actualizar el tamaño del buffer, también es necesario modificar la función Add_HW_SW_ServW2ST_Service() del fichero sensor_service.c, donde se debe indicar que el tamaño del buffer pasa a ser de 10 bytes.

Una vez realizados todos estos cambios, el firmware del Sensortile.box ya está listo para ser grabado en la memoria Flash del módulo multisensor. En cuanto a la recolección de los datos desde el Gateway, resulta mucho más simple que en los casos anteriores, pues basta con activar la notificación de la característica de batería y esperar la recepción de dicha notificación que, como ya se ha comentado, contiene la presión, humedad y temperatura del entorno además del nivel de batería. En el Listado 16, se muestra el proceso seguido para obtener los datos recolectados por el Sensortile.box.

```
28 def base(mac):
29     global pres, hum, temp, bat
30     while True:
31         print("\nConectando a " + mac)
32         p = btle.Peripheral(mac,"random")
33         print("Sensor conectado")
34         p.setDelegate(MyDelegate())
35         ReadData(p)
36         p.disconnect()
37         print("Sensor desconectado")
38         time.sleep(30.0)
```

Listado 16: Proceso para obtener los datos del Sensortile.box

Para comenzar, se inicializa la conexión mediante el uso de la función **btle.Peripheral("MAC","random")**. Tras ello, se asocia la clase MyDelegate(), cuyo contenido se muestra en el Listado 17, donde se puede apreciar que cuando el valor del cHandle es igual a 20, se trata de una notificación que contiene tanto la batería del dispositivo, como los datos capturados por el sensor ambiental.

```
17 class MyDelegate(btle.DefaultDelegate):
18     def handleNotification(self, cHandle, data):
19         global bat, pres, hum, temp
20         print(cHandle)
21         if cHandle == 20:
22             bat, pres, hum, temp = struct.unpack('<hihh',data)
23             bat/= 10.0
24             pres/=100.0
25             hum/=10.0
26             temp/=10.0
```

Listado 17: Clase MyDelegate() del Sensortile.box

Por último, se llama a la función ReadData(), mostrada en el Listado 18, y que es la encargada de activar y desactivar las notificaciones, así como de esperar a recibirlas (líneas 12, 13 y 14). El UUID del servicio del módulo multisensor es **00000000-0001-11e1-9ab4-0002a5d5c51b** y el de la característica relacionada con la batería es **00020000-0001-11e1-ac36-0002a5d5c51b**.

```
 8 def ReadData(p):
 9     global bat,pres, hum, temp
10     services=p.getServiceByUUID("00000000-0001-11e1-9ab4-0002a5d5c51b")
11     characteristicBat = services.getCharacteristics("00020000-0001-11e1-ac36-0002a5d5c51b")[0]
12     p.writeCharacteristic(characteristicBat.valHandle+1, _NOTIFICATION_ON)
13     p.waitForNotifications(100.0)
14     p.writeCharacteristic(characteristicBat.valHandle+1, _NOTIFICATION_OFF)
15     return
```

Listado 18: Función ReadData() del Sensortile.box

Para comprobar el correcto funcionamiento se realizó una prueba, cuyos resultados se muestran en la Ilustración 31, en la que se conecta al sensor y se obtienen y muestran en pantalla los datos relacionados con la presión, temperatura y humedad del entorno además de la batería del dispositivo.

```
pi@raspberrypi:~/TFM-Scripts/Individual $ python3 sensortilebox.py

Conectando a FF:7A:8C:1F:48:2A
Sensor conectado
Presion: 1000.23mBar - Temp: 22.0°C - Hum: 58.0% - Bat: 33.0%
Sensor desconectado

Conectando a FF:7A:8C:1F:48:2A
Sensor conectado
Presion: 1000.22mBar - Temp: 22.0°C - Hum: 58.0% - Bat: 32.0%
Sensor desconectado
```

Ilustración 31: Comprobación conectividad Sensortile.box – Gateway

5.4.4. BlueTile

En el módulo multisensor BlueTile se realizan las modificaciones en el firmware empleando el proyecto BLE_SensorDemo proporcionado por ST Microelectronics. El primer cambio que se hace es en el fichero sensor.c, concretamente en la función Sensor_Device_Init(), donde se elimina la inicialización de aquellos sensores que no se van a usar, dejando exclusivamente la inicialización de los sensores de temperatura, presión y humedad del entorno.

```
if (xFeaturePresence.PressurePresence)
      Init_Pressure_Temperature_Sensor();
if (xFeaturePresence.HumidityTemperaturePresence)
      Init_Humidity_Sensor();
```

También, es necesario eliminar de la función UserAppTick() los envíos de información relacionados con el sensor de movimiento y el micrófono. Además, de forma similar al Sensortile.box, los datos ambientales y los relativos a la batería van en diferentes notificaciones. Con el fin de reducir las comunicaciones de dos a una, se opta por realizar cambios en los datos enviados desde el sensor ambiental para que estos a su vez contengan la información relativa a la batería. En la Ilustración 32 se puede apreciar el formato seguido por los datos del sensor ambiental antes (a) y después (b) de las modificaciones.

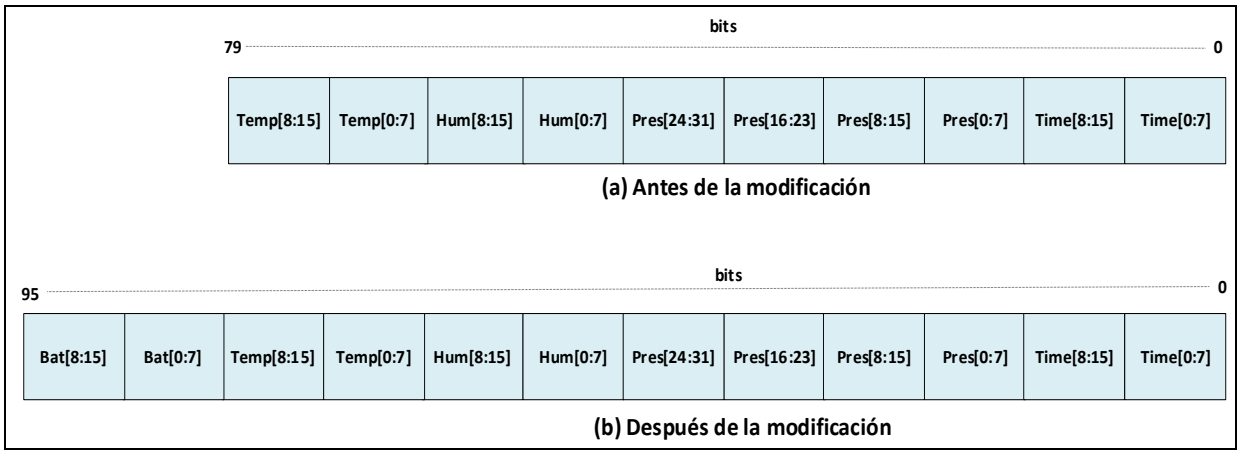

Ilustración 32: Formato de los datos antes (a) y después (b) de las modificaciones

Para lograrlo es necesario modificar en la función UserAppTick() la gestión de las notificaciones ambientales, tal y como se muestra en el Listado 19. En este caso, una vez finaliza el temporizador relacionado con el sensor ambiental, se lee la presión, temperatura y humedad del entorno, y finalmente también el estado de la batería. Todos estos datos son enviados mediante la función Environmental_Update(), que es la que se encarga de actualizar y mandar la notificación con los datos leídos.

```
if (xFeatureNotification.EnvironmentalNotification) {
    if (sensorTimer_expired) {
        sensorTimer_expired = false;
        HAL_VTimerStart_ms(SENSOR_TIMER, ENV_SENSOR_UPDATE_RATE);
        if (xFeaturePresence.PressurePresence) {
            memset(data_raw_pressure.u8bit, 0x00, sizeof(int32_t));
            lps22hh_pressure_raw_get(0, data_raw_pressure.u8bit);
            pressure = LPS22HH_FROM_LSB_TO_hPa(data_raw_pressure.i32bit);
            SensorValue = pressure;
            MCR_BLUEMS_F2I_2D(SensorValue, intPart, decPart);
            PressToSend = intPart * 100 + decPart;
        }
        if (xFeaturePresence.HumidityTemperaturePresence) {
            HTS221_Get_Measurement(0, &humidity, &temperature);
            HumToSend = humidity;
            SensorValue = ((float) temperature) / 10;
            MCR_BLUEMS_F2I_1D(SensorValue, intPart, decPart);
            TempToSend = intPart * 10 + decPart;
        }
        ADC_DMA_Configuration(ADC_Input_BattSensor);
        while (ADC_GetFlagStatus(ADC_FLAG_EOC) == RESET)
            ;
        batteryVoltage = batteryVoltage * 0.99 + ADC_ConvertBatterySensor(ADC_GetRawData(), ADC_ReferenceVoltage_0V6) * 0.01;
        Environmental_Update(PressToSend, HumToSend, TempToSend, (uint32_t) (batteryVoltage*1000));
    }
}
```

Listado 19: Función UserAppTick() de BlueTile

Al querer enviar también el nivel de batería a través de las notificaciones del sensor ambiental, es necesario realizar modificaciones en la función Environmental_Update() del fichero gatt_db.c.

Como se puede apreciar en el Listado 20, se añade el estado de la batería en la última posición del buffer enviado.

```
484 tBleStatus Environmental_Update(int32_t Press, uint16_t Hum, int16_t Temp, uint32_t voltage) {
485     uint32_t SoC = 0;
486     uint8_t BuffPos = 0;
487 
488     SoC = (voltage - 2000);
489     if (SoC > 1000)
490         SoC = 1000;
491 
492     STORE_LE_16(buff, (getTimestamp()));
493     BuffPos = 2;
494 
495     STORE_LE_32(buff + BuffPos, Press);
496     BuffPos += 4;
497 
498     STORE_LE_16(buff + BuffPos, Hum);
499     BuffPos += 2;
500 
501     STORE_LE_16(buff + BuffPos, Temp);
502     BuffPos += 2;
503 
504     STORE_LE_16(buff + BuffPos, SoC);
505 
506     return aci_gatt_update_char_value(HWServW2STHandle, EnvironmentalCharHandle, 0, EnvironmentalCharSize, buff);
507 }
```

Listado 20: Función Environmental_Update() de BlueTile

Tras todos estos cambios el firmware del BlueTile estaría casi listo para funcionar, pero este dispositivo está programado para entrar en modo reposo transcurrido un tiempo. Para evitarlo se debe modificar la función HAL_VTimerTimeoutCallback() del fichero sensor.c, dejando el contenido de dicha función al igual que se muestra en el Listado 21.

```
605 void HAL_VTimerTimeoutCallback(uint8_t timerNum) {
606 
607     switch (timerNum) {
608     case SENSOR_TIMER:
609         sensorTimer_expired = true;
610         break;
611     case RESET_TIMER:
612         SdkEvalLedOff(LED1);
613         SdkEvalLedOff(LED2);
614         SdkEvalLedOff(LED3);
615         aci_gap_set_non_discoverable();
616         NVIC_SystemReset();
617         break;
618     default:
619         break;
620     }
621 }
```

Listado 21: Función HAL_VTimerTimeoutCallback() de BlueTile

Ahora el firmware del BlueTile ya está listo para ser grabado en la memoria Flash del módulo multisensor. La forma de recolectar los datos desde el Gateway es muy similar a la empleada para recolectar los del Sensortile.box. La única diferencia existente es que en el caso del BlueTile la notificación que envía los datos ambientales y de batería, proviene del sensor ambiental, mientras que en el Sensortile.box, los datos ambientales y de batería provienen de las notificaciones de batería. El contenido del script empleado para capturar la información emitida por el BlueTile se muestra en el Listado 22.

```
from bluepy import btle
import struct
import time

_NOTIFICATION_ON = struct.pack("BB", 0x01, 0x00)
_NOTIFICATION_OFF = struct.pack("BB", 0x00, 0x00)

def ReadEnvironmental(p):
    services=p.getServiceByUUID("00000000-0001-11e1-9ab4--0002a5d5c51b")
    charEnv = services.getCharacteristics("001c0000--0001-11e1-ac36-0002a5d5c51b")[0]
    p.writeCharacteristic(charEnv.valHandle+1, _NOTIFICATION_ON)
    p.waitForNotifications(10.0)
    p.writeCharacteristic(charEnv.valHandle+1, _NOTIFICATION_OFF)

class MyDelegate(btle.DefaultDelegate):
    def handleNotification(self, cHandle, data):
        global pres, hum, temp, bat
        if (cHandle == 14):
            data = struct.unpack('<hihhh', data)
            pres = data[1]/100.0
            hum = data[2]/10.0
            temp = data[3]/10.0
            bat = data[4]/10.0

def base(mac):
    global pres, hum, temp, bat
    while True:
        print("\nConectando a " + mac)
        p = btle.Peripheral(mac,"random")
        print("Sensor conectado")
        p.setDelegate(MyDelegate())
        ReadEnvironmental(p)
        time.sleep(5)
        print("Presion: " + str(pres) + "mBar - Temp: " + str(temp) + "°C - Hum: "
                + str(hum) + "% - Bat: " + str(bat) +"%")
        p.disconnect()
        print("Sensor desconectado")
        time.sleep(60)

if __name__=='__main__':
    base("F2:80:09:7F:AC:09")
```

Listado 22: Script de captura BlueTile

Este script es muy similar al empleado con el Sensortile.box, variando la función ReadEnvironmental(), que en lo único que difiere con la función ReadData() del Sensortile.box

es en el UUID de la característica a la que accede, en este caso, el UUID correspondiente al sensor ambiental es **001c0000-0001-11e1-ac36-0002ª5d5c51b**. Otra diferencia que existe con respecto al script empleado para capturar los datos del Sensortile.box es la clase MyDelegate(). En este caso, el cHandle que interesa es aquel igual a 14, que se corresponde con el sensor ambiental.

Para comprobar el correcto funcionamiento se realizó una prueba, cuyos resultados se muestran en la Ilustración 33, en la que se conecta al BlueTile y se leen y muestran por pantalla los datos relacionados con la presión, temperatura y humedad del entorno además de la batería del dispositivo.

```
pi@raspberrypi:~/TFM-Scripts/Individual $ python3 bluetile.py

Conectando a F2:80:09:7F:AC:09
Sensor conectado
Presion: 1008.17mBar - Temp: 17.6ºC - Hum: 59.2% - Bat: 77.3%
Sensor desconectado

Conectando a F2:80:09:7F:AC:09
Sensor conectado
Presion: 1008.17mBar - Temp: 17.5ºC - Hum: 65.5% - Bat: 77.3%
Sensor desconectado

Conectando a F2:80:09:7F:AC:09
Sensor conectado
Presion: 1008.16mBar - Temp: 17.7ºC - Hum: 58.5% - Bat: 77.3%
Sensor desconectado
```

Ilustración 33: Comprobación conectividad BlueTile – Gateway

5.5. Configuración de la nube

La nube que se va a emplear en este trabajo es ThingsBoard. Ofrece dos modos de instalación, en la nube o en local (on-premise). Ya que el prototipo va a ser desplegado en el edificio departamental de informática de la Escuela Politécnica de Ingeniería, se ha optado por la instalación en local en una máquina virtual Ubuntu de un servidor de la Universidad de Oviedo, cuya dirección IP es 156.35.163.174.

Para llevar a cabo su instalación se siguen los pasos indicados en el manual de usuario [56], los cuales se resumen a continuación.

Lo primero es necesario instalar OpenJDK 8, ya que Thingsboard funciona empleando Java 8. Después se configura el sistema operativo para que este emplee OpenJDK8.

```
sudo apt install openjdk-8-jdk
sudo update-alternatives –config java
```

Tras ello, se procede a descargar el paquete de instalación de thingsboard-3.1.deb. Se opta por la versión 3.1 ya que la versión 3.2 presenta varios problemas de instalación. Posteriormente se instalará Thingsboard como servicio.

```
wget https://github.com/thingsboard/thingsboard/releases/download/v3.1/thingsboard-3.1.deb
sudo dpkg -i thingsboard-3.1.deb
```

Como base de datos a emplear, se selecciona PostgreSQL, que tras instalarse se modifica la contraseña, cambiándola por la cadena “123456789”. Además, también es necesario crear una base de datos de nombre ThingsBoard, donde se almacene toda la información recopilada por los módulos multisensor.

Para asociar la base de datos PostgreSQL con la plataforma ThingsBoard, es necesario modificar el contenido del fichero /etc/thingsboard/conf/thingsboard.conf para que incluya las líneas de código mostradas en el Listado 23. En ellas se puede apreciar entre otros datos, la dirección IP asociada a la base de datos PostgreSQL, el nombre de usuario empleado y la contraseña necesaria para acceder a ella.

```
27 # DB Configuration
28 export DATABASE_ENTITIES_TYPE=sql
29 export DATABASE_TS_TYPE=sql
30 export SPRING_JPA_DATABASE_PLATFORM=org.hibernate.dialect.PostgreSQLDialect
31 export SPRING_DRIVER_CLASS_NAME=org.postgresql.Driver
32 export SPRING_DATASOURCE_URL=jdbc:postgresql://localhost:5432/thingsboard
33 export SPRING_DATASOURCE_USERNAME=postgres
34 export SPRING_DATASOURCE_PASSWORD=123456789
35 export SPRING_DATASOURCE_MAXIMUM_POOL_SIZE=5
36
37 # Specify partitioning size for timestamp key-value storage.
38 export SQL_POSTGRES_TS_KV_PARTITIONING=MONTHS
```

Listado 23: Código a añadir en el fichero thingsboard.conf

Con todo esto ya listo, lo único que falta es la instalación del servicio ThingsBoard y la puesta en marcha de este.

```
sudo /usr/share/thingsboard/bin/install/install.sh --loadDemo
sudo service thingsboard start
```

Para comprobar que todo se ha realizado correctamente, se accede a la dirección IP 156.35.163.174:8080 y se comprueba como sale la página de inicio de la plataforma ThingsBoard, mostrada en la Ilustración 34.

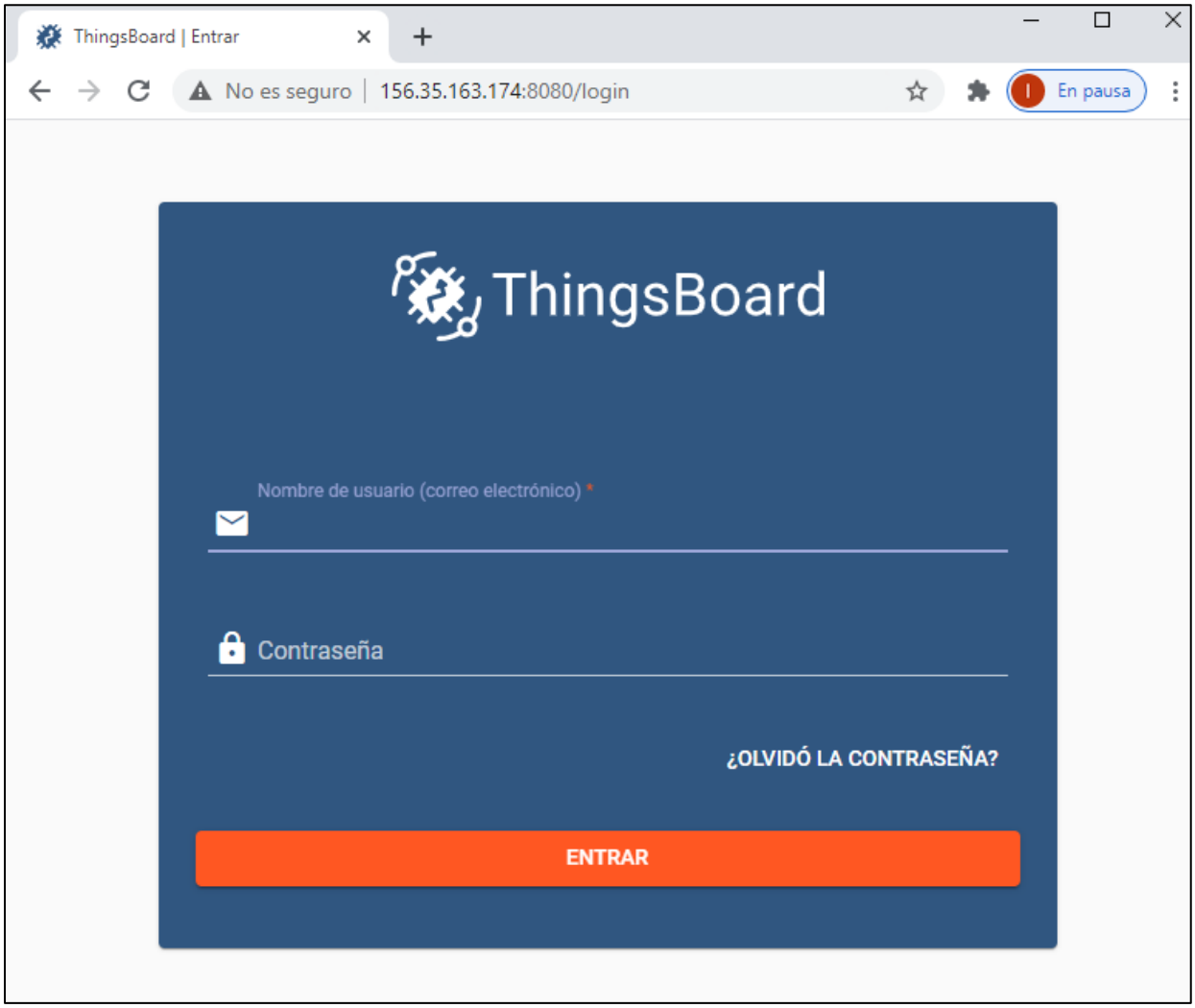

Ilustración 34: Página de inicio de ThingsBoard

A modo de ejemplo y para demostrar su correcto funcionamiento, se inicia sesión con el usuario sysadmin@thingsboard.org, y se crea un nuevo propietario correspondiente al Edificio Departamental de Informática de la Escuela Politécnica de Ingeniería, mostrado en la Ilustración 35.

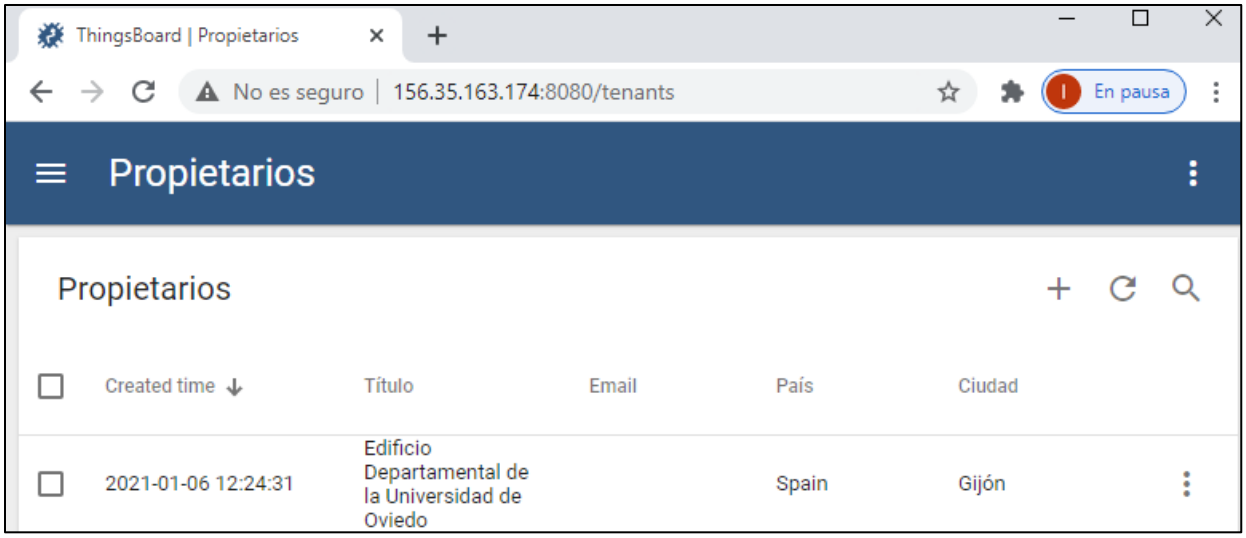

Ilustración 35: Definición del propietario en ThingsBoard

A este propietario se le asigna un usuario para que tenga acceso. Para ello, se utiliza la cuenta de Gmail departamentoinformaticauniovi@gmail.com. Una vez asignado el usuario (ver Ilustración 36), se cierra sesión y se inicia de nuevo con ese usuario, con el que se crean los activos correspondientes, en este caso, el Módulo 1 del Edificio Departamental Oeste.

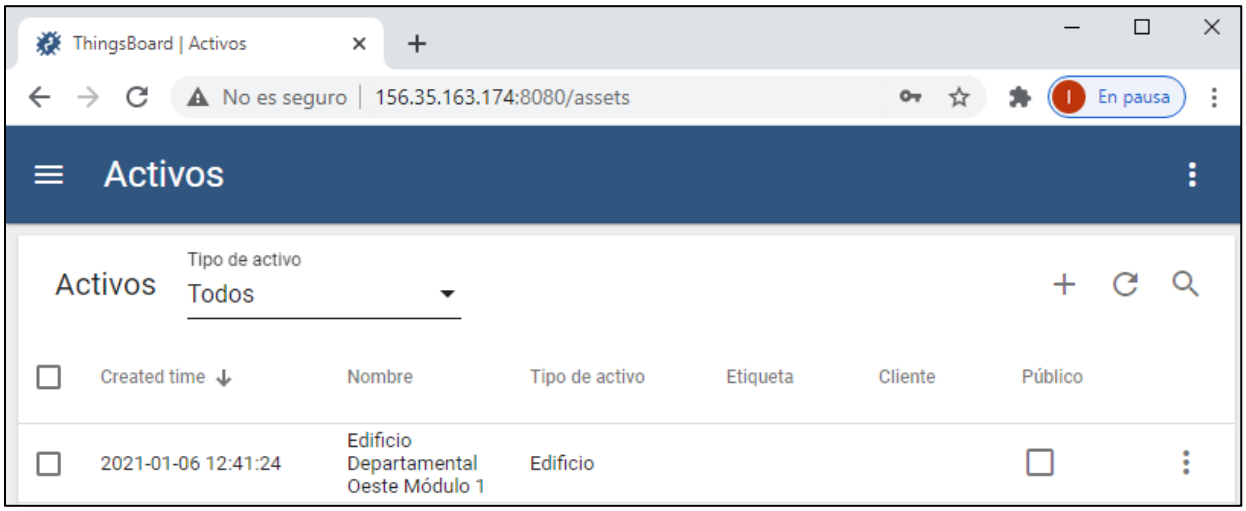

Ilustración 36: Activo Edificio Departamental Oeste ThingsBoard

También hay que crear los diferentes dispositivos empleados, en el caso de este trabajo esos dispositivos serán los módulos multisensor. Para realizar la prueba de comprobación de funcionamiento, se crea el dispositivo Sensortag CC2650 – Lab y se le asocia con el activo creado

previamente, ya que este módulo multisensor se encuentra ubicado en el Departamento Oeste Módulo 1, como se muestra en la Ilustración 37.

Ilustración 37: Relaciones entre Activo y Dispositivo en ThingsBoard

Asociar dispositivos con activos sirve para que cuando se creen diferentes clientes con acceso a la información relacionada con todos los dispositivos del Departamento Oeste Módulo 1, no sea necesario ir asignando cada dispositivo uno a uno, asignando únicamente el activo relacionado con el Edificio en el que se ubican. De esta forma, si en algún momento se amplía el prototipo a todo el edificio departamental, se podría diferenciar entre módulos, dando acceso exclusivo a los responsables de cada módulo.

Para llevar a cabo la comunicación entre el Gateway o Repetidor y ThingsBoard, se empleará un cliente MQTT mediante el bróker de mensajes Mosquitto. Para ello, se desarrolla un script Python de nombre SendToThingsboard.sh que permite publicar los datos recolectados por el módulo multisensor en ThingsBoard y que recibe como parámetros tanto el AccessToken asociado al dispositivo en la plataforma ThingsBoard como un archivo JSON con el formato mostrado en la Ilustración 38.

```
{
  "timestamp": 301234,
  "temperatura": 22.3,
  "humedad": 68.5,
  "presion": 1015.08,
  "luz": 567.22,
  "bat": 75.6
}
```

Ilustración 38: Formato JSON del mensaje a enviar

El contenido del script SendToThingsboard.sh se muestra en el Listado 24.

```
#!/bin/sh

if test $# -ne 2; then
        echo 'SendToThingsboard.sh <accessToken> <datatosend>'
        exit 1
fi

IP_Thingsboard="156.35.163.174"
mosquitto_pub -d -h "$IP_Thingsboard" -t "v1/devices/me/telemetry" -u "$1" -m "$2" > /dev/null
exit 0
```

Listado 24: Script SendToThingsboard.sh

Este script será lanzado cada vez que se reciban los datos provenientes del módulo multisensor. Para este ejemplo, se ha modificado el script de recolección de datos provenientes del Sensortag CC2650, de forma que una vez se reciban todos ellos, se cree un JSON que los contenga, que será pasado como parámetros al script SendToThingsboard.sh junto con el AccessToken correspondiente al módulo multisensor, el cual se obtiene a través de la plataforma de ThingsBoard y es único para cada dispositivo. Además, una ventaja que presenta ThingsBoard es que no es necesario configurar los datos que va a recibir por dispositivos, ya que estos se configuran automáticamente tras recibir un archivo JSON. En la Ilustración 39 se observa el correcto funcionamiento de lo realizado, donde se ejecuta y se comprueba que la información del dispositivo en ThingsBoard se actualiza con los datos capturados por el módulo multisensor.

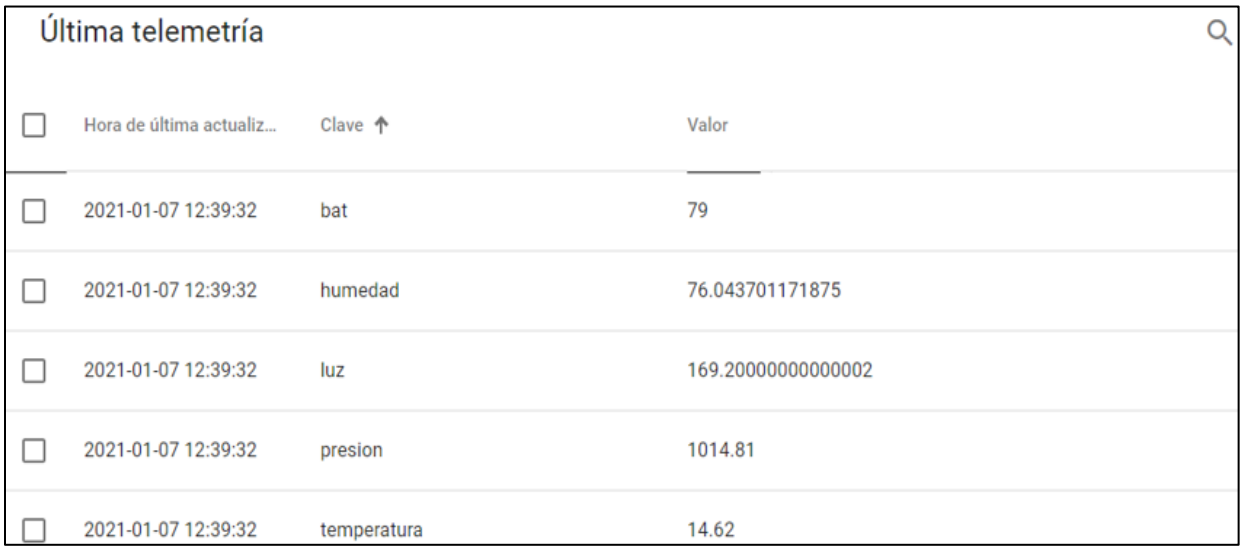

Ilustración 39: Datos almacenados del dispositivo Sensortag CC2650 en ThingsBoard

5.6. Análisis de la comunicación entre Gateway y Repetidores

En el presente apartado se realiza un estudio de la red mallada empleando el protocolo B.A.T.M.A.N. como protocolo de enrutamiento entre los diferentes elementos que conforman la red. Aquellos datos que resultan de gran interés en este tipo de redes son el indicador de fuerza de la señal recibida (RSSI) entre los dispositivos, un estudio del throughput alcanzado por los dispositivos de la red llevando un análisis en función del número de saltos realizado entre comunicaciones, y finalmente un análisis de la latencia y el jitter.

Para analizar el RSSI entre Gateways o Repetidores, se emplea el comando **sudo iwlist scan**, que retorna todas las posibles redes existentes con múltiple información, donde se encuentra el RSSI entre ambos dispositivos. Para lograrlo, se ha optado por variar la distancia entre dispositivos y realizar seis medidas para cada distancia, estando estas separadas por un periodo de 5 minutos.

Para realizar el análisis del throughput la herramienta batctl proporciona una prueba de rendimiento, en la cual se envían 14 MB de datos. Esta prueba tiene el objetivo de evaluar el rendimiento obtenido en función del número de saltos realizados, pudiendo ser estos de uno, dos o tres saltos. Se ejecutará 30 veces por ronda y se calculará la media para cada ronda. Además, se realizarán 6 rondas separadas entre sí por un periodo de 5 minutos.

En cuanto al estudio de la latencia y del jitter, la herramienta batctl proporciona un comando de ping que permite utilizar la dirección MAC de los dispositivos en lugar de su dirección IP. Para analizar estos parámetros, se realizarán 30 envíos de paquetes por ronda y se calculará la media de los tiempos obtenidos para cada una de las rondas. Al igual que en el caso anterior, se llevarán a cabo 6 rondas separadas entre sí por un periodo de 5 minutos.

5.6.1. Análisis de la distancia máxima entre dispositivos basado en el RSSI

En este primer experimento, se comprueba cuál es la distancia máxima entre dos dispositivos de la red mesh. Para ello se prueban diferentes distancias entre dos dispositivos y se comprueba cual es el RSSI existente entre ellos, estimando de esta forma cuál es la distancia óptima en base a dicho valor.

En la Tabla 6, se recogen los diferentes valores de la intensidad de la señal, además del estado de la red a la que están asociados y lo que conlleva ese nivel RSSI con respecto a los otros. En base a lo recogido en esta tabla, cualquier valor cercano o superior a -70 dBm es adecuado para ser empleado en el ámbito IoT.

Intensidad de la señal	Estado	
-30 dBm	Genial	Señal máxima posible, únicamente se logra al estar situado a pocos metros del punto de acceso.
-67 dBm	Muy bueno	Mínima intensidad de señal necesaria para aplicaciones que requieren de una entrega y recepción de paquetes muy fiable.
-70 dBm	Bueno	Mínima intensidad de señal para el envío y recepción de paquetes de forma fiable.
-80 dBm	Malo	Mínima intensidad de señal para llevar a cabo una conectividad básica. El envío de paquetes es algo no fiable.
-90 dBm	Inoperable	Improbable llevar a cabo ninguna funcionalidad

Tabla 6: Estado de la conexión según el RSSI

A continuación, se recogen los resultados obtenidos tras llevar a cabo este experimento.

	Prueba RSSI entre Gateway y Repetidor						
Distancia (m)	**N.º 1**	**N.º 2**	**N.º 3**	**N.º 4**	**N.º 5**	**N.º 6**	**Media**
2	-51 dBm	-44 dBm	-50 dBm	-50 dBm	-49 dBm	-47 dBm	-48.50 dBm
5	-47 dBm	-48 dBm	-51 dBm	-53 dBm	-51 dBm	-53 dBm	-50.50 dBm
8	-63 dBm	-61 dBm	-65 dBm	-63 dBm	-64 dBm	-60 dBm	-62,67 dBm
11	-65 dBm	-68 dBm	-66 dBm	-66dBm	-64 dBm	-66 dBm	-65.83 dBm
14	-73 dBm	-70 dBm	-76 dBm	-75 dBm	-65 dBm	-68 dBm	-71.17 dBm
17	-78 dBm	-79 dBm	-77 dBm	-79 dBm	-82 dBm	-81 dBm	-79.33 dBm
20	-82 dBm	-83 dBm	-82 dBm	-80 dBm	-83 dBm	-78 dBm	-81.33 dBm
23	-84 dBm	-85 dBm	-77 dBm	-85 dBm	-84 dBm	-81 dBm	-82.67 dBm
26	-82 dBm	-77 dBm	-85 dBm	-86 dBm	-87 dBm	-87 dBm	-84 dBm

Tabla 7: Análisis del RSSI entre Gateway y Repetidor

En base a los resultados obtenidos en la Tabla 7 y mostrados en la Ilustración 40, se puede apreciar que hasta en torno a 17 metros los valores del RSSI se encuentran por encima de los -80 dBm, aunque concretamente, en la prueba número 5 y número 6 a una distancia de 17 metros entre

ambos dispositivos el RSSI se encuentra por debajo de los -80 dBm. Por tanto, durante este trabajo se ha optado por establecer una distancia máxima entre Gateways o Repetidores de 14 metros, distancia a la que el RSSI medio obtenido tras varias pruebas se encuentra en -71.16 dBm, valor adecuado para ser empleado en el ámbito IoT.

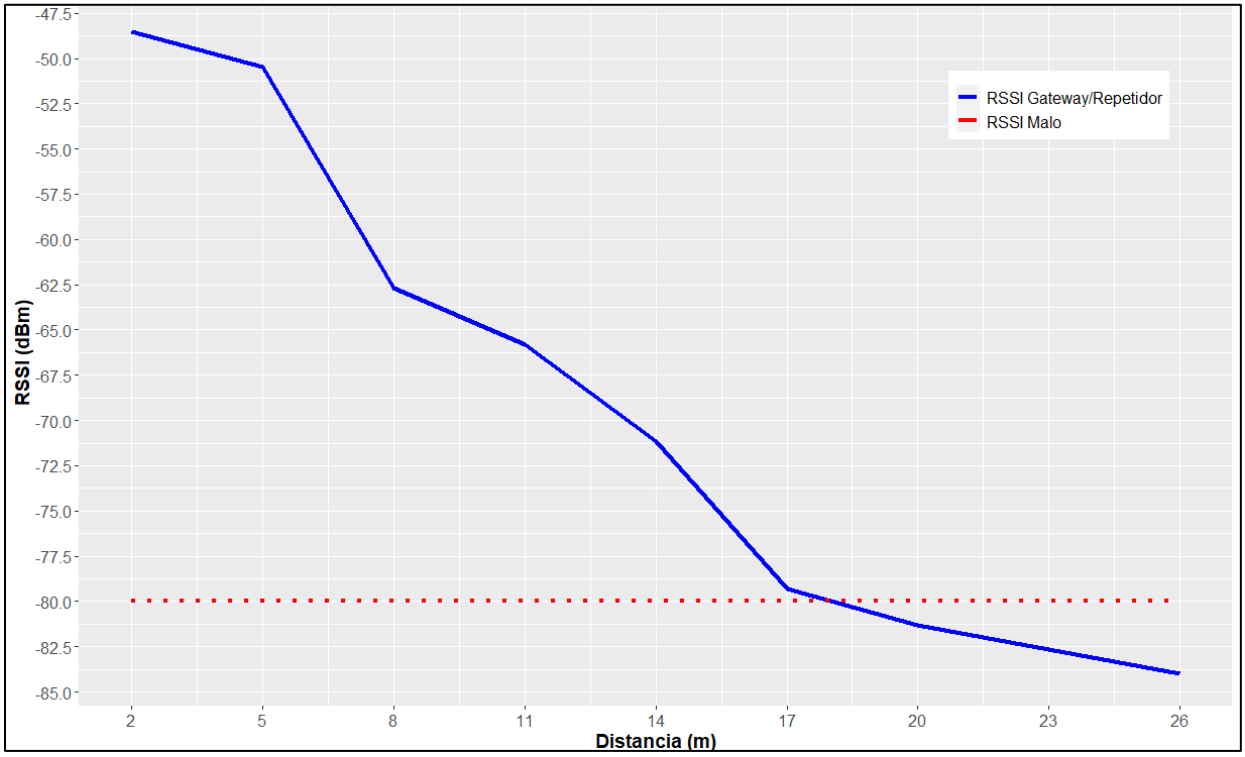

Ilustración 40: Evolución del RSSI del Gateway/Repetidor según la distancia

5.6.2. Análisis del throughput, latencia y jitter

Tras estimar la distancia optima entre los dispositivos de la red, se realizarán varios estudios en los que se analizan tanto el throughput, la latencia y el jitter.

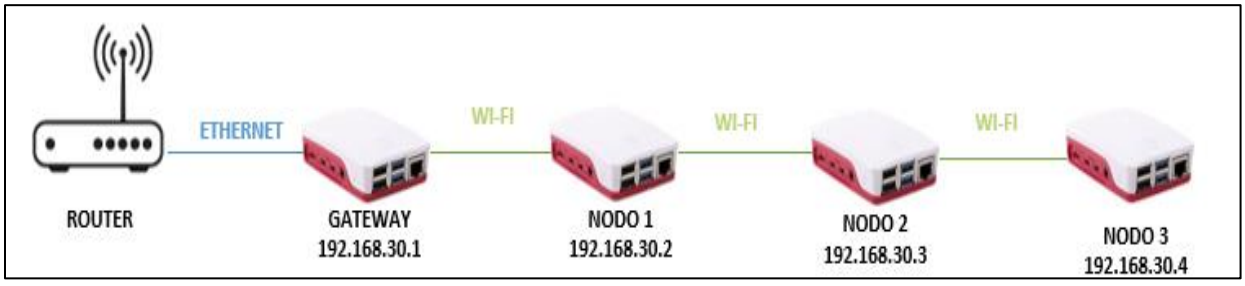

Ilustración 41: Escenario 1

Como se puede apreciar en la Ilustración 41, el escenario empleado para estas pruebas está formado por cuatro dispositivos interconectados en forma de cadena, es decir, el Gateway se encuentra conectado vía Ethernet con el Router y a través de WiFi (empleando el protocolo B.A.T.M.A.N.) con el nodo 1, a su vez, el nodo 1 se encuentra conectado con el nodo 2 vía WiFi (empleando el protocolo B.A.T.M.A.N.) y el nodo 2 se encuentra conectado con el nodo 3 también a través de WiFi (empleando el protocolo B.A.T.M.A.N.). Todos ellos se encuentran a una distancia de 14 metros entre sí.

En primer lugar, se realizará un análisis del rendimiento de la red para uno, dos y tres saltos analizando, por un lado, la comunicación entre el nodo 1 y el Gateway (un salto), por otro lado, la comunicación entre el nodo 2 y el Gateway (dos saltos) y, finalmente, la comunicación entre el nodo 3 y el Gateway (tres saltos).

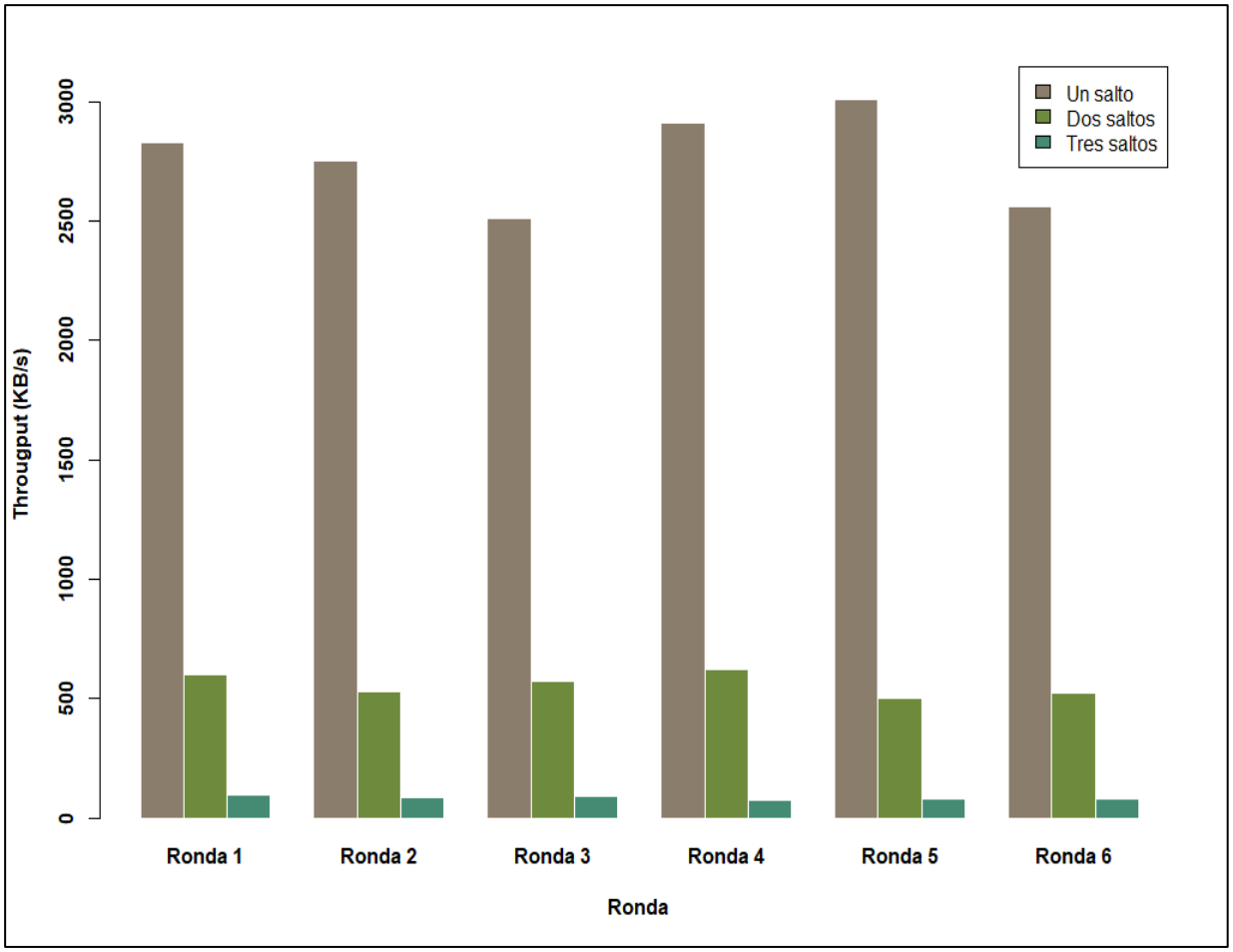

Ilustración 42: Evolución del rendimiento según el número de saltos

	Ronda 1	Ronda 2	Ronda 3	Ronda 4	Ronda 5	Ronda 6	Media
Un salto	2831.41	2752.92	2512.70	2913.18	3008.33	2559.30	2762.97
Dos saltos	599.73	526.30	574.21	620.02	502.48	522.18	557.49
Tres saltos	98.55	83.95	93.20	76.53	78.48	81.82	85.42

Tabla 8: Análisis del throughput según el número de saltos

En la Tabla 8 y en la Ilustración 42, se muestran los resultados recolectados tras realizar los experimentos de rendimiento, mediante la ejecución del comando **sudo batctl tp MAC**, comando proporcionado por la herramienta batctl que ejecuta un test en el que se envían 14 MB de datos entre dos nodos. Cabe destacar que todos ellos se encuentran en B/s. Como se puede apreciar, a medida que se incrementan el número de saltos, el rendimiento también se ve reducido en gran medida. Para el caso de un salto, el throughput se encuentra en el rango comprendido entre 2512.70 y 3008.33 kB/s, en el caso de dos saltos, entre 522.18 y 620.02 kB/s y en el caso de tres saltos, comprendido entre 76.53 y 98.55 kB/s. En el prototipo demostrador realizado para monitorizar el Departamento de Informática, la cantidad de bytes enviados es mínima, ya que se pretende enviar los datos recolectados por los Repetidores/Gateways de los módulos multisensor a la nube cada seis minutos.

Para estimar el número de bytes que supone cada uno de estos paquetes a enviar, se usó el comando **sudo batctl tcpdump wlan0**, comando proporcionado por la herramienta batctl que permite ver todos los paquetes enviados y recibidos a través de la interfaz indicada, así como el tamaño de estos.

```
16:09:21.421265 BAT dc:a6:32:60:1c:ca > dc:a6:32:60:1c:bd: UCAST, ttvn 3, ttl 50
, IP 192.168.30.2.22 > 192.168.30.1.40676: TCP, Flags [...PA.], length 108
```

Ilustración 43: Tamaño del paquete enviado

Los resultados obtenidos tras ejecutar este comando se muestran en la Ilustración 43 y determinan que el tamaño de cada uno de los paquetes enviados cada seis minutos, independientemente del tipo de módulo multisensor, tienen un tamaño de 108 bytes. Suponiendo que la red desplegada constase de 5 Repetidores y un Gateway, que cada Repetidor/Gateway gestionase un total de 10 módulos multisensor y que todos los paquetes se enviasen simultáneamente, el tráfico de la red en ese instante, debido a comunicaciones a la nube, sería de:

$$(1\ Gateway\ + 5\ Repetidores)\ x\ (10\ módulos\ multisensores)\ x\ 108\ bytes = 6480\ bytes$$

En estas condiciones el rendimiento de la red cuando existen tres saltos se encuentra comprendido entre 76.53 kB/s y 98.55 kB/s, por lo que sería más que suficiente para gestionar los 6480 bytes correspondientes a los datos enviados a la nube, sin suponer esto ningún problema para su correcto funcionamiento.

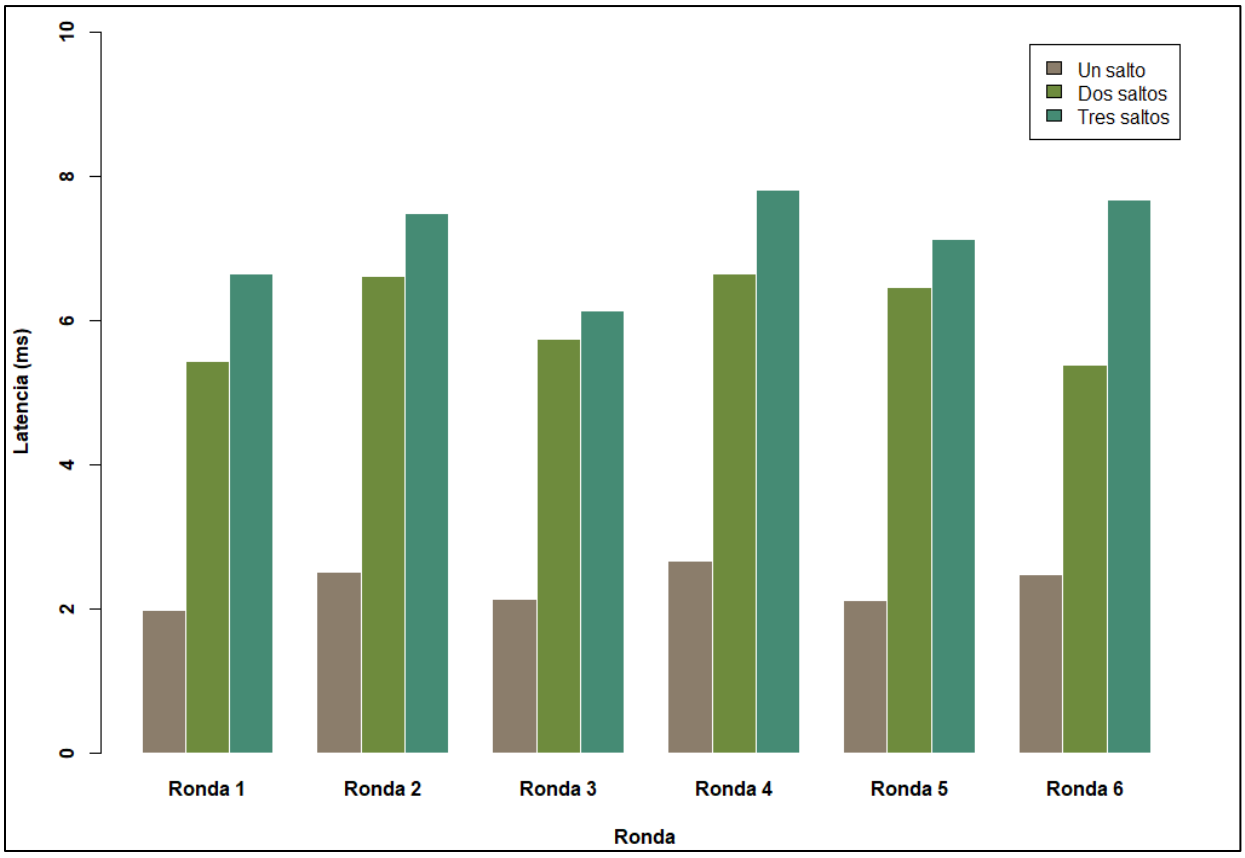

Ilustración 44: Evolución de la latencia según el número de saltos

	Ronda 1	Ronda 2	Ronda 3	Ronda 4	Ronda 5	Ronda 6	Media
Un salto	1.98 ms	2.50 ms	2.13 ms	2.67 ms	2.11 ms	2.47 ms	2.31 ms
Dos saltos	5.43 ms	6.61 ms	5.75 ms	6.65 ms	6.46 ms	5.39 ms	6.05 ms
Tres saltos	6.65 ms	7.48 ms	6.14 ms	7.82 ms	7.13 ms	7.67 ms	7.15 ms

Tabla 9: Análisis de la latencia según el número de saltos

Como se puede ver en la Tabla 9 y en la Ilustración 44, la latencia se ve incrementada según aumenta el número de saltos, estando comprendida entre 1.98 y 2.67 ms cuando solo hay un salto entre dispositivos, entre 5.39 y 6.65 ms cuando hay dos saltos y, finalmente, entre 6.14 ms y 7.82

ms en el caso de tres saltos. En todos los casos la latencia se encuentra por debajo de los 10 ms, valores apropiados para su uso en el ámbito IoT.

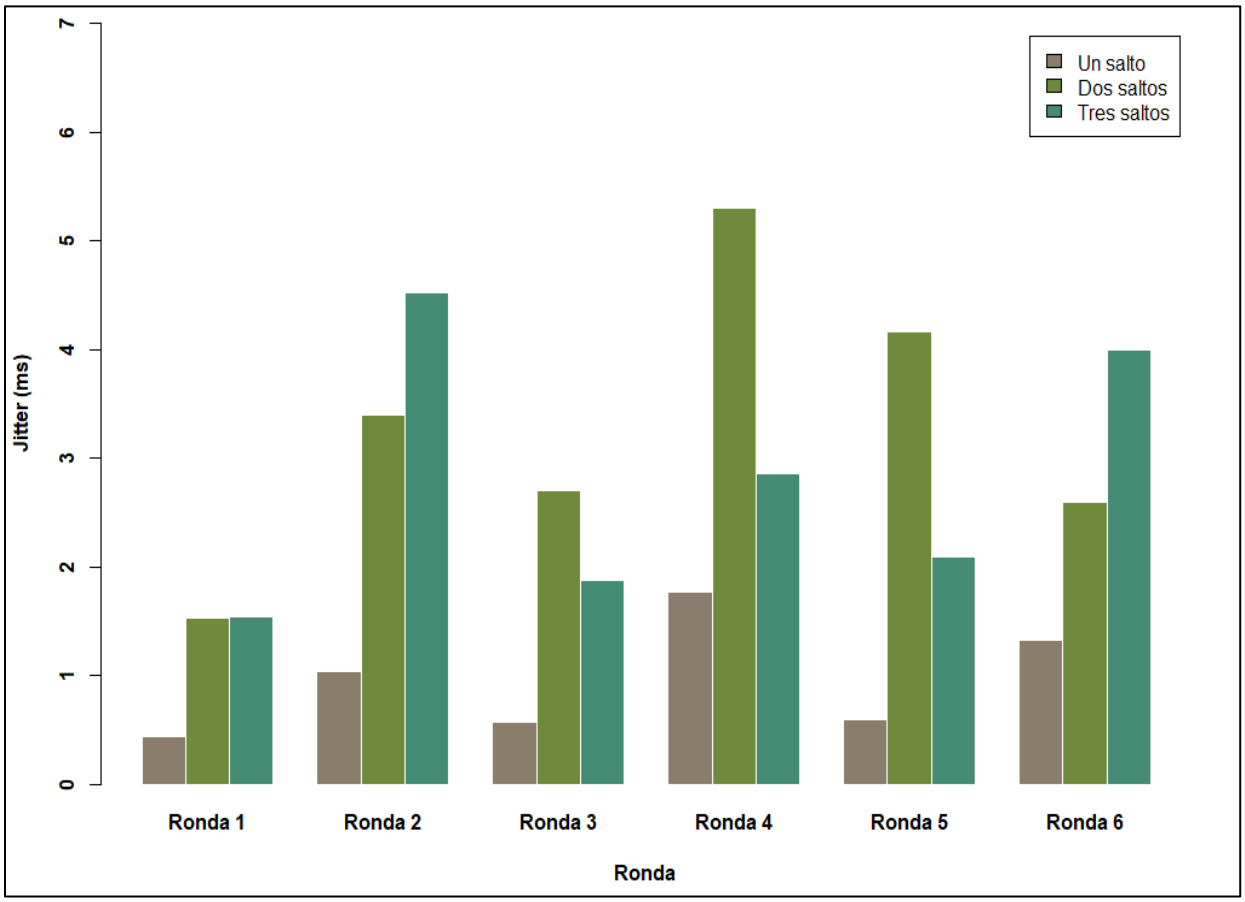

Ilustración 45: Evolución del jitter según el número de saltos

	Ronda 1	Ronda 2	Ronda 3	Ronda 4	Ronda 5	Ronda 6	Media
Un salto	0.44 ms	1.04 ms	0.57 ms	1.77 ms	0.59 ms	1.32 ms	0.95 ms
Dos saltos	1.53 ms	3.40 ms	2.70 ms	5.30 ms	4.16 ms	2.59 ms	3.28 ms
Tres saltos	1.54 ms	4.52 ms	1.87 ms	2.86 ms	2.09 ms	3.99 ms	2.81 ms

Tabla 10: Análisis del jitter según el número de saltos

La evolución del jitter es algo que, a diferencia de la latencia, no se ve incrementado según aumenta el número de saltos. En la Tabla 10 y la Ilustración 45 puede apreciarse como el jitter fluctúa con los saltos y rondas, sin presentar una tendencia creciente con el número de saltos. Independientemente de esto, para todos los saltos en todas las rondas realizadas el jitter se encuentra por debajo de los 6 ms, por lo que no supondría un gran problema a la hora de llevar a cabo las comunicaciones en la red.

5.7. Análisis de la distancia máxima entre Gateway/Repetidor y módulos multisensor

De forma análoga a lo realizado en el apartado 5.6. se analiza cuál es la distancia máxima a la que deben encontrarse los módulos multisensor con respecto al Gateway o Repetidor que los gestiona. Para llevar a cabo este estudio, se desarrolla un script Python que se conecta a un módulo multisensor y retorna el valor del RSSI correspondiente.

Para cada uno de los módulos multisensor se estudia la fluctuación del RSSI según varía la distancia, realizando seis pruebas y posteriormente calculando la media en cada una de esas distancias. Todos los resultados obtenidos tras realizar estas pruebas se recogen en la Tabla 11, Tabla 12, Tabla 13 y Tabla 14.

	Prueba RSSI entre Gateway y Sensortag CC2650						
Distancia (m)	**N.º 1**	**N.º 2**	**N.º 3**	**N.º 4**	**N.º 5**	**N.º 6**	**Media**
1	-52 dBm	-52 dBm	-52 dBm	-56 dBm	-54 dBm	-52 dBm	-53 dBm
2	-56 dBm	-56 dBm	-56 dBm	-56 dBm	-56 dBm	-57 dBm	-56.17 dBm
3	-60 dBm	-56 dBm	-60 dBm	-59 dBm	-59 dBm	-60 dBm	-59 dBm
4	-57 dBm	-67 dBm	-67 dBm	-57 dBm	-67 dBm	-57 dBm	-62 dBm
5	-63 dBm	-70 dBm	-70 dBm	-63 dBm	-63 dBm	-72 dBm	-66.83 dBm
6	-63 dBm	-66 dBm	-66 dBm	-66 dBm	-66 dBm	-73 dBm	-66.67 dBm
7	-68 dBm	-68 dBm	-68 dBm	-69 dBm	-71 dBm	-69 dBm	-68.83 dBm
8	-71 dBm	-65 dBm	-72 dBm	-73 dBm	-73 dBm	-72 dBm	-71 dBm
9	-73 dBm	-68 dBm	-70 dBm	-82 dBm	-77 dBm	-76 dBm	-74.33 dBm

Tabla 11: Análisis del RSSI entre Gateway y Sensortag CC2650

	Prueba RSSI entre Gateway y Smartbond DA14585						
Distancia (m)	**N.º 1**	**N.º 2**	**N.º 3**	**N.º 4**	**N.º 5**	**N.º 6**	**Media**
1	-55 dBm	-51 dBm	-51 dBm	-56 dBm	-48 dBm	-56 dBm	-52.83 dBm
2	-60 dBm	-55 dBm	-60 dBm	-60 dBm	-57 dBm	-60 dBm	-58.67 dBm
3	-62 dBm	-59 dBm	-62 dBm	-58 dBm	-58 dBm	-61 dBm	-60 dBm
4	-65 dBm	-67 dBm	-66 dBm	-62 dBm	-65 dBm	-66 dBm	-65.17 dBm
5	-70 dBm	-71 dBm	-65 dBm	-71 dBm	-66 dBm	-65 dBm	-68 dBm
6	-65 dBm	-72 dBm	-71 dBm	-71 dBm	-65 dBm	-65 dBm	-68.17 dBm
7	-69 dBm	-69 dBm	-73 dBm	-74 dBm	-69 dBm	-67 dBm	-70.17 dBm
8	-73 dBm	-75 dBm	-73 dBm	-73 dBm	-76 dBm	-71 dBm	-73.5 dBm
9	-79 dBm	-78 dBm	-81 dBm	-83 dBm	-78 dBm	-80 dBm	-79.83 dBm

Tabla 12: Análisis del RSSI entre Gateway y Smartbond DA14585

	Prueba RSSI entre Gateway y Sensortile.box						
Distancia (m)	**N.º 1**	**N.º 2**	**N.º 3**	**N.º 4**	**N.º 5**	**N.º 6**	**Media**
1	-57 dBm	-56 dBm	-56 dBm	-56 dBm	-55 dBm	-55 dBm	-55.83 dBm
2	-59 dBm	-62 dBm	-61 dBm	-59 dBm	-58 dBm	-58 dBm	-59.5 dBm
3	-61 dBm	-60 dBm	-59 dBm	-61 dBm	-60 dBm	-60 dBm	-60.17 dBm
4	-61 dBm	-61 dBm	-63 dBm	-61 dBm	-60 dBm	-62 dBm	-61.33 dBm
5	-62 dBm	-62 dBm	-63dBm	-62 dBm	-62 dBm	-63dBm	- 62.33 dBm
6	-66 dBm	-66 dBm	-65 dBm	-67 dBm	-65 dBm	-66 dBm	-65.83 dBm
7	-68 dBm	-67 dBm	-66 dBm	-67 dBm	-68 dBm	-70 dBm	-67.67 dBm
8	-70 dBm	-71 dBm	-71 dBm	-71 dBm	-72 dBm	-71 dBm	-71 dBm
9	-78 dBm	-78 dBm	-75 dBm	-74 dBm	-76 dBm	-74 dBm	-75.83 dBm

Tabla 13: Análisis del RSSI entre Gateway y Sensortile.box

	Prueba RSSI entre Gateway y BlueTile						
Distancia (m)	**N.º 1**	**N.º 2**	**N.º 3**	**N.º 4**	**N.º 5**	**N.º 6**	**Media**
1	-57 dBm	-55 dBm	-60 dBm	-55 dBm	-55 dBm	-55 dBm	-56.17 dBm
2	-62 dBm	-59 dBm	-62 dBm	-62 dBm	-59 dBm	-59 dBm	-60.5 dBm
3	-61 dBm	-63 dBm	-61 dBm	-61 dBm	-65 dBm	-65 dBm	-62.67 dBm
4	-64 dBm	-64 dBm	-62 dBm	-65 dBm	-65 dBm	-64 dBm	-64 dBm
5	-64 dBm	-64 dBm	-66 dBm	-67 dBm	-67 dBm	-64 dBm	-65.33 dBm
6	-67 dBm	-67 dBm	-67 dBm	-74 dBm	-66 dBm	-75 dBm	-69.33 dBm
7	-73 dBm	-71 dBm	-72 dBm	-73 dBm	-71 dBm	-71 dBm	-71.83 dBm
8	-78 dBm	-78 dBm	-78 dBm	-77 dBm	-77 dBm	-75 dBm	-77.17 dBm
9	-81 dBm	-74 dBm	-81 dBm	-76 dBm	-81 dBm	-81 dBm	-79 dBm

Tabla 14: Análisis del RSSI entre Gateway y BlueTile

En base a los resultados obtenidos al realizar las pruebas, mostrados en la Ilustración 46, se puede apreciar como el sensor que mayor alcance ofrece con un nivel RSSI mayor es el Sensortag CC2650, el cuál situándose a una distancia de 9 metros del Gateway tiene un RSSI de -74.34 dBm de media; sin embargo, en ocasiones, el RSSI de ese dispositivo se encuentra por debajo de los -80 dBm, suponiendo esto un posible problema a la hora de enviar paquetes. El segundo que mayor alcance proporciona es el Sensortile.box, con un RSSI medio obtenido a una distancia de 9 metros de -75.83 dBm. Además, en este caso el RSSI más bajo es de -78 dBm, lo que, en principio, no supondría ningún problema a la hora de enviar paquetes. Finalmente, en el caso del Smartbond DA14585 y del BlueTile, los RSSI medios obtenidos andan próximos a los -80 dBm, siendo -79.84 dBm y -79 dBm respectivamente a una distancia de 9 metros del Gateway. Además, en ambos casos, se obtienen valores de RSSI por debajo de los -80 dBm, pudiendo suponer esto un problema a la hora de enviar paquetes. Por todo ello, se ha decidido establecer un rango máximo de 9 metros de distancia respecto al Gateway en los casos del Sensortag

CC2650 y Sensortile.box y un rango máximo de 8 metros en los casos del Smartbond DA14585 IoT y el BlueTile.

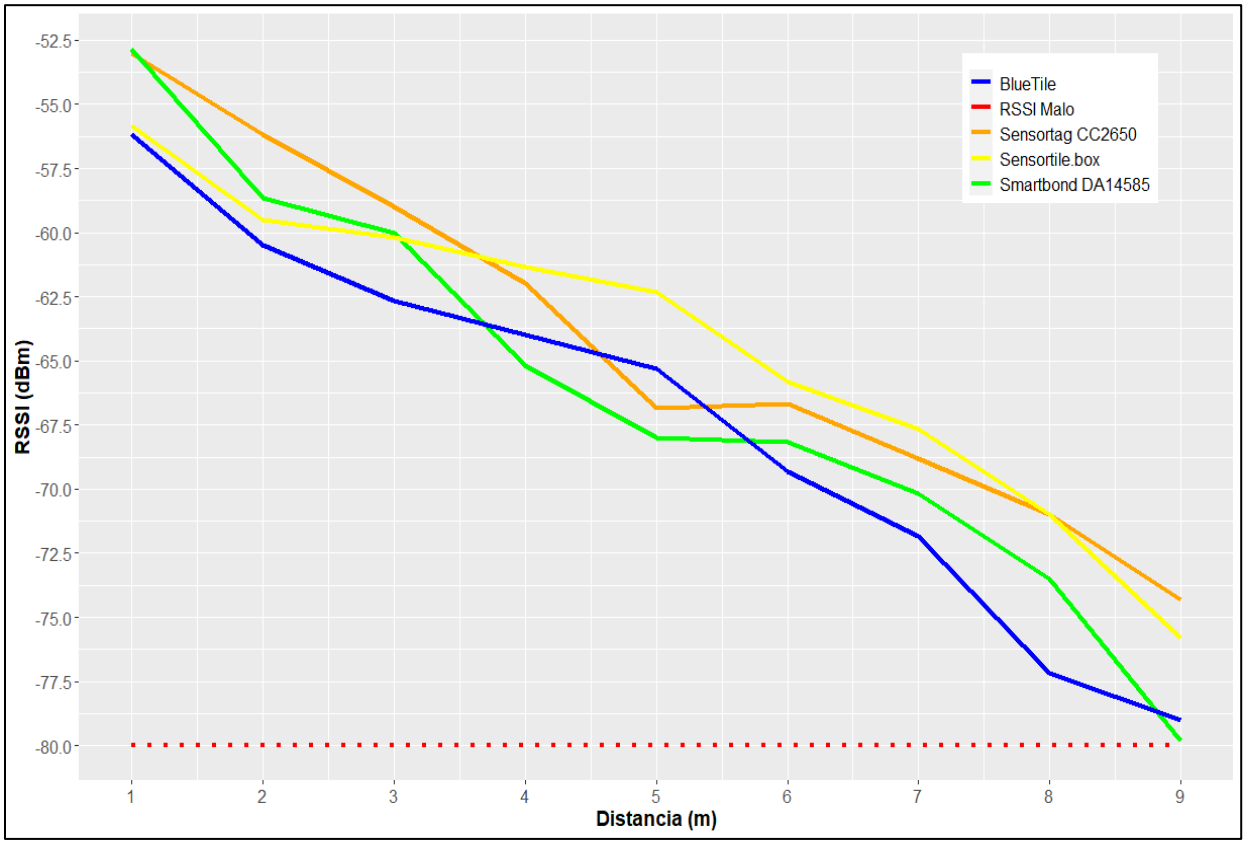

Ilustración 46: Evolución del RSSI de los módulos multisensor según la distancia

6. Despliegue del prototipo demostrador

Para demostrar el correcto funcionamiento de la topología de red diseñada, se ha decidido implantar un prototipo de monitorización en el Módulo 1 del Edificio Departamental Oeste de la Escuela Politécnica de Ingeniería. Con dicho prototipo se pretende monitorizar la temperatura, humedad, presión y luminosidad de diferentes zonas comunes del departamento.

La topología de red empleada está compuesta por dos capas, una primera capa formada por Gateways y Repetidores interconectados formando una red mallada mediante el protocolo B.A.T.M.A.N. y una segunda capa en la que se interconectan los módulos multisensor a los Gateways/Repetidores utilizando el protocolo de bajo consumo Bluetooth Low Energy. Además, la comunicación con la nube se realiza desde los Gateway/Repetidores mediante el protocolo MQTT, utilizando el bróker de mensajes Mosquitto.

En base a los resultados obtenidos en los apartados **Análisis de la comunicación entre Gateway y Repetidores** y **Análisis de la distancia máxima entre Gateway/Repetidor y módulos multisensor**, se ha establecido una distancia máxima de 14 metros entre Gateways y Repetidores con el fin de que las comunicaciones entre ellos se realicen exitosamente. También se ha definido una distancia máxima de 9 metros entre Gateway/Repetidor y Sensortag CC2650 y Sensortile.box y una distancia máxima de 8 metros entre Gateway/Repetidor y Smartbond DA14585 IoT y BlueTile.

Para este prototipo se han empleado 3 Raspberry Pi 4, de las cuales una de ellas actúa como Gateway y las otras 2 actúan como Repetidores de la red mallada. También se ha empleado 1 Raspberry Pi 3 Model B+ que ejerce de Repetidor. En cuanto a los módulos multisensor utilizados, se ha decidido usar el Sensortag CC2650 y el Smartbond DA14585 para monitorizar salas de uso común, como puede ser la Sala de Reuniones o las salas correspondientes a los seminarios, ya que a diferencia del Sensortile.box y el BlueTile disponen de luxómetro, y por tanto pueden captar la luminosidad de la sala, indicando de esta forma si las persianas se encuentran abiertas o cerradas, si hay alguna luz encendida, etc. Por tanto, los módulos multisensor Sensortile.box y BlueTile se emplearán en zonas donde la luminosidad no aporte ninguna información relevante como, por ejemplo, los vestíbulos o los pasillos.

En la Ilustración 47, Ilustración 48, Ilustración 49 e Ilustración 50 se muestra la distribución de los módulos multisensor y de los Gateways y los Repetidores en el sótano, planta baja, primera planta y segunda planta del Departamento de Informática, respectivamente.

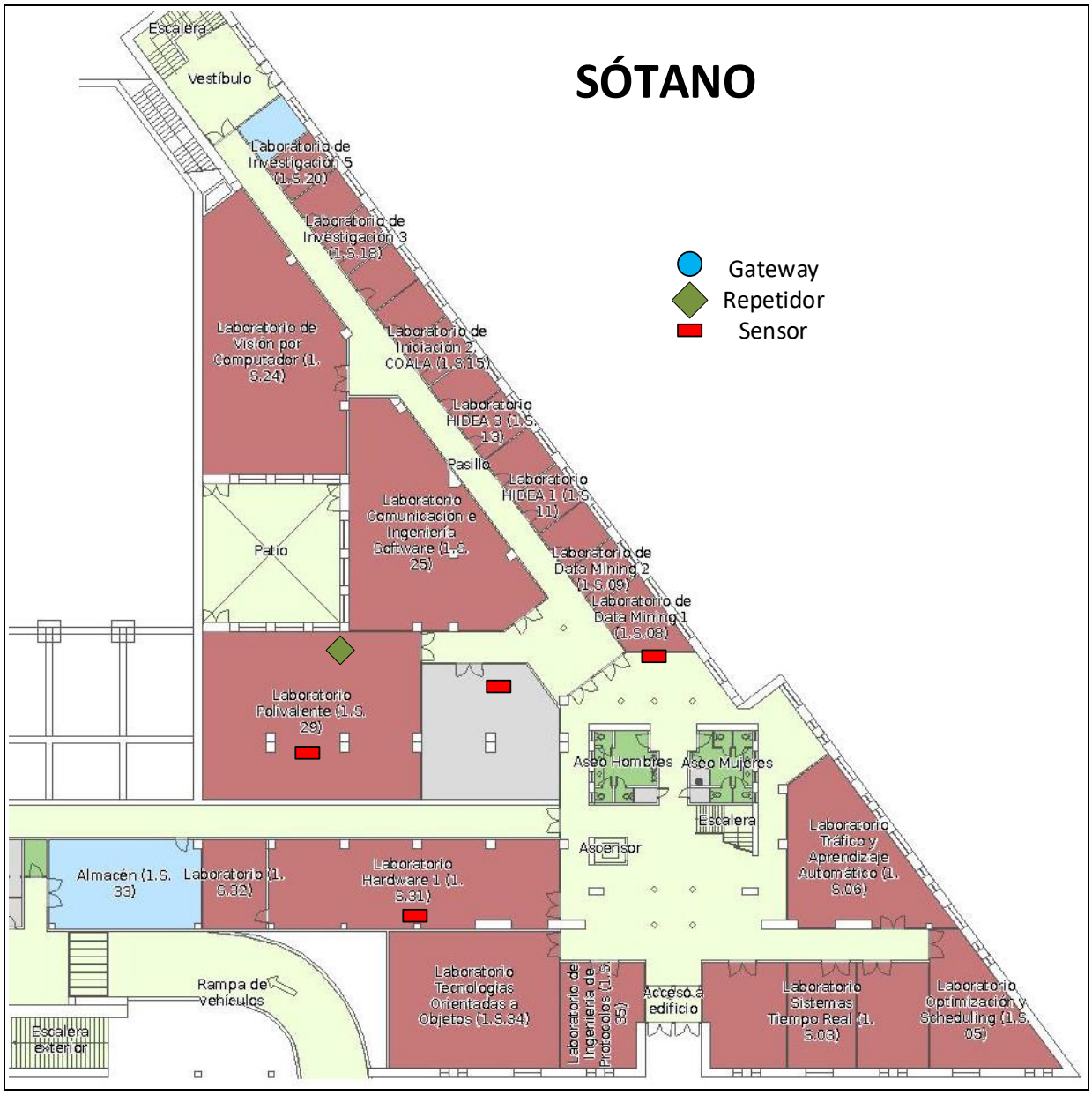

Ilustración 47: Distribución de los módulos multisensor (Sótano)

En el sótano del Departamento, se ha situado un Repetidor en el Laboratorio Polivalente (1.S.29) que gestiona cuatro módulos multisensor, de los cuales dos son Sensortile.box ubicados en el Laboratorio Polivalente (1.S.29) y en el Laboratorio Hardware (1.S.31), otro es un Sensortag

CC2650 ubicado en la sala de servidores y el último es un BlueTile situado en el vestíbulo del sótano.

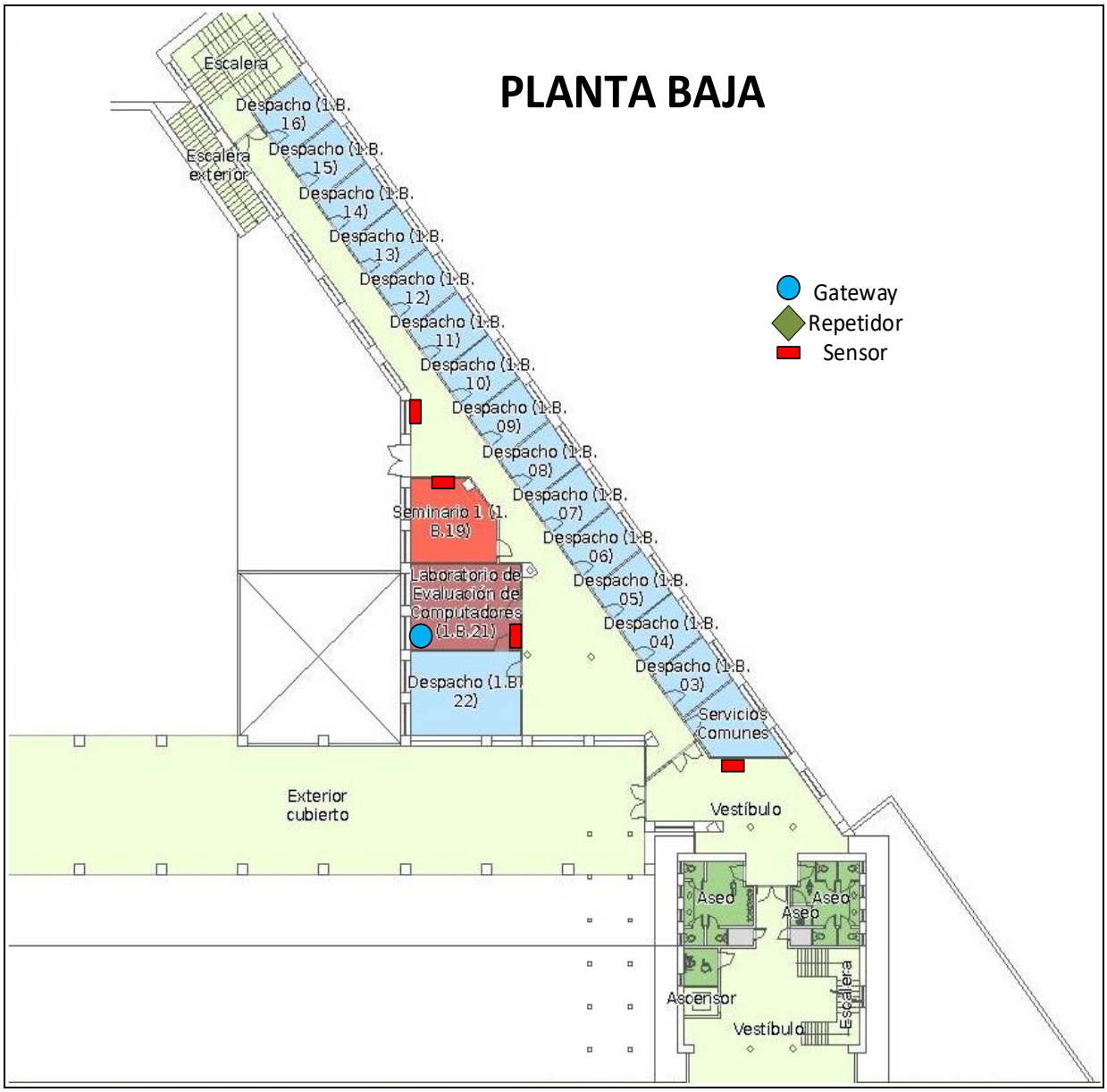

Ilustración 48: Distribución de los módulos multisensor (Planta 0)

En la Planta Baja, se ha puesto un Gateway en el Laboratorio de Tecnologías Multimedia (1.B.21) que gestiona cuatro módulos multisensor diferentes, de los cuales dos de ellos son BlueTile y se encuentran ubicados en el vestíbulo y en el pasillo a la altura del Despacho (1.B.10). Los otros dos que se han colocado son un Sensortag CC2650 en el Seminario 1 (1.B.19) y un Smartbond DA14585 en el Laboratorio de Tecnologías Multimedia (1.B.21).

Ilustración 49: Distribución de los módulos multisensor (Planta 1)

En la primera planta, se ha ubicado otro Repetidor en la Sala de Postgrado que gestiona tres módulos multisensor diferentes, de los cuales uno es BlueTile, y se encuentra situado en el Vestíbulo, donde se ubica la máquina de café. Los otros dos módulos multisensor son un Sensortag CC2650 ubicado en la Sala de Postgrado y un Smartbond DA14585 ubicado en el Despacho (1.1.02) que se corresponde con la actual Sala de Reprografía.

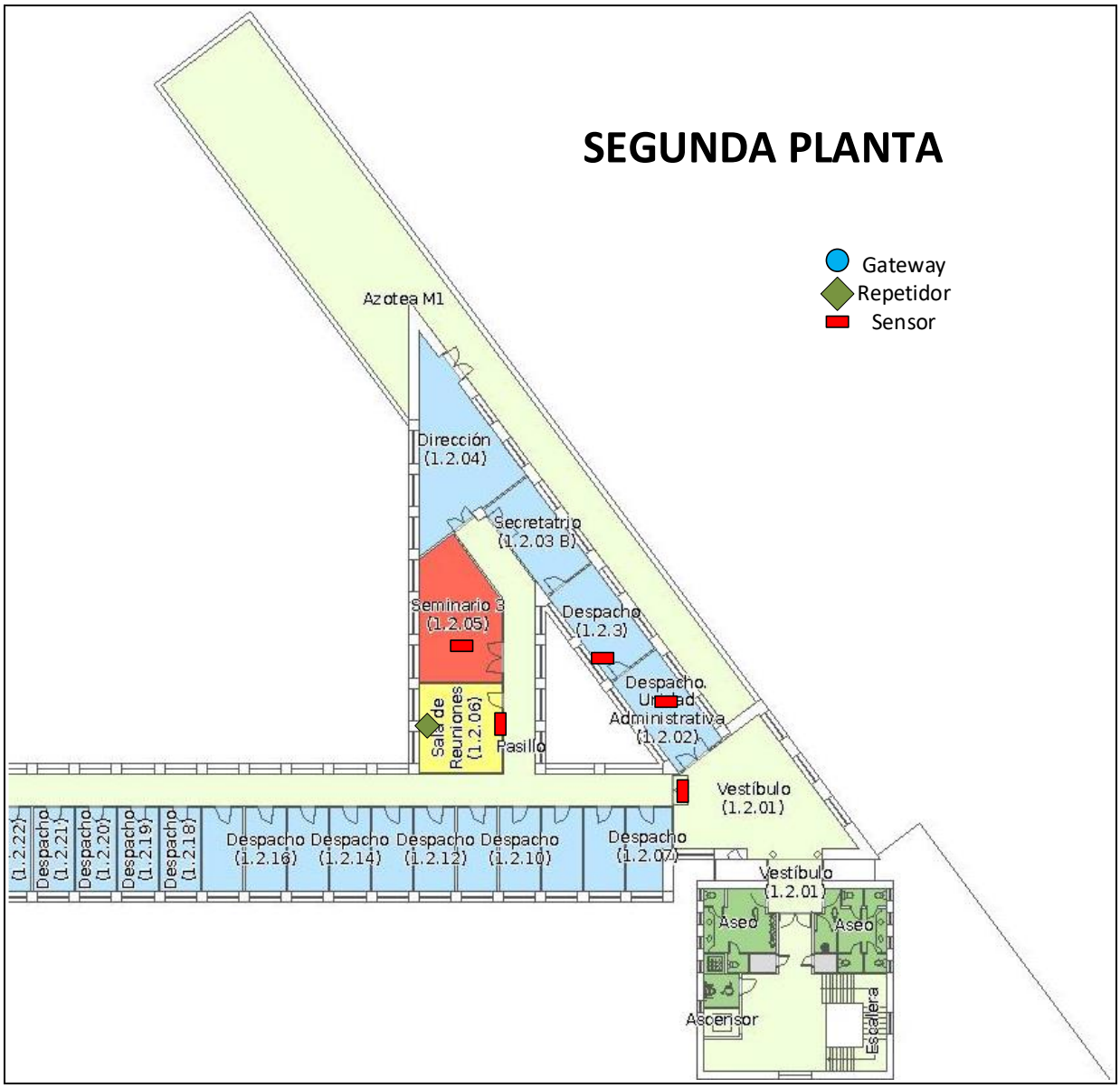

Ilustración 50: Distribución de los módulos multisensor (Planta 2)

Finalmente, en la Segunda Planta, se ha instalado un Repetidor en la Sala de Reuniones (1.2.06) que se encarga de gestionar cinco módulos multisensor, de los cuales hay un BlueTile ubicado en el Vestíbulo (1.2.01), dos Smartbond DA14585 en la Sala de Reuniones (1.2.06) y en el Seminario 3 (1.2.05), y dos Sensortag CC2650 en el Despacho (1.2.3) y en el Despacho de Unidad Administrativa (1.2.02).

En total se han empleado 16 módulos multisensor, de los cuales hay 5 Sensortag CC2650, 4 Smartbond DA14585, 2 Sensortile.box y 5 BlueTile. Además, cabe destacar que la batería

correspondiente al Sensortile.box tarda en consumirse en torno a tres días, ya que utiliza una batería recargable. Por ello, todos los dispositivos de este tipo se han instalado conectados directamente a un enchufe.

Como cada Gateway/Repetidor va a manejar una red de módulos multisensor heterogénea, se ha decidido generar un script que permita recolectar simultáneamente la información que proviene de los cuatro tipos diferentes de módulos multisensor, realizar las transformaciones que correspondan y enviarlas a la plataforma ThingsBoard, en lugar de emplear un script diferente para cada módulo multisensor. El código correspondiente a este script se incluye en el **Anexo 3. Script Python final de recolección y envío de datos a la nube.**

Además, ha sido necesario añadir en la Plataforma ThingsBoard cada uno de los módulos multisensor empleados, con el fin de usar el Access Token que les corresponda para actualizar en la nube la información relacionada con el módulo multisensor.

Planta	Lugar	Modelo	Dirección MAC
Sótano	Laboratorio Polivalente (1.S.29)	Sensortile.box	E0:60:86:BD:1D:BF
Sótano	Sala de Servidores	Sensortag CC2650	24:71:89:E9:2A:85
Sótano	Vestíbulo Sótano	BlueTile	F2:80:09:7F:AC:09
Sótano	Laboratorio Hardware (1.S.31)	Sensortile.box	DA:03:0B:76:73:74
P0	Pasillo – Despacho (1.B.10)	BlueTile	E0:4F:5C:56:BB:30
P0	Vestíbulo Planta 0	BlueTile	F6:CD:F2:B1:7F:77
P0	Seminario 1 (1.B.19)	Sensortag CC2650	54:6C:0E:79:30:80
P0	Laboratorio Multimedia (1.B.21)	Smartbond DA14585	80:EA:CA:70:A1:58
P1	Sala de Postgrado (1.1.25)	Sensortag CC2650	54:6C:0E:53:01:71
P1	Vestíbulo Planta 1	BlueTile	DD:56:6C:33:CB:C3
P1	Despacho (1.1.02)	Smartbond DA14585	80:EA:CA:70:B1:A0
P2	Vestíbulo Planta 2	BlueTile	E2:73:E8:2A:DB:CF
P2	Sala de Reuniones (1.2.06)	Smartbond DA14585	80:EA:CA:70:B1:66
P2	Seminario 3 (1.2.05)	Smartbond DA14585	80:EA:CA:70:A1:6D
P2	Despacho (1.2.3)	Sensortag CC2650	54:6C:0E:4D:3C:81
P2	Despacho Ud. Administrativa (1.2.02)	Sensortag CC2650	24:71:89:E8:99:02

Tabla 15: Distribución de los módulos multisensor en el Departamento de Informática

Todos los dispositivos mostrados en la Tabla 15, han sido registrados en la plataforma ThingsBoard y se han generado 4 dashboards diferentes, uno por cada planta del departamental en el que se muestran los datos recogidos por los módulos multisensor.

Una vez instalado el prototipo de funcionamiento y puesta en marcha su ejecución, se puede observar a través de la plataforma IoT ThingsBoard el estado de cada una de las zonas monitorizadas mediante cada uno de los paneles creados. Dichos paneles se han asignado al usuario personaldptoinformatica@gmail.com.

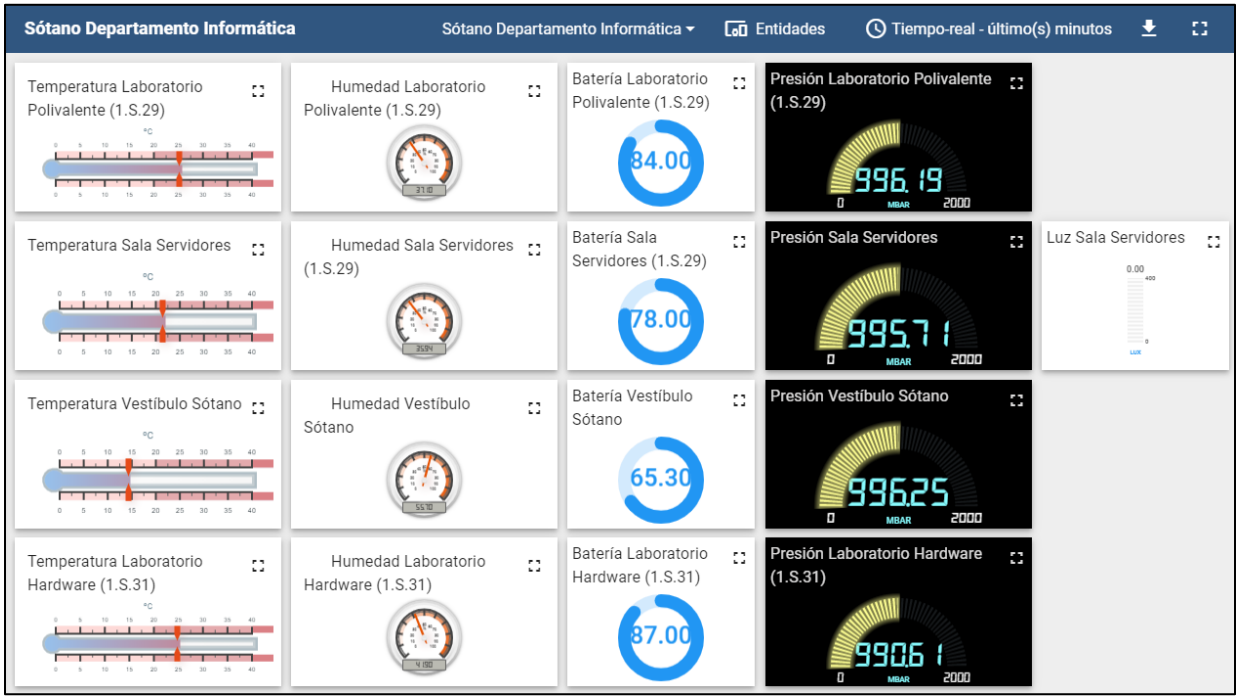

Ilustración 51: Dashboard Sótano Departamento de Informática

En la Ilustración 51, se muestra el dashboard correspondiente al sótano del departamento de informática. En él se puede apreciar cuatro zonas monitorizadas, de las que dos son laboratorios que muestran unas características similares, temperaturas de en torno a los 25 ºC y humedad relativa próxima al 40%. También se muestra el estado de la sala de servidores, en ella la temperatura oscila entre los 20 y 22 ºC a lo largo del día y el porcentaje relativo de humedad es similar al de los laboratorios. Además, en esta sala se ha decidido monitorizar la luz para detectar si hay alguien dentro o si se ha dejado encendida. Por último, se monitoriza el vestíbulo del sótano, que presenta temperaturas muy bajas, habitualmente por debajo de los 15 ºC, y una humedad superior a la existente en los otros casos. En todas las zonas monitorizadas la presión se encuentra siempre en torno a 1000 mbar. Además de la información de las variables de entorno, también se muestra el porcentaje de batería de cada uno de los módulos multisensor empleados.

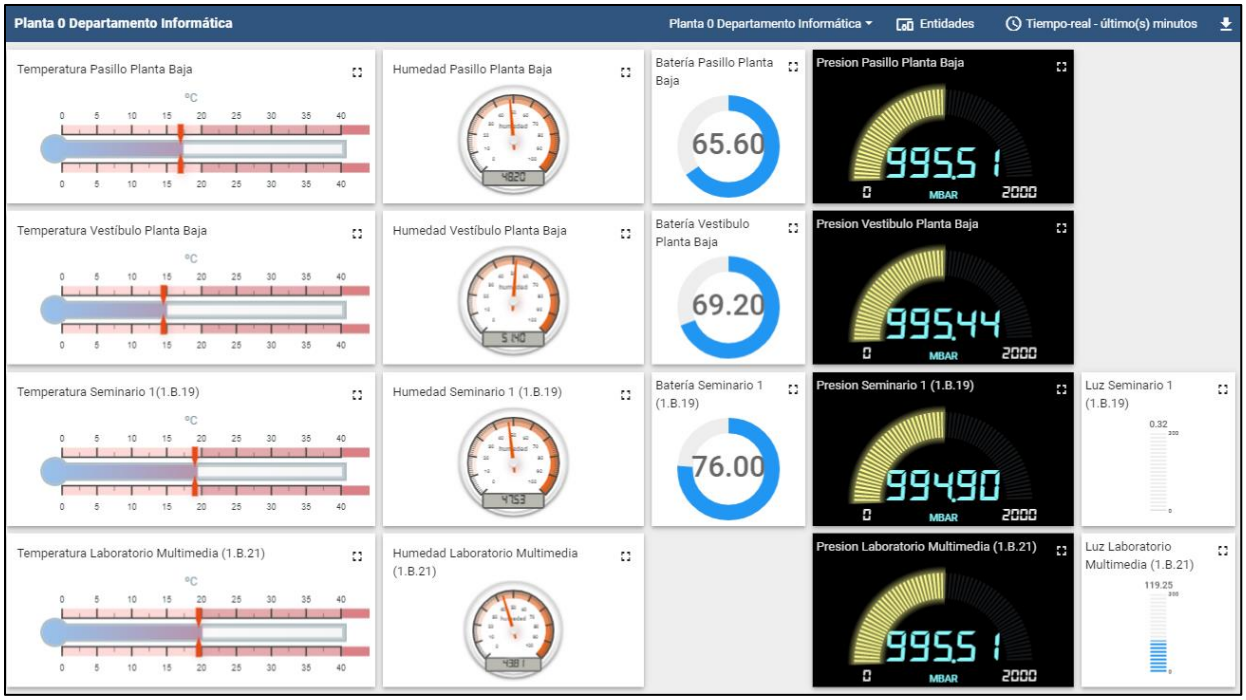

Ilustración 52: Dashboard Planta Baja Departamento de Informática

La siguiente planta monitorizada se corresponde con la planta baja, cuyos datos se recogen en el dashboard mostrado en la Ilustración 52. En esta planta se monitoriza un laboratorio y un seminario cuya temperatura está entre 18 y 20 ºC. Además, en estas dos salas, se monitoriza también la luz existente, para controlar si se ha dejado alguna persiana abierta o alguna luz encendida. También se ha monitorizado el Vestíbulo y el pasillo, estando la temperatura comprendida entre los 14 y los 16 ºC. La presión, al igual que en el caso del Sótano, se encuentra en torno a los 1000 mbar en todas las zonas, y la humedad relativa no fluctúa mucho entre una zona u otra.

En la primera planta se han monitorizado tres zonas, la sala de Postgrados, la sala de Reprografía, que se corresponde con el Despacho 1.1.02, y el vestíbulo. Todos los datos recolectados de esta planta se pueden ver en el dashboard mostrado en la Ilustración 53. De forma similar a los casos anteriores, la temperatura de las salas se encuentra siempre por encima de los 18 ºC. Sin embargo, la del vestíbulo no supera los 17 ºC en ningún momento, llegando a tener temperaturas mínimas de hasta 13 ºC por las noches. En este caso, la humedad relativa varía significativamente en función de la zona. En cuanto a la presión, sucede lo mismo que en los casos anteriores, está siempre alrededor de 1000 mbar.

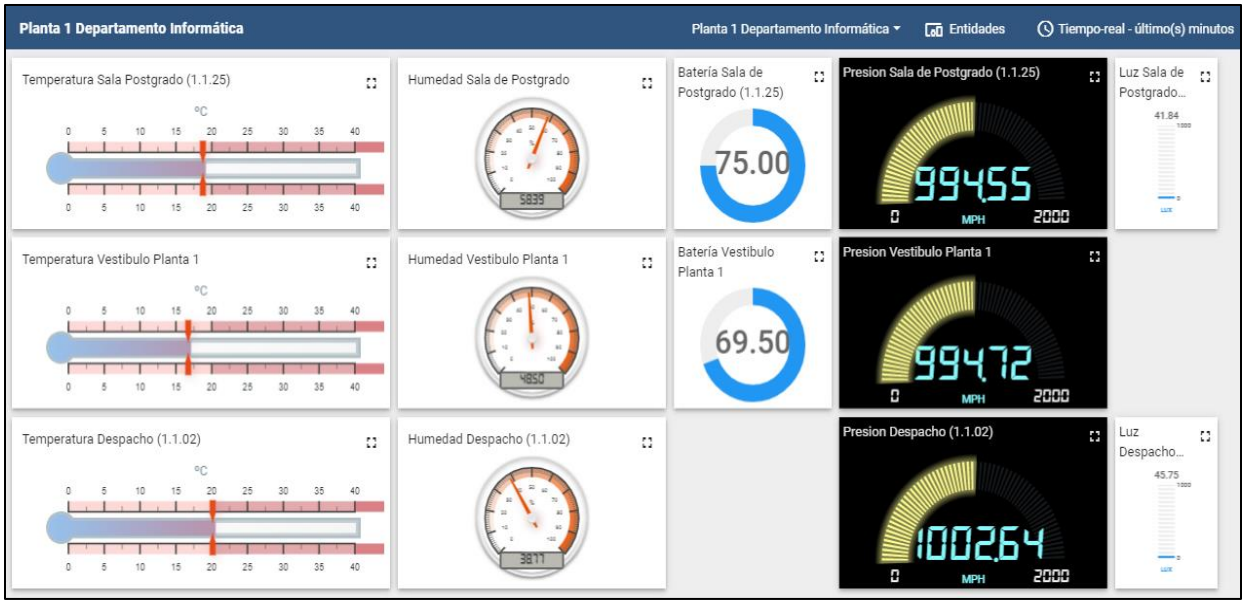

Ilustración 53: Dashboard Primera Planta Departamento de Informática

Ilustración 54: Dashboard Segunda Planta Departamento de Informática

El último dashboard se corresponde con la segunda planta y se muestra en la Ilustración 54. Se puede apreciar cómo se monitorizan cuatro salas y el vestíbulo. En todas las salas, la temperatura se encuentra por encima de los 20 ºC mientras que en el caso del vestíbulo la temperatura es bastante inferior, alcanzando máximos de 17 ºC aproximadamente. También se puede ver como la humedad en la Sala de Reuniones y en el Seminario 3 es inferior a la existente en el Despacho 1.2.3, en el Despacho de Unidad Administrativa y en el Vestíbulo. Igual que en los casos anteriores, la presión se encuentra próxima a los 1000 mbar. Además, en este caso también se registra el nivel de luminosidad presente en cada una de las salas.

6.1. Resultados obtenidos

Una vez instalado el prototipo y comprobado su correcto funcionamiento, se ha optado por llevar a cabo un estudio similar al realizado para analizar el nivel RSSI entre dispositivos, latencia, jitter y throughput.

En primer lugar, se define la topología empleada en el prototipo, la cual se puede ver en la Ilustración 55. Cabe destacar que las distancias mostradas entre los módulos multisensor y los nodos son una aproximación. El Gateway se encuentra en la Planta Baja conectado al router directamente a través de un cable Ethernet. En cuanto a la conectividad entre los Repetidores y el Gateway, se realiza empleando el protocolo B.A.T.M.A.N. Todos se encuentran a un salto, aunque en ocasiones, cuando la comunicación entre el Gateway y el Repetidor ubicado en la Planta 2 resulta inestable, se realizan las comunicaciones empleando dos saltos, del Gateway al Repetidor de la Planta 1 y del Repetidor de la Planta 1 al de la Planta 2 y viceversa. Además, cada nodo de la red se encarga de recolectar los datos de los módulos multisensor ubicados en su misma planta mediante el uso del protocolo Bluetooth Low Energy y, de una vez recolectados, realizar las transformaciones necesarias para enviarlos posteriormente a la plataforma IoT ThingsBoard utilizando el protocolo MQTT.

Sobre esta topología se ha realizado un estudio similar al realizado anteriormente. Primero un análisis del nivel RSSI entre Repetidores y Gateway, y después entre los módulos multisensor y el Gateway o Repetidor encargado de ellos. Para finalizar, se ha hecho un análisis del throughput, la latencia y el jitter entre los Repetidores y el Gateway.

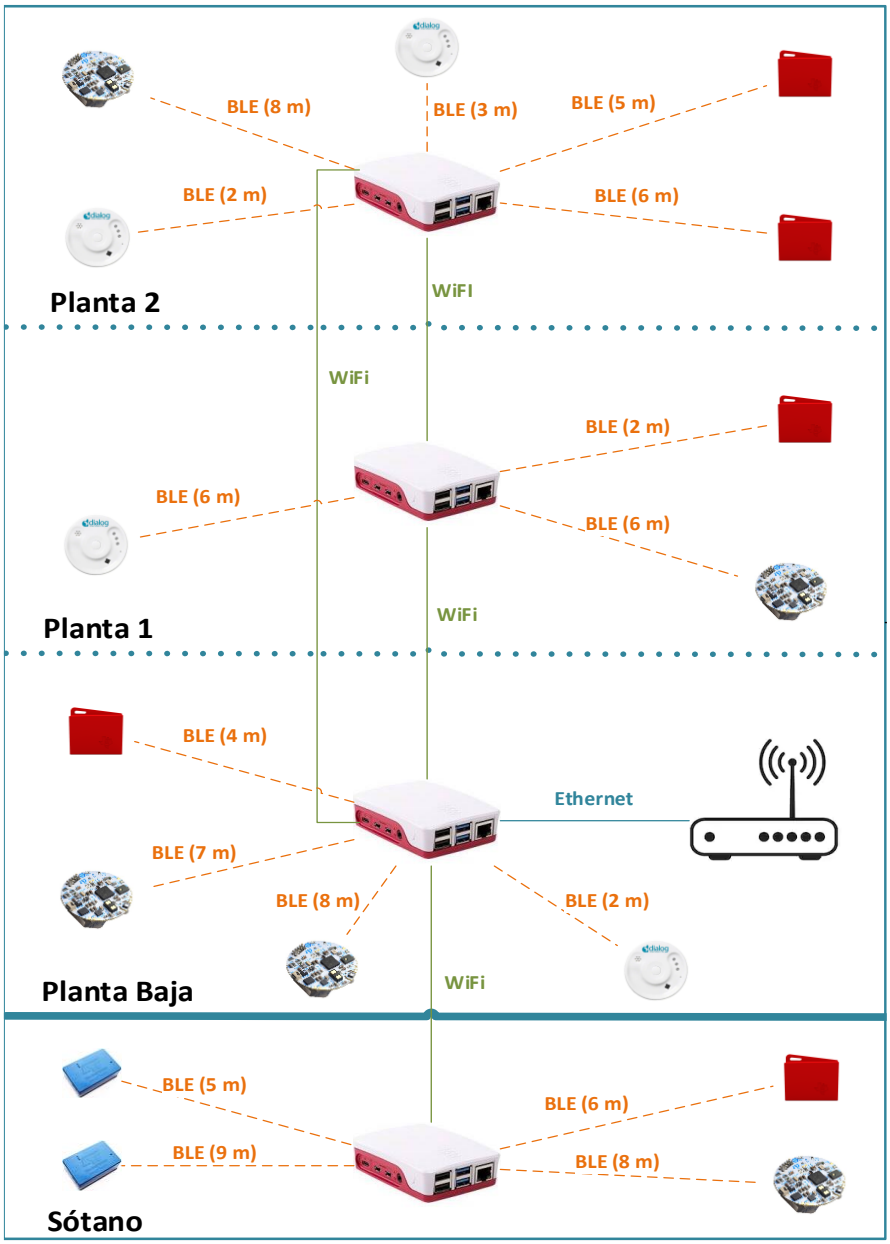

Ilustración 55: Topología seguida en el prototipo desarrollado

En la Tabla 16, se recogen los resultados obtenidos tras analizar el nivel RSSI entre los Repetidores y el Gateway.

	Prueba RSSI entre Repetidor y Gateway						
Repetidor	**N.º 1**	**N.º 2**	**N.º 3**	**N.º 4**	**N.º 5**	**N.º 6**	**Media**
Sótano	-75 dBm	-77 dBm	-77 dBm	-76 dBm	-75 dBm	-76 dBm	-76 dBm
Planta 1	-62 dBm	-63 dBm	-62 dBm	-63 dBm	-63 dBm	-62 dBm	-62.5 dBm
Planta 2	-63 dBm	-65 dBm	-66 dBm	-65 dBm	-66 dBm	-63 dBm	-64.67 dBm

Tabla 16: Análisis del RSSI entre Repetidores y Gateway en el prototipo desarrollado

Como puede verse, ninguno de ellos tiene un nivel RSSI inferior a -80 dBm, por lo que su distanciamiento no supondría ningún problema a la hora de enviar paquetes a través de la red. Además, se puede apreciar como en el caso del Repetidor ubicado en el sótano el RSSI es mucho mayor. Esto se debe básicamente a que, entre la Planta Baja, la Planta 1 y la Planta 2 no hay un techo que las separe directamente, ya que hay un patio interior que las comunica. Sin embargo, en el caso del sótano, sí existe un techo que lo separa completamente de la Planta Baja el cual produce atenuaciones y dificulta la comunicación.

Los resultados obtenidos tras analizar el nivel RSSI entre los módulos multisensor y los nodos (Gateway y Repetidores) se recogen en la Tabla 17.

Ubicación del nodo	Lugar donde se encuentra el módulo multisensor	Dirección MAC	RSSI
Sótano	Laboratorio Polivalente (1.S.29)	E0:60:86:BD:1D:BF	-72 dBm
Sótano	Sala de Servidores	24:71:89: E9:2A:85	-69 dBm
Sótano	Vestíbulo Sótano	F2:80:09:7F:AC:09	-79 dBm
Sótano	Laboratorio Hardware (1.S.31)	DA: 03:0B:76:73:74	-78 dBm
P0	Pasillo – Despacho (1.B.10)	E0:4F:5C:56:BB:30	-72 dBm
P0	Vestíbulo Planta 0	F6:CD:F2:B1:7F:77	-78 dBm
P0	Seminario 1 (1.B.19)	54:6C:0E:79:30:80	-65 dBm
P0	Laboratorio Multimedia (1.B.21)	80:EA:CA:70:A1:58	-57 dBm
P1	Sala de Postgrado (1.1.25)	54:6C:0E:53:01:71	-56 dBm
P1	Vestíbulo Planta 1	DD:56:6C:33:CB:C3	-75 dBm
P1	Despacho (1.1.02)	80:EA:CA:70:B1:A0	-65 dBm
P2	Vestíbulo Planta 2	E2:73:E8:2A:DB:CF	-79 dBm
P2	Sala de Reuniones (1.2.06)	80:EA:CA:70:B1:66	-54 dBm
P2	Seminario 3 (1.2.05)	80:EA:CA:70:A1:6D	-65 dBm
P2	Despacho (1.2.3)	54:6C:0E:4D:3C:81	-73 dBm
P2	Despacho Ud. Administrativa (1.2.02)	24:71:89:E8:99:02	-74 dBm

Tabla 17: Análisis del RSSI entre sensores y nodos en el prototipo desarrollado

Todos los módulos multisensor se encuentran por encima de los -80 dBm. Sí es cierto, que este valor puede cambiar debido a posibles interferencias y se puede producir algún error de comunicación. Es por ello por lo que el script de captura se encuentra preparado para hacer frente a estos posibles problemas y, en caso de que sucedan, no se produzca ningún fallo que pueda causar la detención de la monitorización del Departamento de Informática.

En cuanto al análisis del throughput, se ha optado por medir el rendimiento entre los Repetidores y el Gateway. En la Tabla 18, se recogen los resultados obtenidos tras realizar este experimento, todos ellos en kB/s.

	Análisis del rendimiento entre Repetidores y Gateway						
Repetidor	**N.º 1**	**N.º 2**	**N.º 3**	**N.º 4**	**N.º 5**	**N.º 6**	**Media**
Sótano	399.24	401.82	395.08	387.10	401.00	411.74	399.33
Planta 1	1921.42	2082.15	1824.10	2073.21	1983.02	2012.73	1982.77
Planta 2	804.05	839.36	904.07	917.87	872.40	925.33	877.18

Tabla 18: Análisis del throughput en el prototipo desarrollado

El throughput es mucho menor entre el Gateway y el Repetidor ubicado en el sótano, siendo este próximo a los 400 kB/s, algo que puede justificarse ya que el nivel RSSI entre ambos también es mucho menor que en el resto de los casos. En cuanto al throughput entre el Gateway y los Repetidores ubicados en las Plantas 1 y 2, se puede apreciar como a mayor distancia entre ellos, menor es el throughput existente. En el caso del ubicado en la primera planta, el throughput se encuentra en torno a los 2 MB/s, sin embargo, en el caso del situado en la segunda planta, el throughput se encuentra próximo a los 900 kB/s. En todos los casos el throughput existente es más que suficiente para llevar a cabo las comunicaciones necesarias sin ningún tipo de problema.

También se analizó la latencia y el jitter existente en las comunicaciones entre los Repetidores y el Gateway. Los resultados obtenidos de estas pruebas se muestran en la Tabla 19 y Tabla 20.

	Análisis de la latencia entre Repetidores y Gateway						
Repetidor	**N.º 1**	**N.º 2**	**N.º 3**	**N.º 4**	**N.º 5**	**N.º 6**	**Media**
Sótano	2.87 ms	2.71 ms	2.88 ms	2.72 ms	2.66 ms	2.61 ms	2.74 ms
Planta 1	2.02 ms	1.98 ms	2.01 ms	1.92 ms	2.02 ms	2.01 ms	1.99 ms
Planta 2	2.48 ms	2.48 ms	2.44 ms	2.42 ms	2.36 ms	2.33 ms	2.42 ms

Tabla 19: Análisis de la latencia en el prototipo desarrollado

	Análisis del jitter entre Repetidores y Gateway						
Repetidor	**N.º 1**	**N.º 2**	**N.º 3**	**N.º 4**	**N.º 5**	**N.º 6**	**Media**
Sótano	1.66 ms	1.46 ms	1.67 ms	1.22 ms	1.16 ms	1.10 ms	1.38 ms
Planta 1	0.62 ms	0.42 ms	0.55 ms	0.43 ms	0.52 ms	0.40 ms	0.49 ms
Planta 2	0.92 ms	0.97 ms	0.93 ms	1.23 ms	1.06 ms	1.01 ms	1.02 ms

Tabla 20: Análisis del jitter en el prototipo desarrollado

La latencia y el jitter es mucho mayor entre el Repetidor ubicado en el sótano y el Gateway que en el resto de los casos. Esto se debe principalmente a que son los que menor RSSI tienen. En cuanto a la latencia y jitter existente entre los Repetidores ubicados en las plantas 1 y 2 con el Gateway, es menor en el caso del de la primera planta, ya que se encuentra más cercano al Gateway. En todos ellos, la latencia se encuentra por debajo de los 3 ms y el jitter por debajo de los 2 ms, lo que no supondría ningún problema a la hora de llevar a cabo las comunicaciones entre los nodos de la red.

Además, en todos los casos, se puede apreciar que cuanto menor es el nivel RSSI existente entre Repetidor y Gateway, menor es el throughput y mayor es la latencia y el jitter.

7. Conclusiones y trabajo futuro

Los objetivos planteados han sido logrados completamente, desarrollando una topología de red de doble capa, compuesta por una primera capa de nodos heterogéneos que actúan de Gateways o Repetidores formando una red mallada interconectada mediante el uso del protocolo B.A.T.M.A.N. y, por una segunda capa formada por un conjunto de módulos multisensor heterogéneos conectados al Gateway o Repetidor correspondiente mediante el uso del protocolo Bluetooth Low Energy.

Esta topología ha sido instalada en el Edificio Departamental de Informática con el fin de realizar un prototipo demostrador que permita comprobar su correcto funcionamiento, realizando sobre ella múltiples análisis de latencia, jitter, throughput y comprobaciones del nivel RSSI entre dispositivos, con el fin de dimensionarla correctamente y proporcionar unas comunicaciones fiables, con un porcentaje de pérdida de paquetes muy reducido. Para que las comunicaciones se realizaran con la mayor fiabilidad posible, se determinó que la distancia máxima entre los Gateway y Repetidores debe ser de 14 metros, mientras que la distancia máxima entre los módulos multisensor y el Gateway o Repetidor que los gestiona, debe ser de 9 metros en el caso de los Sensortile.box y los Sensortag CC2650, y de 8 metros en el caso de los Smartbond DA14585 IoT y BlueTile.

Además, para almacenar los datos recolectados por el prototipo se ha empleado la plataforma IoT ThingsBoard, instalada en una máquina virtual de la Universidad de Oviedo, a la que se envían los datos desde los Gateway y Repetidores mediante el protocolo MQTT utilizando el bróker de mensajes Mosquitto.

Gracias a todo esto, se ha desarrollado una topología de red capaz de autogestionarse, autoconfigurarse y ser fácilmente escalable a la hora de añadir o quitar dispositivos de la red. Además, se trata de una topología tolerante a fallos tanto a nivel de red, al poder configurarse varios nodos como Gateways de la red mallada, como a nivel de comunicaciones, al existir siempre múltiples caminos a la hora de redirigir los paquetes a través de la red mallada. Esta topología resulta de gran interés para ser instalada en zonas donde la cobertura inalámbrica por parte de un router se ve reducida, como bien puede ser una planta industrial, ya que permite dar

acceso a Internet a los Repetidores sin que ellos tengan conexión directa con el router, siempre y cuando el Gateway sí la tenga. Como resultado de la investigación llevada a cabo en este trabajo, es posible dimensionar y desplegar de forma rápida y eficiente una red de sensores dispersa para una aplicación objetivo.

Este trabajo fin de master forma parte de la primera tarea de investigación de mi tesis doctoral, relativa a la monitorización de motores eléctricos en tiempo real para la detección de anomalías y el mantenimiento predictivo, por lo que el trabajo futuro a desarrollar se corresponde con el plan de investigación de la tesis.

- Primer año:

  Adaptar el prototipo desarrollado a un prototipo de monitorización de motores eléctricos orientado a la detección de anomalías y a la prevención de fallos de funcionamiento, para posteriormente, en cuanto la situación sanitaria lo permita, instalarlo en la planta industrial de CAPSA.

  Para ello será necesario, en primer lugar, la recolección de los datos relevantes, que se corresponden con temperatura, vibraciones y consumo eléctrico del motor. Sobre las vibraciones se aplicará la Transformada de Fourier para que los datos pasen del dominio del tiempo a la frecuencia y posteriormente se seleccionen aquellos armónicos relevantes que, junto a la temperatura y a los datos relativos al consumo eléctrico, serán almacenados en la nube.

  También se estudiarán los patrones de comunicación, valorando el uso tanto de opciones sincrónicas como asíncronas. Otro aspecto clave para tener en cuenta, es la codificación de la información a la hora de ser transmitida de forma eficiente, valorando el uso de diferentes métodos de serialización de datos estructurados, así como el posible uso de algoritmos de compresión sin pérdida para reducir en mayor medida el ancho de banda utilizado.

- Segundo año:

  Durante el segundo año, se emplearán herramientas de modelado y simulación de técnicas de gestión de recursos en entornos de computación Fog con la intención de distribuir de

forma óptima las tareas de procesamiento de aplicaciones IoT entre las capas de la arquitectura Fog y la nube.

También se realizará la construcción y el análisis de diferentes modelos predictivos mediante servicios en la nube, realizando un estudio comparativo en profundidad de todas las ofertas de servicios de machine learning ofertadas en los tres grandes proveedores cloud (AWS, Azure y GCP), abordando entre otros aspectos las herramientas disponibles para la construcción de modelos, las estructuras de almacenamiento compatibles de los históricos de datos y la existencia de conjuntos de datos de ejemplo para la realización de pruebas.

- Tercer año:

  En el tercer año, se optimizará la aplicación IoT orientada al mantenimiento predictivo, empleando los métodos desarrollados para distribuir de forma óptima las tareas de procesamiento correspondientes a la aplicación IoT orientada al mantenimiento predictivo de motores eléctricos.

  Finalmente, se analizará la influencia de los datos seleccionados como salida de la red de sensores sobre la calidad de los resultados en la detección de anomalías de funcionamiento de los motores eléctricos y, la calidad de los resultados obtenidos en la predicción de fallos de los motores por parte del modelo entrenado.

Referencias

[1] KARIMI, Kaivan; ATKINSON, Gary. What the Internet of Things (IoT) needs to become a reality. White Paper, FreeScale and ARM, 2013, p. 1-16.

[2] METALLIDOU, Chrysi K.; PSANNIS, Kostas E.; EGYPTIADOU, Eugenia Alexandropoulou. Energy Efficiency in Smart Buildings: IoT Approaches. IEEE Access, 2020, vol. 8, p. 63679-63699.

[3] MAGADÁN, L., et al. Low-cost real-time monitoring of electric motors for the Industry 4.0. Procedia Manufacturing, 2020, vol. 42, p. 393-398.

[4] GIL, David, et al. Internet of things: A review of surveys based on context aware intelligent services. Sensors, 2016, vol. 16, no 7, p. 1069.

[5] (Octubre 2020). Obtenido de Cloud Computing – Fog Computing & Edge Computing: https://www.winsystems.com/cloud-fog-and-edge-computing-whats-the-difference/

[6] ZAKHAROV, Alexander, et al. Intellectual data analysis system of building temperature mode monitoring. En 2019 International Russian Automation Conference (RusAutoCon). IEEE, 2019. p. 1-6.

[7] MAGADÁN, Luis, et al. Real-Time Monitoring of Electric Motors for Detection of Operating Anomalies and Predictive Maintenance. En International Summit Smart City 360°. Springer, Cham, 2019. p. 301-311.

[8] AKYILDIZ, Ian F.; WANG, Xudong. A survey on wireless mesh networks. IEEE Communications magazine, 2005, vol. 43, no 9, p. S23-S30.

[9] MAUVE, Martin; WIDMER, Jorg; HARTENSTEIN, Hannes. A survey on position-based routing in mobile ad hoc networks. IEEE network, 2001, vol. 15, no 6, p. 30-39.

[10] LALAR, Sachin; YADAV, A. Comparative study of routing protocols in MANET. OJCST, 2017, vol. 10, p. 174.

[11] ABDULLEH, Muthana Najim, et al. Comparative study of proactive, reactive and geographical manet routing protocols. Communications and Network, 2015, vol. 7, no 02, p. 125.

[12] GROVER, D.; SINGH, G. Implementation: DSR & AODV. International Journal of Computer Application, 2012, vol. 2, no 5, p. 66-79.

[13] ABOLHASAN, Mehran; WYSOCKI, Tadeusz; DUTKIEWICZ, Eryk. A review of routing protocols for mobile ad hoc networks. Ad hoc networks, 2004, vol. 2, no 1, p. 1-22.

[14] MOHSENI, Shima, et al. Comparative review study of reactive and proactive routing protocols in MANETs. En 4th IEEE International Conference on Digital ecosystems and technologies. IEEE, 2010. p. 304-309.

[15] SHIVAHARE, Basu Dev; WAHI, Charu; SHIVHARE, Shalini. Comparison of proactive and reactive routing protocols in mobile adhoc network using routing protocol property. International Journal of Emerging Technology and Advanced Engineering, 2012, vol. 2, no 3, p. 356-359.

[16] ADE, S. A.; TIJARE, P. A. Performance comparison of AODV, DSDV, OLSR and DSR routing protocols in mobile ad hoc networks. International journal of information technology and knowledge management, 2010, vol. 2, no 2, p. 545-548.

[17] HE, Guoyou. Destination-sequenced distance vector (DSDV) protocol. Networking Laboratory, Helsinki University of Technology, 2002, p. 1-9.

[18] CLAUSEN, Thomas, et al. Optimized link state routing protocol (OLSR). 2003.

[19] BOUSHABA, Abdelali, et al. Multi-point relay selection strategies to reduce topology control traffic for OLSR protocol in MANETs. Journal of Network and Computer Applications, 2015, vol. 53, p. 91-102.

[20] JOHNSON, David B., et al. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. Ad hoc networking, 2001, vol. 5, no 1, p. 139-172.

[21] BOUHORMA, Mohammed; BENTAOUIT, H.; BOUDHIR, A. Performance comparison of ad-hoc routing protocols AODV and DSR. En 2009 International Conference on Multimedia Computing and Systems. IEEE, 2009. p. 511-514.

[22] PERKINS, Charles; BELDING-ROYER, Elizabeth; DAS, Samir. RFC3561: Ad hoc on-demand distance vector (AODV) routing. 2003.

[23] PEREZ, MA Ortuno, et al. Abbreviated dynamic source routing: Source routing with non-unique network identifiers. En Second Annual Conference on Wireless On-demand Network Systems and Services. IEEE, 2005. p. 76-82.

[24] JOHNSON, David; NTLATLAPA, Ntsibane; AICHELE, Corinna. Simple pragmatic approach to mesh routing using BATMAN. 2008.

[25] GRAARUD, Espen Grannes, et al. A New Dawn for the Dark Knight: Securing BATMAN. 2011.

[26] KLEIN, Alexander; BRAUN, Lothar; OEHLMANN, Fabian. Performance study of the better approach to mobile adhoc networking (batman) protocol in the context of asymmetric links. En 2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM). IEEE, 2012. p. 1-7.

[27] (Octubre 2020). Obtenido de BATMAN IV – batman-adv – Open Mesh: https://www.open-mesh.org/projects/batman-adv/wiki/BATMAN_IV

[28] (Octubre 2020). Obtenido de BATMAN V – batman-adv – Open Mesh: https://www.open-mesh.org/projects/batman-adv/wiki/BATMAN_V

[29] (Octubre 2020). Obtenido de BATMAN V – batman-adv – Open Mesh: https://www.open-mesh.org/projects/batman-adv/wiki/ELP

[30] BASTIDAS, Sixto Enrique Campaña; PELÁEZ, Jorge Mario Londoño. ESTUDIO DE REDES DE SENSORES Y APLICACIONES ORIENTADAS A LA RECOLECCIÓN Y ANÁLISIS DE SEÑALES BIOMÉDICAS. Revista GTI, 2013, vol. 12, no 33, p. 85-99.

[31] PIYARE, Rajeev; TAZIL, M. Bluetooth based home automation system using cell phone. En 2011 IEEE 15th International Symposium on Consumer Electronics (ISCE). IEEE, 2011. p. 192-195.

[32] SIEKKINEN, Matti, et al. How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4. En 2012 IEEE wireless communications and networking conference workshops (WCNCW). IEEE, 2012. p. 232-237.

[33] BULIĆ, Patricio; KOJEK, Gašper; BIASIZZO, Anton. Data transmission efficiency in Bluetooth Low Energy versions. Sensors, 2019, vol. 19, no 17, p. 3746.

[34] PAEK, Jeongyeup; KO, JeongGil; SHIN, Hyungsik. A measurement study of BLE iBeacon and geometric adjustment scheme for indoor location-based mobile applications. Mobile Information Systems, 2016, vol. 2016.

[35] ÁLVAREZ, Flor, et al. Bluemergency: Mediating Post-disaster Communication Systems using the Internet of Things and Bluetooth Mesh. En 2019 IEEE Global Humanitarian Technology Conference (GHTC). IEEE, 2019. p. 1-8.

[36] ADOMNICAI, Alexandre; FOURNIER, Jacques JA; MASSON, Laurent. Hardware security threats against Bluetooth mesh networks. En 2018 IEEE Conference on Communications and Network Security (CNS). IEEE, 2018. p. 1-9.

[37] WIXTED, Andrew J., et al. Evaluation of LoRa and LoRaWAN for wireless sensor networks. En 2016 IEEE SENSORS. IEEE, 2016. p. 1-3.

[38] LAVRIC, Alexandru; PETRARIU, Adrian Ioan. LoRaWAN communication protocol: The new era of IoT. En 2018 International Conference on Development and Application Systems (DAS). IEEE, 2018. p. 74-77.

[39] Want, R. (2011). Near field communication. *IEEE Pervasive Computing*, *10*(3), 4-7.

[40] MEKKI, Kais, et al. Overview of cellular LPWAN technologies for IoT deployment: Sigfox, LoRaWAN, and NB-IoT. En 2018 ieee international conference on pervasive computing and communications workshops (percom workshops). IEEE, 2018. p. 197-202.

[41] JANSSEN, Thomas, et al. Outdoor fingerprinting localization using sigfox. En 2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN). IEEE, 2018. p. 1-6.

[42] GOMEZ, Carles, et al. A Sigfox energy consumption model. Sensors, 2019, vol. 19, no 3, p. 681.

[43] EE, Gee Keng, et al. A review of 6LoWPAN routing protocols. Proceedings of the Asia-Pacific Advanced Network, 2010, vol. 30, p. 71-81.

[44] GARG, Ruchi; SHARMA, Sanjay. A study on need of adaptation layer in 6LoWPAN protocol stack. International Journal of Wireless and Microwave Technologies (IJWMT), 2017, vol. 7, no 3, p. 49-57.

[45] RAMYA, C. Muthu; SHANMUGARAJ, M.; PRABAKARAN, R. Study on ZigBee technology. En 2011 3rd International Conference on Electronics Computer Technology. IEEE, 2011. p. 297-301.

[46] MUTEBA, Franck; DJOUANI, Karim; OLWAL, Thomas. A comparative Survey Study on LPWA IoT Technologies: Design, considerations, challenges and solutions. Procedia Computer Science, 2019, vol. 155, p. 636-641.

[47] FOULADI, Behrang; GHANOUN, Sahand. Security evaluation of the Z-Wave wireless protocol. Black hat USA, 2013, vol. 24, p. 1-2.

[48] (Noviembre de 2020). Obtenido de BLE-STACK Bluetooth Low Energy software stack: http://www.ti.com/tool/BLE-STACK

[49] (Noviembre de 2020). Obtenido de DA14585 IoT MultiSensor Development Kit: https://www.dialog-semiconductor.com/products/da14585-iot-multi-sensor-development-kit

[50] (Noviembre de 2020). Obtenido de STEVAL-MKSBOX1v1 – Sensortile.box wireless multi sensor development kit with user friendly app for IoT and wearable sensor applications: https://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mems-motion-sensor-eval-boards/steval-mksbox1v1.html#overview

[51] (Noviembre de 2020). Obtenido de STEVAL-BCN002V1B – BlueTile – Bluetooth LE enabled sensor node development kit: https://www.st.com/en/evaluation-tools/steval-bcn002v1b.html#overview

[52] (Diciembre de 2020). Obtenido de Developer Study Guide how to deploy bluez on a Raspberry Pi Board as a Bluetooth Mesh Provisioner: https://www.bluetooth.com/wp-content/uploads/2020/04/Developer-Study-Guide-How-to-Deploy-BlueZ-on-a-Raspberry-Pi-Board-as-a-Bluetooth-Mesh-Provisioner.pdf

[53] (Diciembre de 2020). Obtenido de How to Deploy BlueZ v5.50 on Raspberry Pi: https://3pl46c46ctx02p7rzdsvsg21-wpengine.netdna-ssl.com/wp-content/uploads/2019/03/Tutorial-How-to-set-up-BlueZ_Part2-3.pdf

[54] (Diciembre de 2020). Obtenido de Zephyr Project: https://www.zephyrproject.org/

[55] (Diciembre de 2020). Obtenido de WiFi Channels, Frequency Bands & Bandwidths – Electronic Notes: https://www.electronics-notes.com/articles/connectivity/wifi-ieee-802-11/channels-frequencies-bands-bandwidth.php

[56] (Enero de 2021). Obtenido de Installing ThingsBoard CE on Ubuntu Server | ThingsBoard: https://thingsboard.io/docs/user-guide/install/ubuntu/

Anexos.

Anexo 1. Planificación temporal

En la Ilustración 56 se muestran las tareas de las que está compuesta el trabajo realizado durante el desarrollo del proyecto, indicando además su duración, así como las fechas en las que han sido realizadas y el orden cronológico seguido.

DIAGRAMA DE GANTT

	Nombre de tarea	Duración	Comienzo	Fin
1	Definición de la topología de red a desarrollar	9 días	lun 14/09/20	jue 24/09/20
2	Análisis y selección de protocolos de bajo consumo para la comunicación entre módulos multisensor y Gateway/Repetidor	7 días	vie 25/09/20	lun 05/10/20
3	Análisis y selección de protocolos ad-hoc inalámbricos para la comunicación entre Gateway y Repetidores	10 días	mar 06/10/20	lun 19/10/20
4	Selección de la nube a emplear y del protocolo de comunicación con la nube.	6 días	mar 20/10/20	mar 27/10/20
5	Configuración de los Gateway y los Repetidores	11 días	mié 28/10/20	mié 11/11/20
6	Configuración de los módulos multisensores	32 días	jue 12/11/20	sáb 26/12/20
7	Configuración del Sensortag CC2650	9 días	jue 12/11/20	mar 24/11/20
8	Configuración del Smartbond DA14585 IoT	7 días	mié 25/11/20	jue 03/12/20
9	Configuración del Sensortile.box	8 días	vie 04/12/20	mar 15/12/20
10	Configuración del BlueTile	9 días	mié 16/12/20	sáb 26/12/20
11	Configuración de la nube	9 días	lun 28/12/20	jue 07/01/21
12	Análisis de la comunicación entre Gateway y Repetidores	3 días	vie 08/01/21	mar 12/01/21
13	Análisis de la distancia máxima entre Gateway/Repetidor y módulos multisensores	2 días	mié 13/01/21	jue 14/01/21
14	Instalación del prototipo en el Departamento de Informática	5 días	vie 15/01/21	jue 21/01/21
15	Análisis de las comunicaciones entre dispositivos en el prototipo instalado en el Deoartamento de Informática	2 días	vie 22/01/21	lun 25/01/21
16	Documentación del proyecto	98 días	lun 14/09/20	mié 27/01/21

Ilustración 56: Trabajo realizado durante el desarrollo del proyecto

En la Ilustración 57 se muestra el diagrama de Gantt correspondiente.

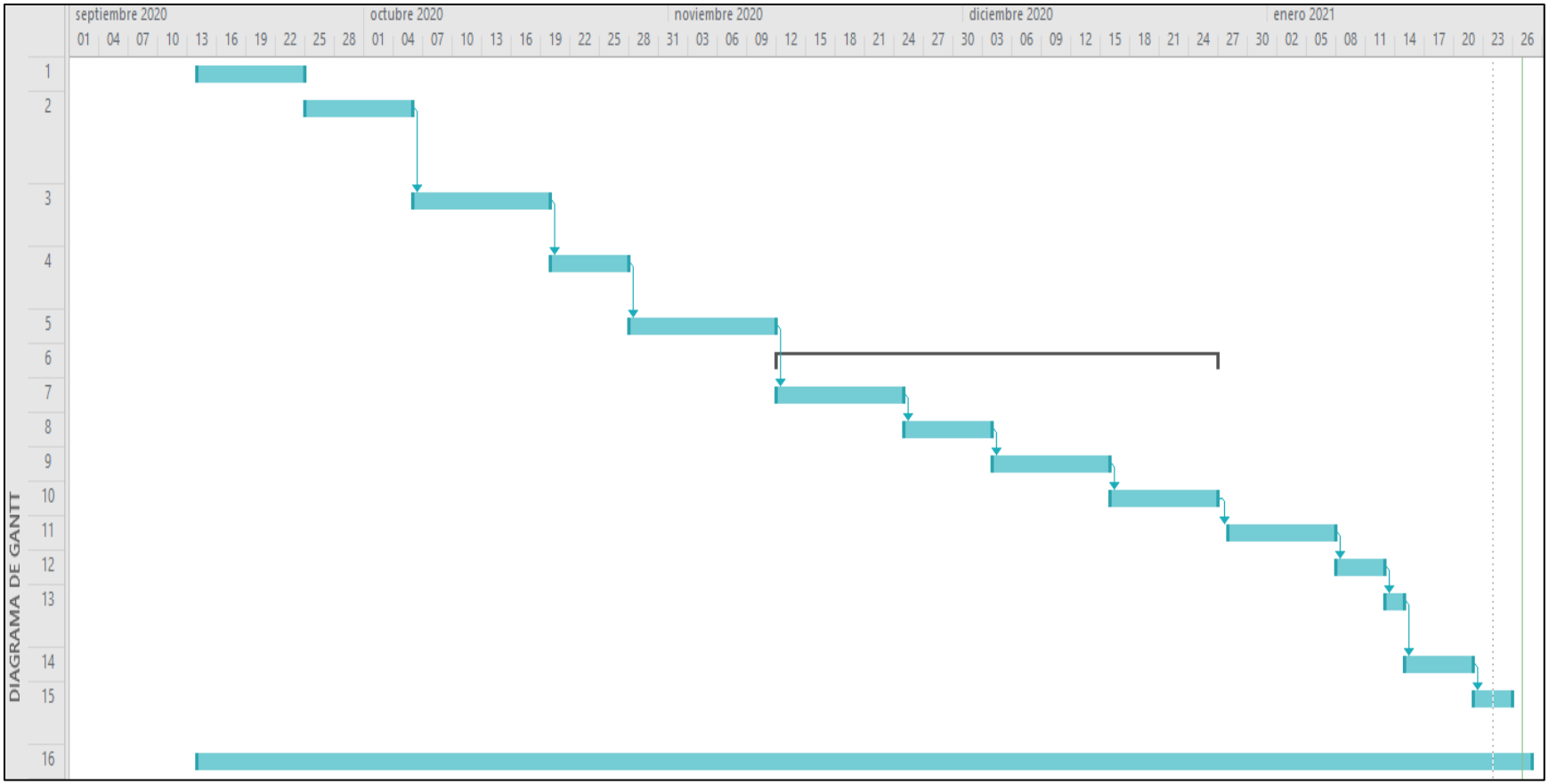

Ilustración 57: Diagrama de Gantt

Anexo 2. Actualización del Firmware de los módulos multisensor

En este anexo se explicará el proceso llevado a cabo para actualizar el firmware de los diferentes módulos multisensor una vez que este se haya generado.

Actualización del Firmware del Sensortag CC2650

Una vez generado el firmware de recolección de datos a emplear por el Sensortag CC2650, es necesario actualizarlo en su memoria Flash. Para ello, se conecta el Sensortag CC2650 al SimpleLink SensorTag Debugger y este a su vez mediante un cable USB al ordenador donde se vaya a ejecutar el software SmartRF Flash Programmer 2, el cual permitirá llevar a cabo la actualización del firmware.

Una vez realizada la conexión, desde el programa SmartRF Flash Programmer 2, se selecciona el archivo .hex correspondiente a los proyectos sensortag_cc2650_app, sobre el que se realizaron las modificaciones pertinentes. También es necesario seleccionar el archivo .hex relativo al proyecto sensortag_cc2650_stk sobre el que no se realizó ninguna modificación. Tras ser seleccionados se escoge el dispositivo que se desea programar, en este caso el módulo multisensor, y se marcan las acciones de borrado, programación y verificación.

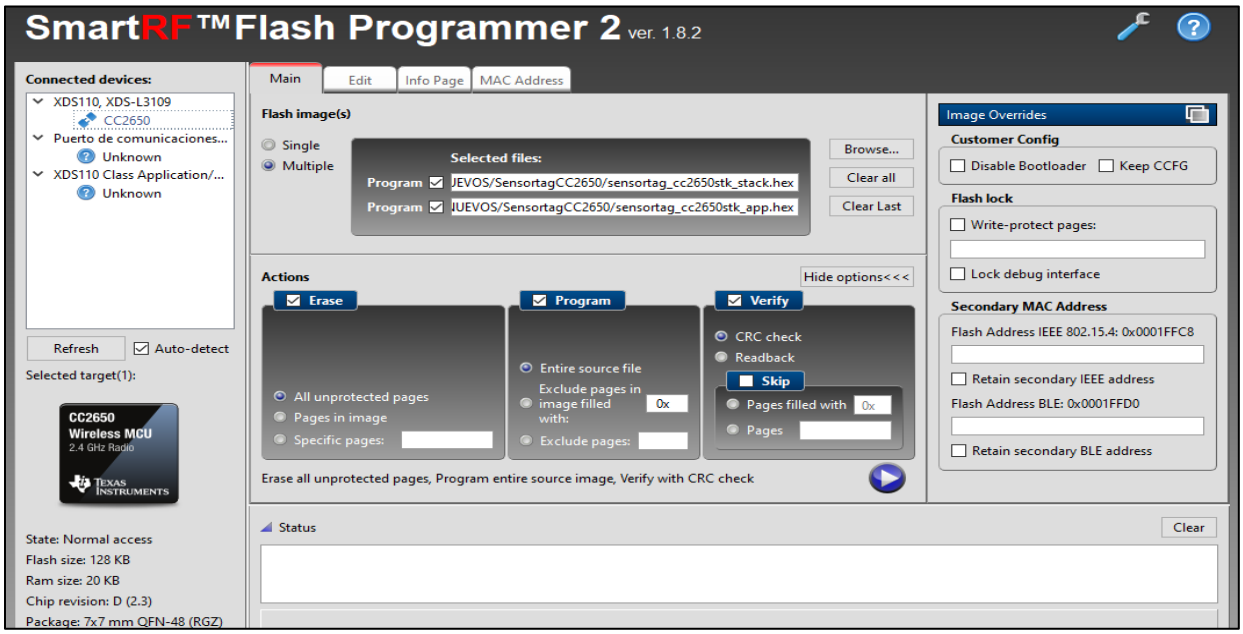

Ilustración 58: Configuración para la actualización del firmware del Sensortag CC2650

Toda esta configuración puede apreciarse en la Ilustración 58. Una vez realizada la configuración, lo siguiente es clicar en el botón play. En la Ilustración 59, se muestran los dos resultados que se pueden obtener una vez se pulsa el botón play, que la grabación en la memoria Flash del firmware se haya realizado exitosamente (a) o que esta se realizase mal y haya provocado algún error (b).

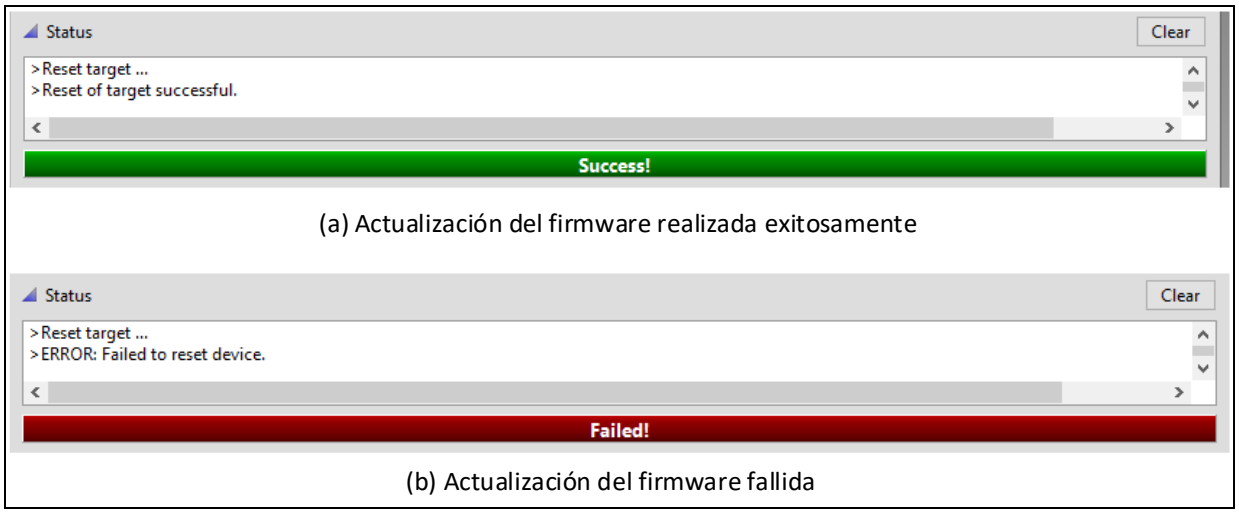

Ilustración 59: Actualización del firmware exitosa (a) y fallida (b) en el Sensortag CC2650

Si la actualización del firmware se ha realizado correctamente, el dispositivo está listo para usar.

Actualización del Firmware del Smartbond DA14585 IoT

La actualización del firmware del Smartbond DA14585 IoT es más complicada en comparación con el resto de los módulos multisensor. Para obtener el archivo .hex correspondiente al firmware a grabar en el módulo multisensor no basta con compilar correctamente el proyecto, sino que es necesario que, una vez se ha compilado este, se seleccione el archivo .hex generado y se mueva a la carpeta utilities/mkimage_utils_scripts. Desde esa carpeta se ejecuta el archivo make_image_iot.bat al que se le pasa como parámetro el nombre del archivo .hex generado previamente. Una vez ejecutado, se obtiene un archivo binario de nombre multi_iot585.bin, que será el empleado a la hora de actualizar el firmware del Smartbond DA14585 IoT.

Posteriormente se conecta el Smartbond DA14585 IoT a la D-SCPINTERFACEBRD y esta, a su vez, al ordenador donde se ejecuta el software SmartSnippets Toolbox v5.0.10. Se creará una configuración por defecto en la que se indica su nombre, en este caso será da14585, el JTAG

empleado y el modelo de módulo multisensor usado (DA14585). Toda esta configuración puede apreciarse en la Ilustración 60.

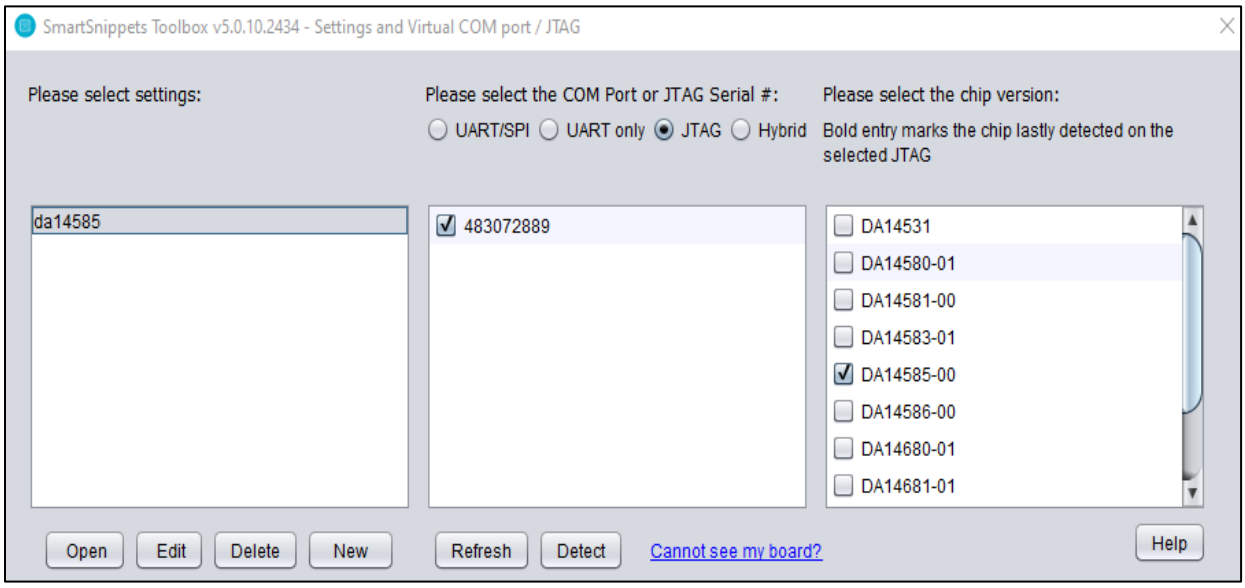

Ilustración 60: Configuración para actualización del firmware del Smartbond DA14585 IoT

Después se clica en el botón Open, el cual abre una interfaz gráfica que permite realizar muchas acciones sobre el módulo multisensor, entre las que cabe destacar, una actualización de la memoria Flash del dispositivo o una estimación de la duración de la batería en base a los parámetros configurados en el firmware.

Para llevar a cabo la actualización del firmware, es necesario configurar la placa empleada, accediendo a la opción Board Setup y seleccionando el puerto UART y la velocidad de transmisión empleada. En este caso, se corresponde con el puerto P0_4, P0_5 con una velocidad de transmisión de 57600 Bd.

Posteriormente se debe indicar cuales son los pines empleados por la SPI Flash, marcando que el pin empleado para SPI_CLK es el P0_0, para SPI_EN es el P0_3, para SPI_DI es el P0_5 y para SPI_DO es el P0_6. Por último, hay que indicar la configuración seguida por la I2C Eeprom, marcando que el pin empleado para SCL se corresponde con el P0_2 y el usado para SDA se corresponde con el P0_3. Toda esta configuración se muestra en la Ilustración 61.

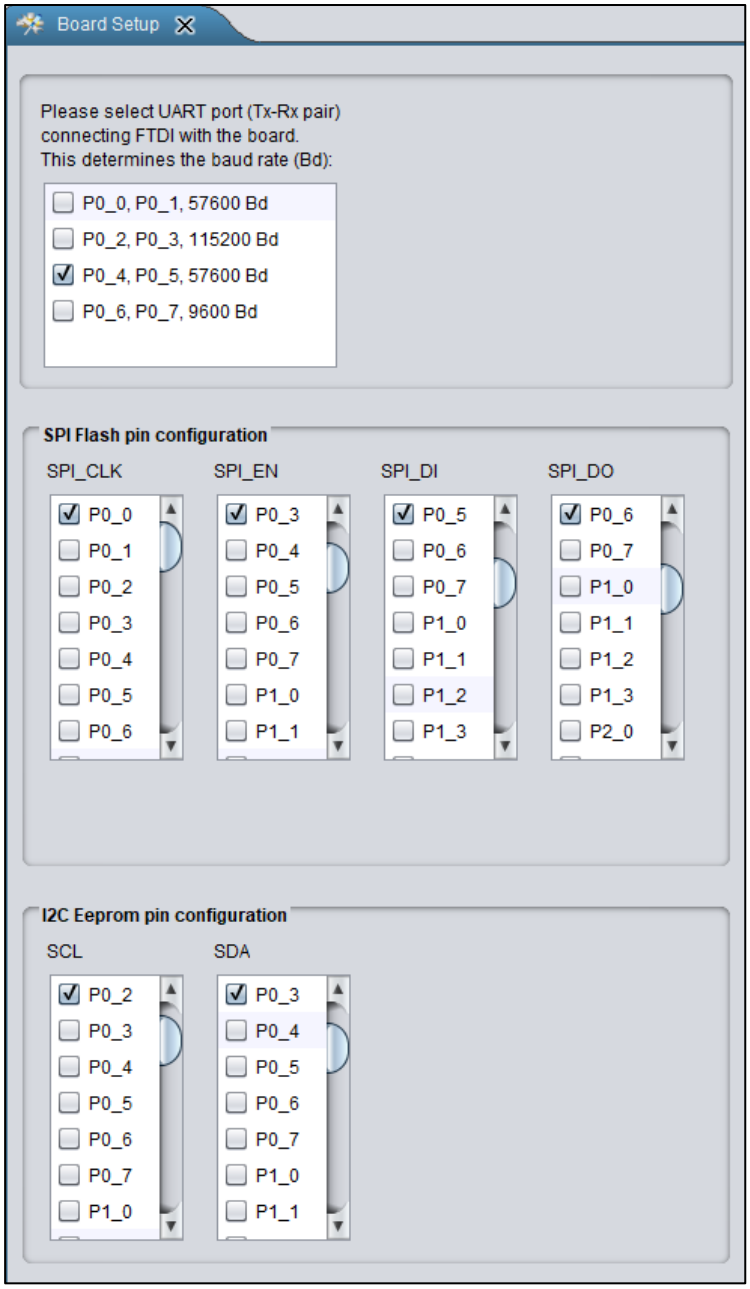

Ilustración 61: Configuración de pines del Smartbond DA14585 IoT

Posteriormente, debe seleccionarse la opción SPI Flash Programmer, donde se cargará el firmware que se va a grabar clicando en el botón "Load hex/bin file" y seleccionando exclusivamente la opción "Do not modify", dejando sin seleccionar la opción "Bootable". El resto se deja tal y como se muestra en la Ilustración 62 y se clica primero en el botón "Connect" y después en "Burn & Verify".

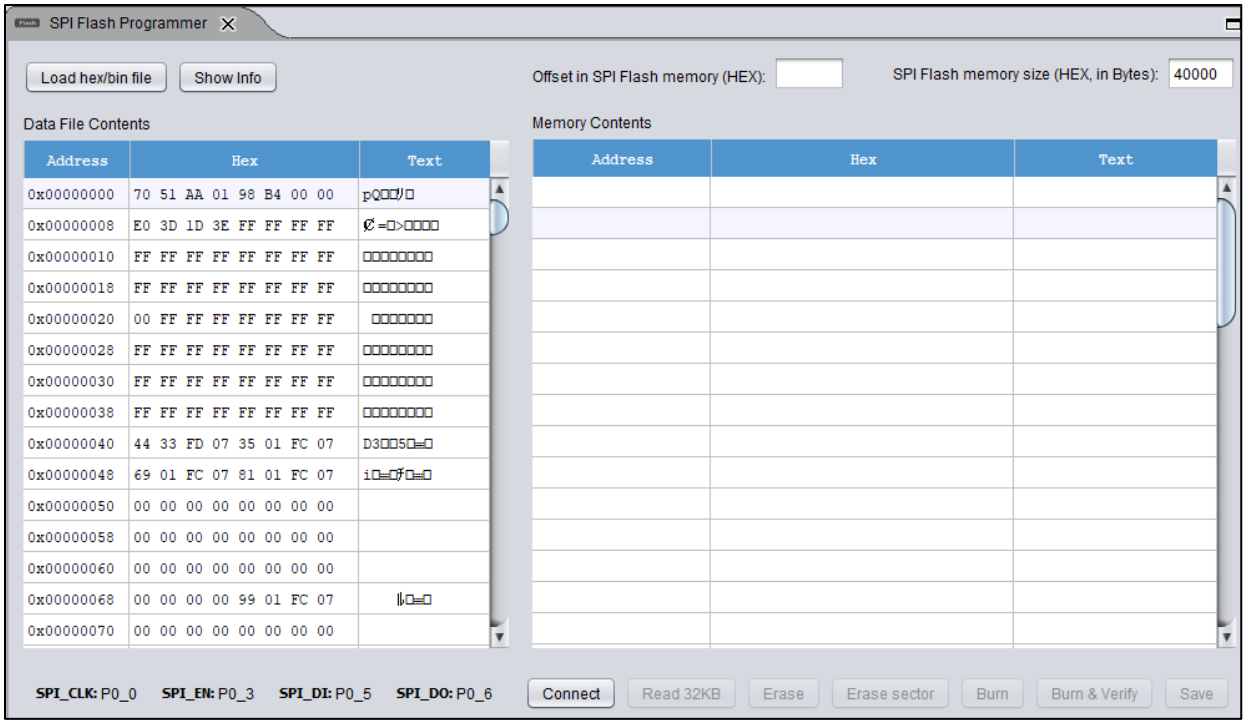

Ilustración 62: Actualización del firmware del Smartbond DA14585 IoT

Una vez grabado y verificado, para aplicar los cambios en la memoria Flash del módulo multisensor, es imprescindible pulsar el botón de Reset de la placa D-SCPINTERFACEBRD. Tras realizarlo se podrá apreciar como los leds del módulo multisensor se encuentra parpadeando, lo que significa que la actualización del firmware se realizó correctamente.

Actualización del Firmware del Sensortile.box

La instalación del nuevo firmware en el Sensortile.box se lleva a cabo empleando el software STM32 Cube Programmer. Lo primero es conectar el Sensortile.box al ST LINK V2 y este a su vez al ordenador. Una vez hecho esto, se ha de comprobar que este instalada la versión más reciente del firmware del ST LINK V2, clicando en el botón “Firmware Update”, el cual muestra la versión actual del firmware y si esta puede ser actualizada. En caso de poder ser actualizada debe hacerse para que todo funcione correctamente.

Una vez actualizada, es necesario configurar la forma de conexión con el Sensortile.box a través del ST LINK V2, seleccionando el número de serie del ST LINK V2 a utilizar, el puerto a emplear, en este caso el Serial Wire Debug (SWD), una frecuencia de 4000 kHz, modo normal y

lo más importante, un reinicio completo del Hardware del dispositivo. Toda esta configuración se muestra en la Ilustración 63.

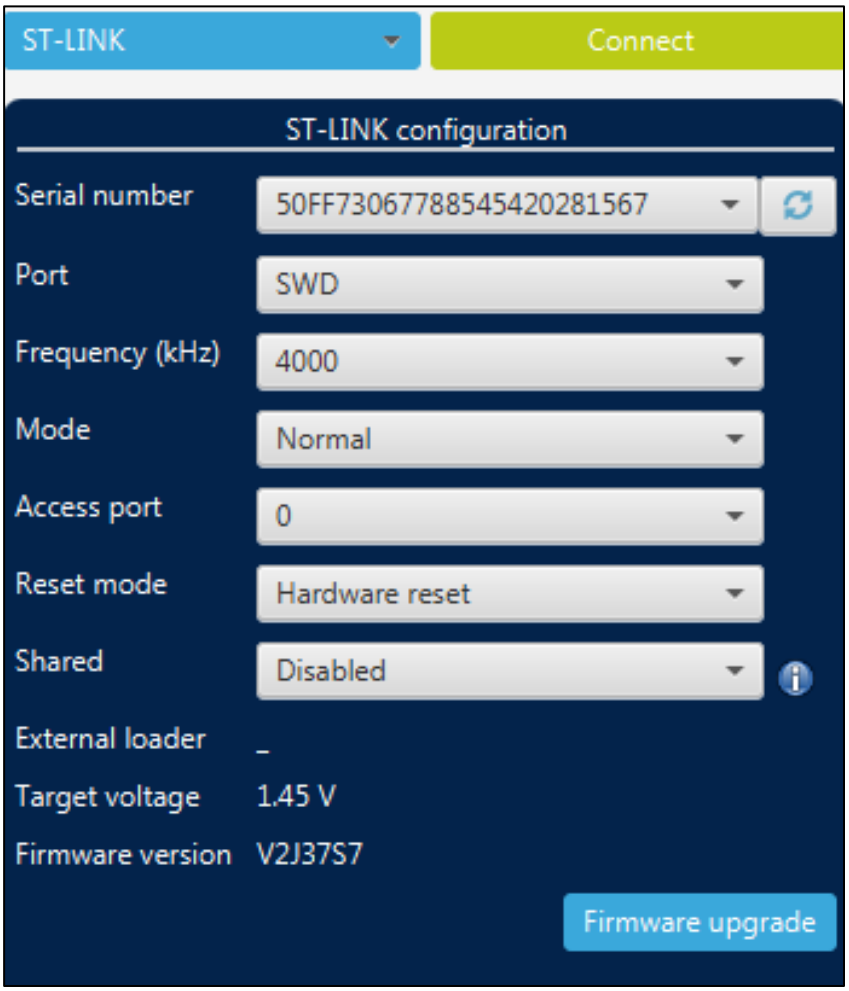

Ilustración 63: Configuración para la actualización del firmware del Sensortile.box

Después de realizada esta configuración, se pulsa sobre el botón "Connect" para iniciar la conexión con el módulo multisensor y se clica sobre el segundo botón del panel izquierdo para abrir la ventana correspondiente a la acción "Erasing & programming", que permite borrar y reprogramar la memoria Flash del Sensortile.box. Tras esto aparece la interfaz mostrada en la Ilustración 64. Se debe seleccionar el archivo binario correspondiente al firmware del módulo multisensor y una vez seleccionado marcar las opciones "Verify programming" y "Run after programming". Tras ello clicar sobre el botón "Start programming" que inicia la actualización del firmware del módulo multisensor.

Ilustración 64: Actualización del firmware del Sensortile.box

Finalmente, si la actualización del firmware se ha realizado de forma correcta, aparece un aviso indicándolo y el led del Sensortile.box comienza a parpadear.

Actualización del Firmware del BlueTile

El proceso llevado a cabo para actualizar el firmware del BlueTile es realmente rápido y sencillo, aunque en ocasiones presenta un problema en el que el módulo multisensor no se actualiza de forma correcta. Este problema puede detectarse cuando, una vez se ha instalado el firmware, el BlueTile presenta una luz completamente blanca en su led, lo que indica que la actualización no se ha realizado bien. En este caso en concreto, el archivo .hex que hay que grabar inicialmente para que funcione bien es el BLE_SensorDemo (with OTA ResetManager).hex. Después se vuelve a grabar el firmware generado durante este trabajo y se puede apreciar cómo, en lugar de una luz blanca, se muestra una luz azul lo que indica que está bien actualizado.

El proceso seguido para actualizar el firmware es el siguiente. En primer lugar, se conecta el BlueTile a la placa STEVAL-BCN002V1D y esta a su vez al ordenador desde el que se ejecuta el software de actualización de firmware BlueNRG-X Flasher Utility. Desde el programa, se selecciona el firmware a grabar en la memoria Flash del módulo multisensor, así como las acciones que se desean realizar, en este caso, verificar y actualizar la memoria del dispositivo. Por último, se selecciona el puerto COM correspondiente al STEVAL-BCN002V1D y se clica en el botón “Flash”. Toda esta configuración se muestra en la Ilustración 65.

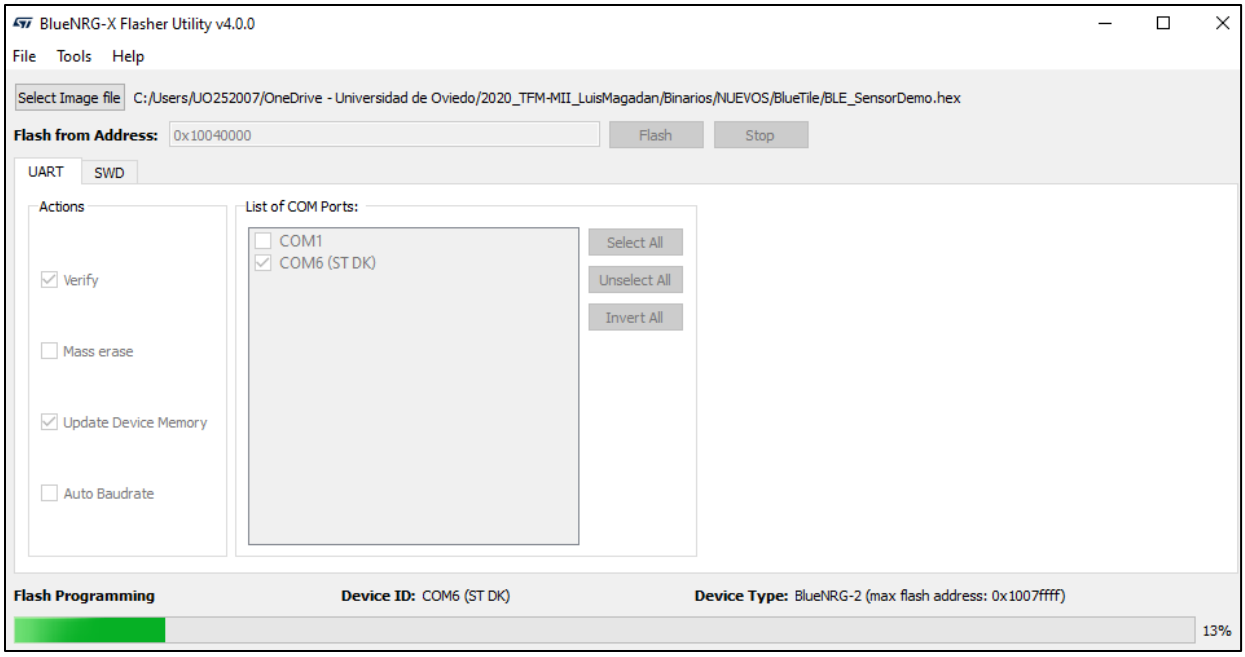

Ilustración 65: Actualización del firmware del BlueTile

Anexo 3. Script Python final de recolección y envío de datos a la nube

A continuación, se muestra el contenido del script de Python empleado finalmente para recolectar la información de los diferentes módulos multisensor y enviarla posteriormente a la plataforma IoT. Para cada uno de los módulos multisensor se debe pasar su dirección MAC, el AccessToken asignado en la plataforma IoT y un número que equivale al modelo del módulo multisensor siendo el 0 el Sensortag CC2650, el 1 el Smartbond DA14585, el 2 el Sensortile.box y finalmente el 3 el BlueTile.

```
from bluepy import btle
import struct
import time
import json
import subprocess
import binascii

ON = struct.pack("B",0x01)
OFF = struct.pack("B",0x00)
_NOTIFICATION_ON = struct.pack("BB", 0x01, 0x00)
_NOTIFICATION_OFF = struct.pack("BB", 0x00, 0x00)

# Sensortag CC2650 auxiliar functions
# ReadHumidity() -> Reads the data obtained by the humidity sensor
def ReadHumidity(p):
    global hum
    services = p.getServiceByUUID("f000aa20-0451-4000-b000-000000000000")
    charact = services.getCharacteristics()
    charactCtrl = charact[1]
    charactData = charact[0]
    charactCtrl.write(ON, withResponse=True)
    time.sleep(5)
    (temp, hum) = struct.unpack('<HH',charactData.read())
    hum = 100.0 *(hum/65536.0)
    charactCtrl.write(OFF, withResponse=False)

# ReadBattery() -> Returns the amount of battery of the device
def ReadBattery(p):
    global bat
    services = p.getServiceByUUID("0000180f-0000-1000-8000-00805f9b34fb")
    charact = services.getCharacteristics()[0]
    bat = ord(charact.read())

# ReadBarometer() -> Returns pressure and temperature both from the barometer sensor
def ReadBarometer(p):
    global temp, pres
    services = p.getServiceByUUID("f000aa40-0451-4000-b000-000000000000")
    charact = services.getCharacteristics()
    charactCtrl = charact[1]
    charactData = charact[0]
    charactCtrl.write(ON, withResponse=True)
    time.sleep(5)
    (tempL, tempM, tempH, presL, presM, presH) = struct.unpack('<BBBBBB',charactData.read())
    temp = (tempH*65536 + tempM*256 + tempL)/100.0
    pres = (presH*65536 + presM*256 + presL)/100.0
    charactCtrl.write(OFF, withResponse=True)

# ReadOptical() -> Returns the brightness level of the environment
def ReadOptical(p):
    global opt
    services = p.getServiceByUUID("f000aa70-0451-4000-b000-000000000000")
```

```
        charact = services.getCharacteristics()
        charactCtrl = charact[1]
        charactData = charact[0]
        charactCtrl.write(ON, withResponse=True)
        time.sleep(5)
        raw = struct.unpack('<h',charactData.read())[0]
        m = raw & 0xFFF;
        e = (raw & 0xF000) >> 12;
        opt = 0.01 * (m<<e)
        charactCtrl.write(OFF, withResponse=False)

    # Smartbond DA14585 IoT auxiliar functions
    # EnableNotifications() -> Enables the notifications from the sensor to be sent with new data
    def EnableNotifications(D):
        service = D.getServiceByUUID("2ea78970-7d44-44bb-b097-26183f402400")
        charactData = service.getCharacteristics("2ea78970-7d44-44bb-b097-26183f402410")[0]
        charactData_handle = charactData.getHandle()
        D.writeCharacteristic(charactData_handle + 1,b"\x01\00", True)

    # DisableNotifications() -> Disables the notifications from the sensor
    def DisableNotifications(D):
        service = D.getServiceByUUID("2ea78970-7d44-44bb-b097-26183f402400")
        charactData = service.getCharacteristics("2ea78970-7d44-44bb-b097-26183f402410")[0]
        charactData_handle = charactData.getHandle()
        D.writeCharacteristic(charactData_handle + 1,b"\x00\00", True)

    # EnableSensors() -> Enables all the sensors used (temperature, humidity, optical and pressure)
    def EnableSensors(D):
        service = D.getServiceByUUID("2ea78970-7d44-44bb-b097-26183f402400")
        charactControl = service.getCharacteristics("2ea78970-7d44-44bb-b097-26183f402409")[0]
        charactControl.write(b"\x01", True)

    # DisableSensors() -> Disable the sensors used
    def DisableSensors(D):
        service = D.getServiceByUUID("2ea78970-7d44-44bb-b097-26183f402400")
        charactControl = service.getCharacteristics("2ea78970-7d44-44bb-b097-26183f402409")[0]
        charactControl.write(b"\x00", True)

    # Sensortile.box auxiliar functions
    # ReadData() -> Enable notifications and wait until data is received
    def ReadData(p):
        global bat, pres, hum, temp
        services=p.getServiceByUUID("00000000-0001-11e1-9ab4-0002a5d5c51b")
        characteristicBat = services.getCharacteristics("00020000-0001-11e1-ac36-0002a5d5c51b")[0]
        p.writeCharacteristic(characteristicBat.valHandle+1, _NOTIFICATION_ON)
        p.waitForNotifications(100.0)
        p.writeCharacteristic(characteristicBat.valHandle+1, _NOTIFICATION_OFF)
        return

    # BlueTile auxiliar functions
    # ReadEnvironmental() -> Read all the data received from the environmental sensor
    def ReadEnvironmental(p):
        services=p.getServiceByUUID("00000000-0001-11e1-9ab4--0002a5d5c51b")
        charEnv = services.getCharacteristics("001c0000--0001-11e1-ac36-0002a5d5c51b")[0]
        p.writeCharacteristic(charEnv.valHandle+1, _NOTIFICATION_ON)
        p.waitForNotifications(10.0)
        p.writeCharacteristic(charEnv.valHandle+1, _NOTIFICATION_OFF)

    # Classes MyDelegate() for the different devices used
    # MyDelegateSmartbond() -> waits and returns for the data from the Smartbond DA14585
    class MyDelegateSmartbond(btle.DefaultDelegate):
        def handleNotification(self, cHandle, data):
            global h, p, t, l
            raw_data = binascii.hexlify(data)
            raw_data = str(raw_data)[2:-1]
            sensor_report = raw_data[4:]
            if sensor_report.find("060203")==0:
                temp_data = sensor_report[6:14]
                aux = temp_data[6:]+temp_data[4:6]+temp_data[2:4]+temp_data[:2]
```

```
121                 t = int(aux,16)
122                 t/= 100.0
123             if sensor_report.find("040203")==0:
124                 pres_data = sensor_report[6:14]
125                 aux = pres_data[6:]+pres_data[4:6]+pres_data[2:4]+pres_data[:2]
126                 p = int(aux,16)
127                 p/= 100.0
128                 hum_data = sensor_report[20:28]
129                 aux = hum_data[6:]+hum_data[4:6]+hum_data[2:4]+hum_data[:2]
130                 h = int(aux,16)
131                 h/=1024.0
132             if sensor_report.find("090203")==0:
133                 lux_data = sensor_report[6:14]
134                 aux = lux_data[6:]+lux_data[4:6]+lux_data[2:4]+lux_data[:2]
135                 l = int(aux,16)
136                 l/=4.0
137 
138     # MyDelegateSensortile() -> waits and returns the data from the Sensortile.box
139     class MyDelegateSensortile(btle.DefaultDelegate):
140         def handleNotification(self, cHandle, data):
141             global bateria, presion, humedad, temperatura
142             if cHandle == 20:
143                 bateria, presion, humedad, temperatura = struct.unpack('<hihh',data)
144                 bateria/= 10.0
145                 presion/=100.0
146                 humedad/=10.0
147                 temperatura/=10.0
148 
149     # MyDelegateBluetile() -> waits and returns the data from the Bluetile
150     class MyDelegateBluetile(btle.DefaultDelegate):
151         def handleNotification(self, cHandle, data):
152             global presion1, humedad1, temperatura1, bateria1
153             if (cHandle == 14):
154                 data = struct.unpack('<hihhh', data)
155                 presion1 = data[1]/100.0
156                 humedad1 = data[2]/10.0
157                 temperatura1 = data[3]/10.0
158                 bateria1 = data[4]/10.0
159 
160     # Main function called with the mac address of the device, the access token and model
161     # Model: 0 ->  Sensortag CC2650 // 1 -> Smartbond DA14585 // 2 -> Sensortile.box // 3 -> BlueTile
162     def base(mac, accessToken, modelo):
163 
164         if (modelo == 0):
165             global temp, pres, opt, hum, bat
166             print('\nConectando a ' + mac)
167             D = btle.Peripheral(mac)
168             print("Sensor conectado")
169             ReadBarometer(D)
170             ReadHumidity(D)
171             ReadOptical(D)
172             ReadBattery(D)
173             jsondata=json.dumps({"temperatura":temp,"humedad":hum,"presion":pres,"luz":opt,"bat":bat})
174             subprocess.run(['bash','./SendToThingsboard.sh',accessToken,jsondata])
175             D.disconnect()
176             print("Sensor desconectado")
177 
178         elif (modelo == 1):
179             global h, p, t, l
180             h = -1
181             p = -1
182             l = -1
183             t = -1
184             print("\nConectando a " + mac)
185             D = btle.Peripheral(mac)
186             print("Sensor conectado")
187             D.setDelegate(MyDelegateSmartbond())
188             EnableNotifications(D)
189             EnableSensors(D)
```

```
190             while (l==-1 or p==-1 or h==-1 or t==-1):
191                 D.waitForNotifications(5)
192             jsondata = json.dumps({"temperatura":t,"humedad":h,"presion":p,"luz":l})
193             subprocess.run(['bash','./SendToThingsboard.sh',accessToken,jsondata])
194             DisableSensors(D)
195             DisableNotifications(D)
196             D.disconnect()
197             print("Sensor desconectado")
198 
199         elif (modelo == 2):
200             global presion, humedad, temperatura, bateria
201             print("\nConectando a " + mac)
202             D = btle.Peripheral(mac,"random")
203             print("Sensor conectado")
204             D.setDelegate(MyDelegateSensortile())
205             ReadData(D)
206             jsondata =
                json.dumps({"temperatura":temperatura,"humedad":humedad,"presion":presion,"bat":bateria})
207             subprocess.run(['bash','./SendToThingsboard.sh',accessToken,jsondata])
208             D.disconnect()
209             print("Sensor desconectado")
210 
211         elif (modelo == 3):
212             global presion1, humedad1, temperatura1, bateria1
213             print("\nConectando a " + mac)
214             D = btle.Peripheral(mac,"random")
215             print("Sensor conectado")
216             D.setDelegate(MyDelegateBluetile())
217             ReadEnvironmental(D)
218             time.sleep(5)
219             jsondata =
             json.dumps({"temperatura":temperatura1,"humedad":humedad1,"presion":presion1,"bat":bateria1})
220             subprocess.run(['bash','./SendToThingsboard.sh',accessToken,jsondata])
221             D.disconnect()
222             print("Sensor desconectado")
223 
224     if __name__ == "__main__":
225         while True:
226             try:
227                 base('54:6C:0E:4D:3C:81','WueWopFaCz6J1ML5Lv3Q',0) # Sensortag CC2650
228             except:
229                 print("Error")
230             try:
231                 base('80:EA:CA:70:B1:66','Qlvml0pRfVCgZfKakxLh',1) # Smartbond DA14585
232             except:
233                 print("Error")
234             try:
235                 base('DA:03:0B:76:73:74','1XuDo0N4aw3IbccTFRVJ',2) # Sensortile.box
236             except:
237                 print("Error")
238             try:
239                 base('F6:CD:F2:B1:7F:77','UkIM9or9y8AUkhr5OoEy',3) # BlueTile
240             except:
241                 print("Error")
242             time.sleep(360)
```