UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Módulo de gestión de red Bluetooth MESH para dispositivo móvil y sistema empotrado

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

AUTOR/A: Zhang , Zhihao

Tutor/a: Blanes Noguera, Juan Francisco

CURSO ACADÉMICO: 2021/2022

Resumen

La tecnología de comunicaciones inalámbrica BLUETOOTH está implantada desde hace varios años gracias al empuje proporcionado por la adopción en terminales de telefonía móvil debido principalmente a su bajo consumo energético necesario para compatibilizar su uso con la alimentación a través de baterías. Las principales plataformas utilizadas en los terminales smartphone (ANDROID e IOS) han incorporado su uso de forma nativa lo que ha permitido la conectividad entre distintos dispositivos tales como automóviles o todo tipo de periféricos inalámbricos. La tecnología BLUETOOTH hace uso de una banda de radiofrecuencia ubicada en el rango de 2,4GHz lo que reduce la distancia permitida de comunicación entre dispositivos, pensada para una decena de metros. Una de las estrategias diseñadas para incrementar dichas distancias es la conformación de una red MESH. Esta aproximación se basa en la creación de una infraestructura de comunicaciones por parte de todos los nodos presentes en la instalación. Esta red de comunicaciones permite el enrutamiento MULTICAST de la información así como la comunicación entre los distintos nodos que conforman la red. Con esto, se permite la comunicación entre 2 nodos situados a una distancia superior a la que permite la comunicación directa gracias al uso de la red generada por los nodos que hacen la función de enrutadores. La tecnología BLE MESH (Bluettooth Low Energy MESH) fue propuesta en 2014 siendo implementada por parte de los fabricantes de integrados electrónicos en 2017. Esta implementación se ha limitado a la distribución de una serie de librerías de aplicación en kits de laboratorio sin que se haya implantado de forma amplia en ningún producto de consumo por la complejidad subyacente en el uso de dichas librerias. El objetivo del proyecto es el desarrollo de una capa de gestión de red MESH Bluetooth que facilite el desarrollo de aplicaciones tanto en dispositivo móvil (Android o iOS) como en microcontroladores, de forma que se facilite la incorporación de esta tecnología en el IoT industrial.

Palabras clave: Bluetooth low energy mesh, ble mesh, Typescript, IoT

Abstract

BLUETOOTH wireless communications technology has been in use for several years thanks to its adoption in cell phone terminals, mainly due to its low energy consumption necessary to make its use compatible with battery power supply. The main platforms used in smartphone terminals (ANDROID and IOS) have incorporated its use natively which has allowed connectivity between different devices such as cars or all kinds of wireless peripherals. BLUETOOTH technology makes use of a radio frequency band located in the 2.4GHz range, which reduces the communication distance between devices, about ten meters. One of the strategies designed to increase these distances is the creation of a MESH network. This approach is based on the creation of a communications infrastructure by all the nodes present in the installation. This communications network allows MULTICAST routing of information as well as communication between the different nodes that make up the network. This allows communication between 2 nodes located at a greater distance than direct communication thanks to the use of the network generated by the nodes that act as routers. The BLE MESH (Bluettooth Low Energy MESH) technology was proposed in 2014 being implemented by electronic integrated manufacturers in 2017. This implementation has been limited to the distribution of a series of application libraries in lab kits without being widely implemented in any consumer product due to the underlying complexity in the use of these libraries. The objective of the project is the development of a Bluetooth MESH network management layer that facilitates the development of applications in both mobile devices (Android or iOS) and microcontrollers, in order to facilitate the incorporation of this technology in the industrial IoT.

Key words: Bluetooth low energy mesh, ble mesh, Typescript, IoT

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

La tecnología de comunicaciones inalámbrica BLUETOOTH está implantada desde hace varios años gracias al empuje proporcionado por la adopción en terminales de telefonía móvil debido principalmente a su bajo consumo energético necesario para compatibilizar su uso con la alimentación a través de baterías. Las principales plataformas utilizadas en los terminales smartphone (ANDROID e IOS) han incorporado su uso de forma nativa lo que ha permitido la conectividad entre distintos dispositivos tales como automóviles o todo tipo de periféricos inalámbricos.

Las personas suelen pensar que todos los Bluetooth son iguales, en realidad existen varios tipos de Bluetooth.

En el año 1998, Ericsson, IBM, Intel, Nokia, y Toshiba formaron the Bluetooth Special Industry Group(SIG) para redactar la especificación de la tecnología Bluetooth. A partir de este año, Bluetooth ha evolucionado de la versión 1.0 a 5.3. El cambio más importante en la tecnología Bluetooth es la versión 4.0. A partir de esta versión, Bluetooth se puede dividir en 2 tipos: Bluetooth Classic(BR/EDR) y Bluetooth Low Energy(BLE).

El primer tipo se utiliza para transmitir flujos de datos como audio y el segundo tipo se usa para transmitir pequeñas operaciones.

Bluetooth no sólo se utiliza en los móviles o tabletas, sino también se puede usar para los dispositivos IoTs(Internet de las cosas). En comparación con las otras tecnologías de la comunicación, Bluetooth tiene las siguientes ventajas:

- Usa la banda de frecuencias libres, es decir, es gratis a la hora de transmitir y recibir.
- Permite controlar dispositivos cotidianos, tales como tabletas, móviles, portátiles, etc.

La desventaja de esta tecnología es que tiene poco alcance, pero eso se puede resolver montando una red mesh con el fin de retransmitir los mensajes. Este trabajo se centra en crear y gestionar una red mesh usando Bluetooth Low Energy Mesh.

Bluetooth Mesh es una tecnología lanzada en el año 2017 y está diseñada especialmente para los dispositivos IoTs. En la capa física de transmisión, BLE Mesh usa la misma tecnología que BLE y gracias a esta característica, los dispositivos que soportan BLE se pueden comunicar con BLE Mesh.

Ble Mesh permite la conexión de todos los dispositivos entre ellos. Distinto al internet que usamos, la red Mesh no tiene un enrutador para decidir dónde se envían los mensajes, sino que cada nodo difunde el mensaje a su alrededor hacia todos los demás. Esto permite una conexión fiable, ya que todos los nodos pueden enviar a todos. La red mesh también tiene técnicas para no saturar la red, tal como poner tiempo de vida a cada mensaje o la lista temporal para guardar mensaje ya transmitido. En este trabajo se van a estudiar todas estas características.

1.1 Motivación

Bluetooth es una tecnología conocida y además se usa en muchos dispositivos. Según lo que se indica en la página oficial de Bluetooth, en el año 2026 se financiarán 7 miles de millones de dispositivos que tienen Bluetooth, por lo tanto, esto quiere decir que el mercado de Bluetooth es grande.

La tecnología BLE MESH fue propuesta en 2014 siendo implementada por parte de los fabricantes de integrados electrónicos en 2017. Esta implementación se ha limitado a la distribución de una serie de librerías de aplicación en kits de laboratorio sin que se haya implantado en ningún producto de consumo. Por ello, a pesar de que existe tanto en España como en el extranjero distintas empresas

que llevan implantando la tecnología BLE en distintos mercados durante los últimos años, no hemos podido encontrar todavía otra empresa que haya realizado una solución basada en esta tecnología aplicada al campo de la maquinaria. En la actualidad, las soluciones basadas en tecnología BLE utilizan la conexión directa entre nodos sin la capacidad de compartir información ni de enrutamiento del flujo de datos dirigido a de otros destinatarios.

Bluetooth es una tecnología madurada, no es una tecnología perfecta, pero tiene características destacadas de las cuales sacar rendimiento en el entorno industrial. Además, Bluetooth Mesh es una tecnología más o menos nueva. Durante el desarrollo, no se han podido encontrar librerías ya hechas para gestionar la capa baja en Javascript y por este motivo se ha decidido desarrollar una. De esta forma el mismo código se puede ejecutar en cualquier dispositivo que tenga un navegador, como, por ejemplo, los dispositivos móviles o tabletas.

Con la ayuda de la empresa con la que se ha colaborado para el proyecto (Dismuntel), ha sido posible el desarrollo usando los módulos de Silicon Lab.

Existen varias formas de implementar Bluetooth Mesh. Para entender mejor el funcionamiento de Bluetooth Mesh y desarrollar la librería propia se ha tomado como ejemplo la utilización del SDK de Silicon Lab.

1.2 Objetivos

El objetivo de este trabajo es desarrollar un conjunto de librerías para facilitar el desarrollo e integración de aplicaciones de BLE Mesh que se puedan ejecutar sobre Cordova para los dispositivos móviles y en el microcontrolador EFR32BG13 de Silicon Lab. Este trabajo se enmarca dentro de una colaboración de mayor alcance con la empresa Dismuntel, motivo por el que se han utilizado los sistemas de Silicon Lab. Este objetivo global del trabajo se puede subdividir en 3 puntos:

1. Desarrollar una librería de Bluetooth Mesh para dispositivo móvil que cumpla con la especificación, que permita la incorporación a aplicaciones fácilmente independientemente de la plataforma (iOS o Android).

2. Desarrollar una librería para facilitar el uso de la red en aplicaciones IoT sobre microcontrolador,que cumpla con los requerimientos de bajo consumo de la empresa. Esta librería también tiene que ser fácil de usar y fácil de modificar, ya que lo que se va a desarrollar es una primera versión del trabajo.

3. Validar los desarrollos sobre dispositivo real, y específicamente sobre los sistemas empotrados de control a emplear por la empresa.

1.3 Estructura de la memoria

Esta memoria está compuesta por 6 capítulos. El primero es la introducción, que se explica qué es lo que se va a desarrollar en este trabajo. Se sigue con el estado del arte, este capítulo está dividido en dos partes. Por una parte, se analizan las tecnologías de la malla que existen para IoT, y por otra parte se busca los productos comerciales de Bluetooth Mesh.

El capítulo 3 es el análisis, donde se estudia la especificación de Bluetooth Mesh, qué se puede hacer con la red, cómo se usa y cómo se implementa la red. El capítulo 4 es el diseño y la implementación del trabajo, aquí se explica la estructura de la librería, su organización y las funcionalidades que se han conseguido.

Posteriormente se sigue con el capítulo 5 que es prueba, en este capítulo se explican las pruebas que se han realizado para la librería.

Finalmente hay una sección de conclusión, donde se analizan los objetivos que se han conseguido y cómo.

CAPÍTULO 2

Estado del arte

Se podría definir la tipología de red como la forma en que se conectan las computadoras para intercambiar datos entre sí.

Existen varias tipologías de red, tales como malla, bus, estrella, punto a punto, etc. Este apartado se centrará en la tipología malla.

La principal característica de la topología malla es que existen varias rutas para enrutar un mensaje, es decir, la información puede viajar en diferentes caminos de manera que cuando un nodo se cae por cualquier razón, la red tiene la funcionalidad de autoconfigurarse para reformar la red. El proceso de autoconfiguración suele requerir pocos costes.

Existen dos formas de transmitir la información por una red mesh,

- Enrutamiento: es la forma que utiliza internet. Depende del destino, el nodo envía un mensaje hacía un nodo u otro.

- Inundación: esta forma envía el mensaje hacía todos los nodos que hay alrededor. Es una forma simple, ya que no tiene que gestionar una lista de cómo se enrutan los mensajes, pero su principal problema es que satura la red. BLE MESH usa esta forma de transmitir los mensajes, pero ha aplicado unas técnicas para no saturar la red. Estas técnicas se estudiarán más adelante.

A continuación, se van a exponer distintas tecnologías que pueden formar una red malla para los dispositivos IoT.

2.1 Tenología inalámbrica de red en malla

2.1.1. Zigbee

Zigbee es una tecnología que se caracteriza por el bajo consumo y baja tasa de transferencia de datos. Se usa principalmente en hogares y aplicacoiones IoT. El estándar es el IEEE 802.15.4[1] y también usa la banda ISM en la industria, pero las empresas diseñan prácticamente sobre 2.4 Ghz, ya que es una banda de frecuencia libre en todo el mundo.

Zigbee, cuando forma una red malla, elige enrutamiento como forma de transmitir los mensajes. Un nodo dentro de una red ZigBee puede tener diferentes roles: coordinador, enrutador y dispositivo final.

El coordinador: es único en la red y su trabajo es la gestión de la red y decidir cómo se enrutan los mensajes. También hace la función de enrutador.

El enrutador: puede haber varios en la red y su función es enrutar los mensajes. Este tipo de nodos no pueden entrar en modo “dormir”. Suelen ser dispositivos conectados a la corriente.

El dispositivo final: son capaces de enviar o recibir paquetes de la red, pero no tienen capacidad de enrutar. Además, pueden entrar en modo de bajo consumo, o "dormir", para alargar la vida de la batería, volviéndose a conectar tan solo cuando es necesario. Solo se pueden comunicar con sus padres (enrutador o coordinador).

2.1.2. OpenThread

OpenThread[2] es una implementación de Thread que fue desarrollado por Google y lanzado en el año 2017. OpenThread cumple toda la especificación de Thread 1.1.1, el objetivo de esta implementación es acelerar el desarrollo de productos de Google Nest, por tanto, se centra en las casas inteligentes o edificios comerciales.

[1] https://standards.ieee.org/ieee/802.15.4/7029/

[2] https://openthread.io/

Thread es una tecnología basada en 6LowPAN. En la capa de transporte y la red usa las tecnologías conocidas, que son IPv6 y UDP. Los dispositivos también están basados en IEEE 802.15.4 como Zigbee 2.1.1.

A diferencia del Zigbee, en cada red de Thread puede haber varios enrutadores (en este caso, coincide con el coordinador de Zigbee). Cuando se cae un enrutador, la red se configura para enviar mensajes a otros enrutadores.

Thread solo se gestiona hasta la capa de la red, es decir, la capa de aplicación es libre para los usuarios. Esto ayuda mucho a los desarrolladores a diseñar su propia capa de la aplicación. Además, es posible conectarse directamente a WiFi y forma parte del Internet que se está usando

2.1.3. Bluetooth Mesh

Bluetooth es una tecnología que permite la comunicación punto a punto o de punto a multipunto, pero no permite la comunicación de multipunto a multipunto. Por este motivo se crea Bluetooth Mesh, para permitir este tipo de conexión.

La red Bluetooth Mesh tiene un nodo especial para gestionar la red, que es el provisioner. El provisioner puede ser un dispositivo final o también puede ser un móvil. A diferencia de otras tecnologías, el provisioner de Bluetooth Mesh sólo se necesita para añadir un dispositivo a la red, una vez formada la red, el provisioner puede desaparecer de la red.

Los mensajes se difunden por inundación, cada mensaje tiene un campo TTL que se decrementa por cada reenvío. Los nodos de la red tienen un caché para guardar el mensaje que ha sido reenviado recientemente para no volver a enviarlo otra vez, de esta forma, la red Mesh no se satura.

Aparte del provisioner, en la red puede haber 4 tipos más, cada nodo puede tener varios tipos activados.

1. Proxy: son los nodos que tienen el servicio proxy activado, los dispositivos que no son de Mesh, pero sí que tiene BLE (teléfono móvil), se pueden conectar a estos proxies para comunicarse en la red.

2. Relay: son los nodos que reenvían los mensajes, cuando estos nodos reciben un mensaje que es de la red y no está en su caché, reenvían el mensaje hacia todos los nodos de su alrededor

3. Low power node(LPN): son los nodos de bajo consumo, intenta buscar un nodo friend y se pone en modo “dormir” cuando no tiene algo que hacer.

4. Friend: son los nodos que permite a los nodos LPN establecer una relación y se guardan los mensajes de los LPN cuando estos están en modo “dormir”

Cada nodo puede tener todos estos modos activados, pero cuando el nodo LPN se pone a dormir, aunque tenga el relay activado no va a reenviar el mensaje.

Otra característica de Bluetooth Mesh es que está tecnología se encarga también de la capa de aplicación. SIG ya ha definido unos modelos básicos: activar, desactivar, controlar la luminosidad, etc. Esto hace que los dispositivos de diferentes compañías pueden inter operar entre sí.

2.1.4. Comparación

Se ha buscado unos datos básicos de estas tecnologías que se muestra en la tabla 2.1. Algunos de los datos se ha obtenido del trabajo fin de máster, "Bluetooth Mesh Networking. Aplicaciones y pruebas de concepto"[1]

Características	Zigbee	Google Thread	Bluetooth LE
Transmisión	semidúplex	semidúplex	dúplex/semidúplex
Velocidad de bajada y subida	20-250 kbps	20-250 kbps	125 kbps-2 Mbps
Latencia	15 msec	<100 msec	<3 msec
Rango	30-100 m	30-100 m	10-150 m
Número máx nodos	65000	250-300	32000

Tabla 2.1: Comparación de las tecnologías

En la tabla se observa que Bluetooth LE tiene una buena latencia y la velocidad de bajada y subida. Aunque no es la red que soporta más nodos, pero 32000 nodos puede cubrir muchos casos de usos.

Aparte de estos datos, Bluetooth Mesh también es la única tecnología que no requiere un nodo especial para tener la red funcionando, con un dispositivo móvil que suele ser la interfaz de los clientes finales, puede formar una red de forma rápida y no requiere el móvil para su funcionamiento. Estos es la causa principal de porque se ha elegido está tecnología en este trabajo.

2.2 Productos comerciales

En este apartado se van a nombrar algunos productos de Bluetooth Mesh. En el apartado 2.1.3 se ha mencionado que los productos de Bluetooth Mesh tienen mucha interoperabilidad, por lo tanto, el producto comercial no tiene que ser un proyecto completo. Es decir, las empresas pueden sacar un producto que permite conectarlo a la red, pero no dar ninguna aplicación que permita gestionar la red y controlar el producto, porque la parte de la gestión y el control lo pueden hacer otros productos de otras empresas.

En la página oficial de Bluetooth[2], se pueden encontrar todos los productos que están cualificados. En la siguiente tabla 2.2 se va a mostrar algunos de como ejemplo representativo de lo que podemos encontrar a nivel comercial en Bluetooth Mesh.

Como se puede ver en la tabla, no hay muchos productos de las empresas españolas, la mayoría son de China. Otro aspecto que se puede observar en la tabla es que algunos productos que normalmente no se necesita varias, pero soporta la tecnología Bluetooth Mesh, tal como cuentakilómetros, la máquina de limpiar plato, diccionario inteligente, etc. Estos productos en sí no se aprovecha mucho la ventaja de Bluetooth Mesh, pero si existe una aplicación general que conecta todos estos productos en una misma red entonces se puede general una red bastante grande que puede cubrir la mayoría caso de usos.

Fecha pub.	Compañia	Producto	Descripción
14/09/2021	SK Magic Co.,Ltd.	Triple Care Dishwasher	Una máquina para limpliar platos
31/08/2021	Tong Yah Electronic Technology Co., Ltd.	RX-4 Meter	Cuenta kilometros
25/08/2021	American Lighting Inc	Spektrum+ LED Downlight	Luz inteligentes
18/08/2021	Qrio Inc.	Qrio Hub	Hub para controlar candado inteligente
28/07/2021	Huawei Technologies Co., Ltd.	HUAWEI Sound X	Altavoces
28/07/2021	Siteco GmbH	Street-Remote	Farola
09/07/2021	Shenzhen Step Electronic and Lighting Co., Ltd	Brevo Hub	Hub que permite controlar con Alexa y Hey Google
17/06/2021	Ninebot (Changzhou) Tech Co., Ltd.	Ninebot NB-BLE5V1-GS	Patinete
11/06/2021	Intermatic Incorporated	Presence Sensors	Sensor para detectar el movimiento
03/06/2021	Anhui Toycloud Technology Co., Ltd.	Alpha Egg dictionary pen S	Diccionario inteligente

Tabla 2.2: Productos

CAPÍTULO 3

Análisis de la especificación

El proyecto que se va a realizar es un proyecto completo, ya que contiene la programación de la placa, la programación del móvil e implementación del estándar Bluetooth Mesh. Para poder llevar a cabo el proyecto es necesario realizar un estudio sobre estas áreas. En este capítulo se va a dividir el estudio en diferentes apartados.

3.1 Ideas básicas de BLUETOOTH Mesh

En este apartado se explicará el funcionamiento básico de Bluetooth Mesh. Para simplificar, se va a usar BLE (Bluetooth Low Energy) y BLE Mesh (Bluetooth Mesh).

Para empezar, se necesitan aclarar algunas palabras ya utilizadas anteriormente:

- Device/Dispositivo: En BLE Mesh un dispositivo es aquel dispositivo que no está conectado a la red.
- Node/Nodo: Cuando un dispositivo está aprovisionado entonces se convierte en un nodo.
- Provisioner/Aprovisionador: Rol especial para aprovisionar los dispositivos de la red.

Para crear una red de BLE Mesh, la única cosa necesaria es la clave de la red, que está formada por 16 bytes. Cualquier nodo si tiene la clave, puede entender la capa de red. Una vez generada la clave de 16 bytes, el aprovisionador ya puede aprovisionar los dispositivos.

Cuando un aprovisionador aprovisiona un dispositivo, se le da dos cosas: la clave de la red, para que el dispositivo entienda los mensajes y una dirección única de la red.

Respecto a la dirección, existen 4 tipos de direcciones, tal como se muestra en la siguiente tabla.

Values	Address Type
0b0000000000000000	Unassigned Address
0b0xxxxxxxxxxxxxx(excluding 0b0000000000000000)	Unicast Address
0b10xxxxxxxxxxxx	Virtual Group
0b11xxxxxxxxxxxx	Group Address

Tabla 3.1: Address type

1. Unassigned Address: como su nombre indica, es una dirección prohibida a usar.

2. Unicast Address: la dirección de un nodo, una forma más correcta de nombrar sería: la dirección de un elemento.

3. Virtual Group: en este documento se nombrará grupo a este tipo de dirección. Es una dirección de un grupo que se usa para agrupar los nodos.

4. Group Address: son todas las direcciones especiales que se han definido por SIG. De momento solo existen unos cuantos, por ejemplo, todos los nodos, todos los nodos proxy, todos los nodos relay, todos los nodos friend, etc.

Un nodo de la red puede tener varias direcciones, esto es porque cada nodo puede tener varios elementos. Esto también es lógico, si se piensa en un interruptor inteligente, un interruptor puede tener varios enchufes, y como cliente, suele querer que se puede controlar cada enchufe por separado y no todo junto.

En el apartado 2.1.3 se ha dicho que BLE Mesh se encarga también de la capa de aplicación, esto se consigue gracias a los modelos (models). Cada elemento puede tener varios modelos, los modelos se pueden entender como los mensajes que recibe un elemento.

Existen 2 tipos de modelos, uno que ha sido definido por SIG, y otro que es vendor model y que puede estar definido por otras empresas. Un ejemplo de modelo es Generic OnOff Server. Cuando un elemento tiene este modelo significa que recibe un mensaje del tipo Generic OnOff, que puede activar o desactivar una E/S. Otro modelo puede ser Light Lightness Server, que puede controlar la luminosidad, y por tanto actuar en un rango discreto.

Un ejemplo sería la luz inteligente de una bombilla. Como sólo tiene una bombilla, solo tiene un elemento. Pero es posible controlar la bombilla de diferentes formas: encender, apagar o ajustar la luminosidad. Por lo tanto, este elemento puede tener dos modelos: Generic OnOff Server y Light Lightness Server.

Como el formato de mensaje es controlado por SIG, cada producto tiene que crear la funcionalidad ajustándose a los modelos, así pues, los productos de diferentes compañías pueden inter operar entre sí.

Para enviar o recibir mensajes de los modelos, se necesita la clave de la aplicación. Los mensajes de los modelos se encriptan dos veces. Primero con la clave de aplicación, para que solo los nodos interesados entiendan el contenido del mensaje y después se encripta con la clave de la red, para que cualquier nodo de la red pueda saber si el mensaje es para él o debe reenviarlo a su alrededor.

La dirección de cada nodo varía en la red, además el nodo no sabe quién hay en la red, por eso, cuando se envían los mensajes, el destino normalmente es hacia la dirección de un grupo.

La comunicación en BLE Mesh es parecida a la publicación/ suscripción. Cada modelo tiene una dirección de publicación y una lista de dirección de suscripción. Es decir, un modelo, hasta que no se vuelve a configurar, sólo puede enviar a un grupo, pero puede pertenecer a varios grupos.

Por ejemplo, siguiendo con el ejemplo de la luz inteligente, la luz puede suscribirse a un grupo que es oficina 1, pero también puede suscribirse a la planta

2. El edificio puede tener 2 tipos de interruptores diferentes, uno que está dentro de cada oficina y controla las luces de toda la oficina y otro que se pone en los pasillos para controlar las luces de toda la planta.

Normalmente BLE Mesh consume bastante energía, cada nodo de la red puede tener diferentes modos activados: Relay, Proxy, Friend y LPN. Como un nodo normalmente tiene que estar continuamente escuchando el mensaje, gasta mucha energía y la forma para reducir este consumo es el modo LPN.

Cuando un nodo tiene LPN activado intenta buscar un friend a su alrededor, una vez encontrado, el nodo se duerme periódicamente. Este nodo no va a retransmitir los mensajes. Si durante el periodo en el que está en dormir, hay algún mensaje dirigido a estos nodos, los nodos Friend se guarda el mensaje. Cuando un nodo LPN despierta, realiza una petición a su Friend para preguntar si hay algún mensaje importante o no.

El Friend y LPN tienen que estar directamente conectados, por esta razón, BLE Mesh no es una buena tecnología si los dispositivos se mueven a menudo y no es posible recargarlos frecuentemente.

En la figura 3.1 se muestra un ejemplo de la red Mesh, no hace falta que siempre tener todos estos nodos, además un nodo puede tener varios modos activado, esto solo es un ejemplo para mostrar cuál es el comportamiento de cada nodo.

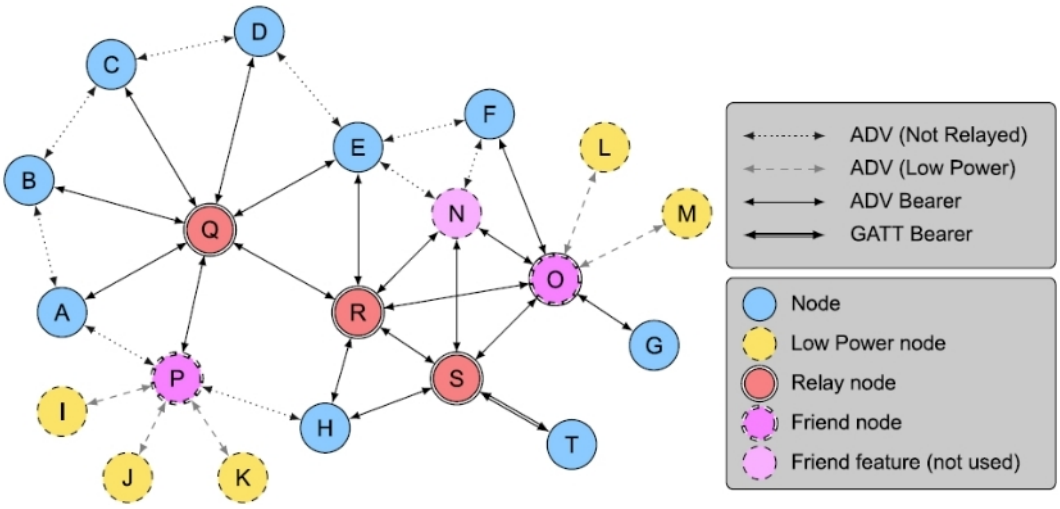

Figura 3.1: Ejemplo de una red Mesh

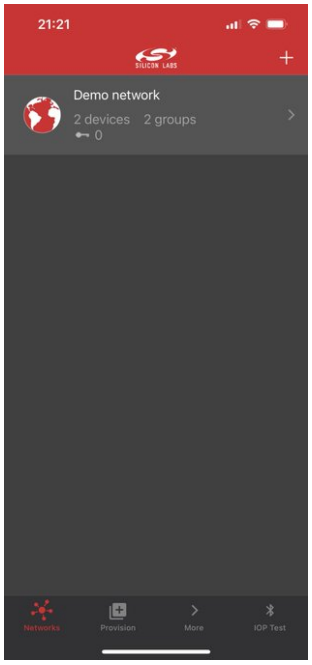

Figura 3.2: Bluetooth mesh by Silicon Labs

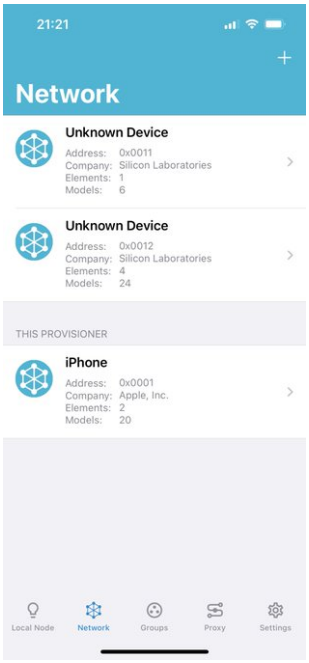

Figura 3.3: nRF Mesh

3.2 Aplicaciones móviles

Antes de empezar a implementar la librería, se han buscado las aplicaciones móviles que existen en el mercado actual. Esto se hace con el fin de tener una idea de qué funcionalidades debe tener una aplicación móvil de la red BLE Mesh, aunque teniendo en cuenta el apartado anterior ya se puede suponer algunas funciones, tal como aprovisionar y enviar y recibir mensajes de los modelos básicos. Se han buscado las aplicaciones en las tiendas de aplicaciones de móvil (App Store y Google Play). Existen diversas aplicaciones de BLE Mesh, pero muchas de ellas son de pago o están diseñadas para el producto propio de la empresa.

Al final, se han encontrado dos aplicaciones que son más o menos universales, es decir, que funcionan para cualquier dispositivo que soporta BLE Mesh. Una es la aplicación de Silicon Labs, que es una aplicación en la que se muestran ejemplos de sus proyectos. Otra es la aplicación de NORDIC, que es otra empresa dedicada a la venta de módulos de comunicación BLE. La figura 3.2 es de Silicon Labs y la figura 3.3 es de NORDIC.

Hemos hecho una tabla 3.2 de comparación de las dos aplicaciones.

Funcionalidad	Silicon Labs	nRF Mesh	Comentario
Crear una red	si	si	
Crear una subred	no	si	
Gestionar NetKey (modificar)	no	si, pero no desde la aplicación	Es posible exportar e importar la configuración de la red, pero desde la aplicación no es posible modificar NetKey primaria
Gestionar AppKey	no	si	si, pero con un límite hasta 5, aunque en este proyecto con 1 es suficiente
Gestionar por elementos	no	si	
Dirección que puede publicar	solo grupo	todo	
Configurar la publicación y suscripción de cada elemento	no, solo permite un modelo y un elemento	si	
Controlar el modelo desde la aplicación móvil	si	si	
Provisionar dispositivo	si	si	

Modificar feature de cada nodo	si	si, pero complicado	En nRF a veces no se puede activar el modo relay, pero es posible, y aunque tiene LPN activado, en la información indica que no está activo
Conectar a la red a través de un proxy	si	si, además puedes elegir a cual	
Gestionar Scene	si	si	Esta funcionalidad no se ha investigado, pero es una función que establece todos los estados a un estado predefinido
Crear Grupos	si	si	
Personalizar la forma de controlar nodos	no	si	
Vendor Model	no hemos probado	no hemos probado	todas las características que están puestos se puede controlar con SIG model, por eso no hemos visto nada de vendor model

Tabla 3.2: Comparación de las dos aplicaciones

En resumen, la aplicación de Silicon Labs está más orientada a un usuario final, tiene más restricciones, pero es más fácil de usar, el usuario solo tiene que aprovisionar y elegir un modelo y ya puede empezar. En cambio, la aplicación de NORDIC ofrece muchas más posibilidades, pero también requiere que el usuario tenga más conocimiento sobre BLE Mesh.

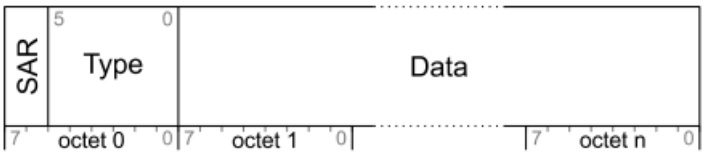

Figura 3.4: Formato de Proxy PDU

3.3 Especificación de BLE Mesh

En las secciones anteriores se ha visto cómo se usa BLE Mesh, pero para implementarlo esto no es suficiente, por ejemplo, la aplicación tiene que enviar un mensaje de Generic OnOff, pero no se conoce el formato del mensaje, los mensajes se encriptan con dos claves, pero no se conoce con qué algoritmo, la aplicación debe tener una funcionalidad que es el aprovisionamiento, pero no se conoce qué pasos se deben realizar durante este proceso.

Para redactar esta sección se ha tomado como referencia Mesh Profile 1.0.1[3] y Mesh Model 1.0.1[4]. Estos dos documentos son las especificaciones de la página oficial de Bluetooth. Mesh Profile explica cómo hay que implementar un nodo de la red y Mesh Model explica cómo son los modelos de SIG.

En esta sección se van a estudiar estos dos documentos y se divide en distintas subsecciones.

3.3.1. Proxy PDU

Proxy PDU (Unidad de datos de protocolo) es un tipo de pdu esencial para la aplicación móvil. Cuando una aplicación móvil se conecta a la red BLE Mesh, está usando las características de BLE. Se conecta al servicio Provisioning o el servicio Proxy para enviar datos. Cada mensaje que se envía desde el móvil, es un proxy pdu.

Proxy PDU tiene los siguientes formatos: SAR: es un campo de 2 bits, se usa para indicar que el mensaje es completo o es un segmento. El valor se muestra en la tabla 3.3. Type: es un campo de 6 bits, pero solo existen 4 valores válidos. Este campo se usa para indicar el tipo del mensaje de BLE Mesh 3.4.

Valor	Descripción
0b00	Un mensaje completo.
0b01	Es el primer segmento de un mensaje.
0b10	Es un segmento que ni es principio ni es final.
0b11	Es el último segmento de un mensaje.

Tabla 3.3: Valores del campo SAR

Valor	Descripción
0x00	Network PDU, este tipo de pdu es el que más se usa. El mensaje Generic OnOff y otros mensajes para configurar el nodo todo pertenece a este tipo
0x01	Mesh Beacon, solo existe dos tipos, Unprovisioned device beacon y Secure Network beacon, para la aplicación móvil, solo se recibe Secure Network beacon y es un mensaje para indicar el estado de la red.
0x02	Proxy Configuration, el contenido de este tipo de pdu es parecido a Network PDU, pero solo se usa entre el móvil y el nodo proxy. Los mensajes son comandos para filtrar los mensajes que se pasan por proxy.
0x03	Provisioning PDU, todos los mensajes de aprovisionamiento son de este tipo.
0x04-0x3F	Reservado para futuro.

Tabla 3.4: Valores del campo Type

El tamaño de Proxy PDU depende del dispositivo, para soportar un mensaje máximo de tipo aprovisionamiento, necesita 69 bytes y para soportar un mensaje máximo de otros tipos se necesitan 33 bytes. Si el móvil o el dispositivo no tiene estos tamaños, entonces hay que segmentar el mensaje. Esto no quiere decir que un mensaje de BLE Mesh solo puede tener 33 bytes, porque los mensajes de BLE Mesh también se segmentan por otras capas.

3.4 Algoritmos

En la especificación de BLE Mesh se han definido unas funciones para generar claves o encriptar. Las funciones son:

- s1: para generar salt.
- k1: para generar clave.
- k2: para generar información sobre la clave de la red.
- k3: para generar el identificador de la clave de la red.
- k4: para generar aid de una clave de aplicación.

Las funciones están definidas a partir del apartado 3.8.2.4 de la especificación.

Todas estas funciones se implementan aplicando varias veces el algoritmo de encriptación.

Para implementar la librería básicamente se necesitan 4 algoritmos.

1. AES_CMAC: para implementar las funciones que se han nombrado antes.
2. AES_CCM: para encriptar el contenido.
3. AES_ECB: para ofuscar los metadatos de la capa de la red.
4. elliptic p256: para generar clave de aprovisionamiento.

3.5 Provisioning PDU

La forma de cómo un móvil aprovisiona el dispositivo y la forma de cómo un nodo aprovisiona un dispositivo es un poco diferente. Este trabajo se centra en la forma del móvil.

Existen 9 tipos de mensajes de aprovisionamiento, que son los siguientes:

1. Provisioning Invite: es un mensaje que el aprovisionador envía al dispositivo para iniciar el proceso de aprovisionamiento.

2. Provisioning Capabilities: es una contestación del dispositivo al aprovisionador, en este mensaje indica el tipo de autenticación que se usa y el algoritmo de encriptación que soporta, en este momento, solo existe un algoritmo para el proceso de aprovisionar, que es "FIPS P-256 Elliptic Curve".

3. Provisioning Start: este mensaje es enviado por el aprovisionador, donde indica qué tipo de autenticación ha elegido y el algoritmo de encriptación.

4. Provisioning Public Key: este mensaje es enviado por las dos partes para intercambiar las claves públicas. Como sólo existe un algoritmo, de momento en este mensaje se incluye el componente X e Y de la clave.

5. Provisioning Input Complete: este mensaje sirve para indicar que el usuario ha autenticado correctamente, si no existe autenticación, este mensaje se omite.

6. Provisioning Confirmation: este mensaje también es enviado por las dos partes, es un valor para autenticación entre el aprovisionador y el dispositivo.

7. Provisioning Random: este mensaje intercambia el valor aleatorio que se ha usado para calcular la confirmación.

8. Provisioning Data: en este mensaje se incluye las claves de la red, estado de la red y la dirección del primer elemento.

9. Provisioning Complete: este mensaje es enviado por el dispositivo para indicar que ha sido provisionado correctamente.

10. Provisioning Failed: este mensaje se envía para indicar los errores que han ocurrido durante el proceso de aprovisionamiento, si todo va correcto, este mensaje se omite.

En este trabajo, sólo se va a tratar el caso de un dispositivo que no requiere autenticación, en la figura 3.5 muestra el intercambio de mensajes de un proceso de aprovisionamiento

A partir del primer mensaje, que es provisioning invite, hay que guardar todos los contenidos del mensaje, quitando el campo de tipo de mensaje. Porque todo este se usará para calcular la confirmación.

Para calcular la confirmación hay que realizar los siguientes cálculos:

1. Usando la clave de elliptic del móvil y la clave del dispositivo para generar un secreto de ecdh(ecdhSecret).

2. Se suman todas los mensajes del aprovisionador y el dispositivo (quitando el campo del tipo) y se pasa a la función s1 para generar confirmationSalt.

3. Se usa ecdhSecret, confirmationSalt y “7072636b” (hexadecimal) para generar confirmationKey con la función k1.

4. Se genera un número aleatorio de 16 bytes y lo concatena con el input del usuario (en este caso como no se tiene el método de autenticación, pues es 16 bytes de 0).

5. El resultado de concatenación se pasa al algoritmo AES_CMAC con la confirmationKey y se genera la confirmación.

En el cálculo, sólo el número aleatorio es desconocido para el dispositivo y el móvil, por lo tanto, cuando el móvil recibe el número aleatorio del dispositivo se realiza otra vez el cálculo para comprobar si coincide con la confirmación que ha enviado el dispositivo.

Antes de enviar los datos de la red, también hay que encriptarlos, esta vez se encripta con el algoritmo AES_CCM. Este algorimo necesita 4 parámetros y uno opcional. En el proceso de aprovisionamiento sólo se necesita 4: la clave, nonce, datos a encriptar y el número de bytes de MIC(Message integrity check (MIC)).

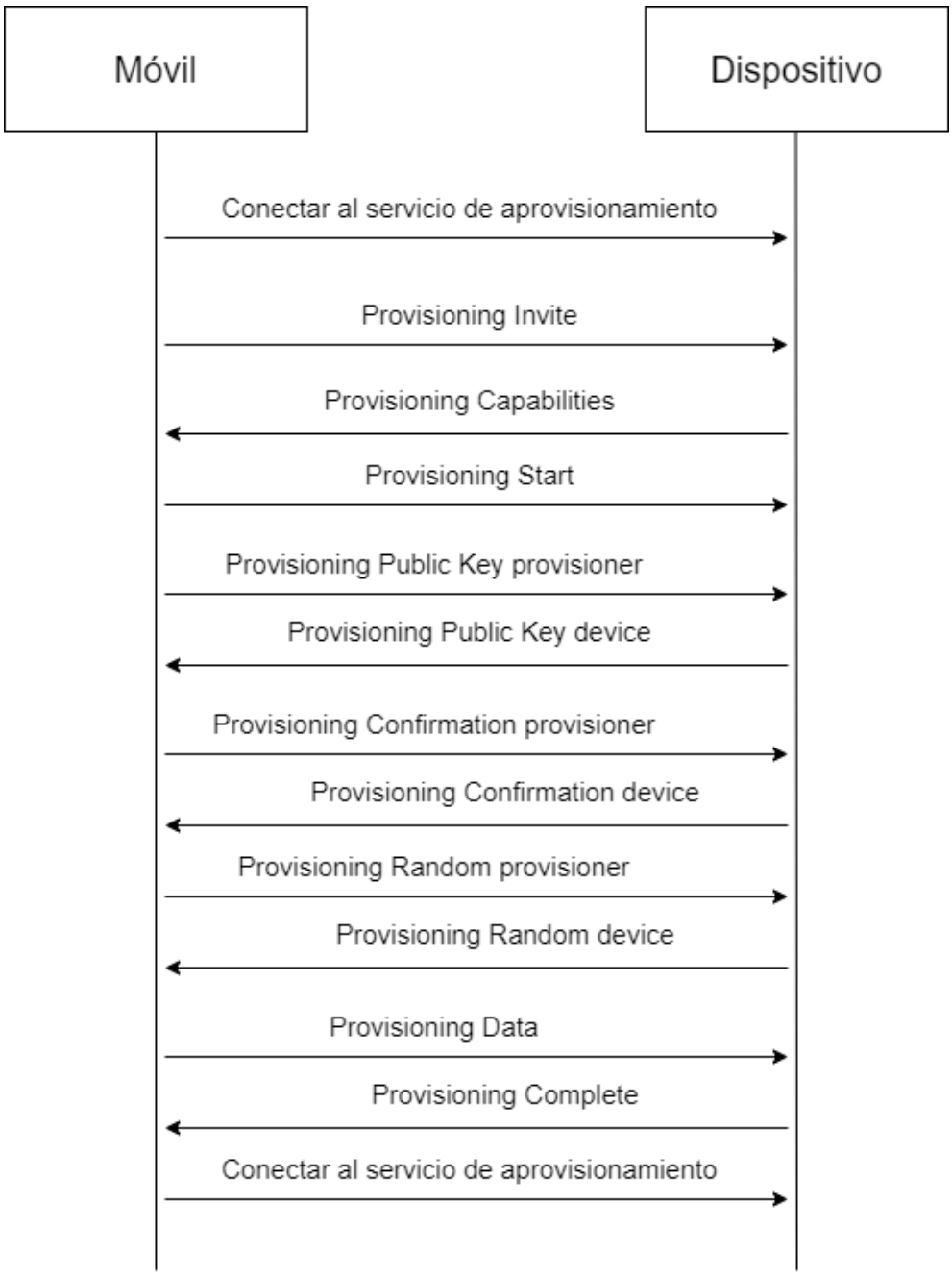

Figura 3.5: Procedimiento de un proceso de aprovisionamiento

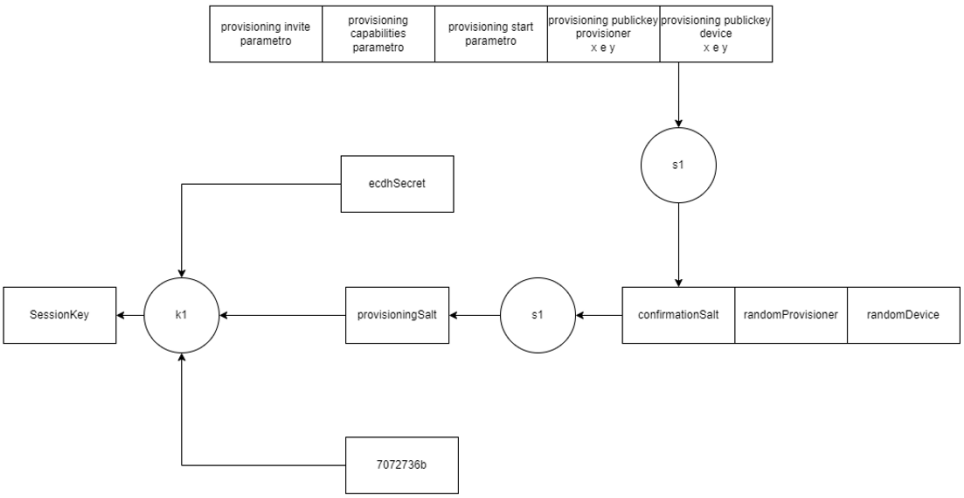

Figura 3.6: La clave de sesión

La generación de la clave para encriptar se muestra en la figura 3.6.

El valor de nonce es parecido, sólo que a la función k1 se le pasa "7072736e" y solo coge los primeros 13 bytes. Para los datos de aprovisionamiento, el número de bytes de MIC tiene que ser 8 bytes

Una vez acabado el proceso de aprovisionamiento hay que generar una clave de dispositivo, esta clave se usa para realizar cualquier configuración del nodo, y la forma de generarla es la misma que la figura 3.6 pero a la función k1, se le pasa "7072646b".

Se puede observar que la clave de dispositivo ha usado los números aleatorios que se han generado durante el proceso de aprovisionamiento. Si se pierde esta clave, ya no es posible recuperarla, por lo tanto, se debe guardar esta clave.

3.6 Network PDU

Este tipo de pdu es el más importante de la red BLE Mesh, porque todos los mensajes de los modelos se empaquetan en un network pdu. El mensaje de este pdu está dividido en diferentes capas como se muestra en la figura 3.7.

En esta sección se van a explicar las capas por separado.

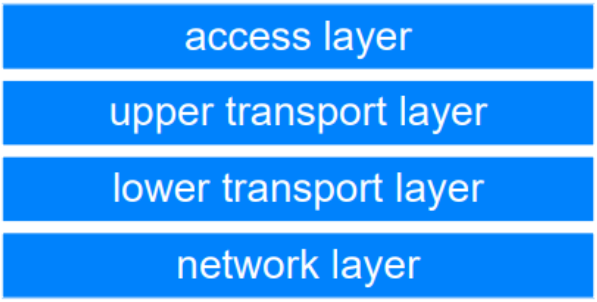

Figura 3.7: Capas de network pdu

3.6.1. Access Layer

Esta capa también se puede entender como la capa de aplicación, todos los parámetros de los modelos se guardan aquí. En la especificación se indica que los valores de esta capa se codifican con formato "little endian" pero la especificación no explica muy bien esta parte, porque existe alguna excepción y falta algunas aclaraciones. Los campos de esta capa dependen de las operaciones, pero se puede decir que está formado por dos campos tal como se muestra en la figura 3.8.

Figura 3.8: Format de la capa de acceso

Código de operación concatenando los parámetros necesarios. El código de operación se puede dividir en 3 tipos: 1 octeto, 2 octetos o 3 octetos como se muestra en la tabla 3.5. Los códigos de 1 octeto y 2 octetos son operaciones que

Formato	Tipos
0xxxxxxx (excluding 01111111)	1 octeto
01111111	Reservado para futuro
10xxxxxx xxxxxxxx	2 octetos
11xxxxxx zzzzzzzz	3 octetos

Tabla 3.5: Tipo de código

han sido definidas por SIG, y los códigos de 3 octetos son para las compañías.

Los códigos de SIG se pueden encontrar en el apartado 4.3.4 de Mesh Profile y el apartado 7 de Mesh Model.

Las operaciones de Mesh Profile son las operaciones que se usan para configurar el nodo y las operaciones de Mesh Model son las operaciones de los modelos. Este es el primer lugar que produce confusión. Por ejemplo, la operación de 3 octetos está dividida en dos partes, código de la operación, representada por x y código de la compañía, representada por z, si el código de la operación es 0x23 y el código de la compañía es 0x0136 el resultado es 0xE33601.

El resultado se obtiene de la siguiente forma:

1. Concatenar los dos códigos, todos los campos de access layer se deben concatenar hacia la izquierda, es decir, la concatenación del código de la operación y el código de la compañía es 0x013623 y no 0x230136 porque después de cambiar a little endian, se invertirá otra vez el orden de los campos.

2. Una vez concatenado, cambiar a “little endian”, que es 0x233601, 0x23 es de la operación 0x3601 es la compañía, pero en “little endian”.

3. Finalmente, se pone los primeros dos bits a 11, 0xE33601.

Esta es la forma de calcular el código de 3 octetos. Se supone que es lo mismo para los otros 2 tipos de códigos, pero los códigos que están puestos en la especificación ya tienen realizada esta transformación o desde el principio ya están diseñados para “Little endian”.

Por ejemplo, el código de la operación de borrar la clave de aplicación es 0x8000 y la operación de obtener la clave de aplicación es 0x8001, va incrementando de 1 en 1. Si se realiza el mismo cálculo, el resultado es 0x8080 y 0x8180, pero estos son incorrectos, porque cuando se envían estas 2 operaciones, se debe enviar 0x8000 y 0x8001. Esto hace que la formación de access layer sea opcode concatenado con "toLittleEndian(parámetros)"si se copian los códigos tal cual de la especificación.

Una vez entendida la parte de código de la operación se explican los parámetros. La figura 3.9 muestra los campos de la operación de añadir la clave de

aplicación, si se imagina que el primer campo es 0x456123 y el segundo campo es 0x63964771734fbd76e3b40519d1d94a48, y se aplica el mismo cálculo de antes:

1. Concatenar los dos campos. Se consigue, 0x63964771734fbd76e3b40519d1d94a48456123
2. Cambiar a "little endian" y se obtiene 0x236145484ad9d11905b4e376bd4f7371479663

El resultado del anterior cálculo está mal, en la especificación no se ha indicado, pero en el caso de las claves, no se debe cambiar a "little endian". Esto también pasa con la operación de añadir grupo a la lista de suscripción, el campo de "label uuid" también se debe dejar tal cual. No se sabe la razón de por qué es así, pero puede ser que cuando un campo envía el valor no se debe cambiar a "little endian".

Field	Size (octets)	Notes
NetKeyIndexAndAppKeyIndex	3	Index of the NetKey and index of the AppKey
AppKey	16	AppKey value

Figura 3.9: Campo de añadir la clave de aplicación

3.6.2. Upper transport layer

Puede ser de dos tipos: de acceso, que guarda el mensaje de la capa de acceso, o de control, que son comandos especiales para controles.

En este trabajo solo se va a analizar el tipo acceso, porque los comandos de controles son todos para los nodos friend o heartbeat, que no es interesante para un dispositivo móvil.

El tipo acceso se puede entender mirando la figura 3.10. A partir de esta capa todo se codifica en "big endian" y los campos se concatenan hacia la derecha, es decir, si el dato encriptado es 0x10a9 y mic es 0x9582e23a, entonces el resultado de la concatenación es 0x10a99582e23a.

Básicamente esta capa se usa para encriptar el contenido de la capa de acceso, como se puede ver en la figura 3.11. La longitud de mic se deja a decisión de los usuarios. Puede ser de 4 bytes o de 8 bytes, cuanto más largo es más seguro, pero

Figura 3.10: Concepto de upper transport layer

como también ocupa más y hay que transmitir más datos, aunque sólo tiene 4 bytes de diferencia, el mensaje se segmenta en la siguiente capa y 4 bytes puede causar que se envíe un segmento más.

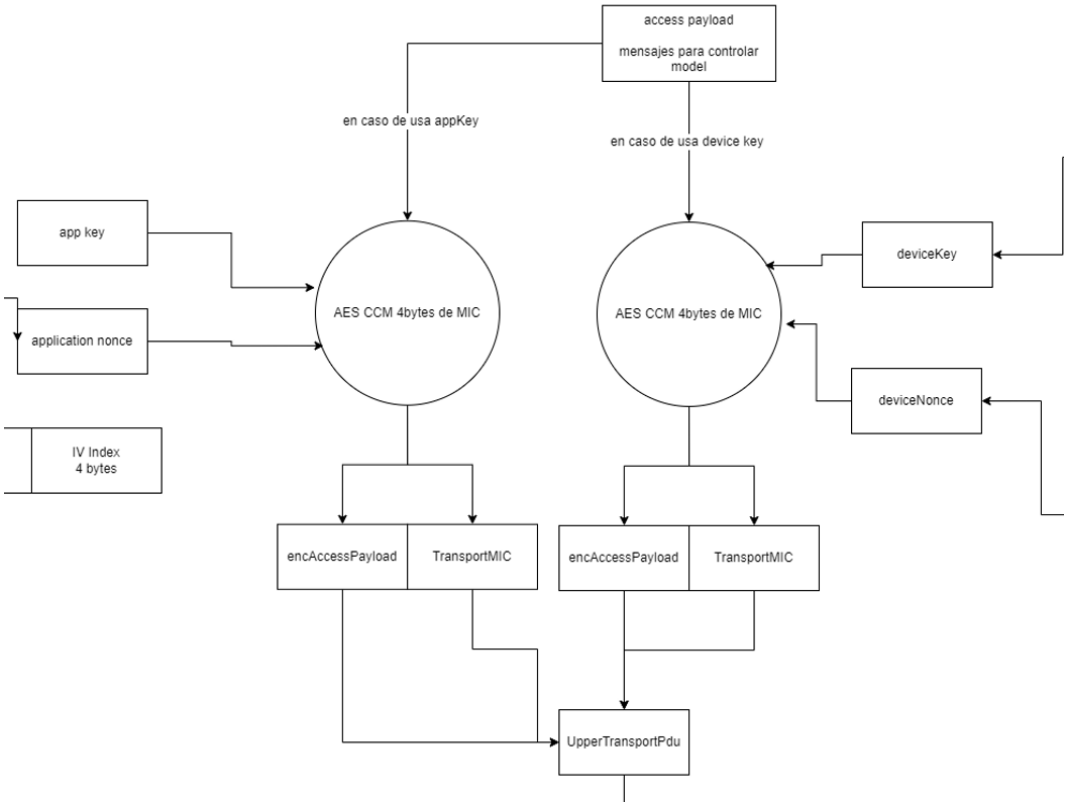

Figura 3.11: Upper access transport layer

El contenido se encripta con el algoritmo AES_CCM, también mencionado en la sección de aprovisionamiento 3.5. Este algoritmo necesita 4 más 1 opcional, clave, nonce,dato, longitud de mic y dato opcional.

Existe dos tipos de clave. Si el contenido de la capa de acceso es de las operaciones que están definidas en Mesh Model, entonces hay que encriptar con la clave de aplicación, si es de Mesh Profile hay que encriptar con la clave de dispositivo que se ha generado durante el proceso de aprovisionamiento.

Depende de qué clave se usa, el tipo de nonce también varía, si usa la clave de la aplicación hay que usar nonce de la aplicación y si usa la clave del disposi-

tivo hay que usar el nonce de dispositivo. La forma de cómo se calculan los dos tipos de nonce se muestra en la figura 3.12 y 3.13.

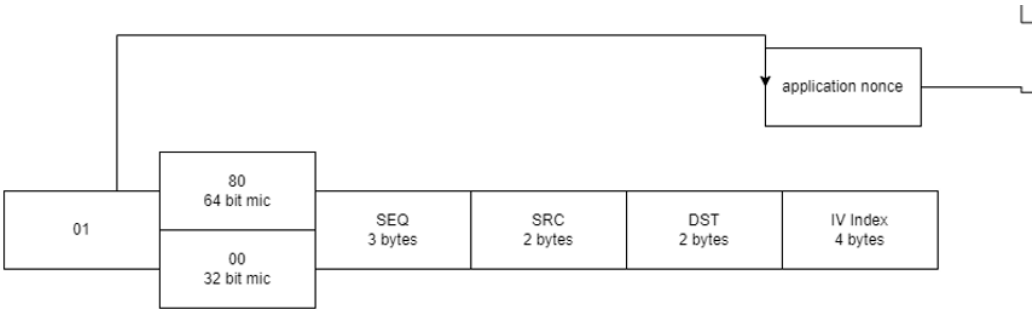

Figura 3.12: Application nonce

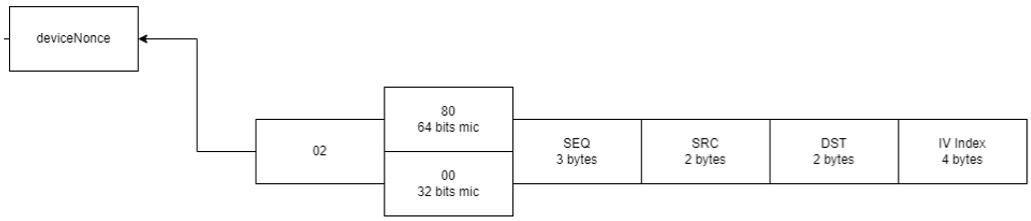

Figura 3.13: Device nonce

En la figura aparecen los campos desconocidos, que se van a explicar uno por uno:

- El primer campo es un valor fijo, indica el tipo del nonce.
- El segundo campo se llama ASZMIC, solo tiene un bit y seguido de 7 ceros como padding para llenar un byte, si el número de mic es de 8 bytes, este campo tiene que ser 1 y si es de 4 bytes, este campo es 0.
- El tercer campo se llama SEQ, es un número de secuencias que se incremente cada vez que se envía un network pdu, los nodos de BLE Mesh fijan SRC junto con SEQ, si este número es menor que el valor que tiene guardado, es posible que alguien haya captado el mensaje anteriormente y está intentandohacer un replay attack.
- El cuarto campo es la dirección del elemento que ha enviado el mensaje, tiene que ser de tipo unicast.
- El quinto campo es la dirección del destino, aquí puede ser cualquier tipo de dirección, un elemento, un grupo o todos los nodos.

- El sexto campo se llama iv index, es parecido al número de secuencia, cada elemento tiene su propio número de secuencia, pero todos los nodos comparten el mismo valor de iv index. Cuando el número de secuencia llega al límite, se incrementa iv index para alargar la vida de la red.

Antes de explicar el dato opcional para encriptar, se va a ver cómo se genera una dirección de grupo. El grupo de BLE Mesh se genera a partir de un uuid, los cálculos son los siguientes:

1. Generar salt con "76746164"(hexadecimal) y la función s1.
2. Pasando uuid y salt a la función AES_CMAC.
3. Coge 14 bits menos significativos y le concatena con 0b10.

La dirección del grupo, en realidad es hash de un valor uuid, por eso, cuando se añade una dirección de grupo a la lista de suscripción de un elemento hay que pasarle uuid y no la dirección.

Cuando el destino es una dirección de grupo, hay que pasar el valor de uuid como dato opcional para encriptar, pero cuando se usa la clave del dispositivo, el destino siempre es un elemento, por lo tanto, el dato opcional solo es posible que se necesite en el caso de usar la clave de la aplicación.

3.6.3. Lowe transport layer

En la capa superior 3.6.2 se ha mencionado que mic puede ser de 4 bytes u 8 bytes, pero en ningún campo se indica eso, como un elemento puede tener varias claves de aplicación asociadas, tampoco se sabe con qué clave se ha encriptado. Todos estos campos se ponen en la capa de transporte inferior.

En la capa inferior también se distingue por acceso y control, este trabajo sólo se centra en el tipo acceso. Si la capa superior se encarga de encriptar, pues la capa inferior se encarga de segmentar. Los campos de un pdu segmentado y no segmentado son distintos. En la figura 3.14se muestra los campos de un pdu segmentado y la figura 3.15 se muestra los campos de un pdu no segmentado.

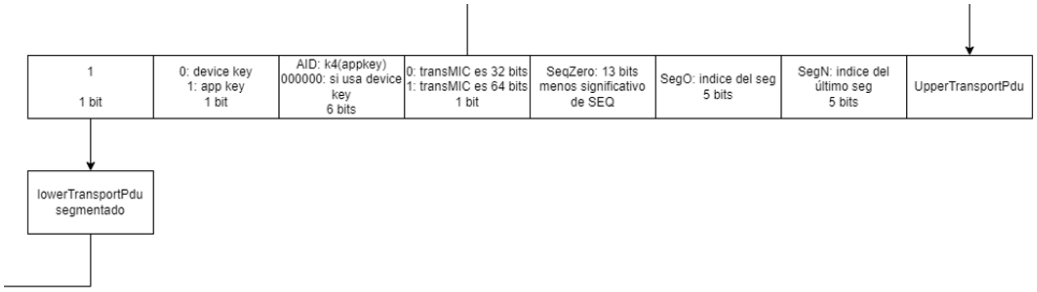

Figura 3.14: Pdu segmentado de la capa inferior de transporte

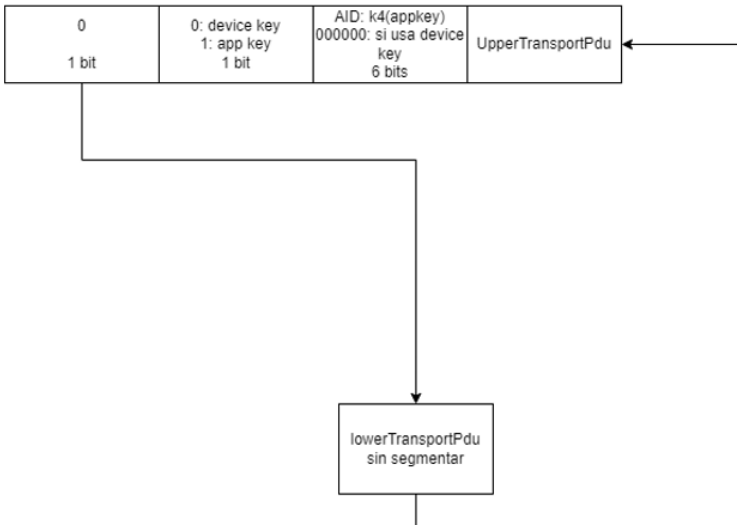

Figura 3.15: Pdu no segmentado de la capa inferior de transporte

Como se puede ver en las figuras, los campos de un pdu no segmentado son menos que un pdu segmentado, esto significa que en un pdu no segmentado cabe más datos de la capa superior.

Básicamente, se puede mirar el tamaño del dato de la capa superior, si no supera 15 bytes entonces cabe en un pdu no segmentado, pero si supera al 15 bytes, hay que segmentarlo de 12 bytes en 12 bytes. Si el tamaño no supera al 12 bytes, es posible segmentar el contenido en un segmento. Los pdus segmentados, requieren ack por la capa inferior y los pdus no segmentados no requiere ack, por lo tanto, cuando el mensaje necesita un ack, segmentarlo en un segmento es más eficiente que no segmentar, pero ya es un caso muy especial.

También se va a analizar los campos uno por uno.

- El primer campo es de un bit, para indicar si el pdu se ha segmentado o no.

- El segundo campo también es de un bit, para indicar si en la encriptación se ha usado la clave del dispositivo o la clave de la aplicación.

- El tercer campo se llama AID, es de 6 bits y sirve para indicar qué clave de la aplicación se ha usado. Como la clave de la aplicación tiene 16 bytes y AID solo 6 bits, es muy posible que exista 2 claves con el mismo valor de AID, en este caso hay que probar de desencriptar con las dos. En el caso de que se use la clave del dispositivo, este campo se llena de zeros.

- El cuarto campo se llama transMIC, es donde se indica el número de bytes de mic de la capa superior. Como se puede ver en el pdu no segmentado no existe este campo, por lo tanto, si mic es de 8 bytes, no se puede enviar con el pdu no segmentado.

- El quinto campo se llama seqZero, en este campo se guardan los 13 bits menos significativos del número de secuencia del elemento. El número secuencia se incrementa por cada envío, eso significa que, si hay dos segmentos, el valor seq de los dos son distintos, pero cuando se encripta, se ha usado el valor secuencia del primer segmento. Este campo sirve para indicar cuál es el seq del primer segmento. Todos los segmentos que tienen el mismo seqZero pertenecen a un mensaje.

- El sexto campo se llama seqO, sirve para indicar el índice del segmento, empieza desde 0. Como sólo hay 5 bits, esto significa que solo puede segmentar 32 segmentos. BLE Mesh no permite enviar un mensaje mayor que 380 bytes o 376 bytes (32 paquetes * 12 bytes por paquete - 4 u 8 longitud de mic).

- El séptimo campo indica el índice del último segmento, es decir, si hay 5 segmentos, el valor de este campo es 4.

Una vez aplicado todos procedimientos anteriores de la capa inferior, se forma lo que se llama tranport pdu. Cuando el transport pdu ya está formado se pasa a la capa de la red que se va a explicar en la siguiente subsección.

3.6.4. Network layer

La capa de transporte se encarga de encriptar la capa de la aplicación y segmentarlo. La capa de la red se encarga de encriptar otra vez con la clave de la red para que cualquier nodo de la red pueda saber si el mensaje es para él o hay que transmitirlo.

Esta capa consiste en 2 fases. En la primera fase se realiza la encriptación, y la otra fase se encarga de ofuscar los metadatos.

En la figura 3.16 se muestra el resumen de la fase de encriptación, el algoritmo que se ha usado es el mismo que se ha usado para la capa superior de transporte, AES_CCM. En este caso no hace falta pasar el dato opcional en ningún caso, por lo tanto, sólo hay que preparar 4 parámetros, el dato, la clave, nonce y el número de bytes de mic.

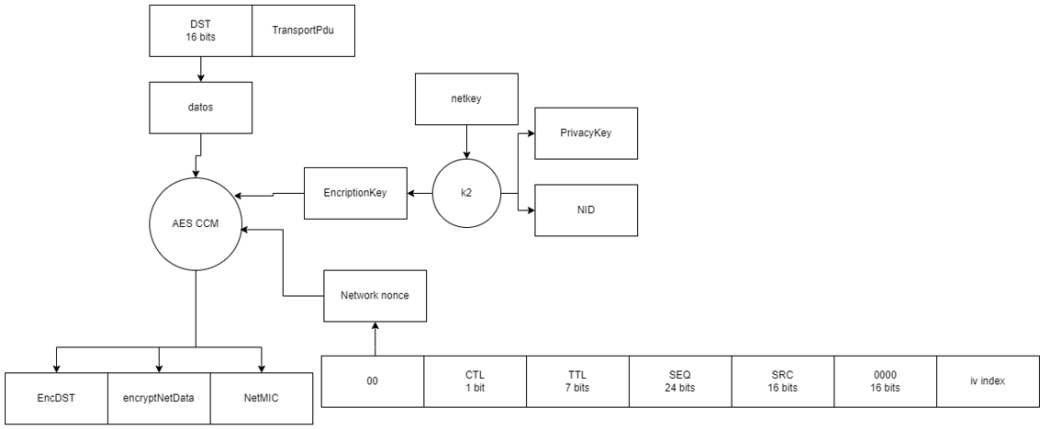

Figura 3.16: Fase de encriptación de netwrork pdu

El dato es la dirección del destino concatenado con el segmento de la capa inferior de transporte. Si hay varios segmentos, hay que generar varios pdu de la red.

La clave que se usa para encriptar es la clave después de aplicar la función k2 a la clave de la red. Esta función genera 3 cosas: privacyKey, encriptionKey y NID. En esta fase solo se usa encriptionKey para encriptar el dato.

El nonce de la red es parecido a los otros dos nonce pero tiene 2 campos más.

- CTL: para indicar si el mensaje es control o acceso.

- TTL: para indicar la vida del pdu. Cada vez que un nodo retransmite el mensaje, decrementa este valor, cuando ese valor llega 1, entonces el nodo ya no retransmite el mensaje.

El número de bytes del mic depende del campo CTL. Si es un mensaje de control, entonces es de 8 bytes y si es un mensaje de acceso, entonces es de 4 bytes.

El resultado se divide por 3 campos. Los primeros 2 bytes es la dirección de destino, los últimos 4 u 8 bytes es el mic de la red, y el resto es el contenido de la capa de transporte.

Cuando ya se tiene el dato encriptado se deben poner los campos de metadatos para saber la vida de la red, número de secuencia… En la figura 3.17 se muestra un resumen de cómo se ofusca los campos.

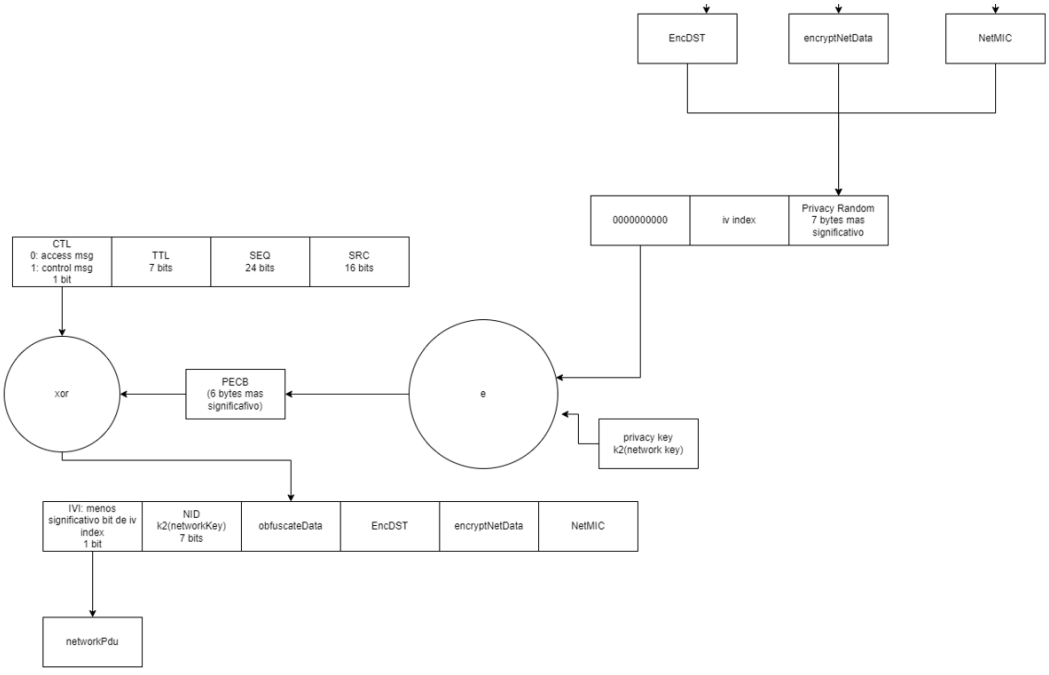

Figura 3.17: Fase de ofuscar de netwrork pdu

1. Se coge la dirección de destino, dato y mic juntos, se cogen los 7 bytes más significativos y se le pasa a la función e que es el algoritmo AES_ECB.

2. Se cogen los 6 bytes más significativos de la salida de la función e para obtener PECB.

3. Se llenan los campos de metadatos, CTL,TTL SEQ y SRC. Una vez rellenado, se hace la operación XOR con PECB. El resultado es el dato ofuscado.

Este proceso sólo ofusca los datos, por lo tanto, si se desea rehacer el proceso se hace de igual forma. Después de obtener el dato ofuscado hay que rellenar dos campos más, uno es IVI que es el bit menos significativo de iv index y NID, que se obtiene llamando la función k2.

Como se puede observar, cuando se ha encriptado el dato se ha usado el campo ttl, y este valor decrementa por cada reenvío, por lo tanto, cada nodo cuando retransmite el mensaje, decrementa el valor y vuelve a encriptar otra vez con el nuevo valor ttl.

3.7 Proxy configuration

Este es un pdu especial que sólo se usa para la configuración entre el móvil y el nodo proxy. Existen 3 configuraciones: 1 para indicar si es lista blanca o lista negra, otras 2 para añadir las direcciones a la lista o borrarlas de la lista.

Esta configuración sirve para filtrar los mensajes que se intercambian entre el nodo y el móvil. Si el móvil sólo se usa para iniciar la comunicación, entonces no es necesario configurar, ya que se autoconfigura. Solo es necesario cuando un móvil se quiere convertir como un nodo de la red.

El formato de una configuración es el mismo que el de la capa de acceso 3.8, pero en este caso todo se pone en “big endian”.

Los códigos están definidos en la tabla 6.5 de Mesh Profile, en este apartado se va a ver cómo se genera un mensaje de configuración.

En este caso, el mensaje se genera directamente pasando a la capa de la red. El campo de CTL se pone a 1 y el destino se pone a 0. Cuando se encripta, se usa proxy nonce en vez de network nonce. El formato de proxy nonce se muestra en la figura 3.18.

Como clave también se usa la clave de encriptación que se puede obtener de la función k2, en este caso se trata como un mensaje de control, por lo tanto, el número de bytes de mic es 8.

Figura 3.18: El formato del proxy nonce

Cuando un móvil se conecta a un nodo proxy, la lista se inicializa siendo del tipo lista blanca y vacía. Cuando le llega un network pdu válido, se añade la dirección fuente automáticamente a la lista, como resultado, si siempre se conecta el móvil a un proxy para enviar operaciones, esta configuración se realiza automáticamente y no es necesario realizarla. Pero si en el móvil también se simula un nodo y se suscribe a una dirección, entonces se debe configurar la dirección cada vez que se conecta a un nodo proxy.

3.8 Secure Network beacon

Existen 2 tipos de beacon, uno para aprovisionar y otro es secure network beacon, que se usa para transmitir el estado de la red. Este trabajo sólo se centra en secure network beacon, ya que el móvil sólo puede enviar este tipo de beacon.

El formato del mensaje se muestra en la figura 3.19. Este mensaje sólo se autentica y no se encripta, cualquier dispositivo puede leer el contenido.

Figura 3.19: El formato de secure network beacon

En los siguientes puntos se explica el significado de cada campo:

- El primer campo indica el tipo de beacon, el valor 1 indica que es un secure network beacon.
- El segundo campo indica el estado de la red, de momento el estado solo tiene dos procesos, el bit 0 indica el estado del refresco de la clave, y el bit 1 indica el estado de la actualización de iv index.

- El tercer campo indica el id de la red, esto se obtiene aplicando la función k3 a la clave de la red.
- El cuarto campo indica el valor de iv index, el significado de este campo se va a explicar en el siguiente punto.
- El quinto campo es un valor para autenticar que ha sido enviado por un nodo que tiene la clave de la red.

En la figura 3.20 se muestra un resumen de cómo generar el valor de autenticar.

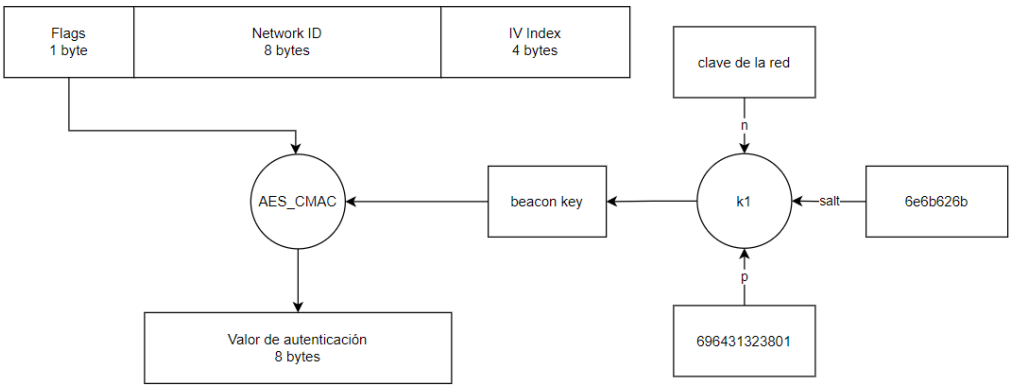

Figura 3.20: Resumen de la autenticación

Cuando el móvil conecta a un nodo proxy, el primer mensaje que le llega es un secure network beacon para indicar el estado de la red, si el nodo tiene varias clave de la red(significa que tienes varias subredes) también se enviará secure network beacon de cada red.

En la red mesh existen 2 procedimientos, 1 es el procedimiento de actualizar la clave de la red y la otra es el procedimiento de actualizar el valor de iv index. El trabajo solo se desarrolla el procedimiento de actualizar el valor de iv index, pero sí que analiza los dos procedimientos.

3.8.1. Procedimiento de actualizar la clave red

Por la razón seguridad, la especificación aconseja que cada cierto tiempo se actualiza la red, o cuando se borra los nodos. La nueva clave no se difunde por la red, sino el aprovisionador se encarga de enviar uno a uno usando la clave del

dispositivo de cada nodo. Cuando se borra un nodo hay que meter en la lista de exclusión, para no enviar la nueva clave a él. Porque puede pasar que el nodo borrado aún tenga la clave y gente maliciosa puede intentar utilizar este nodo para obtener la nueva clave, este se llama "trash-can attack".

Cada nodo tiene 3 fases, que se explica en los siguientes puntos:

1. Cuando el nodo recibe la nueva clave, y el estado de secure network beacon es 0. En esta fase el nodo sigue enviando los mensajes con la clave vieja, pero puede recibir los mensajes que están encriptados con las dos claves. Cuando el aprovisionador envía la nueva clave al nodo, el nodo se cambia a esta fase automáticamente.

2. Cuando un nodo tiene la clave nueva, y le llega un secure network beacon donde el estado de la actualización de la clave es 1, entonces se transmite a fase 2. Esta fase el nodo se encripta con la nueva clave, pero se puede recibir el mensaje con las dos claves.

3. Cuando un nodo está en la fase 2 y le llega un secure network beacon donde el estado de la actualización de la clave es 0, entonces se transmite a la fase 3, que también se entiende como el estado normal. El nodo se encripta y se recibe mensaje sólo con la clave nueva.

La transición de fase también se puede realizar con la operación de configuración y no necesita beacon. En la figura 3.21 se muestra un resumen de transición entres las fases.

El objetivo de este trabajo es realizar una librería para facilitar el uso y la gestión de la red, pero para actualizar la clave no es una operación unitaria, sino que hay que enviar uno a uno, durante el proceso puede pasar cualquier cosa, por eso, para facilitar no se añade esta funcionalidad, en cambio, se pierda la seguridad, pero sí que es posible conseguir la misma finalidad si cada cierto tiempo el usuario borra los nodos y aprovisiona con la nueva clave.

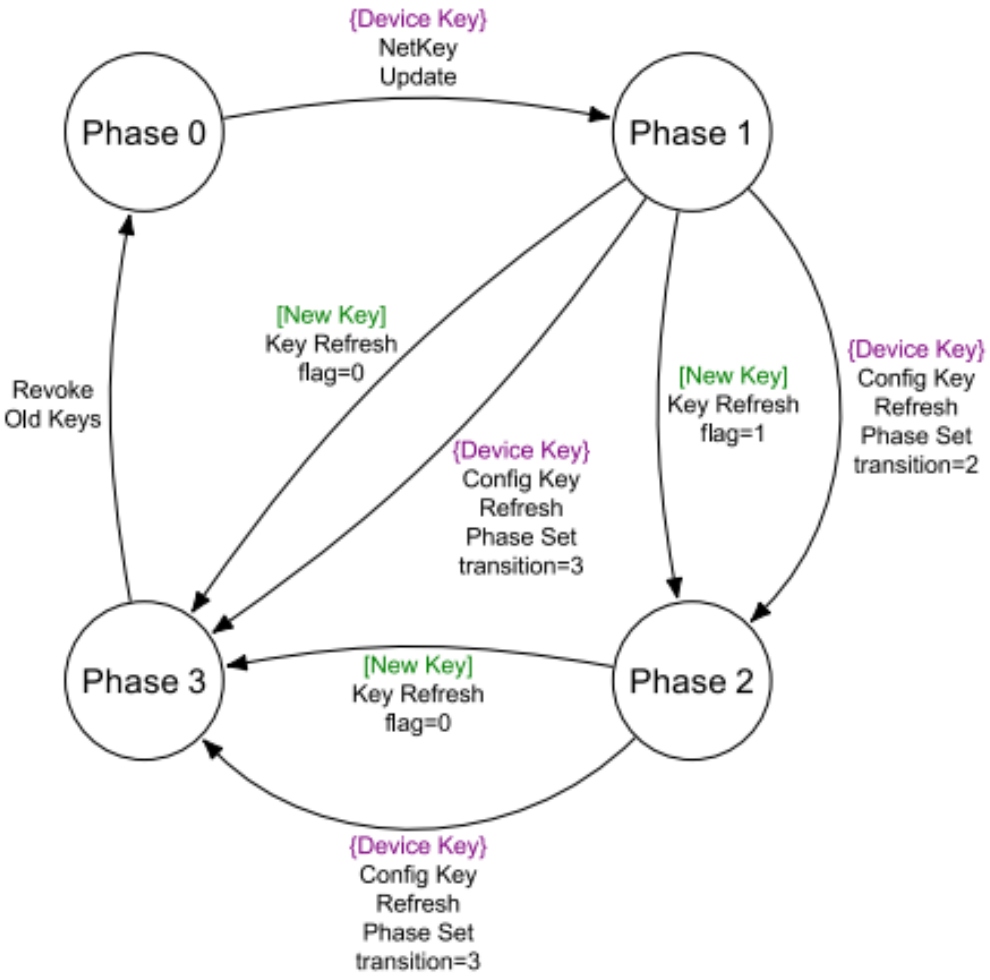

Figura 3.21: Diagrama de la actualización de clave

3.8.2. Procedimiento de actualizar iv index

Cada elemento del nodo tiene un número de secuencia propio, antes de que el número de secuencia se agote hay que actualizar el valor de iv index. El número de secuencia tiene 3 bytes. Si cada segundo el elemento se envía un mensaje, con 194 días se agota el número.

Existen 2 estados del proceso de actualizar iv index, el estado normal y el estado en progreso.

Normalmente están en el estado normal, en este estado, el nodo envía con el valor de iv index que está en el beacon, pero se puede recibir el mensaje de iv index e iv index - 1. Cuando el nodo detecta uno de sus elementos u otros elementos de otro nodo que es posible agotar dentro de 6 días, entonces empieza el proceso de la actualización de iv index.

Cuando un nodo inicializa la actualización, el nodo cambia al estado en progreso, si cuando envía el beacon se envía iv index + 1. Durante este proceso, el nodo envía con el valor iv index pero recibe mensajes de iv index e iv index + 1. En el caso de las placas de Silicon Labs, a partir de 96 horas y antes de 144 horas del proceso de la actualización, se vuelve al estado normal. Una vez vuelve al estado normal, el beacon que envía es en el estado normal y el valor de iv index es iv index +1. A partir de este momento el nodo se recibe mensajes de iv index e iv index +1 pero se envía con iv index +1(el nuevo valor de iv index).

Cuando un nodo está fuera de la red hace mucho tiempo, que el proceso de la actualización ya se ha acabado, es posible inicializar el proceso de recuperamiento de iv index. Durante este proceso si el nodo recibe un beacon con el valor de iv index menor que iv index + 42, entonces se recupera directamente, si es mayor, entonces el nodo hay que volver a aprovisionar otra vez.

Una vez actualizado iv index, el nodo no puede actualizar otra vez dentro de 192 horas que son 8 días.

Cuando durante el proceso de actualizar iv index, si un móvil quiere aprovisionar el dispositivo, hay que pasarle el estado en progreso y el valor de iv index + 1

CAPÍTULO 4

Diseño e implementación

En este capítulo se va a explicar cómo se ha realizado el diseño y la implementación para este trabajo. Se empieza por analizar el requisito por parte de la empresa, y cuáles son las acciones que van a realizar los clientes. Luego se diseña una estructura de la librería para ver cuáles son los componentes que hay que implementar, y por último se implementa cada uno de estos componentes.

4.1 Análisis de requisito

En el capítulo 3 se ha estudiado cómo se genera un mensaje y como es el procedimiento que hay que realizar para mantener la red. Aunque tenga todas las funcionalidades hechas, la mesh sigue siendo difícil de usar por la gestión de la red, tal como guardar las claves, los direcciónes, actualizar iv index, etc.

La idea de este trabajo es realizar una primera versión de mesh que sea fácil de usar, en este capítulo se piensa más o menos cómo es la primera versión de un producto final.

Cuanto menos acciones se requieran al cliente, más fácil será de usar. En los siguientes puntos se muestran las acciones que van a realizar los clientes finales:

- Aprovisionar un dispositivo: el proceso de aprovisionar un dispositivo es bastante largo, por lo tanto, la librería debe ofrecer una función para que se haga todo el proceso.
- Conectar a un nodo proxy.

- Listar el estado de un nodo o de un grupo de nodos.
- Configurar el estado de un nodo o de un grupo de nodos.
- Gestionar la red, solo la gestión que es necesaria para el cliente final, tal como la creación del grupo y el agrupamiento de los nodos.

Aparte de estas acciones existen otros requisitos que es necesario que un nodo de la red los cumpla, los cuales son:

- La red debe ser dinámica, es decir, los nodos pueden moverse físicamente a distintos lugares (fuera de la red), cuando vuelve, debe poder seguir funcionando, da igual si requiere una acción por parte del cliente o no.
- La red debe ser automantenible, cuando el nodo está siendo usado por el cliente final, debe de poder seguir usándose a largo plazo sin requerir ninguna acción, tal como actualizar iv index para alargar la vida.
- Todos los nodos tienen el mismo perfil, no se diferencia por nodos, cualquier nodo tiene la misma funcionalidad de mesh que los demás, todos deben poder retransmitir el mensaje y tener proxy activado para conectar el móvil.
- El nodo debe tener un modo de bajo consumo cuando no está siendo usado.
- Cualquier modificación del estado del nodo, da igual si es por BLE Mesh o no, debe ser informado a los dispositivos móviles que están conectados usando BLE Mesh.

La idea del producto final es que unos dispositivos tengan conectados otros dispositivos, como, por ejemplo, la luz, el motor o cualquier otro producto. Estos dispositivos pueden estar distribuidos en cualquier zona del mundo(donde no se tiene acceso a la red), pero se puede montar una red malla de forma rápida y se puede controlar cada uno de ellos a distancia. Los productos finales cuando se están usando, están conectados a una fuente de corriente, entonces no tiene el problema del consumo, pero cuando no están siendo usados, se guarda en un almacén y está apagado todo, pero tiene BLE Mesh funcionado para que en cualquier momento se puedan encender a distancia.

4.2 Preparación del entorno

Para desarrollar el programa de móvil existen muchos frameworks, React, ionic, Cordova o incluye directamente un proyecto de android studio o Xcode. En este trabajo se va a desarrollar un "module" con "npm" y usando el lenguaje "Typescript".

La ventaja de desarrollar un proyecto de npm es que se puede usar para muchos proyectos de Javascript, además también se puede usar los módulos que han desarrollado otras personas.

La principal ventaja de desarrollar un proyecto de npm es que es posible usar en Android e iOS, ya que el código es javascript y depende del navegador y no de la plataforma.

La ventaja de usar Typescript es que facilita desarrollar un proyecto más robusto, por el hecho de que hay que especificar bien los tipos de parámetros y esto también ayuda al entorno de desarrollo a autocompletar los códigos. Otra ventaja es que el código escrito en Typescript se puede compilar en distintas formas de importación de Javascript, tal como CommonJS, AMD, ESM, UMD, etc.

En este proyecto se va a compilar el código en UMD y luego se usa la herramienta browserify[1] para generar la librería final que se puede importar en cualquier html.

Para probar la librería se genera un proyecto de helloworld de Cordova, en este proyecto se importa el fichero que se genera por browserify y se prueba en Android e iOS. La librería debe funciona igual en las 2 plataformas.

También se ha usado otras herramientas para mantener el proyecto, tal como git, como el trabajo es de una persona, no va a haber conflicto, pues solo se usa la funcionalidad de control de versión de git. Eslint[2], para dar el formato a los códigos. Typedoc, para generar la documentación de API(application programming interface).

[1]Herramienta de npm que permite convertir distintos ficheros de javascript en un solo

[2]Herramienta de npm para dar formato a los códigos

En la parte del programa para el microcontrolador se ha usado Simplicity Studio como el IDE de desarrollo. Es una herramienta de Silicon Labs para desarrollar el programa de sus microcontroladores, cuando se conecta la placa se detecta automáticamente el modelo, también es posible programar una placa personalizada pero con el microcontrolador de Silicon Labs.

4.3 Diseño de la estructura de la librería

En la figura 4.1 muestra un diseño simple de que componentes va a tener la librería. Como la librería es una primera versión, se debe tener en cuenta el acoplamiento de cada componente, porque es posible que se cambie en el futuro.

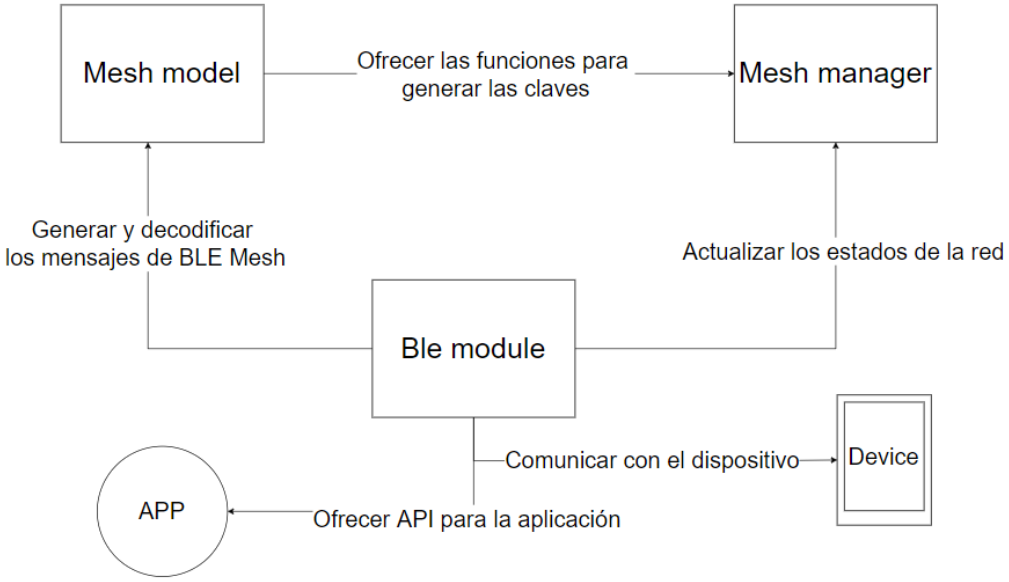

Figura 4.1: Estructura de la librería

Primero se va a explicar la utilidad de cada componente.

Mesh model. Este componente se encarga de realizar todos los trabajos relacionados con las capas de los mensajes que se han explicado en el capítulo 3. La idea es pasándole los parámetros, genera un mensaje que se puede enviar directamente al móvil.

Los mensajes que suele quiere generar un usuario son los mensajes de la capa de acceso, activar o desactivar algo, enviar el mensaje de generic onoff, añadir la

clave, etc. Por lo tanto, la librería debe poder generar todos los pdus de las capas inferiores.

Un mensaje de la capa de acceso es posible ser segmentado por distintos segmentos, así que la salida debe ser una lista donde se guardan todos los segmentos. Como este componente sólo genera el mensaje y para conseguir la máxima compatibilidad, se ha decidido devolver una lista de string donde el contenido es hexadecimal. Como string es un tipo muy común en muchos lenguajes de programación, la salida se puede usar junto con distintas librerías de comunicación Bluetooth.

Ya que la librería tiene todas las funciones de cifrado, porque lo necesita para generar mensajes de distintas capas, pues también ofrece las funciones de generar clave y generar uuid para la dirección de grupo.

Mesh manager. En el capítulo 3 se ha visto que en una red BLE Mesh es necesario guardar muchos objetos, por ejemplo, cada vez que se envía un mensaje hay que incrementar el valor de seq, una red necesita una clave la red y para enviar mensaje de generic onoff es necesario una clave de la aplicación, cada dispositivo tiene su propia dirección y también se genera la clave del dispositivo durante el proceso de aprovisionamiento, etc.

La idea de este componente es gestionar la red, crear las claves, asignar las direcciones correspondientes y guardar las informaciones de la red. En este trabajo se ha elegido IndexedDb [3] como la base de datos, ya existe en casi todos los navegadores, incluido en las aplicaciones que ha hecho con Cordova.

Este componente es muy aconsejable que se cambie por otra base de datos, porque IndexdDb es una base de datos local, y no se puede sincronizar si hay varios dispositivos que están siendo controlados por la misma red.

Ble module. Este componente se encarga de la parte de comunicación entre móvil, al final BLE Mesh en la capa física es la misma que BLE. Cuando el móvil conecta a un dispositivo final es a través de los servicios de BLE. La idea de este componente es gestionar la conexión con los dispositivos y ofrecer las funciones de enviar mensaje.

[3] https://www.w3.org/TR/IndexedDB/

En este componente estaría todas las funciones de enviar comandos a cada dispositivo o grupo. Este componente también usa los otros componentes, pide datos desde Manager y lo pasa a Model para generar los mensajes correspondientes.

La aplicación final es una aplicación desarrollada con Cordova, aquí se ha elegido un plugin "Bluetooth Low Energy (BLE) Central Plugin for Apache Cordova"[4] para la parte de la comunicación, pero si en un futuro se quiere migrar la aplicación a otra plataforma sólo hay que modificar este componente para que soporte la nueva librería de la comunicación.

En las siguientes secciones se va a explicar cada uno de los componentes con más detalle.

4.4 Mesh model

Para desarrollar las capas de BLE Mesh se necesitan 4 algoritmos, AES_CMAC, AES_CCM, AES_ECB y elliptic p256. Se puede buscar estos algoritmos en la página de npm, pero los módulos que están en npm no siempre se pueden usar para el navegador, existen algunos que sólo se puede usar en NodeJS.

En este trabajo, se ha tomado este ejemplo[5] con base en este componente, es un ejemplo de parte cliente de proxy que funciona en un navegador, permite a los usuarios conectar a un dispositivo que tiene proxy activado y puede enviar unos mensajes. En la figura 4.2 se muestra como es la página.

Aunque faltan bastantes funciones que se necesitan para el componente, pero este ejemplo contiene todas las funciones de AES que se necesitan para el cifrado. El algoritmo elliptic p256 se ha elegido del módulo elliptic[5], que sí que funciona en un navegador después de compilar.

En la figura 4.3 se muestra un resumen de este componente, para la división entre los ficheros también se ha tomado el ejemplo como la referencia.

[4] https://github.com/don/cordova-plugin-ble-central
[5] https://www.npmjs.com/package/elliptic

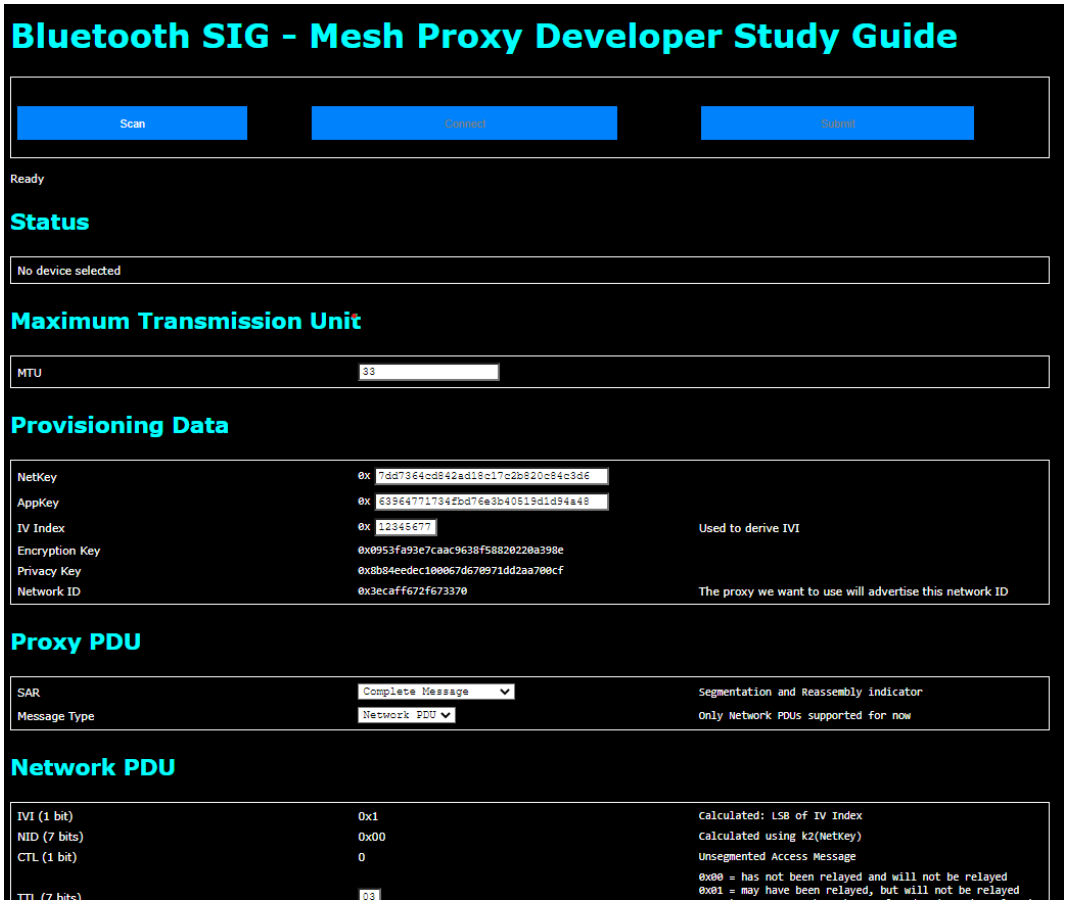

Figura 4.2: Ejemplo de parte cliente de proxy

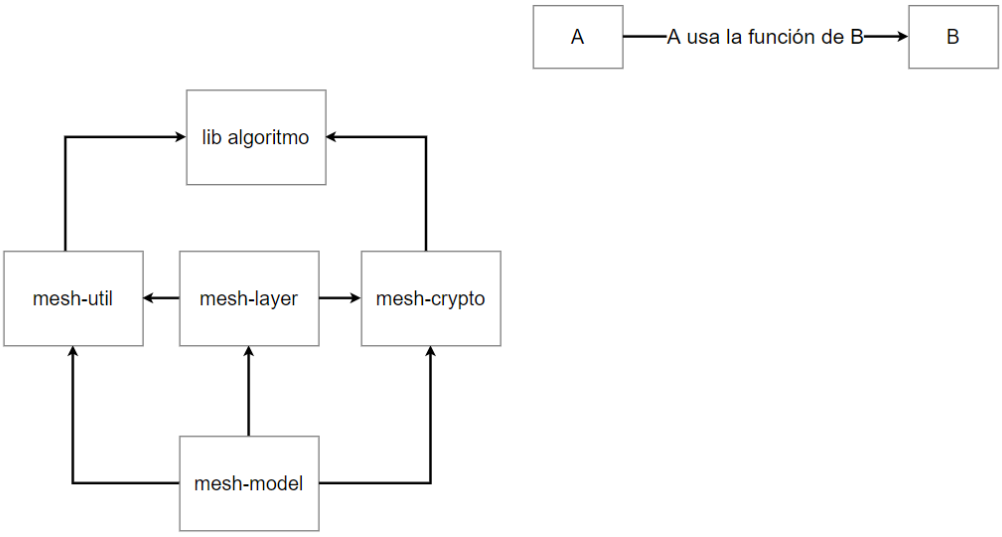

Figura 4.3: Estructura de mesh model

4.4.1. Mesh util

Como su nombre lo indica, aquí están las funciones de las utilidades, en la siguiente lista se muestra las funciones que se tiene:

- Convertir de bytes a hexadecimal e inversa.
- Realizar la operación Xor para 2 Uint8Array que se usa para ofuscar el metadato de la capa de la red.
- Sacar N bits más significativo o menos significativo de un hexadecimal, para descomponer los campos.
- Convertir int a hexadecimal y viceversa.
- Cambiar endian de un hexadecimal.
- Otras funciones pequeñas tal como obtener tiempo actual, obtener el campo ctl y ttl, etc.

Los campos de la capa inferior de transporte segmentado son los campos que no se puede guardar en un byte, por lo tanto, también se ha creado una clase en este fichero, que es FieldArray.

Esta clase tiene 3 funciones, add, set y fieldArrayToHex, cada vez que se llama la función add hay que pasar un int donde indicar el valor, otro int para indicar cuántos bits ocupa el campo y un nombre que es opcional para indicar el nombre del campo. Esta función concatena el campo hacia la izquierda.

La función set solo sirve para los campos que tienen nombre, puede cambiar el contenido del campo, esta función se usa para el campo SegO que se incrementa para cada segmento.

La función fieldArrayToHex devuelve todos los campos concatenados en hexadecimal.

Con la ayuda de esta clase se genera los metadatos de una manera muy sencilla, se puede pasar directamente el valor de cada campo y el número de bits que ocupa, una vez llenado todo el campo, se puede generar todo campo concatenado llamando una función.

4.4.2. Mesh crypto y layer

En el fichero mesh crypto se desarrolla las funciones que están relacionadas con el cifrado, tal como las funciones que se mencionan en la sección 3.4, k1,k2,k3,k4,s1,e,AES_CMAC.

Aparte de las funciones básicas, también están otras funciones tal como generar los diferentes nonce, cifrar el mensaje y descifrar, generar una clave de elliptic p256, ofuscar la capa de la red y rehacer el proceso de ofuscar, convertir uuid a una dirección de 2 bytes.

El fichero layer tiene todas las funciones de convertir el mensaje de la capa superior a una capa inferior. Aquí también tiene dos funciones muy importantes, una que es pasándole un mensaje de acceso, que se puede generar un mensaje de proxy que se puede enviar directamente al dispositivo. Otra función es pasándole un mensaje completo, puede distinguir el tipo del mensaje, si es network, beacon o proxy status, y descifrarlo.

4.4.3. Mesh model

En este fichero están todas las funciones que se usan para otros componentes. Para la forma de diseñar las funciones se ha tomado como referencia el módulo de parte cliente Amazon S3. Cada función recibe un objeto, y cada función tiene su propio tipo de objeto.

En este fichero existe tipos que son generales que se muestra en la figura 4.4. Para generar un mensaje de la red es siempre necesita seq(número secuencia), src(la dirección fuente) ivIndex(iv index de la red) y netKey(la clave de la red). Luego depende del tipo del mensaje, se necesita otros parámetros.

Una función se define como el ejemplo que se muestra en la figura 4.4.

Para generar un mensaje de añadir la clave de la aplicación se necesita pasar un objeto de AddAppKeyInput, donde este tipo es una unión de otros tipos.

La ventaja de hacer así los parámetros es que el usuario puede ignorar la orden de pasar los parámetros, además si está en un proyecto de Typescript, algún

```
type GenericMetadata = {
    seq: string;
    src: string;
    ivIndex: string;
    netKey: string;
};
type OptionalMetadata = {
    ttl: string;
    labelUuid: string;
    dst: string;
    appKey: string;
    deviceKey: string;
    szmic?: boolean;
};
```

Figura 4.4: Tipo generar y opcional de la red

```
type AddAppKeyInput = {
    netKeyIndex: number
    appKeyIndex: number
    keyValue: string
} & GenericMetadata & Pick<OptionalMetadata, 'deviceKey' | 'dst' | 'ttl' | 'szmic'>;
/**
 * This function generate msg for adding app key to a device.
 * The correspond response is config app key status.
 * @param metaData
 * @returns
 */
const configAppKeyAdd = (metaData: AddAppKeyInput): string[] => {
    const { keyValue, appKeyIndex, netKeyIndex, deviceKey, dst, ivIndex, netKey, seq, src, ttl, szmic } = metaData
    const accessPayload = OPCODE_CONFIG_APPKEY_ADD +
        MeshUtil.generateNetKeyAndAppKeyIndex(netKeyIndex, appKeyIndex) +
        keyValue
    return accessToNetwork({
        accessPayload,
        ivIndex, netKey, seq, src, deviceKey, dst, ttl, szmic
    })
}
```

Figura 4.5: La función de añadir la clave de la aplicación

editor le indica los campos que no se han puesto y que no es opcional. Tampoco se va a dejar a compilar si no se ha pasado todos los parámetros necesarios.

4.5 Mesh manager

En este componente solo existirán dos ficheros, uno que es meshdb y otro que es mesh manager. La idea de dividir en dos ficheros es para que se facilite el cambio de la base de datos en el futuro.

En el fichero de mesh manager están todas las funciones de gestionar los elementos de la red, tal como, crear un grupo, borrar un nodo, obtener labeluuid de un grupo, obtener iv index de la red, etc.

Para los demás componentes solo existe este fichero, y en meshdb es donde realmente está la gestión de la base de datos. En este trabajo se ha utilizado IndexDb. Esta base de datos no es relacional, se guardan los objetos como la tabla. En la figura 4.6 se muestran los distintos objetos que se guardan en la base de datos.

Figura 4.6: Las tablas de la base de datos 1.

Como no es una base de datos relacional, no existen ninguna restricción entre diferentes tablas. El usuario puede borrar una clave pero sin borrar la referencia que tiene guardada de dispositivo.

Esta base de datos no cumple con la especificación de BLE Mesh, está diseñada solo para este proyecto.

En los siguientes puntos se va a explicar cada tabla más detalladamente:

- Netkeys: La tabla para guardar las claves de la red.
  - El campo index va a ser el índice de la red, es la clave primaria.
  - El campo networkId es el id de la clave que se envía por secure network beacon, está indexado para facilitar la búsqueda, cuando se conecta a un nodo proxy pero recibe un id que no se encuentra en la base de datos significa que se ha conectado a un nodo de otra red, es posible pasar porque dos claves diferentes pueden tener el mismo id, para simplificar, se ha añadido la restricción de valor único, cuando se repite id, impide que se añade a la base de datos.

- El campo netkey es donde se guarda el valor de la clave.
- El campo oldkey se usa para guardar la clave vieja cuando está actualizando la red, pero la librería no tiene la funcionalidad de actualizar la clave hecha, este campo se reserva para el futuro.

- AppKeys: La tabla para guardar las claves de la aplicación.
  - El campo index, appkey y oldkey es lo mismo que la tabla NetKeys.
  - El campo aid está indexado para encontrar la clave de manera rápida, porque cuando un mensaje que está cifrado en la clave de la aplicación, en la capa inferior de transporte se pone el valor de aid para indicar qué clave ha usado, pero este campo solo tiene 6 bits, es muy probable que dos claves de la aplicación coincidan en el valor de aid. Por lo tanto, en este campo no se pone la restricción del valor único, y cuando se descifra, se debe probar con todas las claves del mismo aid.
  - El campo boundNetIndex indica con qué clave de la red está enlazada esa clave de la aplicación.

- Devices: La tabla para guardar toda la información del nodo.
  - El campo addr es la clave primaria, porque no puede repetirse y es el valor con el que se identifica el nodo, el valor es la dirección del elemento 0 del nodo.
  - El campo mac no es necesario para la red BLE Mesh, pero el usuario tiene que poder identificar la dirección de la red de un nodo físico, en este caso se enlaza la dirección mac del dispositivo con la dirección de la red.
  - Los campos proxy, relay, relayRetransmitCount y relayRetransmitIntervalSteps son las informaciones que se obtienen en el dato de composición.
  - Los campos netKeyIndex y appKeyIndex son dos listas para guardar los índeces de las claves que tienen el nodo.
  - El campo deviceKey es el campo donde se guarda la clave del dispositivo.

- El campo elements es una lista donde se guarda todos los elementos del nodo, cada elemento tiene una dirección y una lista para guardar los models que tienen en este elemento.
- El campo modelId guarda el identificador del model.
- El campo appKeyBounded guarda las claves de la aplicación que están enlazados con el model.
- El campo subAddrList es una lista donde se guarda las direcciones que se están suscribiendo el model.
- El campo pubAddr guarda la dirección de publicación del model.

- VirtualDevices: Cuando un dispositivo móvil envía un mensaje a un nodo proxy hay que indicar una dirección fuente, por eso, a cada móvil se le asigna una dirección como si fuera un nodo de la red.
  - El campo addr es la dirección que se ha asignado al dispositivo.
  - En el campo seq se guarda el número secuencia de la dicha dirección.
  - El campo name solo se usa para que el usuario ponga una etiqueta a la dirección. No está indexada para la búsqueda y tampoco es el valor único.

- Groups: Es la tabla para guardar los grupos de la red.
  - El campo addr es la clave primaria. Es posible tener dos labeluuid que se pueden convertir en la misma dirección, pero para reducir el cálculo que se hace cada nodo, se ha dejado addr como la clave primaria para que no se repita.
  - El campo labelUuid se guarda el valor de labelUuid que se usa para el desciframiento.
  - El campo name se crea para facilitar al usuario identificar el grupo, este campo tiene la restricción del valor único y está indexado para facilitar la búsqueda.

Estas tablas guardan los objetos generales que pueden ser varios, pero existen otros datos como iv index o tiempo transcurrido después de iniciar la actualiza-

ción de iv index. Estos datos que solo existen uno por red, se guardan en una tabla que se llama GenericData que se muestra en la figura 4.7.

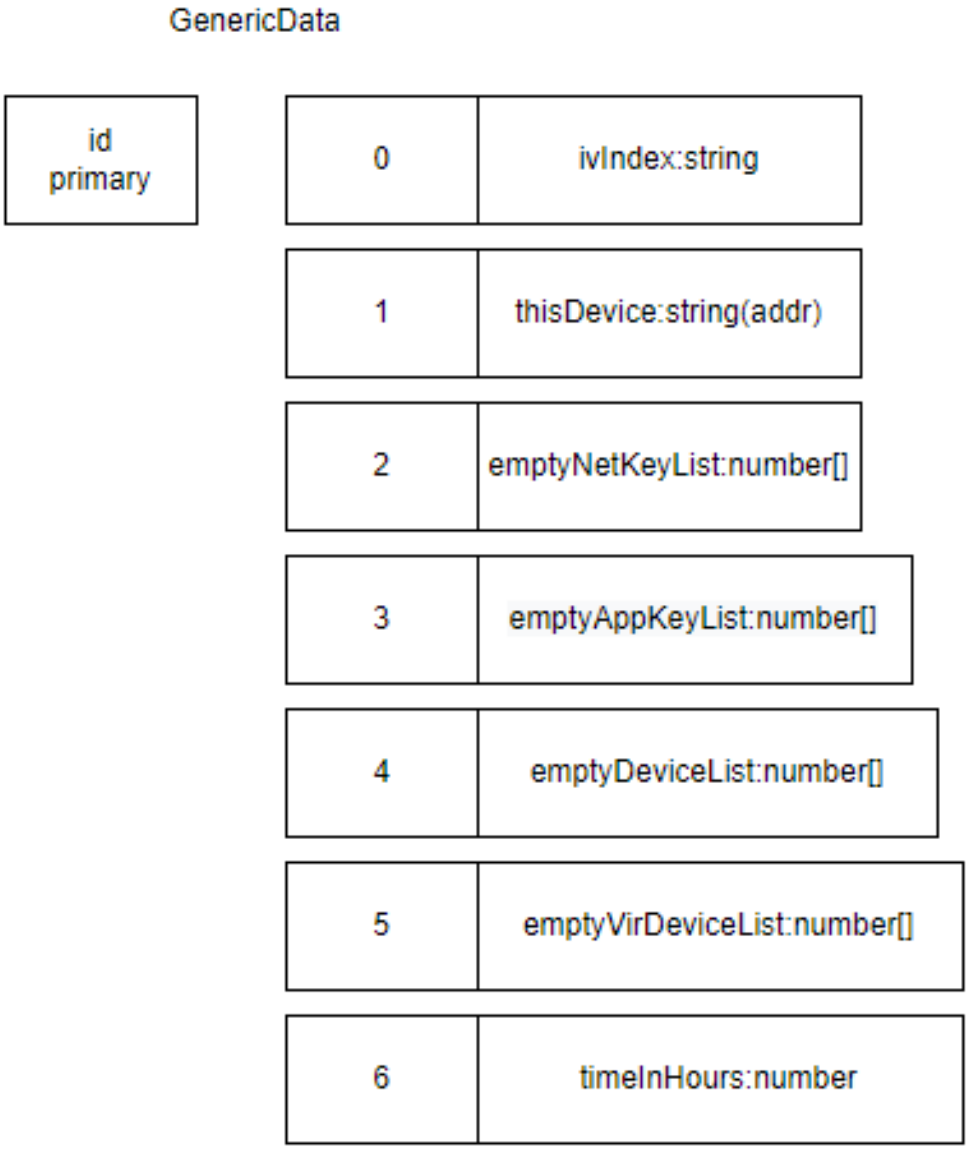

Figura 4.7: La tablas de la base de datos 2.

El campo id es la clave primaria y es un tipo enumerable para indicar que datos se guarda. Esta tabla tiene una estructura de clave y valor. El tipo del valor depende del id.

- id 0: Aquí se guarda el valor de ivindex de la red
- id 1: Aquí se guarda la dirección del dispositivo móvil, para saber qué dispositivo virtual pertenece al móvil actual.

- id 2-5: Aquí se guarda una lista para saber dónde hay hueco.
- id 6: Aquí se guardan las horas que han pasado después de iniciar la actualización de iv index.

Las claves y los dispositivos necesitan una dirección o índice para identificarlo. Se puede crear un puntero que se guarda la última dirección que se ha asignado, pero cuando llega al final entonces deja de funcionar el puntero. La red es dinámica, el usuario puede borrar los nodos o las claves, en este caso se libera las direcciones que están ocupadas anteriormente. Para resolver este problema se ha diseñado una clase que se llama EmptyList.

Las claves y el dispositivo virtual solo van a utilizar una dircción cada vez. Pero el nodo puede tener varios elementos y es posible usar varias direcciones. Cuando se aprovisiona un dispositivo sólo hay que pasarle la dirección del elemento 0, si el nodo tiene más elementos, se cogen las direcciones consecutivas, es decir, si un nodo tiene 3 elementos y le asigna la dirección 1, entonces, 1 es la dirección del elemento 0, 2 es del elemento 1 y 3 es del elemento 2. Por eso, cuando se reserva una dirección para un nodo, hay que comprobar que las siguientes n direcciones tampoco están siendo usadas.

La clase EmptyList tiene una lista, en esta lista se guardan los huecos libres que hay, por ejemplo, si la dirección 1 hasta 10(ambos inclusivos) están libres, entonces es una lista donde contiene 2 elementos [1,10]. La lista tiene 2 propiedades:

1. La lista siempre tiene los elementos en pares, es decir, si sólo existe una dirección libre que es la dirección 5, entonces el contenido de la lista es [5,5].
2. La lista está ordenada de menor a mayor, es decir, si las direcciones 1 a 5 y 7 a 10 son libres, entonces se guardan como [1,5,7,10]. Esto puede ocurrir cuando el usuario borra un nodo y libera los elementos por la mitad.

La clase va a tener 3 funciones, una que es obtener una dirección que tiene n direcciones libres consecutivas. Gracias a la segunda propiedad, esta función se puede realizar de manera sencilla, que es recorrer la lista de dos en dos. Si la diferencia entre el segundo valor y el primer valor es mayor que n - 1, entonces devuelve el primer valor. La complejidad de la función es lineal.

Otra función es reservar n direcciones a partir de una dirección. El proceso de aprovisionamiento acaba después de enviar la dirección, por eso, cuando obtiene una dirección que tiene n huecos libres no significa que hay que reservarlo ya, porque puede pasar el error durante el envío del mensaje. Esta función se llama cuando realmente ha acabado el aprovisionamiento. La función busca la dirección inicio, y mira si el siguiente valor coincide a la dirección inicio + n - 1, si coinciden, entonces se rellena el hueco justamente. Se debe quitar los dos valores de la lista, y si no coincide, hay que modificar la dirección inicio del hueco. Como la lista está ordenada, la complejidad de esta función también es lineal.

Última función es librar n direcciones a partir de una dirección, esta función es la más compleja. Se ha dibujado un diagrama de flujo para esta función que se muestra en la figura 4.8. Esta función también tiene una complejidad lineal.

La idea de la función librar es intentar mantener las 2 propiedades de la lista. Primero se intenta buscar dónde se debe poner el nuevo hueco, y luego mira si el hueco conecta con otros huecos o no.

Todas las funciones tienen una complejidad espacial y temporal lineal que depende del número de huecos que hay. Pero como la lista está ordenada y siempre intenta buscar un hueco que cabe, además para este trabajo todos los nodos van a tener la misma cantidad de elementos, por lo tanto, en el uso real no va a haber muchos huecos y los costes se acercan mucho a constantes.

Las funciones están pensadas solo para el uso correcto, es decir, si se reserva dos veces la misma dirección, entonces la lista pierde el sentido. La base de datos tiene la funcionalidad de transacción, cuando se implementa la función de añadir un dispositivo nuevo, la actualización de la lista y la inserción de un nodo deben estar en una transacción para asegurar el correcto funcionamiento de la lista.

El índice de las claves son 12 bits, esto significa que se puede guardar 4096 claves donde el índice empieza desde 0 a 4095.

En una red BLE Mesh, puede haber 32766 direcciones para los elementos, de 1 a 32767(0x3FFF). En este trabajo se reserva 511(0x0001 a 0x0200) para los dispositivos virtuales, y los restos todo para los nodos (0x0201 a 0x7FFF).

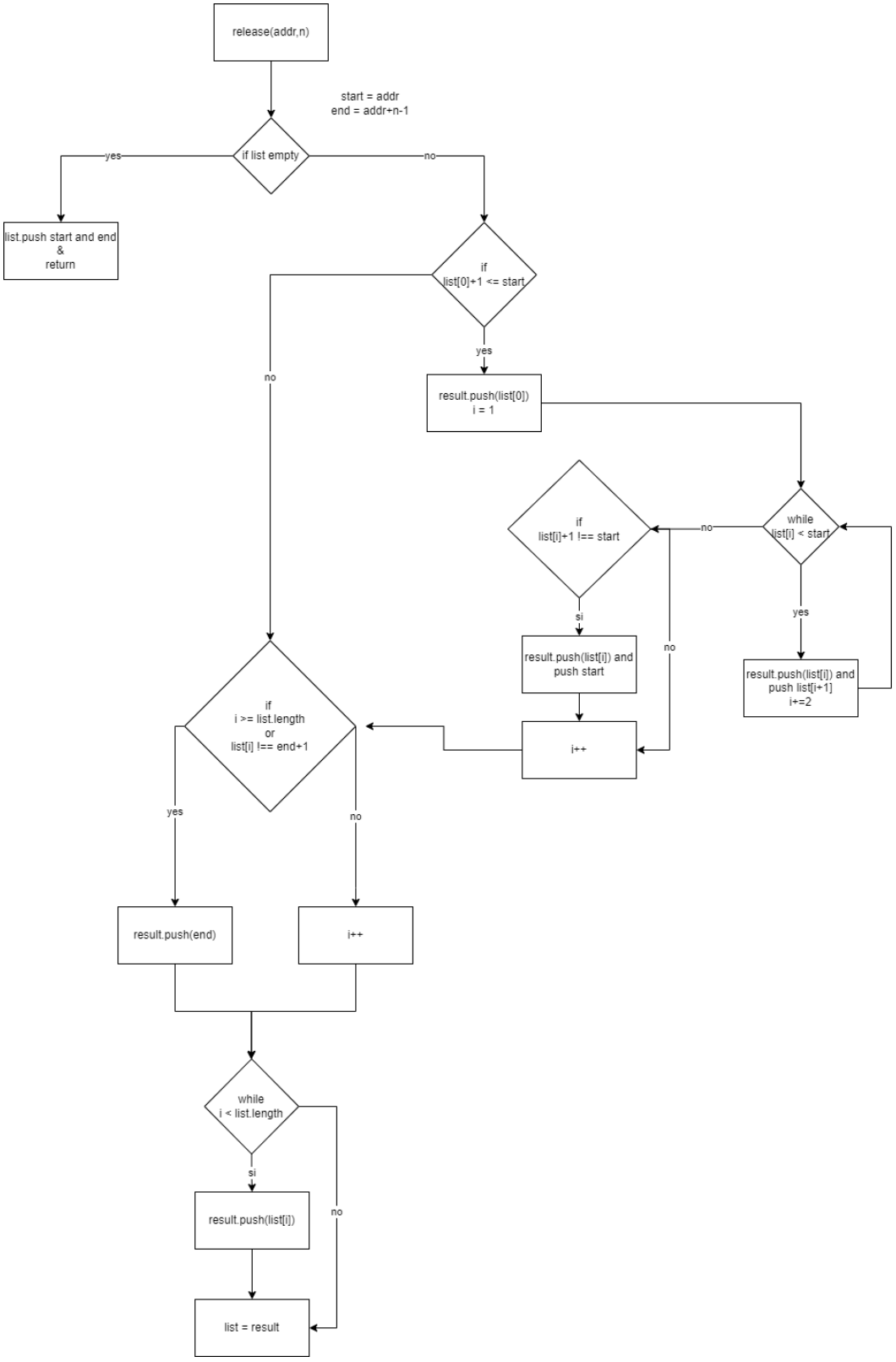

Figura 4.8: El diagrama de flujo de liberar las direcciones.

4.6 Ble module

Este componente se encarga de usar los otros dos componentes para conseguir las funcionalidades que se han analizado en la sección 4.1. Para la parte de comunicación a través de BLE se ha cogido ble central (Bluetooth Low Energy (BLE) Central Plugin for Apache Cordova), este plugin funciona en Android e iOS, aunque tiene bastantes diferencias entre las dos plataformas, pero la parte de escritura y lectura son igual en las dos plataformas.

4.6.1. Análisis del plugin

Antes de usar hay que ver cómo funciona esta librería. Cuando se quiere buscar un dispositivo Bluetooth, hay que usar la función scan. Los dispositivos BLE todo tienen una base de datos que se llama GATT, en esta base de datos se guardan los servicios que tienen activado. Cuando se quiere buscar un dispositivo que no está aprovisionado, hay que escanear los dispositivos que tienen el servicio de aprovisionamiento, que es 0x1827 y cuando se quiere conectar a un dispositivo proxy, hay que escanear el servicio proxy, que es 0x1828.

Un nodo BLE Mesh, no puede tener los dos servicios activados a la vez, cuando está aprovisionado se oculta el servicio de aprovisionamiento y se ofrece el servicio del proxy si es necesario.

Para conectar a un dispositivo se necesita un parámetro id, que es diferente en Android e iOS, en el caso de Android, este id es la dirección mac del dispositivo, un string donde se pone la dirección de mac que se separa por ':'. Pero en el caso de iOS esta id es un valor uuid que se obtiene usando la función scan.

El servicio de BLE puede tener distintas características y cada característica puede tener diferentes propiedades. En el caso del servicio aprovisionamiento y proxy tiene dos características, uno se llama in para enviar mensaje a un nodo, y otro se llama out para recibir el mensaje. La característica in tiene solo la propiedad writeWithoutResponse activada y out sólo tiene notify activada como se muestra en la figura 4.9 y 4.10.

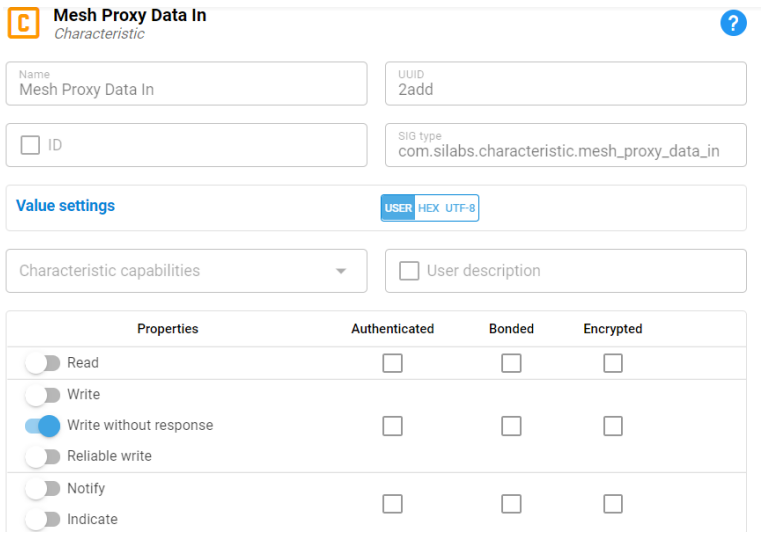

Figura 4.9: La característica de proxy data in

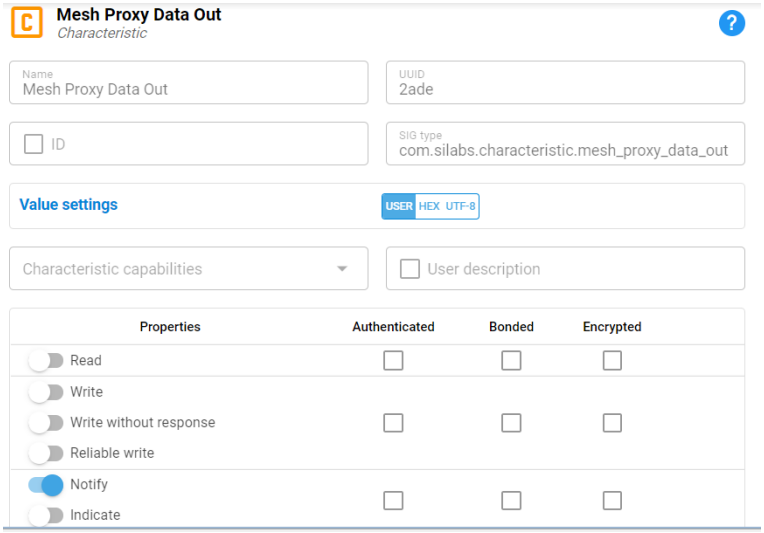

Figura 4.10: La característica de proxy data out

Para enviar un mensaje a un nodo hay que usar la función writeWithoutResponse que necesita pasarle id del dispositivo que se va a enviar(debe estar conectado), el identificador del servicio, el identificador de la característica, el dato, y dos callbacks, una para cuando ha sido éxito y otra para cuando ha fallado.

Para recibir el mensaje hay que llamar startNotification pasarle id del dispositivo, el identificador del servicio, el identificador de la característica, una callback que se ejecuta cada vez que se recibe un mensaje y una callback que indica si se ha producido error cuando empieza la notificación.

Aunque existen muchas más, pero las que se van a usar son escanear, conectar, enviar, recibir y desconectar.

4.6.2. La función de conectar

Un usuario cuando identifica un dispositivo no puede identificarlo con la dirección de la red. Se identifica como un objeto físico. En este proyecto, los dispositivos finales van a tener un código QR donde el contenido es la dirección MAC del dispositivo, como se muestra en la figura 4.11. Por lo tanto, la función conectar debe poder conectarse a un dispositivo usando la dirección mac.

Figura 4.11: El código QR del dispositivo

En Android se puede conectar directamente con la dirección MAC, pero en iOS no tiene la información de la dirección MAC del dispositivo. Para resolver este problema, los dispositivos van a enviar un advertising cada cierto tiempo donde indica su dirección MAC. Gracias a esta funcionalidad del dispositivo, en caso de iOS, la función de conectar escanea unos segundos a su alrededor, si encuentra que un dispositivo que está enviando advertising coincide con la dirección MAC que le ha pasado, pues coge el id(uuid), y se conecta al dispositivo.

En el análisis del plugin 4.6.1 se ha visto que cuando se envía el mensaje a un nodo necesita el id, por lo tanto, este valor debería estar guardado. Además, cuando se conecta puede ser que se necesite el servicio de aprovisionamiento o el servicio de proxy. Como la recepción del mensaje es asincrónica, también se necesita un eventlistener para lanzar un evento cuando se ha recibido el mensaje. Para resolver todos estos problemas se ha diseñado una clase BleDevice.

4.6.3. BleDevice

Es una clase creada para gestionar el dispositivo conectado. En la figura 4.12 se muestra el constructor de la clase. El constructor sólo recibe dos parámetros, uno es el dato que se obtiene una vez conectado al dispositivo, el otro parámetro es opcional donde indica que la conexión es para el aprovisionamiento o es para usar proxy.

Figura 4.12: El constructor de la clace BleDevice

En los siguientes puntos se explica el uso de cada atributo:

- deviceInfo: se guarda el dato cuando se conecta a un dispositivo, aquí está el id del dispositivo.
- init: una variable para indicar si ha inicializado correctamente o no.
- msgList: un diccionario para guardar los mensajes de la capa de transporte.
- proxyMsg: la variable temporal para guardar los segmentos del mensaje proxy.
- initializingPromise: la promesa sirve para esperar cuando se ha acabado el proceso de la inicialización.
- receiver: un eventListener para emitir evento cuando recibe un mensaje con el código de operación.
- genericNotifier: otro eventListener para notificar cuando se ha llegado mensaje de genericonoff.

- isProxy: la variable para indicar hay que enviar al servicio proxy o al servicio aprovisionamiento.
- writing: la variable para indicar si ya hay un mensaje enviando o no.

Para el plugin, la conexión acaba una vez conectado, pero para BLE Mesh hay que realizar más pasos. Primero hay que registrar una callback cuando recibe el mensaje. Cuando un móvil conecta a un servicio proxy, le llega un mensaje de secure network beacon para indicar el estado de la red. La función realmente acaba de conectar después de recibir beacon y actualiza el estado de la red. Por esta razón, se ha creado una promesa de inicialización, cuando se crea este objeto, hay que esperar esta promesa que acaba para saber si se ha conectado o no.

Para recibir el mensaje se realizan los siguientes procesos:

1. Cuando un nodo envía algo al móvil, se le envía proxy pdu. Este pdu es posible estar segmentado, por lo tanto, se ha creado una variable proxyMsg para guardar los segmentos. Una vez recibido todos los segmentos, lanza un evento de meshMsg.

2. Cuando el evento meshMsg está siendo lanzado, el contenido ya es un mensaje de la red, aquí puede ser de 4 tipos, network pdu, provisioning pdu, secure network beacon y proxy filter. Si el tipo es secure network beacon o provisoning, pues se lanza los eventos correspondientes para que otras funciones se traten, pero si es network pdu debe descifrar la capa de la red y comprobar si es un segmento o es un mensaje completo. Cuando se descifra hay que intentar dos veces, porque la red mesh tiene que soportar iv index e iv index -1.

   En el caso de que es un segmento, hay que guardarlo en msgList, donde la clave es la dirección fuente y el valor es un objeto lo cual contiene los metadatos y los segmentos.

   Si es un mensaje no segmentado o es el último segmento del mensaje, entonces se descifra la capa superior de transporte.

3. Una vez ha llegado todos los segmentos se descifra la capa superior del transporte, aquí se descifra con la clave de la aplicación o la clave del dis-

positivo, en el caso de usa la clave de la aplicación es posible que hay varias claves que tiene el mismo aid, y hay que probar con todas las claves con el mismo aid.

Después de descifrar se comprueba si el código de la operación es de genericonoff o no, en el caso de que sí, hay lanzar un evento con genericNotifier con la dirección fuente y el estado actual del nodo.

Da igual si es de generic onoff o no, una vez descifrado se obtiene el código de la operación, y se lanza un evento con el código de la operación.

Cuando el evento secureBeacon está lanzado, hay que comprobar que el nodo conectado pertenece a la misma red que el móvil. También se comprueba el estado de la actualización de iv index y se debe seguir el procedimiento que se ha explicado en la subsección 3.8.2.

El microcontrolador de Silicon Labs intenta acabar el proceso de la actualización entre 4 y 6 días. Por eso, el móvil inicia el proceso de la actualización iv index cuando detecta que el número de secuencia es posible que llegue al límite después de 10 días(con un uso normal), dejando un margen de 4 días para que el usuario no note ningún cambio.

Si la conexión es para el servicio de aprovisionamiento, la promesa de inicialización se resuelve una vez registrado los eventos de tratamiento de mensaje, pero si la conexión es para el uso proxy, la promesa de inicialización se resuelve después de tratar secure network beacon, si no, es posible que el nodo y el móvil tengan una diferencia de iv index muy grande y no va a funcionar ningún mensaje.

Cuando envía el mensaje, el plugin envía uno por uno, por eso, se ha creado una función que recibe una lista de string para enviar todos a la vez. Las funciones envían un comando es asíncrono, para evitar la intersección entre los segmentos de distintos mensajes se ha creado una variable writing para indicar si hay otro mensaje que está enviando o no. Como Javascript no es multihilo, por lo tanto, una variable normal es suficiente para controlar eso y no se necesita semáforo. Cuando la función detecta que writing es verdadero, se espera un muy pequeño tiempo aleatorio de unos 50 ms y se vuelve a comprobar otra vez.

4.6.4. Las funciones de enviar mensaje

En el fichero mesh-model hay las funciones de crear los mensajes de la red, pero había muchos parámetros que un usuario normalmente no sabe qué es, por ejemplo, ttl, seq, iv index, etc. En este fichero se crean otras funciones para cada comando, pero todos los campos que está relacionado con la red se rellena consultando a la base de datos.

El proceso de una función es cómo se explica en los siguientes puntos:

1. Comprobar que el móvil está conectado a un nodo proxy y ha inicializado correctamente.
2. Generar el mensaje del comando.
3. Crear una promesa.
4. Obtener el objeto en el caso si hay que actualizar la base de datos.
5. Crear un timer que pasando x tiempo lanza un error y borra la callback que está registrado.
6. Crear una función callback para cuando se recibe el mensaje, en esta función se realiza la actualización de la base de datos si es necesario, una vez acabado todo se resuelve la promesa que se ha creado anteriormente.
7. Registrar la callback al evento que toca, por ejemplo, es el comando es añadir la clave, entonces hay que esperar un código de AppKey Status.
8. Enviar el mensaje.
9. Espera la promesa.

Como la creación de la promesa y la espera no se puede realizar junto con la creación, hay que asegurar de poner todas las funciones que son posibles de lanzar error antes de la creación, si no, es posible no capturar el error de la promesa.

Para que un nodo puede usar el model de generic onoff hay que realizar dos pasos:

1. Añadir la clave de la aplicación a un nodo.

2. Enlazar la clave de la aplicación añadida al model.

Realizando estos dos pasos, el model ya puede entender todos los mensajes que son dirigidos para él, pero si se quiere que pertenezca a un grupo hay que añadirle la dirección del grupo a su lista de suscripción.

Para facilitar la configuración se crea una función donde el usuario puede especificar que para tal elemento se le añade tal grupo y se usa tal clave. Con esta función, el usuario puede asignar cada nodo a su propio grupo de una forma sencilla que solo es una acción.

4.7 Interacción entre la librería y el microcontrolador

En esta sección se explica cómo se han conseguido los requisitos usando la librería y el microcontrolador de Silicon Labs. La placa que se ha usado es una placa personalizada de la empresa.

Las condiciones que se aparecen en siguientes subsecciones, tal como el tiempo de espera, todos son las condiciones para esta placa determinada, si se usa otra placa u otro sdk es posible que tendría diferentes condiciones.

El desarrollo se ha realizado con Simplicity Studio que se muestra en la figura 4.13. La empresa ha proporcionado 2 placas de Silicon Labs que son EFR32BG13 y otras 2 placas del producto final para el desarrollo, que se puede ver en la figura 4.14. También ha proporcionado un programa base donde está la parte de comunicación del microcontrolador con otros componentes de la placa.

El objetivo principal de esta parte del trabajo es modificar el código de la librería y el programa base para conseguir cumplir los requisitos de la empresa para las nuevas versiones de sistema empotrado bajo BLE Mesh al tiempo que se preserva la estructura del sistema, ya que forma parte de una línea de desarrollo de producto propio de Dismuntel.

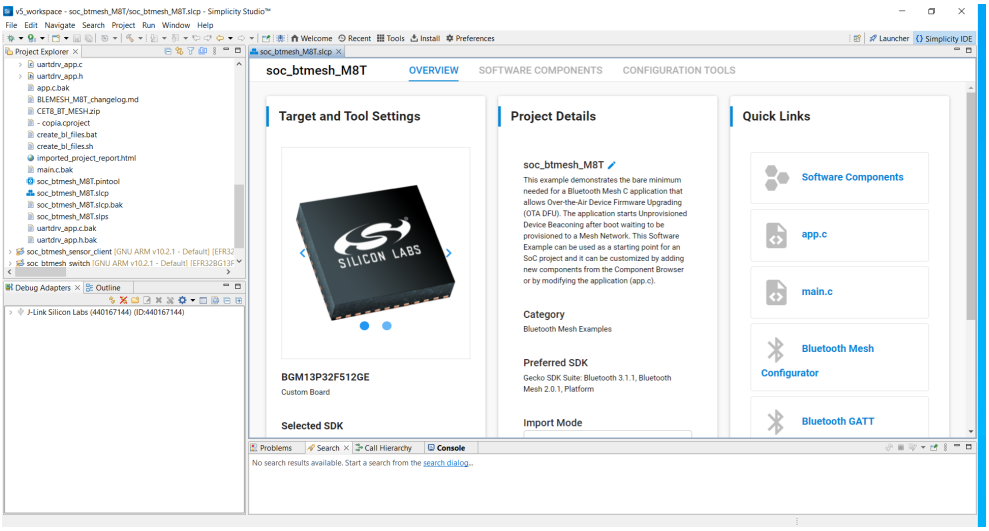

Figura 4.13: Simplicity Studio

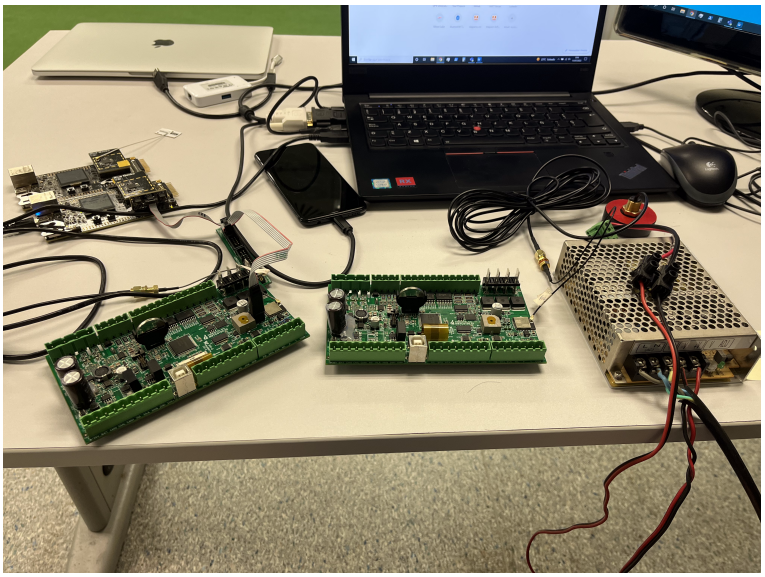

Figura 4.14: Las placas que se han usado en desarrollo

4.7.1. Aprovisionar el dispositivo

El proceso de aprovisionamiento está explicado en la sección 3.5, en este trabajo sólo se ha trabajado con el microcontrolador de Silicon Labs, los pasos para un dispositivo que no tiene OOB activado son:

1. Conectar al servicio de aprovisionamiento.

2. Enviar todos los mensajes necesarios para aprovisionarse.

3. Una vez enviado provisioning complete, hay que desconectar del dispositivo.

Después de desconectar el dispositivo hay que esperar 2-3 segundos, el dispositivo oculta el servicio de aprovisionamiento y expone el servicio de proxy (da igual si el nodo tiene activado el modo de proxy o no). El nodo permite a otro dispositivo conectar una vez al servicio proxy, si en esta conexión no se activa el modo proxy, una vez desconectado del nodo, también oculta el servicio de proxy.

Un requisito del trabajo es que todos los nodos deben ser idénticos, todos deben ser proxy y poder retransmitir los mensajes. Por lo tanto, la función de aprovisionar contiene 2 pasos, el primer paso es aprovisionar, y otro paso es la configuración del nodo.

En el paso de la configuración se activa el modo proxy y relay, también se realiza otras configuraciones que se explica luego. Durante el proceso de la configuración es posible pasar cualquier error, cuando se produce el error se intenta borrar el nodo para que el usuario pueda volver a aprovisionar desde 0, pero siempre existe la posibilidad de que se haya acabado el proceso de la configuración por la mitad, y el móvil no puede ni enviar el mensaje de borrar el nodo, para este caso el dispositivo físico debe ofrecer alguna forma de resetear el nodo.

4.7.2. Bajo consumo

Los nodos de BLE Mesh que tiene el modo relay y proxy activado suelen ser nodos que consumen mucha energía porque necesitan que estén continuamente escuchando los mensajes.

En este trabajo, cuando se requiere el bajo de consumo, no requiere que el nodo pueda contestar en seguida, pues la forma de conseguir el bajo consumo es activar el modo lpn sin tener el friend.

El modo de lpn se puede inicializar por parte del nodo, cuando está activado este modo deja de escuchar el mensaje y puede bajar el consumo desde 11 mAm a 4 mAm. Cuando se desactivan otras funcionalidades, al final es posible conseguir 1 mAm.

En el nodo se le añade un elemento para determinar si hay que entrar al modo de lpn, cuando el móvil conecta con un nodo entonces empieza a publicar un generic on al grupo "PhonePresent" dónde están estos elementos. Cuando este elemento está en el estado off, se duerme un periodo largo, y se despierta un periodo corto para ver si le llega algún mensaje de activar el elemento, si no le llega, pues vuelve al periodo largo de dormir. Otra forma de activar este elemento es cuando detecta que hay un móvil que está conectado al nodo a través de BLE, entonces tampoco entra al modo de dormir.

La razón de publicar el mensaje con el móvil y no los nodos es para no malgastar la lista de protección replay attack. Replay attack es un tipo de ataque donde el malicioso captura el mensaje correcto, por ejemplo, un mensaje de activar un elemento, y lo publica en otro tiempo cuando puede realizar un ataque. Cada nodo de BLE Mesh tiene una lista para guardar el número de secuencia de cada dirección fuente, si recibe un mensaje con el número de secuencia menor o igual que el valor guardado se ignora el mensaje, porque puede ser un mensaje que se ha publicado por un tercero.

Como el nodo suele ser un dispositivo que tiene poca memoria, por lo tanto, no va a poder guardar todas las direcciones, los nodos de Silicon Labs por defecto el tamaño de la lista es 32, esto significa que sólo puede guardar el número de secuencia de 32 elementos diferentes. Cuando esta lista se llena, el nodo ignora cualquier mensaje de un elemento nuevo, si deja que los nodos haga una publicación a sus alrededor para despertar los nodos, esta lista se llena enseguida, también se limita mucho el dispositivo de una red, por esta razón, se pública por el móvil, y el intervalo tiene que ser menor o igual que el periodo corto de despierta para que cuando se despierta segura puede capturar este mensaje.

En la librería se ha decidido publicar el mensaje cada 3 segundos, esto significa que el número de secuencia del móvil se incrementa cada 3 segundos. Cuando actualiza el valor de iv index, se ha decidido dejar una margen de 10 días, el tiempo se ha calculado por esta publicación periódica. El móvil puede dejar conectado a un nodo y sólo por enviar esa publicación, el número de secuencia llega al límite después de 10 días.

El nodo cuando no recibe el mensaje activa el elemento de "PhonePresent" y no hay un móvil conectado se vuelve al periodo de dormir. De esta forma se consigue el bajo consumo y no requiere ninguna acción por parte del cliente. La configuración de este elemento se realiza durante el proceso de la configuración de aprovisionamiento, usando la clave de la aplicación 0.

4.7.3. Elementos del nodo

El dispositivo de este trabajo es un controlador, por lo tanto, para cada componente del dispositivo se crea un elemento, elemento de motor, elemento de luminaria, etc. La agrupación se deja a los usuarios decidir, porque puede ser que quieran agrupar por funcionalidades, activar todos los motores, o agrupar por localización, activar todos los elementos de una determinada zona, etc. Como el usuario puede crear su propio grupo y borrarlo, tampoco se puede saber a qué grupo hay que añadir a cada elemento.

Aunque no hay que configurar la lista de suscripción de los elementos durante el paso de la configuración de aprovisionamiento si hay que configurar la dirección de la publicación de los elementos.

Hay un requisito que es cuando se produce un cambio de estado, hay que informar a los móviles a través de la red BLE Mesh. Para conseguir eso se ha creado un grupo "Mobiles", durante el proceso de la configuración se añade la dirección de publicación a todos los nodos con la clave de la aplicación 0, y cuando se produce el cambio del estado, el nodo llama la función de publicar.

Cuando un móvil se conecta a un nodo, también le envía un mensaje de proxy filter para que pase todos los mensajes del grupo "Mobiles", así si recibe cualquier mensaje, el atributo genericNotifier del BleDevice lanzará un evento para notificar a la aplicación móvil.

CAPÍTULO 5

Pruebas

La prueba de este trabajo consiste en dos partes, la primera parte es probar que el mesh model puede generar los mensajes correctos, y la segunda parte es probar que la librería final cumple los requisitos.

Para la primera parte se ha cogido los ejemplos que están en la especificación Mesh Profile, cada vez que se crea una función de generar pdu de una determinada capa se comprueba que coincide con la salida que está en el ejemplo.

Todas estas pruebas están en un fichero, en la figura 5.1 se muestra la ejecución de las pruebas. Estas pruebas no garantizan 100 % el funcionamiento de las funciones, ya que para algunos mensajes no existen ejemplos, por ejemplo, pdu de control segmentado, la mayoría de los mensajes del model. Estos mensajes se comprueban enviando el mensaje al microcontrolador de Silicon Labs, si el microcontrolador contesta con una respuesta correspondiente entonces se asume que está correcto.

La segunda parte está más relacionada con el microcontrolador, para cada funcionalidad se realizan distintas pruebas y se prueba en Android e iOS para asegurar que funciona en las dos plataformas. En la función de aprovisionamiento hay que comprobar los siguientes puntos:

- Comprobar que se aprovisiona correctamente y tiene todas las configuraciones correctas.
- Comprobar que se aprovisiona varios dispositivos y se le asigna las direcciones correctas.

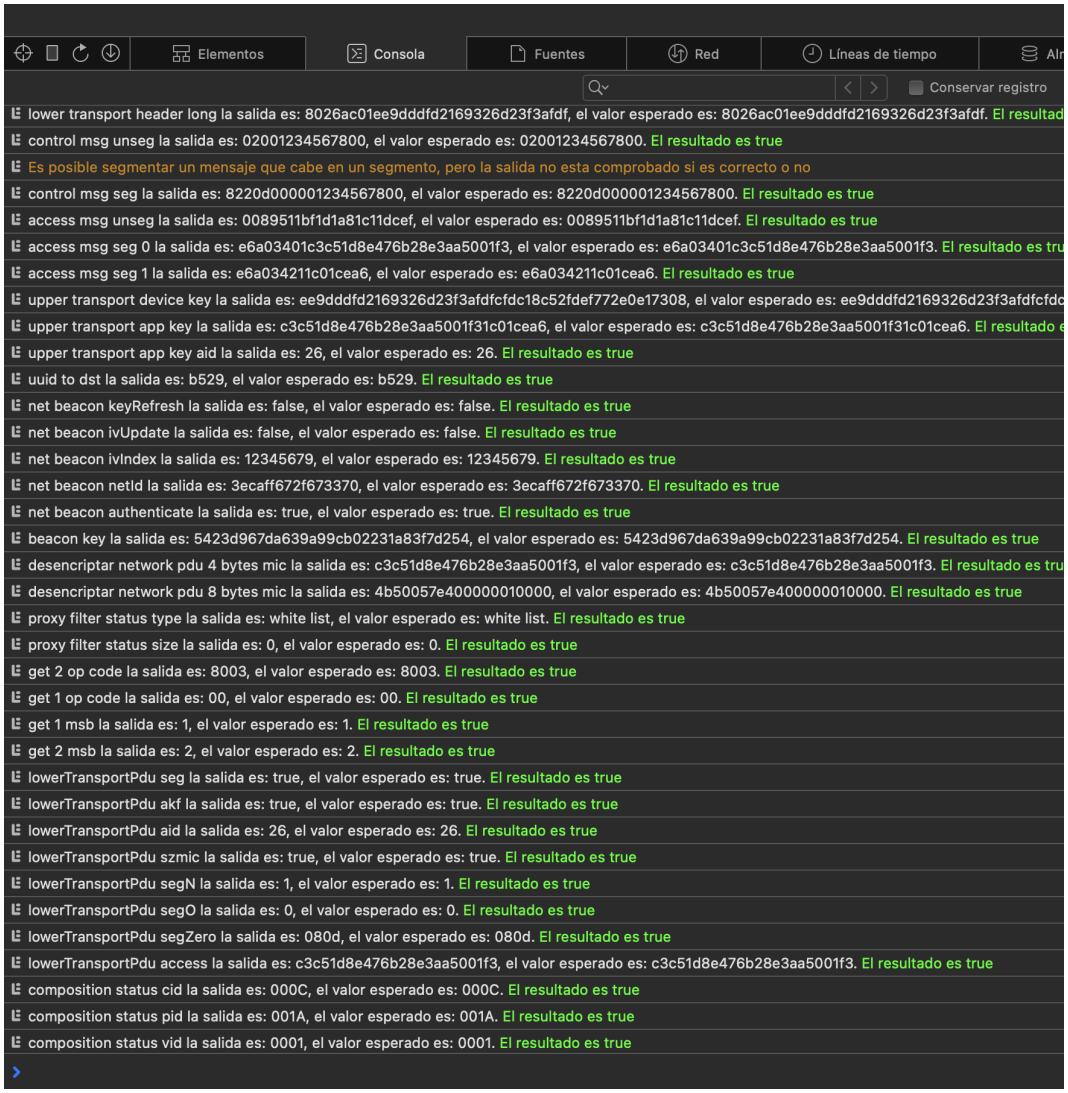

Figura 5.1: El resultado de las pruebas

- Comprobar si durante el proceso de la configuración se produce algún error se borra el nodo o no.

Sobre el mantenimiento de la red se comprueba los siguientes puntos:

- Comprobar cuando conecta a un nodo proxy el móvil publica generic on al grupo “PhonePresent” cada 3 segundos, y los nodos se despiertan.
- Comprobar la actualización de iv index de todos los casos, actualización normal, el móvil tiene un valor 2 mayores que el dispositivo, el móvil tiene un valor 2 menores que el dispositivo.
- Comprobar que tiene el servicio proxy está expuesto.

Sobre el funcionamiento normal de la mesh se comprueba:

- Comprueba que puede conectarse a un node de la red.
- Comprueba que los comandos funcionan y actualiza la base de datos correctamente.
- Comprueba que los nodos si se retransmiten los mensajes.
- Comprueba que cuando se produce un cambio de estado, el nodo informa al móvil.

Sobre el manager se comprueba:

- Comprueba que funciona la exportación e importación de la base de datos.
- Comprueba el correcto funcionamiento de la clase EmptyList.
- Comprueba que la transacción de la base de datos funciona como debe.
- Comprueba la creación de las claves, los grupos, los dispositivos.
- Comprueba que se puede cambiar el dispositivo virtual.

Todas estas pruebas no garantizan que la librería funciona correctamente, pero han sido una primera aproximación a la validación del desarrollo. Con estas pruebas comprobamos que la librería ya cumple todos los requisitos que se han analizado en la sección 4.1.

La librería se ha probado en un proyecto helloworld de Cordova. En la figura 5.2 se muestra un ejemplo de la aplicación que se ha creado para la prueba, donde lo que se ha hecho es conectar al nodo y activar el panel principal del nodo como se puede ver en la figura 5.3. En la salida se puede ver que hay dos mensajes del nodo, el primero es la respuesta de cuando se ha enviado el comando. Como el panel no se puede activar inmediatamente, se contesta que aún está en off. El segundo mensaje que ha llegado es una notificación pasiva del nodo cuando realmente tiene activado el panel principal.

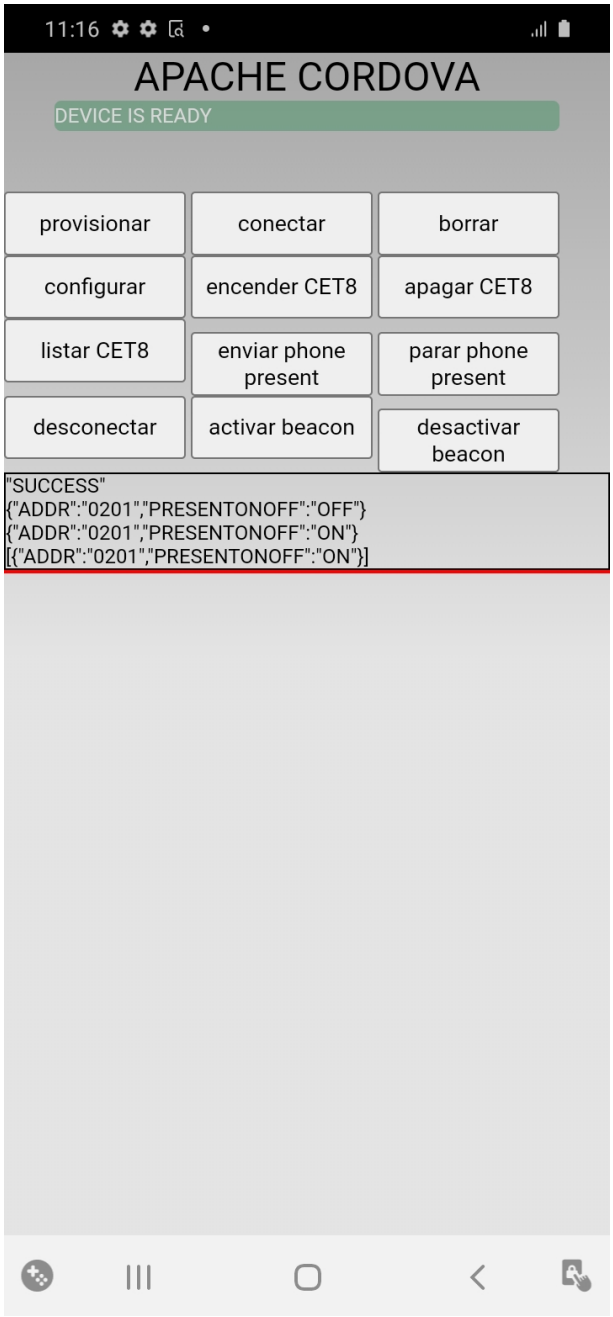

Figura 5.2: La aplicación de prueba

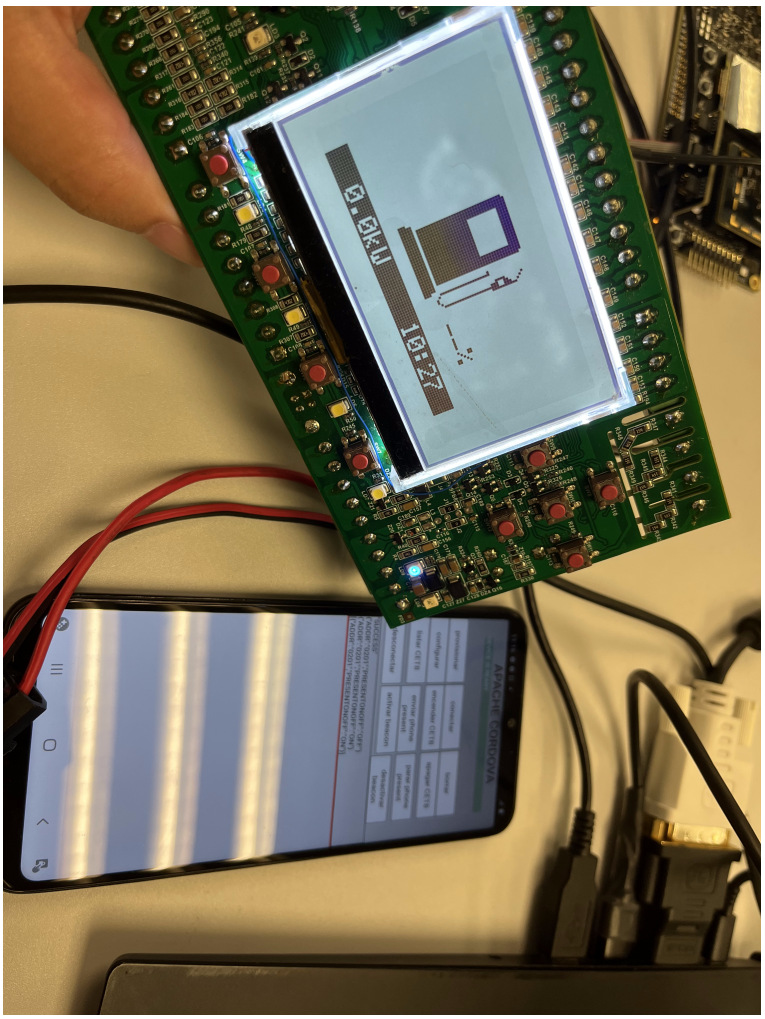

Figura 5.3: Activar el panel principal

CAPÍTULO 6

Conclusiones

Trás desarrollar el trabajo hay que volver a analizar el objetivo que se ha definido en el capítulo 1 para ver si se ha conseguido o no. Se han definido 3 objetivos pues se analiza los 3 por separados:

1. Desarrollo de una librería para aplicaciones móvile que sea genérica. La librería que se ha desarrollado, bajo la tecnología de red mesh model se ha descrito en la sección 4.4. Crear una librería que sólo gestiona las capas de BLE Mesh facilita mucho la flexibilidad de la gestión de la red, porque aunque Nordic sí que ha publicado sus códigos fuentes en un repositorio público, la distribución es de una aplicación entera y no es posible extractar sólo una parte para adaptarse a este trabajo. Otra ventaja de la librería desarrollada es que la librería es posible ejecutarla en un navegador, es decir, con el mismo código, funciona en PC, Android, iOS u otra plataforma que soporta un navegador actualizado. La librería de Nordic está hecha en Java y Swift, esto significa que para cada plataforma hay que desarrollar otra capa adicional, ya que aunque Java si que es un lenguaje multiplataforma, no es tan flexible como Javascript.

2. El segundo objetivo es desarrollar manager y ble module (para sistemas empotrados) de forma que sean fácil de usar y modificar. Este objetivo hay que analizarlo por dos partes, la primera parte es ver si realmente es fácil de usar o no. Para la gestión de una red BLE Mesh, se necesitan guardar las claves, actualizar iv index, gestionar las direcciones, gestionar label uuid del grupo, pero la librería ya hace todas estas cosas. Los grupos se identifica por

nombre y no label uuid, el usuario no va a notar la existencia de las claves del dispositivo, ya que la librería decide sola, el usuario tampoco hace falta preocupar por la actualización de iv index, el proceso se inicializa sólo y se acaba antes de que el número llegue al límite. Por lo tanto, por la parte de facilitar el uso sí que se ha conseguido Por la parte fácil de modificar, se ha separado la base de datos de la gestión como se ha definido en la sección 4.5, es muy fácil cambiar cualquier otra base de datos. Pero ble module no es tan fácil de modificar, pero cuando se quiere cambiar el plugin de la comunicación es posible que cambie también la forma de escanear, la forma de envío y recepción, por lo tanto, este cambio es un poco inevitable.

3. El tercer objetivo de validación se ha conseguido, con el desarrollo del caso de aplicación cumpliendo con los requerimientos de bajo consumo y comunicación con la librería, que se han descrito en la sección 4.6.

En conclusión, se ha conseguido el propósito general del TFM al tiempo que los objetivos de desarrollo por parte de empresa Dismuntel. Gracias a este proyecto se ha aprendido el funcionamiento de BLE Mesh y también se ha aprendido cómo desarrollar una librería mirando la especificación, labor esta compleja y laboriosa por cuanto implica la lectura y compresión de una gran cantidad de documentación técnica.

Bibliografía

[1] L. F. Fernández, "Bluetooth mesh networking. aplicaciones y pruebas de concepto," Master's thesis, Universidad Complutense de Madrid, Sep. 2020.

[2] "Products embracing the next era of bluetooth technology," May 2022. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/recent-enhancements/mesh/mesh-qualified/

[3] M. W. Group, "Mesh profile 1.0.1," Jan 2019. [Online]. Available: https://www.bluetooth.com/specifications/specs/mesh-profile-1-0-1/

[4] ——, "Mesh model 1.0.1," Jan 2019. [Online]. Available: https://www.bluetooth.com/specifications/specs/mesh-model-1-0-1/

[5] M. Woolley, "An introduction to the bluetooth mesh proxy function," Nov 2020. [Online]. Available: https://www.bluetooth.com/bluetooth-resources/bluetooth-mesh-proxy-kit/

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

ETS Enginyeria Informàtica
Camí de Vera, s/n, 46022, València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es

APÉNDICE A

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**			X	
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**		X		

ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**			X	
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Este trabajo se alinea al objetivo de desarrollo sostenible 4,8,9 y 11. En los siguientes puntos se explica las relaciones del trabajo con los ods:

- Educación de calidad. Este trabajo se ha podido conseguir gracias al ejemplo que existe en la página oficial de Bluetooth. Una de las finalidades de este trabajo es servir como una tutoría más de Bluetooth Mesh, ya que la especificación es muy larga. En el capítulo 3 también se ha mencionado, que en el ejemplo de la especificación hay un error que es posible que confunda a los lectores. Aunque el trabajo no va a ayudar a la sociedad en general, pero sí que puede ayudar a otras personas que tienen un conocimiento básico sobre informática a aprender Bluetooth Mesh.

- Trabajo decente y crecimiento económico. Bluetooth Mesh es una tecnología nueva, y Bluetooth es un mercado muy grande. En esta área es posible haber muchas tecnologías nuevas, pero para desarrollar un proyecto sobre Bluetooth Mesh es difícil, ya que requiere conocimientos sobre las redes para que se pueda generar los mensajes de distintas capas. En este trabajo se

ha diseñado e implementado una librería general que realiza todos estos pasos. Esto ayuda a otros desarrolladores para que sólo se preocupen de la utilización de Bluetooth Mesh y no implementar desde el principio las capas. Bluetooth Mesh también tiene otra dificultad para los desarrolladores que es la gestión de la red, en este trabajo también se ha presentado una forma de gestión de la red, aunque no está aprovechando toda la seguridad de Bluetooth Mesh, pero es una forma sencilla de usar para los clientes finales.

- Industria, innovación e infraestructuras. Bluetooth Mesh es una tecnología nueva, incluso la especificación es posible que cambie. En este trabajo se ha usado el modo lpn sin un nodo friend para reducir el consumo, esta forma de usar el modo lpn no es una forma que ha diseñado la especificación. En este trabajo se ha usado una nueva tecnología y se ha innovado una forma nueva de usarla en la industria. Este trabajo puede facilitar a las empresas a desarrollar Bluetooth Mesh sobre sus propios proyectos, también es posible modificar algunas partes para conseguir sus propios objetivos.

- Ciudades y comunidades sostenibles. El trabajo no tiene una influencia directa en mejorar las ciudades y comunidades sostenibles, pero la tecnología que se ha trabajado es muy útil para este objetivo sostenible. Bluetooth Mesh es una tecnología de comunicación segura y escalable, es una tecnología diseñada para IoT. Cuando Bluetooth Mesh esté madurada y sea fácil de configurar y usar, las empresas podrán usar esta tecnología para desarrollar sus propios dispositivos IoT que sí que mejoren la ciudad.

El trabajo no tiene muchas relaciones con otros objetivos que no estén mencionados anteriormente, pero tampoco está en contra de otros objetivos. Al final del trabajo que se ha desarrollado se ha pensado para bajar el consumo, la tecnología usa una banda libre en casi todo el mundo, por lo tanto, tampoco afecta ni al clima ni a los animales.

ETS Enginyeria Informàtica
Camí de Vera, s/n, 46022, València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es