

Descripción del protocolo BLE

Dispositivos y Sistemas Inalámbricos

Francisco M. Delicado Martínez^{*}

Diego Hortelano Haro^{**}

Dpto. Sistemas Informáticos

Escuela Superior de Ingeniería Informática de Albacete

Universidad de Castilla-La Mancha

16 de enero de 2017

Índice

1. Definición y objetivos	3
2. Evolución	4
3. Tipos de dispositivos	5
4. Detalles técnicos	5
5. Aplicaciones	6
6. Pila de protocolos	8
6.1. Capa física	9
6.2. Capa de enlace	11
6.2.1. Dirección del dispositivo	13

^{*}francisco.delicado@uclm.es

^{**}Gran parte del contenido de este documento ha sido extraído del TFG titulado “*Estudio del estándar Bluetooth Low Energy para redes híbridas de sensores en Internet de las Cosas.*”, realizado por Diego Hortelano Haro, y dirigido por Teresa Olivares Montes, y María del Carmen Ruiz Delgado

6.2.2.	Estados sin conexión	14
6.2.3.	Estado de conexión	16
6.2.4.	Formato de paquetes de capa de enlace	19
6.2.5.	Paquete de tipo Data	25
6.3.	Interfaz Host-Controlador, HCI.	28
6.3.1.	Ordenación de los bits.	29
6.3.2.	Paquete HCI para Comandos.	30
6.3.3.	Paquete HCI de Eventos.	31
6.3.4.	Paquete HCI de Datos ACL.	32
6.3.5.	Interacciones entre HCI y Capa de Enlace	34
6.4.	Protocolo de control y adaptación del enlace lógico, L2CAP .	37
6.5.	Protocolo de atributo, ATT	40
6.5.1.	Operaciones del ATT	41
6.5.2.	Formato paquete ATT	43
6.6.	Administrador de seguridad, SM	44
6.6.1.	Procedimientos de seguridad	44
6.6.2.	Algoritmos de <i>pairing</i>	45
6.7.	Perfil de atributo genérico, GATT	47
6.7.1.	Roles	47
6.7.2.	Atributos	48
6.7.3.	Jerarquía de datos en GATT	50
6.7.4.	Descubrimiento de servicios y características	54
6.7.5.	Lectura de Valores de Características	57
6.7.6.	Escritura de Valores de Características	59
6.8.	Perfil de acceso genérico, GAP	60
6.8.1.	Roles	60
6.8.2.	Modos y procedimientos	61
6.8.3.	Modos de seguridad	63
6.8.4.	Procedimientos de seguridad	64
6.8.5.	Dirección del dispositivo aleatoria	65
6.8.6.	Servicio GAP	66

1. Definición y objetivos

Bluetooth Low Energy, también conocido por sus siglas BLE o como Bluetooth Smart, comenzó formando parte de la especificación del núcleo de la versión 4.0 del estándar Bluetooth y su diseño presenta objetivos diferentes a los del Bluetooth clásico.

Tal y como encontramos en [1], el inicio de BLE se encuentra en Wibree, tecnología de muy bajo consumo para la comunicación inalámbrica diseñada por Nokia [2] en 2006 para sustituir a Bluetooth en este tipo de aplicaciones de bajo consumo. A pesar de esta idea inicial, la tecnología Wibree fue incorporada al estándar principal Bluetooth en 2010 por el Bluetooth Special Interest Group [3] con la adopción del Bluetooth Core Specification Version 4.0 [4].

BLE puede definirse como una versión más inteligente y de bajo consumo de Bluetooth [5], diseñado para complementar a este último y tener un consumo tan bajo como fuera posible. Por otro lado, aunque BLE opera en la misma banda ISM y toma prestada gran parte de la tecnología de su antecesor, debe considerarse como una tecnología diferente, pues persigue diferentes objetivos y está orientada a diferentes segmentos de mercado [6].

Por ello, BLE ha tomado un camino completamente diferente al de Bluetooth, optimizándose para un consumo mínimo, en lugar de centrarse en aumentar la velocidad de transferencia. De esta manera, aunque BLE renuncia a transferencias de datos de tamaño elevado, consigue la posibilidad de mantener una conexión durante varios días. Este mínimo consumo hace que su uso sea idóneo en dispositivos alimentados por pequeñas baterías y que aún no disponen de una tecnología inalámbrica apropiada, como microcontroladores o sensores.

De esta manera queda claro que el principal objetivo de BLE ha sido crear la tecnología inalámbrica de corto alcance con el menor consumo posible, pero además no se han descuidado los objetivos establecidos por Bluetooth:

- Permitir su uso a nivel mundial, utilizando para ello la frecuencia de banda de 2.4 GHz, disponible sin coste de licencia en todo el mundo.
- Bajo coste, objetivo estrechamente relacionado con el del bajo consumo. Es posible satisfacer ambos utilizando una menor cantidad de memoria y una menor velocidad de procesamiento; renunciando a ciertas cosas, por supuesto, como a la posibilidad de crear estructuras de red más complicadas, como una malla completa.

- Robustez, al igual que el Bluetooth clásico, BLE hace uso de la tecnología AFH (Adaptive Frequency Hopping) de Bluetooth [7], la cual ayuda a detectar interferencias rápidamente y a evitarlas en el futuro, lo que es muy útil en un espectro de radio tan utilizado como es la banda de frecuencia ISM. El funcionamiento de esta tecnología se detallará en la sección correspondiente a la capa física de la pila de protocolos (sección 6.1).
- Corto alcance, objetivo relativamente sencillo de cumplir en BLE, ya que al ser un sistema de bajo consumo, debe emitir con poca potencia, así como mantener una alta sensibilidad en el receptor para reducir la potencia necesaria para recoger las señales. En cualquier caso, un rango de corto alcance no significa que los dispositivos deban estar excesivamente cerca, BLE está diseñado para ser una red de área personal y ese rango es el que se desea alcanzar (alrededor de 100 metros, suficiente para aplicaciones de sensores domésticas).
- Bajo consumo, objetivo que, aunque ya se encontraba en la versión clásica, BLE ha potenciado, reduciendo el consumo uno o dos órdenes de magnitud.

2. Evolución

A pesar de que BLE es una buena tecnología, ha tenido un tiempo de adopción increíblemente rápido, siendo el número de dispositivos que ya incluyen esta tecnología muy superior al de otras tecnologías inalámbricas en relación al tiempo transcurrido desde su lanzamiento.

Esta rápida adopción tiene una sencilla explicación, y es que se trata de una tecnología ligada a los smart phones, tablets e informática móvil, cuyo uso ha crecido enormemente en un corto período de tiempo. A este hecho debemos añadir la rápida adopción de BLE por las grandes marcas de la industria de la telefonía, como Apple o Samsung, lo que ha dado un gran impulso a esta tecnología.

Así pues, el bajo coste de esta tecnología (es posible adquirir un microcontrolador con BLE a un precio inferior a 4 euros) y la necesidad de conectarse con el exterior de los nuevos dispositivos inteligentes mencionados anteriormente, hacen que éste sea un campo con un alto potencial de crecimiento, ofreciendo una gran oportunidad para proporcionar soluciones innovadoras a problemas comunes.

Otro de los factores clave del éxito de BLE, aunque éste sea menos visible, es el objetivo de su diseño: ha sido creado como un framework extensi-

ble para el intercambio de datos, permitiendo fácilmente el intercambio de información entre diferentes dispositivos sin necesidad de conocer a fondo la tecnología subyacente, a diferencia de Bluetooth clásico que se centra en un conjunto limitado de casos de uso. Los grandes proveedores de smart phones son conscientes de esta ventaja, y han proporcionado APIs flexibles y de bajo nivel para dar libertad a los desarrolladores móviles a la hora de utilizar BLE.

Para concluir este apartado, podemos añadir a las ventajas de BLE ya mencionadas algunas otras: un sencillo modelo de datos; el hecho de que no haya unos costes de licencia intrusivos ni unas cuotas de acceso a las especificaciones básicas; o la existencia de una pila de protocolos generales específicos. Todas ellas han ayudado a que este estándar crezca rápidamente, además de proporcionarle un potencial muy importante.

3. Tipos de dispositivos

La nueva especificación del estándar Bluetooth con la inclusión de BLE ha hecho que los nuevos dispositivos no sean compatibles con los dispositivos Bluetooth de versiones anteriores, lo que ha hecho necesaria la aparición de dos tipos diferentes de dispositivos compatibles con la nueva versión:

- Dispositivos Bluetooth Smart, que utilizan BLE, y solamente pueden comunicarse con dispositivos que también lo utilicen (lo que imposibilita las conexiones con dispositivos de versiones anteriores).
- Dispositivos Bluetooth Smart Ready, los cuales utilizan BLE y Bluetooth clásico, por lo que pueden comunicarse con ambos tipos de dispositivos.




La Tabla 1 recoge los diferentes tipos de Bluetooth existentes en la actualidad, mostrándonos la compatibilidad entre ellos.

4. Detalles técnicos

La información técnica de BLE, tal y como puede verse en la página del Bluetooth SIG [9], es la mostrada a continuación:

- Transferencias de datos en paquetes muy pequeños (de 8 a 27 octetos), realizadas con una velocidad de 1 Mbps.

Tabla 1: Compatibilidad entre versiones Bluetooth [8].

Marca	Versión	Logo	Compatible con
Bluetooth Smart Ready (Dual Mode)	v4.0		Bluetooth Smart Ready Bluetooth Smart Bluetooth Clásico
Bluetooth Smart (Single Mode)	v4.0		Bluetooth Smart Ready Bluetooth Smart
Bluetooth (Clásico)	v2.1 + EDR v3.0		Bluetooth Smart Ready Bluetooth Clásico

- Al igual que Bluetooth clásico, BLE hace uso de AFH (adaptative frequency hopping) para minimizar las interferencias de otras tecnologías que utilicen la banda ISM de 2.4 GHz, lo que mejora los enlaces y el rango de los mismos.
- Bluetooth Smart permite manejar el host desde el controlador, lo que nos da la oportunidad de suspender el primero durante largos periodos de tiempo y volver a reactivarlo cuando vuelve a ser necesario que realice alguna acción, permitiendo un ahorro de energía muy significativo.
- Latencia muy baja, permitiéndonos establecer una conexión y transferir datos en un tiempo inferior a 3 ms, lo que permite unas transmisiones de datos de manera autenticada en milésimas de segundo.
- BLE cuenta con un rango de más de 100 metros, debido al aumento del índice de modulación.
- Para asegurar una mayor robustez contra las interferencias, BLE hace uso de un CRC de 24 bits en todos los paquetes.
- En el tema de la seguridad, BLE utiliza una encriptación AES-128 para autenticar y proteger los paquetes.
- Por último, en lo referente a las topologías de red, BLE utiliza 32 bits para direccionar dispositivos, lo que permite la conexión de una enorme cantidad de dispositivos siguiendo una topología de estrella.

5. Aplicaciones

A grandes rasgos, el principal uso para el que fue concebido BLE es el mismo que el que propició el desarrollo de Bluetooth: comunicar diferentes dispositivos de manera inalámbrica. Pero, como ya hemos dicho

anteriormente, existe una importante diferencia entre ambos, y es que al disponer de un consumo mucho más reducido, es posible utilizar BLE en multitud de nuevos dispositivos que anteriormente no podían hacer uso de esta tecnología, como sensores o relojes, o cualquier otro dispositivo que deba mantenerse con una pequeña batería durante un largo período de tiempo.

De esta manera, y aunque BLE es una tecnología con una vida relativamente corta, ya encontramos diversas aplicaciones o ideas para posibles usos. Uno de los más interesantes es, posiblemente, el utilizarlo para unir las redes inalámbricas de sensores a Internet, entrando en lo que conocemos como Internet de las Cosas (Internet of Things o IoT).

Así pues, la conexión entre sensores y dispositivos de uso diario para los usuarios (como smart phones, tablets, PCs o portátiles, por ejemplo) nos permitirá realizar acciones como:

- Detectar la proximidad de una persona a un lugar en particular (su casa, su coche, su oficina, una tienda determinada...) y actuar en consecuencia.
- Detectar la presencia de alguien, encendiendo o apagando las luces de una habitación, encendiendo la alarma cuando los usuarios salen de casa, etc.
- Utilizar objetos cotidianos para recoger datos del usuario de manera no intrusiva, por ejemplo, tomar la temperatura corporal de alguien utilizando el mando a distancia de la televisión.
- Conocer los datos de multitud de dispositivos de medida de diversos ámbitos, como por ejemplo deportivo o medicinal, de una manera rápida y sencilla, para procesarlos y obtener una información útil para el usuario. Esta información puede ser accesible desde el propio teléfono móvil.

Además de estas posibles aplicaciones relacionadas con los sensores, existe la posibilidad de utilizar BLE para otros fines, como:

- Controlar multitud de electrodomésticos y aparatos desde largas distancias.
- Situar a un usuario dentro de un edificio, localizando su posición de una manera mucho más precisa que la tecnología GPS (en distancias cortas, dentro de una red de dispositivos BLE).

- La posibilidad de situar a un usuario en un edificio nos permite desarrollar un gran abanico de aplicaciones que se comuniquen con él a través de dispositivos como smart phones o tablets, entre otros. En este contexto aparecen los dispositivos BLE conocidos como *beacons* o balizas (donde destaca el *ibeacon* [10] de Apple). Estos dispositivos nos permiten enviar a los usuarios información útil en función de su localización dentro de un edificio en particular: restaurantes, tiendas o centros comerciales... Como un ejemplo concreto podemos citar a McDonald's [11], que instaló *beacons* que permiten a sus clientes beneficiarse de una serie de ofertas a cambio de permitir a la empresa conocer las áreas por las que se mueven.
- Otro uso diferente de esta tecnología es permitir a los clientes realizar pagos de manera más cómoda, existiendo ya algunos proyectos en este campo.

6. Pila de protocolos

Tal y como se describe en el primer volumen de la especificación 4.0 del estándar Bluetooth [12] y en [1], BLE añade una nueva pila de protocolo a Bluetooth. En ella podemos diferenciar tres partes: controlador, host y aplicación:

- La aplicación es la capa superior, y contiene la lógica y la interfaz de usuario. Además, es la encargada de controlar los datos relacionados con la aplicación que se encuentra en ejecución en un momento determinado.
- El host incluye las siguientes capas del protocolo:
 - Perfil de Acceso Genérico (Generic Access Profile, GAP).
 - Perfil de Atributo Genérico (Generic Attribute Profile, GATT).
 - Protocolo Administrador de Seguridad (Security Manager Protocol, SMP).
 - Protocolo de Atributo (Attribute Protocol, ATT).
 - Protocolo de Adaptación y Control de Enlace Lógico (Logical Link Control and Adaptation Protocol, L2CAP).
 - Interfaz Host-Controlador (Host Controller Interface, HCI).
- El controlador engloba las siguientes capas del protocolo BLE:

- Interfaz Host-Controlador (Host Controller Interface, HCI).
- Capa de Enlace (Link Layer, LL).
- Capa Física (Physical Layer, PHY).

La figura 1 muestra esta clasificación de manera gráfica, donde los diferentes protocolos aparecen reflejados en el interior de la parte a la que pertenecen. En las siguientes secciones se describen las diferentes capas de una manera más detallada.

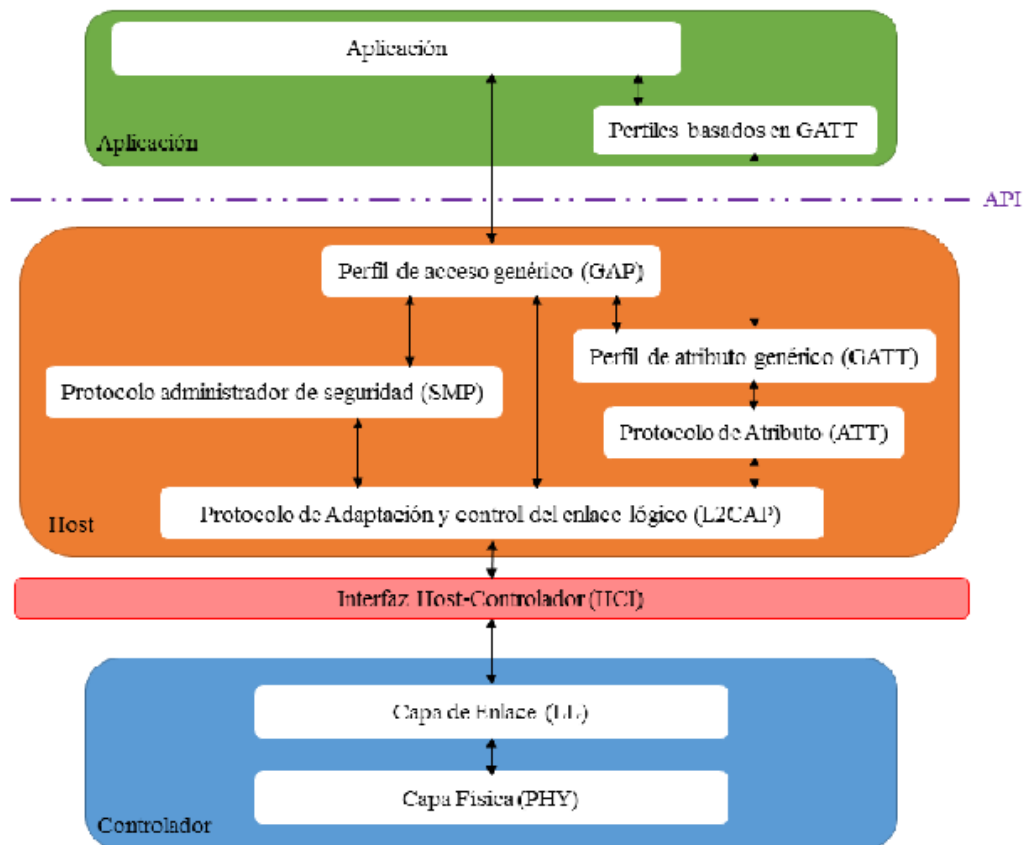


Figura 1: Pila de protocolos BLE, dividida en sus tres partes: controlador, host y aplicación.

6.1. Capa física

La capa física es la encargada de modular y demodular las señales analógicas enviadas y recibidas, transformando las señales digitales a analógicas y viceversa. En [13] encontramos la especificación oficial de esta capa.

Para la comunicación, BLE utiliza la banda de 2.4 GHz ISM (Industrial, Scientific and Medical), dividiéndola en 40 canales de 2 MHz, que van desde 2.4000 GHz hasta 2.4835 GHz. De todos estos canales, los tres últimos (37, 38 y 39) se utilizan para enviar mensajes de *broadcast* y *advertisements* para establecer conexiones, dejando el resto de canales para la transmisión de datos una vez se haya establecido la conexión. Esta división se muestra en la Figura 2.

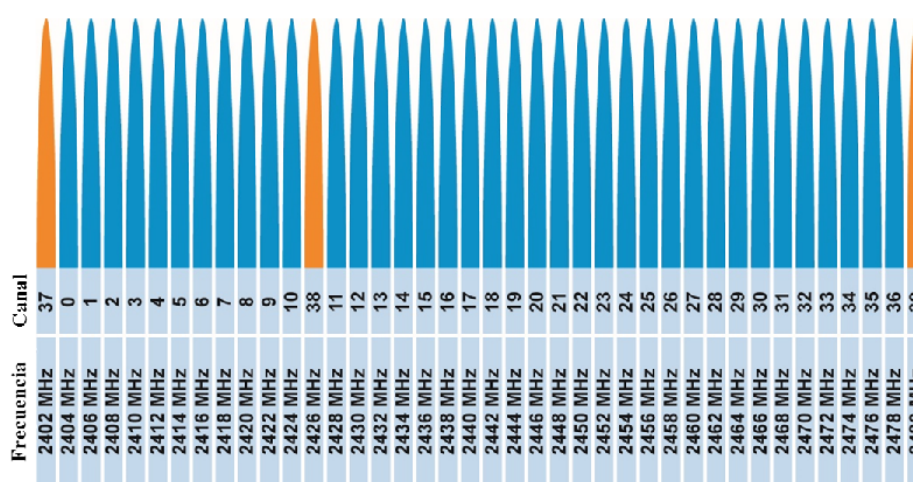


Figura 2: División de los canales BLE en Advertisement y Data [14].

Esta banda de frecuencia tiene importantes ventajas, entre las que destaca el hecho de que se trate de una banda gratuita y sin requisitos de licencia en todo el mundo, lo que permite abaratar los dispositivos que utilizan BLE. Sin embargo, esto mismo hace que no sea la única tecnología que hace uso de la banda ISM, y en ella podemos encontrar estándares como Wi-Fi, ZigBee o Bluetooth tradicional. Por esta razón, resulta muy sencillo que se produzcan interferencias entre las diferentes tecnologías. En la figura 3 se muestra la sobreposición de los canales de varios de estos estándares.

Para tratar de evitar las posibles interferencias, BLE, al igual que Bluetooth clásico, hace uso de la tecnología AFH (*Adaptive Frequency Hopping*) para reducir las interferencias que el resto de tecnologías de radio inalámbricas que utilizan la misma banda de frecuencia (Wi-Fi, Bluetooth clásico o ZigBee, por ejemplo) puedan producir (ver Figura 4b). Mediante esta técnica, el canal por el que se envían datos cambia, evitando posibles colisiones si alguna tecnología está haciendo uso de un canal en la misma frecuencia. Además, a diferencia de RFH (*Random Frequency Hopping*, ver Figura 4a), este cambio de canal no se realiza de manera aleatoria, sino que

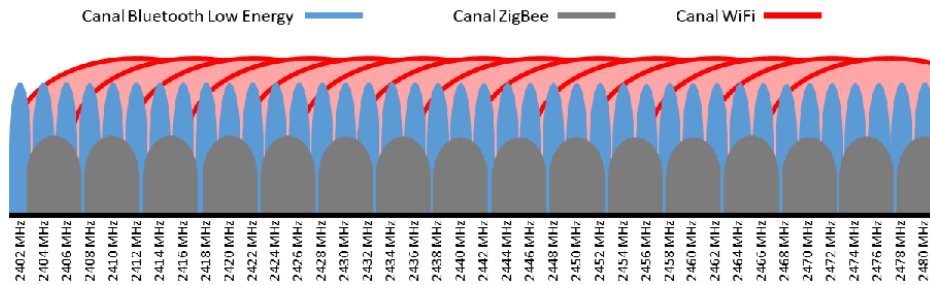
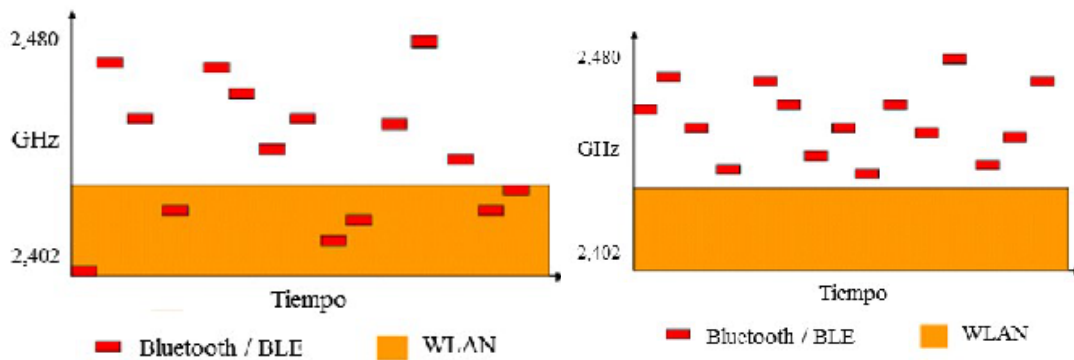


Figura 3: Canales Wi-Fi, ZigBee y BLE en la banda de frecuencia ISM.

se tiene en cuenta los canales de frecuencia que ya se encuentran ocupados (Figura 4).



(a) Random Frequency Hopping (RFH). (b) Adaptive Frequency Hopping (AFH).

Figura 4: Diferencia entre RFH y AFH [15].

Otra herencia que Bluetooth clásico ha dejado a BLE es la modulación utilizada para codificar el flujo de datos, ya que ambos hacen uso de GFSK (Gaussian Frequency Shift Keying, un tipo de modulación donde los 1 y 0 lógicos se representan mediante desviaciones positivas y negativas, respectivamente, de la frecuencia de la onda portadora [16]), al igual que otros protocolos de red inalámbricos. El ancho de banda máximo definido para BLE es de 1 Mbit/s, aunque en la práctica dicho límite nunca se alcanza, debido a los datos extras, sobreflujo de control, que se va introduciendo en cada una de las capas del protocolo.

6.2. Capa de enlace

La capa de enlace se encuentra directamente conectada con la capa física, y consta de una parte hardware y otra parte software, ya que gran parte

de su funcionalidad es fácil de automatizar por hardware pero cara de implementar por software. Esta capa es la única con restricciones temporales reales, ya que debe cumplir con los requisitos que define el estándar, por lo que suele estar aislada del resto de capas para ocultar su complejidad y sus requisitos temporales.

El funcionamiento de la capa de enlace se puede describir como una máquina de estados con cinco estados diferentes [13]:

- *Standby*: No transmite ni recibe paquetes, únicamente puede cambiar de estado.
- *Advertising*: Permite transmitir paquetes por los canales destinados para mensajes de tipo *advertisement* (ver sección 6.2.4.2). Un dispositivo en este estado se conoce como *advertiser*.
- *Scanning*: El dispositivo escucha los canales de *advertisement* para captar los paquetes de este tipo que otros dispositivos estén emitiendo. Un dispositivo en este estado también es conocido como *scanner*.
- *Initiating*: Escucha los canales de *advertisement* buscando paquetes de uno o varios dispositivos en particular, respondiendo a estos paquetes para iniciar una conexión. Si un dispositivo se encuentra en este estado puede denominarse *initiator*.
- *Connection*: Dos dispositivos en este estado se encuentran conectados, pudiendo realizar transmisiones de datos entre ellos. Dentro de este estado se distinguen dos roles diferentes:
 - Maestro: Se trata del dispositivo que ha entrado en el estado *connection* desde el estado *initiating*. Este dispositivo se comunicará con uno o varios dispositivos en el rol de esclavos, definiendo los tiempos de transmisión.
 - Esclavo: Es el rol del dispositivo que ha entrado en la conexión desde el estado *advertising*. Únicamente puede estar conectado a un dispositivo maestro a la vez.

En un momento dado, un dispositivo puede estar en un único estado. El diagrama de estados de esta máquina de estados se muestra en la Figura 5.

A continuación vamos a ver con mayor profundidad los diferentes estados. Para ello comenzaremos definiendo el término "dirección de un dispositivo BLE", un concepto importante relacionado con la capa de enlace. Tras esto, pasaremos a detallar los posibles estados, clasificándolos en dos grupos: por un lado, aquellos que nos permiten intercambiar información

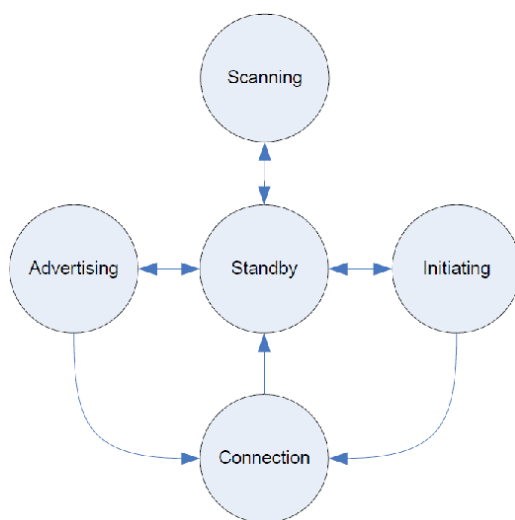


Figura 5: Máquina de estados de la capa de enlace [13].

sin establecer una conexión, y por otro, aquellos que nos permiten establecer conexiones. Para ver una explicación más detallada de cualquier estado o concepto de esta sección puede consultarse [13].

6.2.1. Dirección del dispositivo

La dirección del dispositivo (Device Address), es un identificador fundamental que permite reconocer inequívocamente a los dispositivos. Esta dirección tiene siempre una longitud de 48 bits, y puede ser de dos tipos:

- Pública (public device address): Debe crearse y registrarse siguiendo las normas del IEEE, y no cambiará durante el tiempo de vida del dispositivo. Esta dirección se encuentra dividida en dos campos de 24 bits cada uno, los cuales contienen datos para identificar la compañía y el dispositivo dentro de la misma, tal y como se muestra en la Figura 6a.
- Aleatoria (random device address): Se puede preprogramar en el dispositivo o generarla de manera dinámica cuando el dispositivo se encuentra en funcionamiento. Se puede utilizar como sustituta de la dirección pública o para identificar un dispositivo de manera segura. Como se puede observar en la Figura 6b, esta dirección también cuenta con dos campos, pero en este caso solamente uno de ellos es la dirección y el otro un campo hash para evitar errores. En la sección 6.8, referente al perfil de acceso genérico (GAP), se detallan los

tipos existentes dentro de la dirección aleatoria a niveles superiores del protocolo.

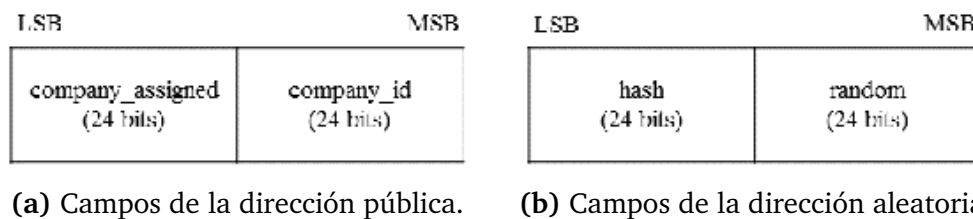


Figura 6: Campos de la dirección pública y aleatoria de un dispositivo [13].

Cada dispositivo debe contener al menos una dirección (puede tener las dos) para identificarse en las diferentes conexiones.

6.2.2. Estados sin conexión

En lo referente a los estados sin conexión de la máquina de estados de la capa de enlace, encontramos cuatro estados diferentes: *standby*, *advertising*, *scanning* e *initiating*.

6.2.2.1. Standby

El estado *Standby* es el estado por defecto de la capa de enlace, y en el no se envían ni reciben paquetes, sino que mantiene al controlador en un modo de espera que le permite consumir menos energía. Un dispositivo en este estado puede acceder a los estados *advertising*, *scanning* o *initiating*.

6.2.2.2. Advertising

En cuanto a los estados *advertising* y *scanning*, ambos trabajan con paquetes del tipo *advertisement* (ver 6.2.4.2) para una descripción más detallada). Estos paquetes son enviados por un dispositivo *advertiser* en ratios de tiempo fijos, definidos por el intervalo de *advertisement*, el cual puede variar de 20 ms a 10.24 sg (más un retraso pseudo-aleatorio, introducido por el controlador Bluetooth para evitar colisiones, cuyo valor oscila entre 0 y 10 ms). Cuanto menor sea el intervalo de *advertisement*, mayor será la frecuencia de emisión, lo que aumentará las posibilidades de que el mensaje llegue al receptor, pero también aumenta el consumo de energía del dispositivo. Aumentando el intervalo conseguimos descender el consumo de energía, pero también la posibilidad de que el paquete llegue al dispositivo receptor. El estándar Bluetooth [13] define subintervalos de valores dentro de este intervalo de *advertising* dependiendo de si la función del

dispositivo *advertiser* es formar parte de una conexión (intervalos de tiempo más cortos), o enviar datos en modo *broadcast* (intervalos de tiempo mayores).

Debido a que existen tres canales de frecuencia para transmitir los paquetes de tipo *advertisement* y los dispositivos *advertiser* y *scanner* no se encuentran sincronizados de manera alguna, los paquetes solamente serán recibidos cuando la escucha coincida, aleatoriamente, con la transmisión, tal y como se muestra en la Figura 7 [1]:

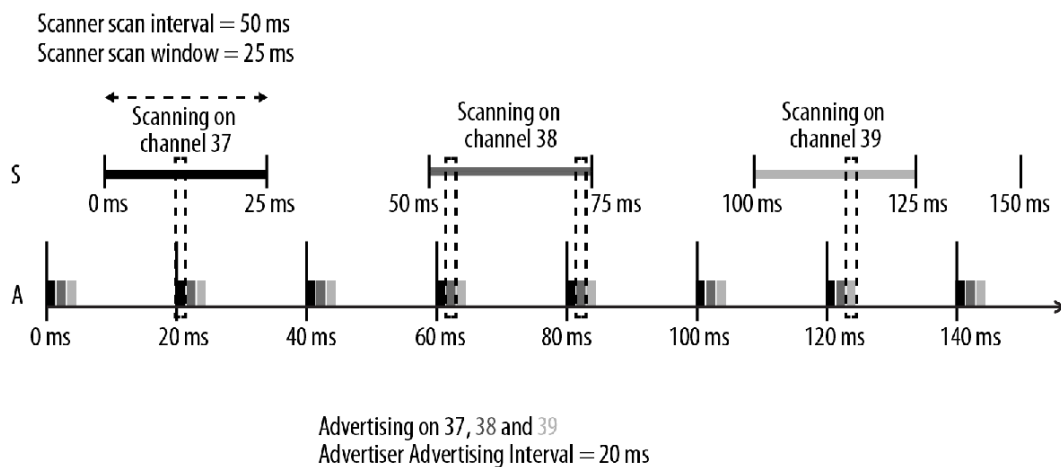


Figura 7: Advertising y Scanning, representación [1].

6.2.2.3. Scanning

Como ya se ha comentado, este estado también utiliza los paquetes del tipo *advertisement*. Solo que ahora el dispositivo que está en este estado se le denomina *scanner*. Los parámetros relativos al dispositivo *scanner* son *scan interval* y *scan window*. Estos parámetros definen la frecuencia y el tiempo por el que un dispositivo *scanner* escuchará la red en busca de mensajes de tipo *advertisement*. Al igual que sucede en el dispositivo *advertiser*, estos parámetros también influyen en el consumo de energía del dispositivo *scanner*. En lo referente al proceso de *scanning*, la especificación define también dos tipos, los cuales se detallan a continuación y se muestran en la Figura 8 [1]:

- *Passive scanning*: El dispositivo *scanner* únicamente escucha la red, por lo que el *advertiser* no tiene constancia de si algún dispositivo está recibiendo sus mensajes.
- *Active scanning*: El dispositivo *scanner* envía un paquete de tipo *scan response* tras la recepción de cada paquete recibido. Esto hace que el

advertiser sea capaz de enviar una mayor cantidad de datos, pero en ningún caso proporciona una manera de que el dispositivo *scanner* envíe datos al *advertiser*.

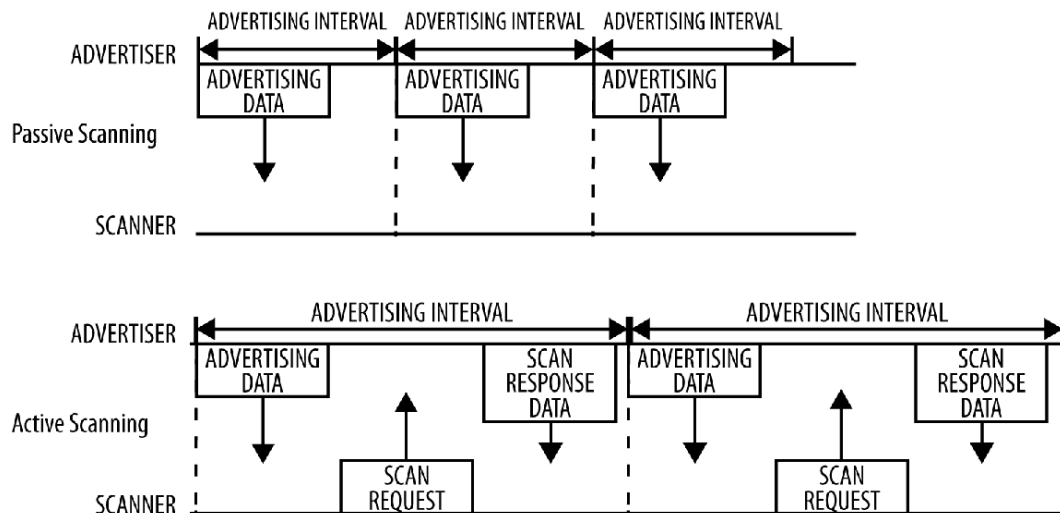


Figura 8: Diferencias entre Active y Passive Scanning [1].

6.2.2.4. Initiating

Por último, el estado *initiating* es muy similar al ya comentado *scanning*, pero a diferencia de éste, un dispositivo *initiator* escucha la red en busca de dispositivos *advertisers* que permitan establecer una conexión, enviando una petición de conexión a los dispositivos deseados. Si este proceso se completa con éxito, los dos dispositivos implicados cambiarán su estado al de *connection*, como se detalla a continuación.

6.2.3. Estado de conexión

A la hora de establecer una conexión entre un par de dispositivos intervienen los estados *advertising* e *initiating*, y se realiza siguiendo los siguientes pasos:

1. El dispositivo en estado *initiating* y que desempeñará el rol de maestro escucha la red en busca de dispositivos *advertiser* que acepten peticiones de conexión. Un dispositivo maestro puede seleccionar al esclavo en base a su dirección o a los datos enviados por el mismo.

2. Una vez se ha encontrado el dispositivo *advertiser* deseado, el *initiator* envía una petición de conexión al mismo. Este paquete de petición contiene el incremento del salto de frecuencia, el cual determina los saltos de frecuencia que utilizarán el maestro y el esclavo durante la conexión (ver AFH, en la sección 6.1).
3. El dispositivo *advertiser* envía una respuesta al *initiator*, haciendo que sus estados cambien a *connection* y quedando establecida la conexión. Como se ha comentado anteriormente, dentro del estado *connection* existen dos roles: el maestro, que será desempeñado por el dispositivo *initiator*, y el esclavo, puesto que será ocupado por el *advertiser*.

De esta manera, podemos definir una conexión como una secuencia de datos que se intercambia entre los dispositivos maestro y esclavo, siguiendo las pautas especificadas en el establecimiento de dicha conexión. Cada uno de estos intercambios de datos se denomina evento de conexión (*connection event*), y se muestra de manera esquemática en la Figura 9. Durante un evento de conexión, el dispositivo maestro y el esclavo envían y reciben alternativamente paquetes.

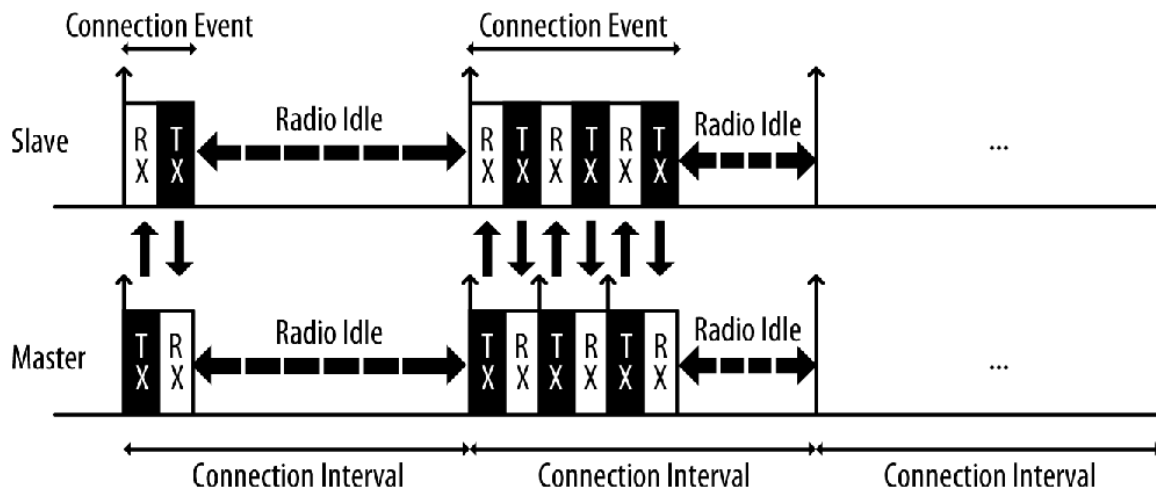


Figura 9: Connection events [1].

Un evento de conexión se considera abierto mientras que los dispositivos continúen enviando paquetes y puede ser cerrado por cualquiera de los dispositivos, bien sea porque se ha finalizado la transferencia de datos, bien porque se han recibido múltiples paquetes con un CRC erróneo, o bien porque se ha dejado de recibir paquetes (para más información sobre esto puede consultarse [13]).

En relación con estos eventos de conexión, podemos encontrar varios parámetros importantes:

- *Connection interval*: Establece el tiempo que transcurre entre el inicio de dos eventos de conexión consecutivos, siendo labor del dispositivo maestro evitar que se solapen, cerrando un evento de conexión antes del inicio del siguiente. El valor de este parámetro oscila entre 7.5 ms (mayor flujo de datos, pero también mayor consumo) y 4 s (menor flujo de datos y menor consumo).
- *Slave latency*: Permite al dispositivo esclavo utilizar un número reducido de eventos de conexión, ya que establece el número máximo de eventos de conexión que puede saltarse sin que el dispositivo maestro cierre el evento de conexión. Se debe tener cuidado para que este valor no interfiera con el *Connection supervision timeout* y se produzca un cierre de la conexión.
- *Connection supervision timeout*: Establece el tiempo máximo entre la recepción de dos paquetes de datos antes de que la conexión se considere perdida y se cierre.
- *Connection Transmit Window*: Permite al dispositivo maestro administrar de manera eficiente los diferentes eventos de conexión u otras actividades que deba realizar. Para ello le dota de flexibilidad a la hora de elegir el comienzo de un evento de conexión, dentro de un margen de tiempo conocido como ventana de transmisión (*transmit window*).

Una vez que dos dispositivos han establecido una conexión, se utilizan los paquetes de datos (ver la sección 6.2.5 para una descripción detallada de los mismos) para transportar los datos del usuario de manera bidireccional entre el dispositivo maestro y el esclavo.

Para aportar seguridad y mantener la confidencialidad e integridad de los datos, la capa de enlace de BLE hace uso de las siguientes medidas:

- CRC: Se comprueba que todos los paquetes recibidos se encuentran libres de errores CRC (*Cyclic Redundancy Check*), y en caso contrario deben volver a enviarse.
- Encriptación: Para proporcionar seguridad en BLE, la capa de enlace proporciona un enlace cifrado para intercambiar datos de manera segura. Las claves son generadas y administradas por las capas superiores, tal y como veremos más adelante, pero es la capa de enlace la

encargada de cifrar y descifrar los paquetes de manera transparente a las capas superiores.

- *White List*: Estas listas contienen únicamente direcciones de dispositivos BLE, y nos permiten filtrar los dispositivos cuando nos encontramos en los estados de *advertising*, *scanning* o *initiating*, ya sea por razones de seguridad o simplemente para trabajar con un subconjunto conocido de antemano de todos los disponibles. De esta manera, un dispositivo *scanner* o *initiator*, puede utilizar esta lista para limitar el número de dispositivos que detectará o con los que podrá conectarse, mientras que un dispositivo *advertiser* utilizará estas listas para seleccionar los dispositivos de los cuales aceptará una conexión. Cuando se recibe un mensaje de un dispositivo cuya dirección no se encuentre en la *White List*, dicho mensaje se desecha.

6.2.4. Formato de paquetes de capa de enlace

BLE hace uso únicamente de dos tipos de paquetes: (*Advertisement* y *Data*), los cuales comparten el mismo formato. Esto supone una enorme ventaja al implementar la pila del protocolo. El conocer más detalladamente estos paquetes y sus diferencias puede servir de gran utilidad en múltiples ocasiones al trabajar con redes de dispositivos BLE, por lo que a continuación se definirán ambos tipos detalladamente.

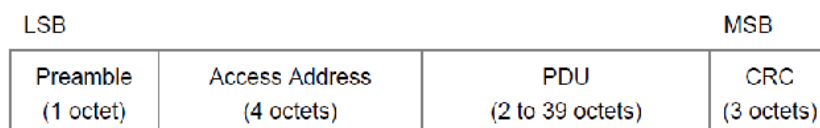


Figura 10: Estructura y campos de un paquete BLE [13].

6.2.4.1. Formato general La capa de enlace cuenta únicamente con un formato para los dos tipos de paquetes. Este formato general se muestra en la Figura 10, en la que podemos ver los campos de los que consta cada paquete: preamble, Access Address, PDU y CRC.

A continuación se muestra una pequeña definición de cada uno de los campos, así como de su contenido:

- **Preamble**: Se encuentra presente en todos los paquetes BLE, tiene una longitud de 1 Byte y se envía en primer lugar. Este campo contiene una sucesión de unos y ceros alternos, y su función consiste en ayudar al receptor a realizar tareas relacionadas con la transmisión,

como la sincronización de la frecuencia. Este campo tiene un valor fijo dependiendo del tipo de paquete:

- En los paquetes de tipo *Advertisement* siempre tiene un valor de 10101010b.
 - En los paquetes de tipo *Data*, este campo tendrá un valor de 10101010b ó 01010101b, dependiendo de si el bit menos significativo (LSB) del campo Access Address es 0 o 1, respectivamente.
- Access Address: Este campo tiene una longitud de 4 Bytes y se envía en segundo lugar. Al igual que el anterior, este campo también varía en función del tipo de paquete del que se trate. De esta manera, tenemos que:
 - Los paquetes *Advertisement* siempre deben llevar la misma dirección de acceso, la cual es 0x8E89BED6.
 - Los paquetes de tipo *Data* llevan una dirección de acceso de 32 bits que se genera en el estado *Initiating* y se envía al dispositivo con el que se establece la conexión. Además, debe ser diferente para cada conexión y cumplir una serie de requisitos definidos en el estándar.
 - PDU: Este campo se envía a continuación del campo Access Address, y varía en función del tipo de paquete. El contenido de este campo para los diferentes tipos se detalla en las secciones 6.2.4.2 y 6.2.5.
 - CRC: Es el último campo en ser transmitido, y consiste en un CRC de 24 bits que se calcula sobre el campo PDU siguiendo los pasos establecidos por el estándar.

Al ser el PDU un campo con longitud variable, la longitud total del paquete puede variar desde 80 bits hasta 376 bits.

6.2.4.2. Paquete de tipo *Advertisement*

Este tipos de paquetes consisten básicamente en mensajes de difusión a través del aire, sin conocimiento previo sobre si existe algún dispositivo que vaya a recibirlos. Estos paquetes se utilizan para realizar alguna de las siguientes acciones:

- Permitir a los esclavos darse a conocer y a los maestros establecer una conexión con ellos.

- Realizar difusiones de datos en aquellas aplicaciones que no requieren establecer una conexión.

Formato

En los paquetes de tipo *Advertisement*, el campo PDU comentado en la sección 6.2.4.1 contiene los campos *Header* y *Payload*, tal y como se refleja en la Figura 11.

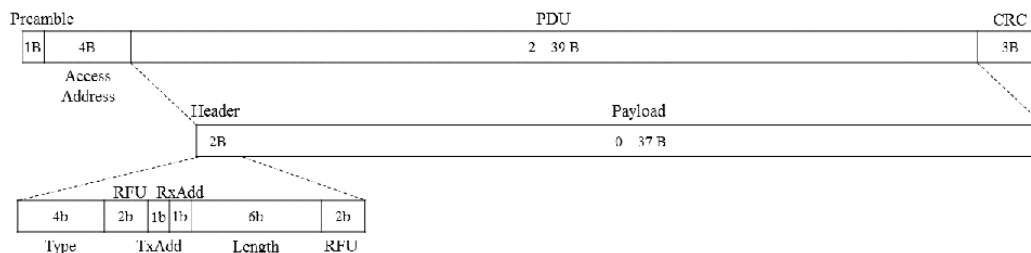


Figura 11: Estructura y campos de un paquete BLE de tipo *Advertisement*.

Dichos campos serán detallados a continuación:

- Header:** este cuenta a su vez con una serie de campos, los cuales se muestran en la Figura 12. La función de estos campos es:
 - Type: Especifica el tipo de PDU de entre todos los posibles en los paquetes de tipo *Advertisement*. Los diferentes tipos de PDU se comentarán más adelante.
 - RFU: Los campos RFU se encuentran reservados para usos futuros, y su valor debe ser 0.
 - Los campos TxAdd y RxAdd especifican si la dirección del dispositivo transmisor y del receptor contenidas en el *Payload* es la dirección pública o aleatoria (ver Dirección del dispositivo en la página 13). Estos campos pueden utilizarse o no dependiendo del tipo de *Payload*. En caso de no utilizarse, se tratan como si fueran campos del tipo RFU.

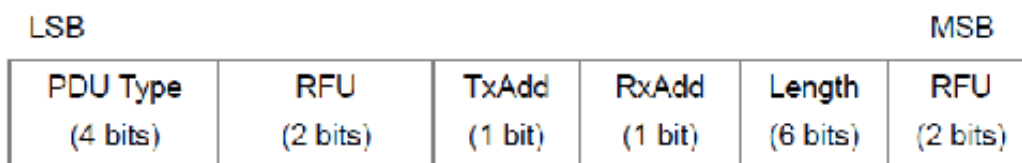


Figura 12: Estructura y campos del campo *Header* de un paquete *Advertisement*.

- **Length:** Este campo define la longitud del campo *Payload* incluido en el paquete, y sus valores varían entre 6 y 37 bytes.
- **Payloads pertenecientes a una PDU de tipo Advertising:** estas PDUs se envían desde un dispositivo que se encuentre en un estado de *Advertising*, y se reciben por dispositivos que se encuentren en los estados de *Scanning* o *Initiating*.

Tabla 2: *Payloads* pertenecientes a una PDU de tipo *Advertising*.

Nombre PDU	Evento	Descripción	Campos en Payload
ADV IND	<i>Connectable undirected advertising event</i>	Permite a un <i>scanner</i> o <i>initiator</i> responder con un <i>scan request</i> o <i>connect request</i>	Dirección del dispositivo <i>advertiser</i> y datos del mismo (opcionales)
ADV DIRECT IND	<i>Connectable directed advertising event</i>	Permite a un <i>initiator</i> responder con una <i>connect request</i>	Dirección del dispositivo <i>advertiser</i> y dirección del dispositivo al que va dirigido
ADV NON CONN IND	<i>Non connectable undirected advertising event</i>	Permite a un <i>scanner</i> recibir información del <i>advertiser</i> contenida en el campo de datos	Dirección del dispositivo <i>advertiser</i> y datos del mismo (opcionales)
ADV SCAN IND	<i>Scannable undirected advertising event</i>	Permite a un <i>scanner</i> responder con un <i>scan request</i> para pedir información adicional sobre el <i>advertiser</i>	Dirección del dispositivo <i>advertiser</i> y datos del mismo (opcionales)

Este tipo de paquetes se puede clasificar de acuerdo a tres propiedades diferentes:

- **Connectability:**
 - *Connectable*, si un *scanner* puede iniciar una conexión tras recibir un paquete de este tipo.
 - *Non-connectable*, si un *scanner* no puede iniciar una conexión. Este tipo de paquetes se utilizan únicamente para realizar transmisiones de datos de tipo *broadcast*.
- **Scannable:**
 - *Scannable*, si permite a un *scanner* que recibe este paquete responder.
 - *Non-scannable*, cuando no se espera respuesta de los dispositivos que reciben estos paquetes.
- **Directability:**
 - *Directed*, si se especifica la dirección del dispositivo al que va dirigido el paquete. Este tipo de paquete no admite datos adicionales, y es siempre del tipo *connectable*.

- *Undirected*, cuando no van dirigidos a ningún dispositivo en particular. Estos paquetes pueden contener datos del usuario.

Las diferentes PDU englobadas dentro del tipo *Advertising* se recogen en la tabla 2. En dicha tabla aparece el nombre con el que se conocen los diferentes tipos de PDUs, el evento que genera el *Advertising* que los contiene, una descripción del uso de ese *Advertising* y los diferentes datos contenidos en la PDU.

Este payload se divide en dos campos principales: la dirección dirección pública o aleatoria del dispositivo *advertiser* (AdvA), de 6 bytes de longitud, y un campo de datos opcional (AdvData), cuya longitud varía entre 0 y 31, tal como se muestra en la Figura 13.

AdvA (6 Bytes)	AdvData (0 – 31 Bytes)
-------------------	---------------------------

Figura 13: Estructura y campos del campo *Payload* de un paquete *Advertisement*.

El formato de los datos contenidos en el campo AdvData se muestra gráficamente en la Figura 14. Como se puede ver en la figura, este campo de datos está dividido en dos partes, conocidas como parte significativa y parte no significativa. La parte significativa contiene una serie de estructuras AD, mientras que la parte no significativa extiende la longitud del campo hasta los 31 bites de longitud, completándolo con ceros.

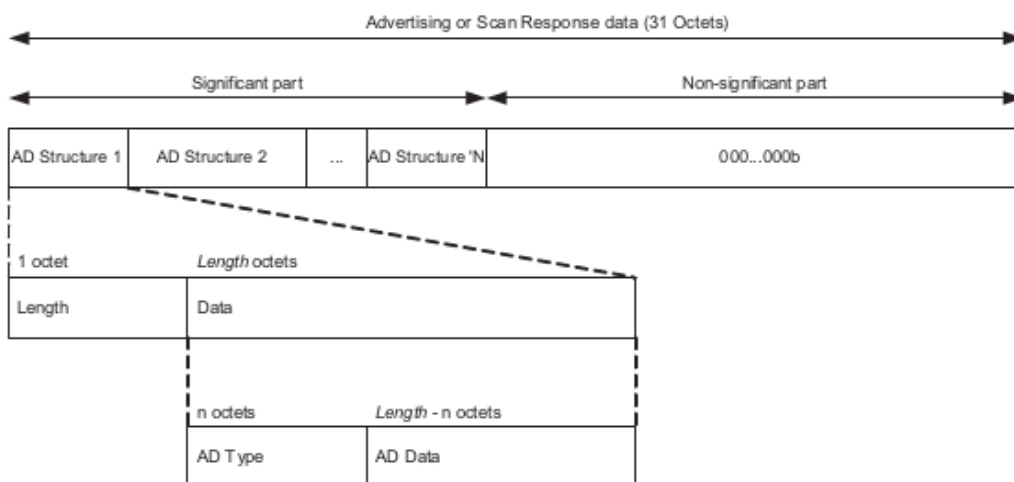


Figura 14: Formato de datos en mensajes *Advertising* o *Scan Response* [17].

Cada estructura AD contiene un campo *Length* de un byte, el cual indica la longitud de la estructura AD. El primer byte del campo de datos es el campo *AD Type*, que nos indica el tipo de datos contenidos. El resto del campo de datos son los datos propiamente dichos, que varían en función del tipo especificado.

Existen diferentes tipos de AD (almacenado en el campo *AD Type*), los cuales se mencionan, de manera agrupada, a continuación. Además, todos ellos se encuentran detallados en el documento de *Assigned Numbers* [18] de Bluetooth SIG.

- *Service UUIDs*: Permite identificar el UUID del servicio que implementa en los datos de sus paquetes *advertising*. Este UUID puede ser de 16 o de 128 bits.
- *Local Name*: Contiene el nombre del dispositivo, ya sea de manera completa o acortada.
- *Flags*: Contienen diferentes bits que se interpretan como valores booleanos, y proporcionan información sobre las opciones que el dispositivo soporta físicamente.
- *Manufacturer Specific Data*: Utilizado para transmitir datos específicos del fabricante.
- *TX Power Level*: Indica el nivel de la potencia de transmisión del paquete.
- *Security Manager Out of Band (OOB)*: Se utiliza por el Administrador de Seguridad (SM) para comunicar determinada información relacionada con el descubrimiento de un dispositivo y el establecimiento de una conexión segura. Este tipo únicamente debe utilizarse en el mecanismo de *out-of-band*.
- *Security Manager TK Value*: Al igual que el tipo anterior, transmite datos relativos a al emparejamiento de dispositivos, y solamente debe utilizarse en el mecanismo de *out-of-band*.
- *Slave Connection Interval Range*: Contiene el rango de intervalos preferidos por el dispositivo periférico, para cualquier conexión lógica. El nodo central tomará en cuenta estos valores al establecer una conexión.
- *Service Solicitation*: Permite a un dispositivo pedir uno o más servicios al establecer una conexión. Esto obliga a un nodo central a proporcionar uno o más de estos servicios para conectarse con el periférico, para que el periférico pueda hacer uso de ellos.

- *Service Data*: Contiene el UUID de un servicio, junto con los datos asociados a dicho servicio.
- **Payload perteneciente a una PDU de tipo *Scanning***: Este tipo de PDUs cuenta con dos modelos diferentes. Dichos modelos se utilizan en los siguientes casos:
 - SCAN_REQ: Se envía por un dispositivo en estado *Scanning*, y lleva por destino un dispositivo en el estado *Advertising*.
 - SCAN_RSP: Se envía por un dispositivo que se encuentre en estado *Advertising*, y es recibida por un dispositivo en el estado *Scanning*.

En la tabla 3 se recogen estas PDUs, junto con una breve descripción de las mismas y de los campos que se incluyen en su *Payload*.

Tabla 3: *Payloads* pertenecientes a una PDU de tipo *Scanning*.

Nombre PDU	Descripción	Campos en Payload
SCAN_REQ	Utilizado por un <i>scanner</i> pedir más información a un <i>advertiser</i>	Dirección del dispositivo <i>scanner</i> y del <i>advertiser</i> al que va dirigido
SCAN_RSP	Utilizado por un <i>advertiser</i> al que le ha llegado un SCAN_REQ pidiendo más información	Dirección del dispositivo <i>advertiser</i> y datos de respuesta (opcionales)

- **Payload perteneciente a una PDU de tipo *Initiating***: En lo referente al tipo *Initiating*, solamente encontramos uno. Esta PDU es enviada por un dispositivo en el estado *Initiating* y recibida por otro dispositivo que se encuentre en el estado *Advertising*.

En la tabla 4 se recoge la PDU de este tipo, junto con una descripción de la misma y de los datos contenidos en su *Payload*.

Tabla 4: *Payloads* pertenecientes a una PDU de tipo *Initiating*.

Nombre PDU	Descripción	Campos en Payload
CONNECT_REQ	Permite a un <i>initiator</i> establecer una conexión con un <i>advertiser</i>	Dirección de los dispositivos <i>initiator</i> y <i>advertiser</i> , además de los datos necesarios para la conexión

6.2.5. Paquete de tipo Data

Los paquetes de tipo *Data* se utilizan para realizar el intercambio de datos del usuario entre el dispositivo maestro y el esclavo, de manera bidireccional, en una conexión. Por ello son paquetes que encapsularán paquetes

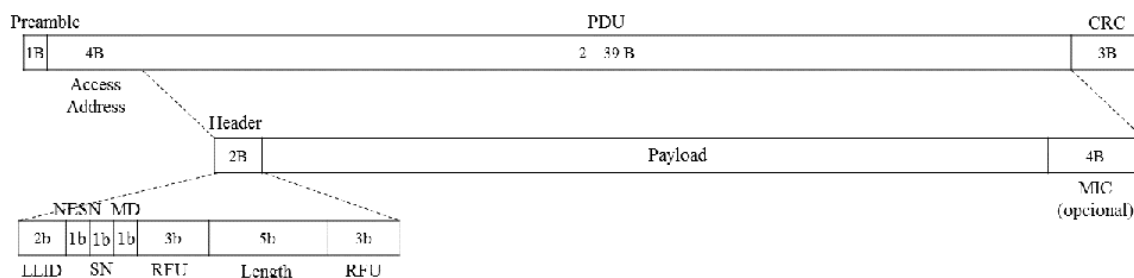


Figura 15: Estructura y campos de un paquete BLE de tipo *Data*.

procedentes de la capa L2CAP. Además, también se utilizan para manejar diferentes procedimientos de control. A continuación se detallará el formato de estos paquetes, diferenciando los que se utilizan para transportar datos de los que se usan en los procedimientos de control.

6.2.5.1. Formato

Los paquetes de tipo *Data* siguen el formato especificado en la sección 6.2.4.1, pero sus PDUs se dividen en los siguientes campos, tal y como se refleja en la Figura 15:

- **Header:** Cabecera de 16 bits, cuyos campos se detallarán más tarde.
- **Payload:** Campo de tamaño variable, que contiene los datos del usuario, procedentes de la capa L2CAP, o aquellos necesarios para realizar los procedimientos de control.
- **MIC:** Este campo es utilizado en conexiones cifradas, siempre y cuando el paquete incluya datos, y el cálculo de su valor se especifica en el estándar [13].

Header

El campo *Header* de los paquetes de tipo *Data* cuentan con los campos que se muestran en la Figura 16. La función de cada uno de estos campos es:

- **LLID:** Indica el tipo de paquete *Data* del que se trata: paquete de control, paquete completo de datos o fragmento de datos.
- **NESN:** *Next Expected Sequence Number*, permite a un dispositivo informar a su par de que se ha recibido un paquete.

- SN: El *Sequence number* se utiliza para identificar el paquete.
- MD: El campo *More Data* indica si existen más fragmentos de datos.
- RFU: Los campos *Reserved for Future Use* deben contener ceros y ser ignorados cuando se reciba el paquete.
- Length: Contiene la longitud (en bytes) del campo *Payload* y del MIC (si se encuentra).



Figura 16: Estructura y campos del campo *Header* de un paquete *Data*.

Payload de datos

Los *Payload* de este tipo se utilizan para transmitir datos de las capas superiores del protocolo. Si la conexión es cifrada el campo MIC deberá contener el valor correspondiente para permitir al receptor utilizar los datos.

Un tipo especial de PDU dentro de este apartado son las PDUs vacías. Estas PDUs vacías llevan unos valores determinados en los campos LLID y Length para indicar el tipo de paquete, y son utilizados por el dispositivo maestro en una conexión para permitir al esclavo enviar cualquier paquete de tipo *Data*.

Payload de control

Los paquetes de este tipo se utilizan para controlar la conexión de la capa de enlace. Su *Payload* tiene la estructura mostrada en la Figura 17.

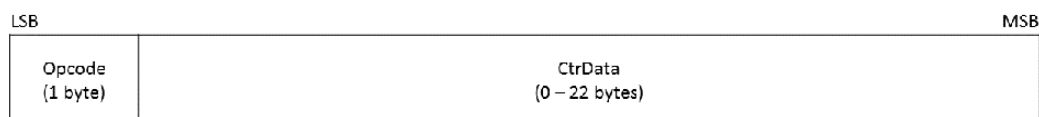


Figura 17: Estructura y campos del campo *Payload* en una PDU de control.

El campo *Opcode* indica el tipo de *Payload* de control del que se trata. Entre estos códigos encontramos aquellos para modificar los parámetros de la conexión, iniciar la encriptación de una conexión o para informar de

errores, entre otros. Para ver la totalidad de tipos diferentes de PDUs de control, puede consultarse [13].

El campo *CtrlData* se utiliza únicamente en los casos en los que el tipo de operación de control requiere el envío de algún dato.

6.3. Interfaz Host-Controlador, HCI.

La Interfaz Host-Controlador (HCI) es un protocolo que permite la comunicación entre un host y un controlador Bluetooth a través de una interfaz serie [1]. El sentido de establecer aquí una separación se encuentra en que el módulo del controlador tiene una pila de protocolos muy estricta temporalmente hablando, y poco adecuada para CPUs muy avanzadas (habituales en la parte del host). Podemos encontrar ejemplos de esta configuración en la mayoría de los dispositivos, tales como ordenadores, tablets o smart phones, donde el host (aplicación) se ejecuta en la CPU principal, la cual envía órdenes a un chip hardware separado, conectado mediante una interfaz UART o USB.

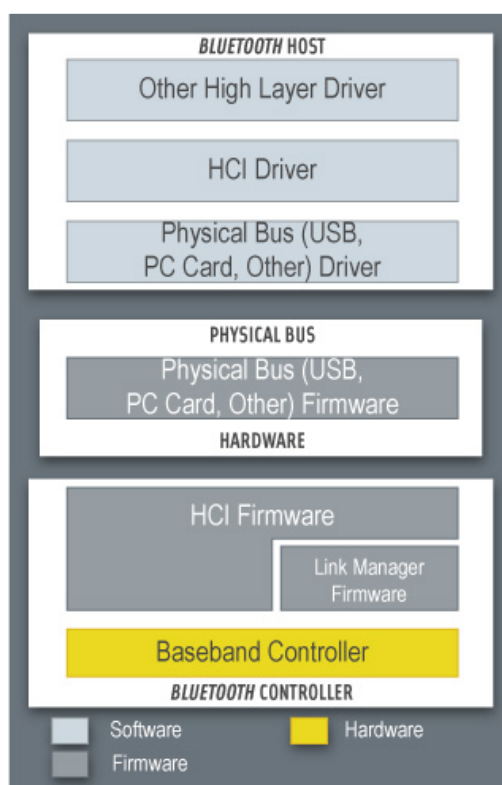


Figura 18: División de la interfaz HCI en sus dos partes: el driver HCI, en el host BLE, y firmware HCI, en el controlador BLE. La HCI se encuentra a ambos lados del bus físico, asilando al resto de capas del mismo [3].

La Figura 18 muestra una vista esquemática de las capas más bajas del estándar BLE. Este estándar [19] define la interfaz HCI como un conjunto de comandos y eventos que permiten la interacción entre el host y el controlador, además de un formato para los paquetes de datos y un conjunto de reglas para el control de flujo. Existen diferentes estándares para la capa de transporte HCI, utilizadas en función de las diferentes interfaces hardware, permitiendo transmitir los mismos comandos, eventos y paquetes de datos. Entre esos estándares destacan USB (para PCs) y UART (en el caso de los dispositivos móviles).

Entre el driver HCI del host y el firmware HCI instalado en el hardware Bluetooth pueden existir múltiples capas. Estas capas intermedias, conocidas como Capa de Transporte Host-Controlador (*Host Controller Transport Layer*), proporcionan la capacidad para transferir datos sin tener conocimiento de los mismos, permitiendo al driver HCI del host intercambiar datos y comandos con el firmware HCI del hardware Bluetooth, sin preocuparse de la separación entre ellas. El objetivo de esta capa de transporte es la transparencia, permitiendo a los drivers ser independientes de la tecnología de transporte subyacente.

De esta forma, la especificación HCI, [20]-Parte E, determina el formato de cada uno de los paquetes HCI que se intercambian el controlador y el host. Existen cuatro tipos de paquetes HCI, que son listados en la Tabla 5.

Tabla 5: Tipos de paquetes HCI

Indicador HCI (1 byte)	Tipo paquete HCI
0x01	Comando HCI
0x02	Datos ACL HCI
0x03	Datos síncrono HCI
0x04	Evento HCI

El Indicador HCI debe ser enviado antes del paquete HCI. Todos los tipos de paquetes HCI tienen un campo de longitud, el cual especifica la cantidad de bytes que forman el paquete HCI. Cuando el total de bytes especificados ha sido recibido, el sistema espera la recepción del siguiente Indicador HCI, que indicará el comienzo del nuevo paquete HCI.

A continuación se describirá el formato de cada uno de los paquetes HCI, con excepción del paquete de datos síncronos HCI.

6.3.1. Ordenación de los bits.

La ordenación de los bits en los campos definidos dentro de los paquetes a lo largo de este documento sigue la misma que la utilizada en la especi-

ficación de la capa de enlace de BLE ([13]), el formato *Little Endian*, que sigue las siguientes reglas:

- El bit menos significativo (LSB) corresponde a b_0 .
- El LSB es el primer bit enviado en las transmisiones.
- En las ilustraciones, el LSB se muestra a la izquierda.

Por otro lado, los valores binarios de los diferentes campos se encuentran indicados siguiendo el formato *Big Endian* (es decir, el bit más significativo o MSB se muestra a la izquierda), siguiendo de nuevo las pautas utilizadas en la especificación.

6.3.2. Paquete HCI para Comandos.

Los paquetes HCI para comandos se utilizan para enviar comandos de configuración o actuación desde el host al controlador. Un controlador podrá aceptar un paquete HCI de comando de una longitud máxima de 255 bytes, excluyendo la cabecera del paquete en cuestión (3 bytes).

El formato de este paquete es mostrado en la Figura 19. Como ya se ha indicado, este paquete debe ir precedido del indicador HCI (0x01) que especifica que la información que le sigue debe ser interpretada como un Comando HCI.

El parámetro `OpCode` de 2 bytes especifica el código del comando enviado. Este parámetro es dividido en dos campos: `OCF` (*OpCode Group Field*) que ocupa los 6 bits más significativos de `OpCode`, mientras que los otros 10 bits menos significativos son utilizados por el campo `OCF` (*OpCode Command Field*)

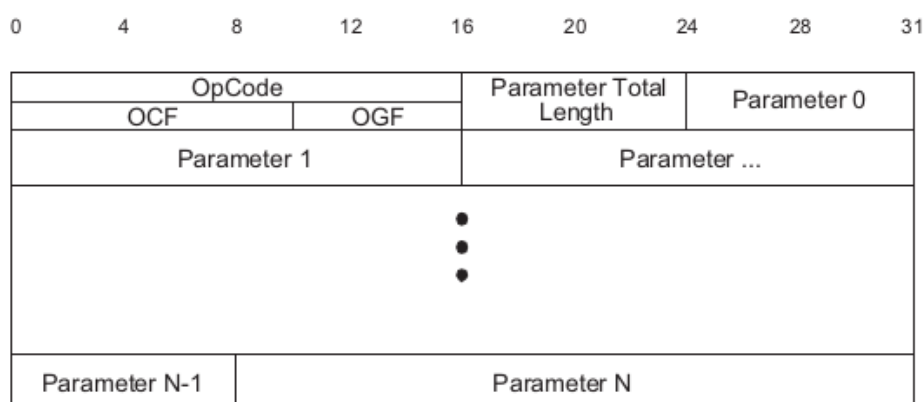


Figura 19: Formato de un paquete HCI para Comando.

Dicho esto el valor de los campos de este tipo de paquetes es el siguiente:

OpCode - OGF (6 bits) : 0x00-0x3F (el valor 0x3F es reservado para comandos de depuración específicos de fabricantes de dispositivos Bluetooth).

OpCode - OCF (10 bits) : 0x0000-0x03FF.

Total Length (1 byte) : longitud en bytes del total de parámetros transmitidos en el paquete.

Parameter X (n bits) : cada comando tiene un número específico de parámetros, y cada uno de ellos tiene una longitud determinada. El número, significado y longitud de cada uno de estos parámetros es definido por el comando en sí. Pero la longitud de todos ellos debe coincidir con el parámetro *Total Length*.

Un listado de cada uno de los **comandos utilizados por la capa HCI** en el caso de una comunicación Bluetooth LE, puede consultarse en [20]-**Parte E, sección 7.8**.

6.3.3. Paquete HCI de Eventos.

Los paquetes HCI de eventos son utilizados por el controlador para notificar al host cuando ha ocurrido un evento que ha podido modificar el comportamiento del controlador. Todo host podrá procesar paquetes asociados a eventos HCI de un máximo de 255 bytes sin tener en cuenta la cabecera del paquete (2 bytes).

El formato de los paquetes asociados a la comunicación de eventos HCI se puede ver en la Figura 20. Este tipo de paquetes deben ir precedidos del Indicador HCI (0x04).

En el caso de Bluetooth LE, el controlador LE envía códigos de *sub-eventos* para indicar que los eventos transmitidos son específicos de LE. Estos códigos de *sub-eventos* irán siempre codificados en el primer Event Parameter.

El significado de cada uno de los campos del paquete asociado a eventos HCI es el siguiente:

Event Code (1 byte) : 0x00 - 0xFF, identifica el tipo de evento al que está asociado el mensaje. El código 0xFF es reservado para eventos de depuración específicos de fabricantes de dispositivos.

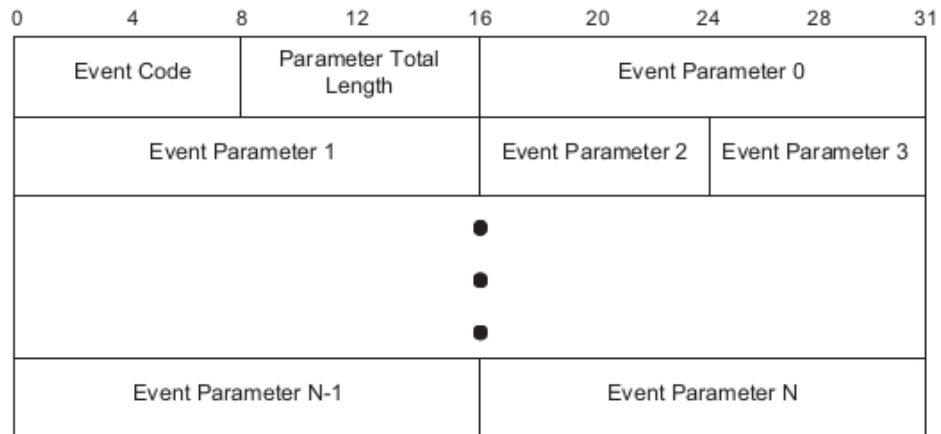


Figura 20: Formato de un paquete HCI para Evento.

Total Length (1 byte) : longitud en bytes del total de parámetros transmitidos en el paquete.

Event Parameter X (n bits) : cada evento tiene un número específico de parámetros, y cada uno de ellos tiene una longitud determinada. El número, significado y longitud de cada uno de estos parámetros es definido por el evento en sí. Pero la longitud de todos ellos debe de coincidir con el parámetro *Total Length*.

Un listado de los **eventos de la capa HCI** se pueden ver en [20]-**Parte E, sección 7.7**

6.3.4. Paquete HCI de Datos ACL.

El paquete HCI de datos ACL se utiliza para el intercambio de información entre el host y el controlador. La información a intercambiar irá insertada en el campo Data del paquete y procederá de la capa *L2CAP*, ver sección 6.4 para información sobre el formato de mensajes.

El formato del paquete HCI de Datos ACL se muestra en la Figura 21. Al igual que todos los paquetes HCI, este irá precedido por un campo Identificador HCI con valor 0x02.

El significado de cada uno de los campos de este paquete es:

Handle (12 bits) : es un identificador de conexión (*connection handle*) a la que pertenecen los datos incluidos en el campo Data.

PB Flag (2 bits) : en el caso de comunicaciones Bluetooth LE estos flag indican si el contenido del campo Data es el primer fragmento/paquete

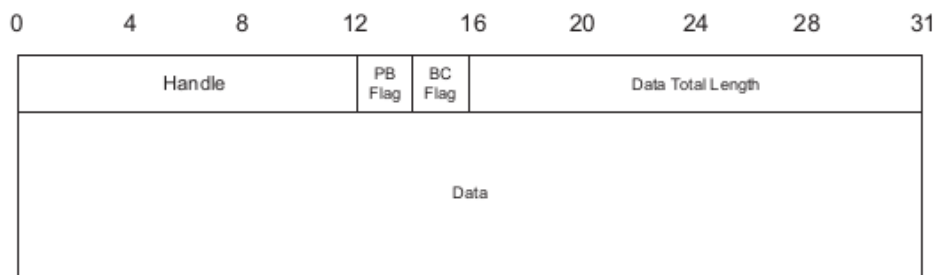


Figura 21: Formato de un paquete HCI de datos ACL.

de las capas superiores, o es un fragmento intermedio. Los posibles valores de este campo son:

PB Flag	Descripción
00	Indica el primer paquete de datos procedentes de capas superiores y enviados desde el host al controlador.
01	Indica fragmento posterior de datos (distinto del primero), procedentes de capas superiores. El sentido de la comunicación (host->controller, o controller->host) viene determinado por el primer paquete asociado a este. El cual será identificado por el mismo Handle y su valor de PB Flag.
10	Indica el primer paquete de datos procedentes de capas superiores y enviados desde el controlador al host.

BC Flag (2 bits) : indica si el paquete ha sido enviado o recibido mediante broadcast o no. La recepción o envío dependerá de si el paquete es enviado del host al controlador (mensaje a emitir) o viceversa. La interpretación de los valores de este campo es:

BC Flag	Descripción
00	El mensaje no es de broadcast.
01	El mensaje es de broadcast.
10	
11	Reservado para uso futuro

Total Length (2 bytes) : longitud total en bytes del campo Data

Data (Total Length bytes) : está formado por el paquete de la capa *L2CAP* que se encapsula dentro de este paquete HCI de datos ACL.

6.3.5. Interacciones entre HCI y Capa de Enlace

En esta sección se intentará dar una visión de cuales son los mecanismos de comunicación de comandos y eventos entre el *Host Controller Interface* y la capa de enlace en Bluetooth LE. Nos centraremos en algunos procedimientos específicos, indicando los mensajes que se intercambian entre ambas entidades en dicho procedimiento. Un examen más exhaustivo de la interacción entre HCI y la capa de enlace se puede hacer en [13]-Parte D.

Aclarar que la descripción se centrará en el lado del cliente, ya que será el dispositivos sobre el que nosotros podemos tener acceso directo (smartphone, PC), pudiendo obtener trazas de todos el tráfico de mensajes que se genera entre el HCI y la capa de enlace de datos. **Hay que tener en cuenta, que en el caso de Bluetooth LE, nosotros no podemos obtener la trama que se envía por la interfaz, sino que la que obtenemos es la que envía el HCI, interfaz hciX de nuestro dispositivo.**

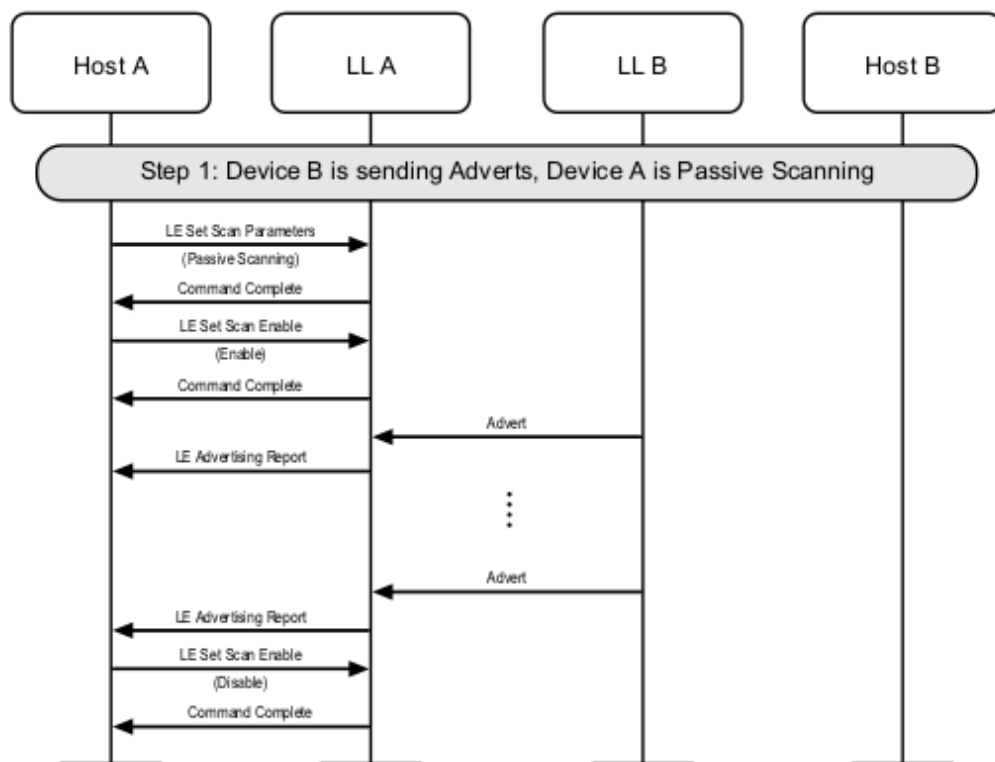


Figura 22: Secuencia de mensajes de un *Scanning* pasivo.

6.3.5.1. Proceso de *Scanning* pasivo

Este proceso es el que habitualmente realiza un cliente cuando intenta descubrir los dispositivos (*peripherals*) que están en su área de cobertura. En la Figura 22, pueden verse los mensajes enviados durante el proceso.

Tan solo indicar que todos los mensajes enviados entre los entes Host A y LL A, son comandos o eventos HCI.

6.3.5.2. Proceso de establecimiento de conexión

El proceso de establecimiento de una conexión entre un cliente y un servidor se puede ver esquematizado en la Figura 23. Al igual que en el caso anterior entre Host A y LL A solo se intercambian comandos o eventos HCI.

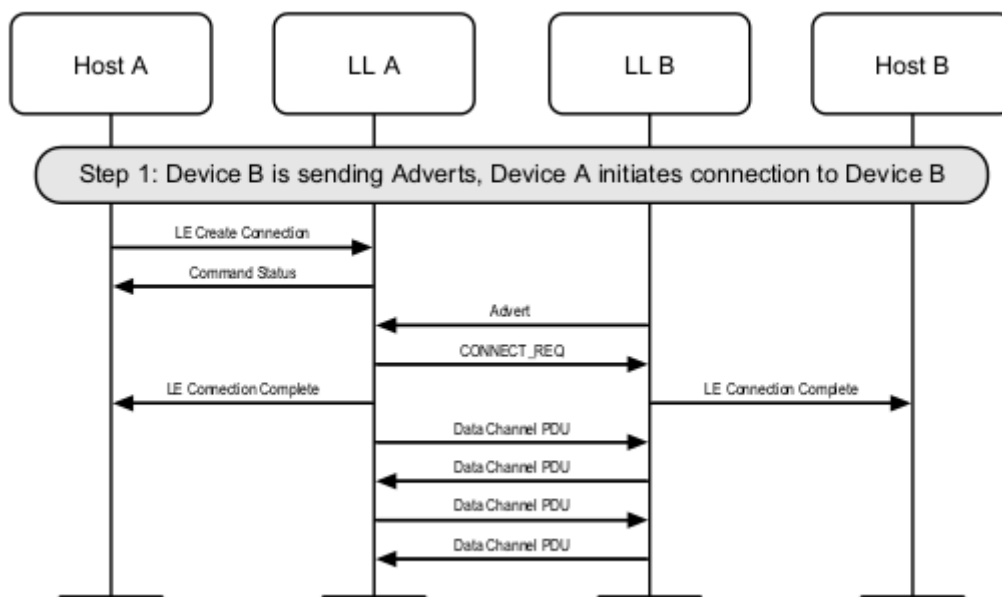


Figura 23: Secuencia de mensajes de una inicialización de conexión.

6.3.5.3. Actualización de la conexión

Este proceso se da cuando el se quiere actualizar los parámetros que definen el funcionamiento de la conexión entre un cliente y un servidor. Al igual que en los procesos anteriores, en éste solo se ven implicados entre Host A y LL A comandos o eventos HCI. La Figura 24 muestra un esquema del proceso.

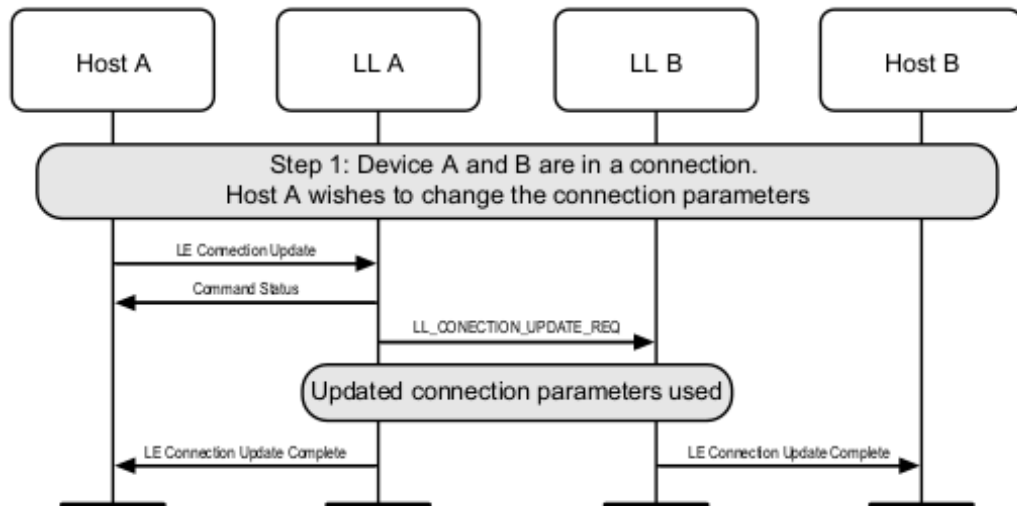


Figura 24: Secuencia de mensajes de una actualización de conexión.

6.3.5.4. Envío de datos

En la Figura 25 se representa el intercambio de información en el proceso de envío de datos. En este proceso se intercambian paquetes de tipo *ACL Data*, que no son otros que paquetes HCI que contienen datos de capas superiores. El esquema es el mismo tanto para la lectura de datos de un

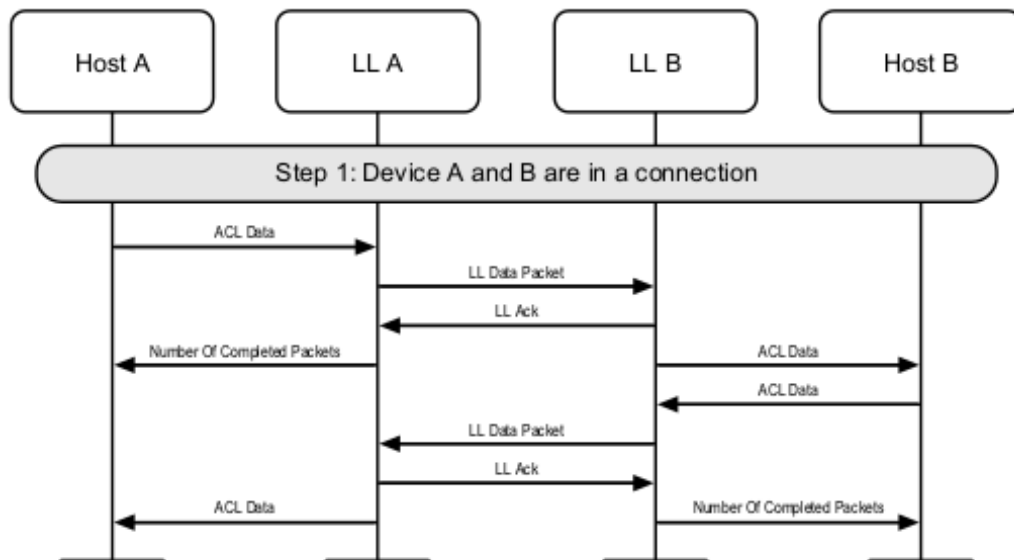


Figura 25: Secuencia de mensajes en el envío de datos durante una conexión.

servidor, como para la escritura de datos en él. Evidentemente el primer

paquete de tipo *ACL Data* llevará información sobre la operación a realizar, lectura o escritura. Mientras que el segundo paquete de tipo *ACL Data* transportará el dato leído (en el caso de una operación de lectura), o la información de confirmación de una lectura.

6.3.5.5. Desconexión

El proceso de desconexión se representa en la Figura 26; en la que evidentemente entre Host A y LL A solo se intercambian comando o eventos HCI.

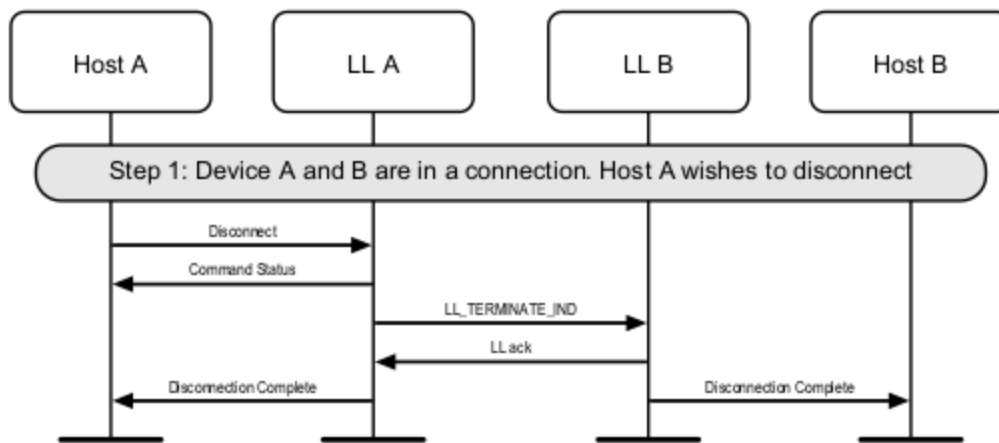


Figura 26: Secuencia de desconexión.

6.4. Protocolo de control y adaptación del enlace lógico, L2CAP

L2CAP, siglas de *Logical Link Control and Adaptation Protocol*, ofrece el servicio de un protocolo multiplexador que recibe paquetes de las capas superiores y los encapsula en el paquete BLE estándar para su envío, realizando la acción inversa en la recepción.

Además de esto, L2CAP también proporciona fragmentación y recombinación de paquetes de las capas superiores, lo que permite dividir paquetes demasiado grandes en paquetes con una carga de hasta 27 bytes (en los que va incluida la cabecera, lo que significa que el tamaño del campo de datos final es de 23 bytes), que es el tamaño máximo soportado por el protocolo BLE. En el momento de la recepción, este protocolo recibe múltiples paquetes fragmentados, los cuales combina para reconstruir el paquete ori-

ginal que proporciona a las capas superiores. Esto permite a las aplicaciones enviar y recibir paquetes de datos de hasta 64 KBytes de longitud.

En BLE, L2CAP tiene dos protocolos sobre él, cuyas rutas debe planificar: el *Attribute Protocol* (ATT), que forma las bases del intercambio de datos en las aplicaciones, y el *Security Manger Protocol* (SMP), que proporciona un framework para generar y distribuir claves de seguridad entre dispositivos.

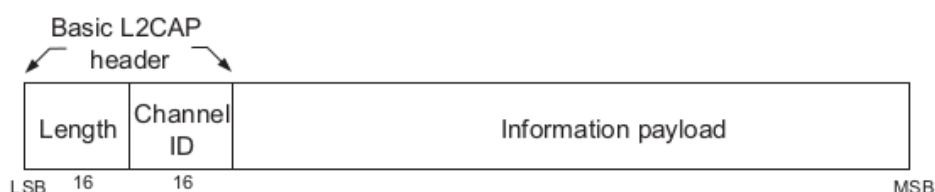


Figura 27: Formato de un paquete L2CAP.

El formato general de un paquete de la capa *L2CAP* es mostrado en la Figura 27. Siendo el significado de cada uno de los campos del paquete descrito a continuación:

Length (2 bytes) : indica la longitud del *payload* del paquete.

Channel ID (2 bytes) : identificador que indica cual es el canal de destino del paquete. Un canal no es sino un identificador lógico que especifica cual es el destino del mensaje. O dicho de otra forma, cual es el contenido del mensaje: si parámetros de configuración de la conexión a nivel L2CAP, o si es alguna de las capas superiores que se encapsulan dentro de este mensaje.

El valor que este campo puede tomar se especifica en la siguiente tabla:

Channel ID	Descripción
0x0004	El paquete contiene información del protocolo ATT.
0x0005	El paquete contiene parámetros de configuración del canal L2CAP.
0x0006	El paquete contiene información del protocolo SMP.

En el caso de ser un paquete L2CAP de configuración del canal L2CAP (Channel ID = 0x0005) el contenido del *payload* será el mostrado en la Figura 28.

El valor de cada uno de los campos del comando L2CAP es el que se lista a continuación:

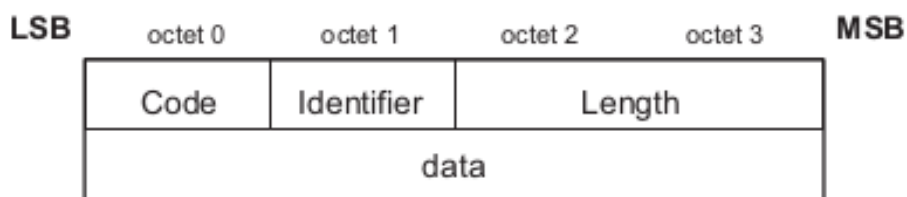


Figura 28: Formato de un comando L2CAP.

Code (1 byte) : indica el tipo de comando que es enviado en este mensaje. En el caso de que en un paquete sea recibido con un valor del campo Code no válido, automáticamente se responderá con un paquete del tipo Command Reject (Comando Rechazado).

Los valores válidos de este campo son:

Code	Descripción
0x00	Reservado
0x01	Command Reject
0x06	Disconnection Request
0x07	Disconnection Response
0x12	Connection Parameter Update Request
0x13	Connection Parameter Update Response
0x14	LE Credit Based Connection Request
0x15	LE Credit Based Connection Response
0x16	LE Flow Control Credit
0x17 - 0xFF	Reservado

El significado de cada uno de estos comandos se puede consultar en [17]-Parte A, sección 4.

Identifier (1 byte) : identificador que relaciona las peticiones (*request*) con las respuestas a las mismas (*response*). Se utiliza un identificador diferente para cada comando enviado por un canal L2CAP.

Length (2 bytes) : indica la longitud en bytes del campo Data incluido en el comando.

Data (Length bytes) : incluye los datos asociados con el comando enviado en este mensaje. Tanto el contenido, como su longitud total, dependerá del tipo de comando de que se trate (campo Code).

Como ya se ha indicado una descripción detallada de los comandos que se pueden intercambiar entre las capas *L2CAP* de dos entes, en una comunicación, se puede ver en [17]-Parte A, sección 4.

6.5. Protocolo de atributo, ATT

El ATT o *Attribute Protocol*, tal y como se detalla en la especificación Bluetooth [17] permite a un dispositivo, conocido como cliente, acceder a un conjunto de atributos, con sus valores asociados, expuestos por otro dispositivo denominado servidor. Estos atributos pueden ser leídos y escritos (operaciones *read* y *write*) por el cliente, mientras que el servidor tiene las opciones de indicar y notificar (acciones *indicate* y *notify*) los cambios producidos en los atributos al cliente. Cada uno de estos atributos tiene asociados los siguientes campos:

- Tipo de atributo, definido por un UUID, especifica lo que el atributo representa. El Bluetooth SIG ha definido diferentes tipos de atributos, que son utilizados por las capas más altas de la especificación Bluetooth. Además, existe la posibilidad de definir nuevos tipos de atributo.
- Identificador de atributo (*attribute handle*) de 16 bits de longitud, que identifica unívocamente a un atributo del servidor, permitiendo al cliente hacer referencia al mismo a la hora de leerlo o modificarlo. También permite a un cliente identificar un atributo que le sea notificado o indicado.
- Conjunto de permisos definidos por las diferentes capas superiores que hacen uso del atributo en cuestión y que no pueden ser accedidos por esta capa. Estos permisos pueden utilizarse para prevenir que las aplicaciones obtengan o alteren valores de atributos que otras aplicaciones estén utilizando.
- Valor del atributo en cuestión, se trata de un array de bytes, con una longitud que puede ser fija o variable, dependiendo del tipo de dato que se encuentre almacenado en dicho atributo. Si la longitud del dato es demasiado grande para transmitirse en un único paquete, el envío se realiza a través de múltiples PDUs, siendo los valores transmitidos invisibles para el protocolo ATT. Cuando se transmite el valor de un atributo, su longitud no se envía en ningún campo específico del PDU, siendo el único valor referente a ésta el correspondiente a la longitud del paquete completo en el campo destinado para ello. Esto implica que solamente podremos enviar el valor de un único atributo en un mensaje, a menos que tanto el servidor como el cliente conozcan las longitudes de los diferentes valores.

Las capas superiores pueden definir un atributo para que pueda ser leído o escrito (o ambas cosas), además de poder añadir restricciones de seguridad para hacerlo menos vulnerable.

Cuando un cliente desea leer o escribir el valor de un atributo de un servidor, envía una petición de lectura o escritura con el identificador del atributo en cuestión. A dichas peticiones, el servidor responderá con el valor del atributo o con un paquete de reconocimiento (ACK), respectivamente. En el caso de una operación de lectura, la labor de comprender el dato recibido recae en el cliente, basándose en el UUID del atributo. De igual manera, en una operación de escritura es el cliente el que debe proporcionar los datos compatibles con el tipo del atributo, pudiendo el servidor rechazar la escritura en caso contrario.

6.5.1. Operaciones del ATT

En el protocolo de atributo encontramos una serie de operaciones que nos permiten desempeñar diferentes acciones. Estas operaciones se muestran agrupadas a continuación [17]:

- *Error Handling*, utilizado por el servidor para responder a cualquier petición cuando ocurre un error. Incluye:
 - *Error Response*: Permite enviar una respuesta de error a una petición de un cliente, en lugar de la información pedida, si no es posible realizar la operación. Esta respuesta de error contiene información sobre el error producido.
- *Server Configuration*, nos permite configurar el propio protocolo ATT, e incluye únicamente:
 - *Exchange MTU Request/Response*: usadas por el cliente y el servidor para intercambiar la información correspondiente al tamaño de paquete máximo con el que pueden trabajar (conocido como *Maximum Transmission Units* o simplemente MTU).
- *Find Information*, permite a un cliente obtener información sobre los diferentes atributos del servidor.
 - *Find Information Request/Response*: Usadas para pedir y proporcionar una lista de atributos dentro de un rango particular de identificadores.

- *Find by Type Value Request/Response*: Permiten al cliente obtener información sobre un atributo en particular, utilizando su tipo y su valor.
- *Read Operations*, utilizadas por el cliente para obtener el valor de uno o más atributos. Estas operaciones incluyen:
 - *Read by Type Request/Response*: Obtiene el valor de uno o más atributos mediante su tipo.
 - *Read Request/Response*: Obtiene el valor de los atributos a través de su identificador.
 - *Read Blob Request/Response*: Obtiene parte del valor de un atributo largo utilizando su identificador.
 - *Read Multiple Request/Response*: Obtiene el valor de uno o más atributos utilizando múltiples identificadores.
 - *Read by Group Type Request/Response*: Obtiene el valor de uno o más atributos, utilizando un tipo de grupo de atributos. Estos tipos de grupos son definidos por capas superiores.
- *Write Operations*, que permiten al cliente seleccionar el valor de uno o más atributos. Estas operaciones incluyen:
 - *Write Request/Response*: Escribe un valor en un atributo determinado y espera una respuesta del servidor.
 - *Write Command*: Escribe un valor en un atributo determinado, pero sin esperar respuesta del servidor.
 - *Signed Write Command*: Similar a la operación anterior, pero en este caso el cliente debe utilizar una firma para verificar su identidad.
- *Queued Writes*, utilizadas por el cliente para escribir valores en los atributos con longitudes superiores a las admitidas en un paquete.
 - *Prepare Write Request/Response*: Encola una operación de escritura en el servidor para un atributo en particular, esperando una respuesta del servidor reconociendo la acción.
 - *Execute Write Request/Response*: Ejecuta todas las operaciones de escritura que se encuentren en la cola. Tras realizar la acción, el servidor notifica del éxito o fallo de la operación.

- *Server Initiated*, usadas por el servidor para hacer saber al cliente el nuevo valor de un atributo.
 - *Handle Value Notification*: Permite al servidor actualizar el valor de un atributo al cliente, de manera asíncrona y sin esperar respuesta o reconocimiento de este.
 - *Handle Value Indication/Confirmation*: Permite al servidor actualizar el valor de un atributo al cliente de manera asíncrona, pero a diferencia de la operación anterior, tras esta actualización el servidor espera un mensaje de confirmación o reconocimiento por parte del cliente.

6.5.2. Formato paquete ATT

El formato general de un paquete ATT se puede observar en la Figura 29.

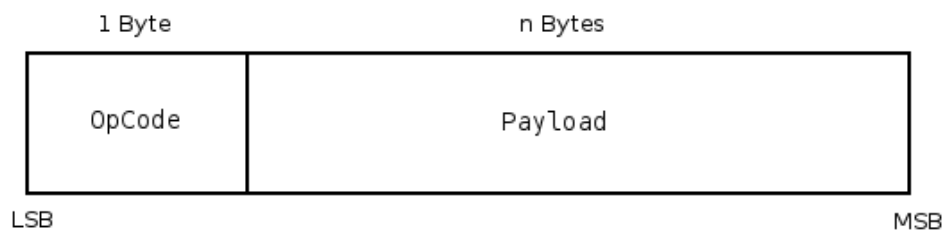


Figura 29: Formato general paquete ATT.

El valor de cada uno de los campos es el que se muestra a continuación:

OpCode (1 byte) : indica el tipo de operación asociado a el mensaje ATT. Los posibles valores son:

OpCode	Descripción
0x01	Error Response
0x02 - 0x03	Mensajes para intercambio de MTU
0x04 - 0x07	Mensajes de petición de información
0x08 - 0x11	Mensajes para lectura de valores del servidor
0x12 - 0x19	Mensajes para escritura de valores en el servidor
0x1B	Envío de una notificación por el servidor
0x1D - 0x1E	Envío de una indicación por el servidor

Payload (n bytes) : en este campo se incluyen los parámetros, o información asociada a cada una de las operaciones relacionadas con cada OpCode. El formato, y contenido de este campo dependerá del valor OpCode.

Una descripción detallada de los paquetes ATT y su formato puede consultarse en [17]-Parte F, sección 3.4.

6.6. Administrador de seguridad, SM

La especificación de la capa *Security Manager* [17] define métodos para el emparejamiento, generación y distribución de claves entre dispositivos, así como un protocolo y una serie de algoritmos de seguridad para dichos métodos. Todo ello proporciona a BLE la capacidad para generar e intercambiar claves de seguridad, permitiendo de esta manera una comunicación segura entre un par de dispositivos mediante un enlace cifrado. Esto consigue dar una mayor confianza a la identificación de un dispositivo remoto y la posibilidad de ocultar la dirección pública, si se desea, para evitar posibles ataques maliciosos.

Cada dispositivo es el encargado de generar y controlar las claves que él distribuye, sin que ningún otro dispositivo afecte en la generación de esas claves. La fuerza de la clave generada dependerá de los algoritmos implementados en el dispositivo.

Dentro del administrador de seguridad se definen dos roles para los dispositivos, en base al rol que tomen dentro de la conexión:

- *Initiator*: se corresponde siempre con el dispositivo maestro o central de la capa de enlace.
- *Responder*: se corresponde con el dispositivo esclavo o periférico de la capa de enlace.

Aunque el encargado de comenzar un procedimiento relacionado con el *Security Manager* siempre es el *Initiator*, el dispositivo *Responder* puede solicitar de manera asíncrona el inicio de cualquiera de estos procedimientos. No obstante, no hay garantías reales para el *Responder* de que el *Initiator* haya recibido su solicitud.

La arquitectura de esta capa se ha diseñado de tal manera que los requisitos de memoria y procesamiento sean mayores para el dispositivo que se encuentra en el rol de *Initiator* que para el *Responder*, ya que los dispositivos que desempeñan el rol de maestro suelen disponer de mayor capacidad de cómputo y memoria que aquellos que se encuentran en el rol de esclavo.

6.6.1. Procedimientos de seguridad

La capa *Security Manager* perteneciente al protocolo BLE proporciona soporte para realizar los siguientes procedimientos:

- *Pairing*: procedimiento que permite generar una clave de seguridad temporal para poder crear un enlace seguro y cifrado.
- *Bonding*: comienza realizando el procedimiento *pairing*, seguido por la generación e intercambio de claves de seguridad permanentes, las cuales son almacenadas en la memoria no volátil del dispositivo. Esto permite crear un enlace permanente entre dos dispositivos que posibilitará la creación de enlaces seguros en siguientes conexiones sin la necesidad de repetir el proceso completo de nuevo.
- *Encryption Re-establishment*: tras la realización del procedimiento de *bonding*, las claves deben encontrarse almacenadas en los dos dispositivos que se han conectado. Si las claves han sido almacenadas, este proceso define como usar dichas claves en conexiones venideras para reestablecer una conexión segura y cifrada sin la necesidad de realizar los procesos anteriores de nuevo.

Así pues, el proceso de *pairing* permite crear un enlace seguro que solamente se mantendrá durante el tiempo de vida de la conexión, a diferencia del procedimiento de *bonding*, que crea una asociación permanente (*bond*) y claves compartidas que se utilizarán en conexiones posteriores hasta que se decida eliminarlas. El procedimiento de *bonding* incluye el de *pairing* (que consta de dos fases), seguido de una fase adicional. Estas fases son, tal y como podemos ver en la Figura 30:

1. Intercambio de características del emparejamiento.
2. Generación de una clave de corto plazo.
3. Distribución de la clave específica de transporte.

6.6.2. Algoritmos de *pairing*

Un proceso de *pairing* requiere un intercambio de paquetes de SMP (*Security Manager Protocol*) para generar una clave temporal denominada STK (*Short Term Key*) a ambos lado de la conexión. El último paso de un procedimiento de *pairing* propiamente dicho es la encriptación del enlace con la clave STK ya generada. Durante el intercambio de paquetes, los pares de la conexión negocian cuál de los siguientes métodos utilizarán:

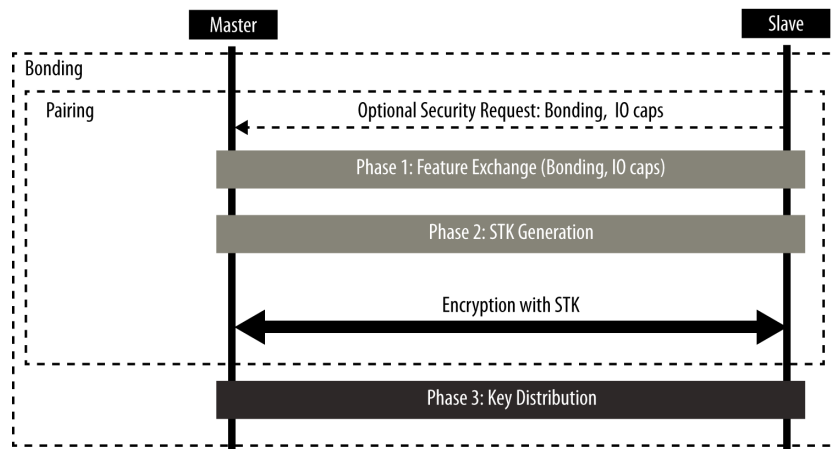


Figura 30: Secuencias de pairing y bonding [1].

- *Just Works*: la clave STK se genera en ambos dispositivos en base a paquetes intercambiados en texto plano. Esto hace que este método no proporcione seguridad contra ataques de tipo *man-in-the-middle* (MITM).
- *Passkey Entry*: uno de los dispositivos muestra una clave de seis dígitos (*passkey*) aleatoriamente generada, la cual deben conocer los dispositivos que deseen conectarse a él, por lo que el usuario debe administrársela. Otra variante de este método consiste en que el usuario introduce la clave en los dos dispositivos de la conexión (lo que puede ser muy útil si ninguno de los dispositivos es capaz de mostrar la clave mediante una pantalla o cualquier otro dispositivo de salida). Este método protege contra ataques de tipo MITM, por lo que se recomienda su uso cuando sea posible.
- *Out Of Band* (OOB): cuando se utiliza este método, se transfieren datos por medios diferentes a BLE (por ejemplo, otra tecnología inalámbrica como NFC). Esto proporciona protección contra ataques de tipo MITM en la interfaz de radio BLE.

La especificación de *Security Manager* incluye también tres tipos de mecanismos de seguridad que pueden utilizarse para ofrecer diferentes niveles de seguridad, bien en una conexión o bien durante el estado *advertising*:

- *Encryption*: se cifran todos los paquetes que se transmiten en una conexión establecida.

- *Privacy*: permite a un dispositivo en modo *advertiser* ocultar su dirección Bluetooth pública, utilizando en su lugar una dirección generada aleatoriamente, la cual pueden reconocer los dispositivos en modo *scanner* que tengan una conexión de tipo *bonding* con el dispositivo emisor.
- *Signing*: el dispositivo puede enviar un paquete firmado sin cifrar sobre una conexión ya establecida.

Cada uno de estos mecanismos puede utilizarse independientemente de los otros, existiendo la opción de utilizar más de uno de forma simultánea.

6.7. Perfil de atributo genérico, GATT

El perfil de atributo genérico, GATT o *Generic Attribute Profile* se encuentra construido sobre el ATT (ver página 40) y establece operaciones comunes y un framework para el transporte y almacenamiento de datos por este protocolo, definiendo en detalle la manera en la que dos dispositivos BLE transfieren datos. A diferencia de GAP (ver página 60), que define las interacciones de bajo nivel entre los dispositivos BLE, GATT se hace cargo únicamente de los procesos y formatos relacionados realmente con las transferencias de datos.

Además, GATT define una serie de objetos de datos genéricos que se pueden utilizar en una gran variedad de perfiles de aplicaciones, los cuales se conocen como perfiles basados en GATT, (*GATT-based profiles*). Estos perfiles mantienen la misma arquitectura cliente/servidor del protocolo ATT, pero encapsulan los datos en servicios.

Como ya se ha comentado, GATT hace uso del protocolo ATT para realizar el transporte e intercambio de datos entre los dispositivos, para lo que organiza estos datos de manera jerárquica en secciones denominadas servicios. A su vez, cada uno de estos servicios agrupa trozos de datos de usuarios conocidos como características. Así pues, cada una de las características contenidas en un servicio contiene la unión de diferentes datos de usuario con meta-datos que los describen. En el siguiente apartado se describen con mayor detalle los roles que desempeñan los dispositivos en este protocolo; tras esto, se definen los atributos; y por último, se especifica la jerarquía de datos establecida en esta capa de BLE.

6.7.1. Roles

La especificación de BLE [17] define dos roles para el perfil GATT, los cuales son utilizados también por el protocolo ATT:

- Cliente: dispositivo que inicia los comandos y peticiones al servidor, y que recibe las repuestas, indicaciones y notificaciones del mismo.
- Servidor: dispositivo que acepta comandos y peticiones entrantes desde un dispositivo cliente, al que envía respuestas, indicaciones y notificaciones.

Estos roles no son fijos para un dispositivo en una conexión, sino que serán asignados cuando un dispositivo inicie un determinado proceso, y se liberarán cuando dicho proceso finalice. Además, existe la posibilidad de que un dispositivo tenga asignados los dos roles simultáneamente.

El perfil GATT se ha diseñado para ser utilizado por una aplicación o por otro perfil, permitiendo a un cliente comunicarse con un servidor. Un servidor puede contener múltiples atributos, y el perfil GATT define como utilizar el protocolo de atributo para descubrir, leer o escribir esos atributos, así como para obtener indicaciones de los mismos.

6.7.2. Atributos

Los atributos son la entidad de datos más pequeña definida por GATT y ATT, y pueden contener información sobre la estructura de los diferentes atributos almacenados en el servidor. Tanto GATT como ATT trabajan únicamente con atributos, pero para permitir la conexión entre servidores y clientes la información se debe organizar de manera jerárquica, como se verá en esta sección.

Conceptualmente hablando, los atributos siempre se encuentran almacenados en el servidor, y son accedidos y modificados por el cliente. Para facilitar su uso y manejo, cada atributo contiene información sobre él mismo y la información que contiene. Dicha información se encuentra almacenada en los campos del atributo detallados a continuación y mostrados en la Figura 31:

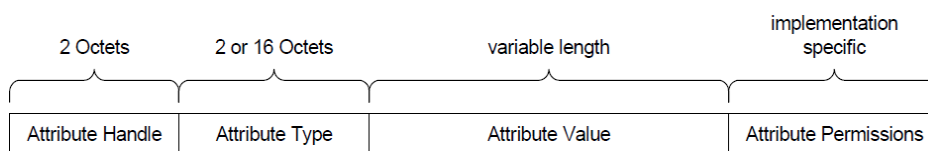


Figura 31: Representación de un atributo [17].

- *Handle*: se trata de un identificador único de 16 bits que permite direccionar cada atributo. Al tener una longitud de 2 bytes, la cantidad total de atributos disponibles en un servidor puede ser elevada

(65535), pero en la práctica el límite suele establecerlo la memoria, por lo que el número máximo suele ser unas pocas docenas. Estos identificadores no tienen por qué ser secuenciales, por lo que un cliente deberá utilizar el servicio *Discovery* para obtener los identificadores de los atributos en los que esté interesado.

- **Type:** indica el tipo del valor del atributo mediante un UUID (*Universally Unique Identifier*) de 2, 4 o 16 bytes (los tamaños más pequeños hacen uso de la parte del UUID definida para BLE). Este es el tipo utilizado por el servicio *Discovery* para proporcionar el identificador del atributo. BLE define diversos tipos de atributos de manera genérica, aunque también permite a los usuarios definir los suyos propios si lo desean. Una lista de los UUID estandarizados pueden consultarse en [21].
- **Value:** este atributo contiene el dato real del atributo. No existen restricciones sobre el tipo de dato que puede contener, aunque su longitud máxima es de 512 bytes. Se debe tener cuidado con lo que se almacena en este campo, ya que es el único donde el cliente puede leer y escribir libremente (con los permisos necesarios), puesto que el resto de campos no pueden ser modificados ni accedidos directamente por el cliente.
- **Permissions:** los permisos son meta-datos que especifican qué operaciones ATT pueden ejecutarse en cada atributo en particular y con qué requisitos de seguridad pueden hacerlo. ATT y GATT definen los siguientes permisos:
 - Permisos de acceso: determina si un cliente puede leer o escribir en el valor de un atributo. Con estos permisos puede permitirse a un cliente leer o escribir el valor del atributo únicamente, así como realizar ambas acciones o ninguna.
 - Cifrado: indica si es necesario un cierto nivel de cifrado para permitir al cliente acceder al atributo. Las opciones disponibles en este caso son: no se requiere de cifrado, se requiere de cifrado sin necesidad de utilizar claves autenticadas y se requiere de cifrado con claves autenticadas.
 - Autorización: permite seleccionar al servidor si se requiere o no que el dispositivo cliente esté autorizado para acceder al valor del atributo.

6.7.3. Jerarquía de datos en GATT

Los perfiles GATT son los elementos superiores en la jerarquía de datos de la capa GATT. Estos perfiles están diseñados para ser utilizados por una aplicación o por otro perfil, y permiten a un cliente comunicarse con un servidor. Como ya se ha comentado, un servidor contiene un determinado número de atributos, y el perfil GATT define cómo utilizar el protocolo de atributo para trabajar con ellos.

Cada perfil GATT especifica la estructura en la que sus datos se intercambian. Esta estructura define principalmente dos elementos utilizados en los perfiles (niveles superiores de la jerarquía): servicios y características, los cuales se encuentran almacenados en atributos. Así pues, un perfil está compuesto de uno o más servicios, necesarios para desempeñar un caso de uso. De igual manera, cada uno de estos servicios se compone de características o referencias a otros servicios. Una característica contiene un valor y opcionalmente puede contener información sobre dicho valor. Dicho de otra manera, los servicios, características y sus componentes contienen los datos correspondientes al perfil al que pertenecen, y todos ellos se almacenan en atributos. Esta descripción se encuentra representada en la Figura 32.

A continuación, se realiza una descripción más detallada de cada uno de los elementos existentes en la jerarquía de datos utilizada por la capa GATT que hemos comentado anteriormente:

6.7.3.1. Servicio

Se trata de una colección de datos, con sus comportamientos asociados, que permiten llevar a cabo una determinada función. La definición de un servicio puede contener referencias a otros servicios, así como características obligatorias y opcionales.

Visto de otra manera, un servicio puede definirse como una agrupación de atributos relacionados conceptualmente en una sección común dentro del conjunto de atributos del servidor. De hecho, los atributos del servidor son realmente una sucesión de definiciones de servicios, las cuales comienzan con un atributo que marca el inicio de un servicio (conocido como declaración de servicio).

Podemos distinguir dos tipos de servicios: primarios y secundarios. El primero de ellos se trata de un servicio que proporciona funcionalidad al dispositivo, además de poder ser incluido por otros servicios. En cambio, un servicio secundario está diseñado únicamente para ser incluido en un servicio primario, en otro servicio secundario o en una capa superior de la especificación.

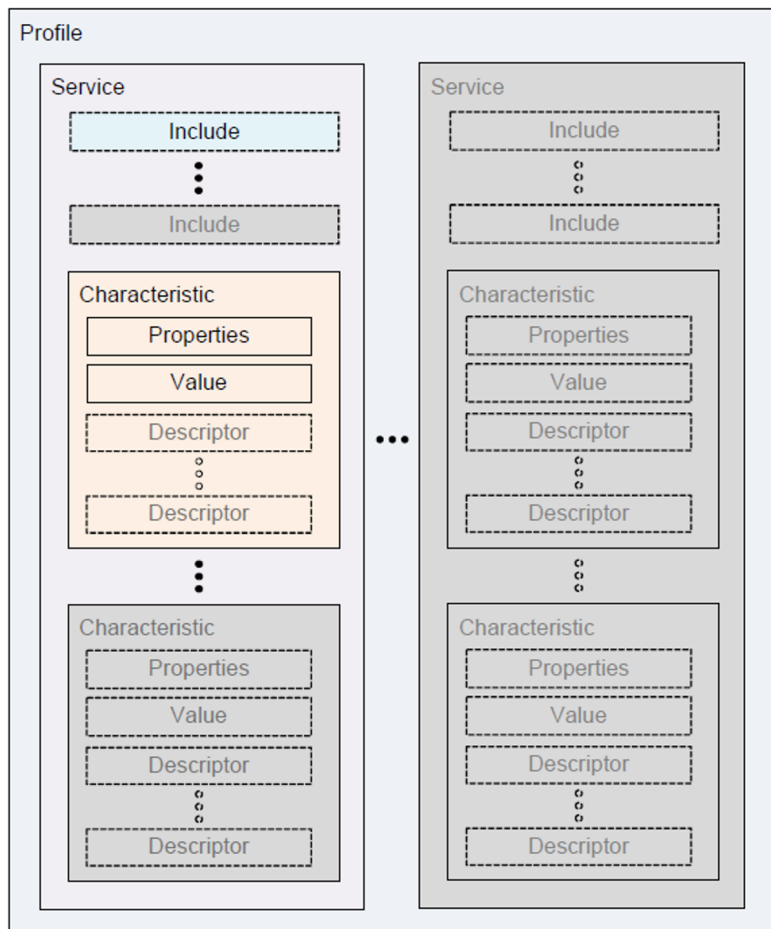


Figura 32: Jerarquía de perfil en GATT [17].

El formato de estos atributos se detalla en la especificación BLE [17]-**Part G, sección 3.1**, y se muestra en la Tabla 6, donde se indican los valores que toman los campos de un atributo (ver Figura 31).

Tabla 6: Valores de los campos del atributo asociado a la definición de un servicio.

Handle	Tipo Atributo	Valor del Atributo	Permisos acceso
0xXXXX	0x2800 - UUID servicio primario	16-bit UUID o 128-bit UUID	Solo lectura, No Autorización, No Autenticación
	0x2801 - UUID Indica servicio secundario		

6.7.3.2. Inclusión de servicios Proporciona un método para especificar la definición de un servicio ya existente en la definición de otro nuevo servicio. Para ello se utiliza una definición *include* al comienzo de la defini-

ción del nuevo servicio. Esto hace que se incluya la definición completa del servicio usado como parte de la definición del nuevo servicio, incluyendo las características y servicios que incluye a su vez el primero a su vez.

El valor de los campos que definen este atributo se muestra en la Tabla 7.

Tabla 7: Valores de los campos del atributo asociado a la inclusión de un servicio.

Handle	Tipo Atributo	Valor del Atributo			Permisos acceso
0xXXXX	0x2802 - UUID para servicio incluido	Handle del servicio	Handle final del grupo	UUID del servicio	Read only, no autenticación, no autorización

El *UUID del servicio* solo se define cuando este es un UUID de 16-bit.

6.7.3.3. Característica

Una característica se ha de entender como un dato del servicio que puede ser accedido por un usuario de dicho servicio. Debido a la estructura jerárquica del GATT una característica debe ser definida dentro de la definición de un servicio. Es decir, que tras la especificación de los atributos asociados a un servicio 6.7.3.1, se especificarán las declaraciones de todas las características asociadas a dicho servicio.

La definición de una característica acabará, cuando comienza la declaración de otra característica del mismo servicio, o cuando comienza la definición o declaración de otro servicio.

Una definición de una característica debe contener al menos una **declaración de la característica** y la **declaración del valor** asociado a la característica. Adicionalmente, la definición de la característica puede incluir una declaración de descriptores, tras la declaración del valor, que proporcionan al usuario meta-datos sobre la característica en sí, y su valor asociado, como pueden ser: un nombre o descripción en forma de texto, formato de representación del valor asociado a la característica, etc. Una descripción detallada de la definición y manejo de las características de un servicio se puede ver en [17]-Part G, sección 3.3.

A continuación se dará una breve descripción de como se realiza la declaración de una características y de su valor asociado.

Declaración de una característica

La declaración de una característica se realiza mediante la definición de un atributo con un valor del *Tipo de Atributo* igual a 0x2803. Especificándose en el campo de *Valor del Atributo*, las propiedades de dicho atributo, es

decir, si se puede leer, escribir, etc., un *handle* para ese dato de la característica, que nos permita acceder al valor de dicho dato de la característica de manera directa, y el UUID de la característica que se está declarando. Como se puede ver en la Tabla 8

Tabla 8: Valores de los campos del atributo asociado a la declaración de una característica.

Handle	Tipo Atributo	Valor del Atributo			Permisos acceso
0xXXXX	0x2803 - UUID para Característica	Propiedades (1 byte)	Handle Dato (2 bytes)	UUID Característica (2 ó 16 bytes)	Read only, no autenticación, no autorización

Donde el campo **Valor del Atributo - Propiedades** se interpreta como un campo de flags, cuyos valores se interpretan según la Tabla 9

Tabla 9: Descripción del campo Propiedades, dentro del Valor del Atributo de una declaración de una característica.

Propiedad	Valor (1 byte)	Descripción
<i>Broadcast</i>	0x01	El dato asociado a esta característica puede ser enviado por broadcast durante el descubrimiento de los servicios.
<i>Read</i>	0x02	El dato puede ser leído directamente por un cliente.
<i>Write Without Response</i>	0x04	El dato puede ser escrito sin confirmación de escritura.
<i>Write</i>	0x08	El dato puede ser escrito pero con confirmación de escritura.
<i>Notify</i>	0x10	El dato se enviará a cualquier cliente que esté conectado cuando sea modificado en el servidor. La notificación se realiza SIN confirmación.
<i>Indicate</i>	0x20	El dato CON confirmación.

En el caso del campo **Valor del Atributo - UUID Característica** de la Tabla 8, un lista de valores estandarizados que puede tomar dicho campo en el caso de utilizar 16-bit Bluetooth UUID se puede ver en [22]. Esta estandarización permite al cliente determinar, a través de su UUID, el propiedades de la característica como: tipo de dato almacenado, longitud del mismo, etc.

Declaración del Valor

La definición del atributo de la *Declaración del Valor de la Característica* contiene el valor o dato asociado a la característica en sí. Su declaración se realiza inmediatamente después de la declaración de la característica a la que está asociado.

El contenido del atributo utilizado para la declaración de dicho valor tiene el formato y valor de campos mostrado en la Tabla 10

Tabla 10: Valores de los campos del atributo asociado a la declaración del Valor una característica.

Handle	Tipo Atributo	Valor del Atributo	Permisos acceso
0xXXXX	UUID Característica: 16-bit Bluetooth UUID, o 128-bit Bluetooth UUID	Valor del dato	Especificado por la declaración de la característica.

Descriptores de características

Estos descriptores son mayormente utilizados para proporcionar al cliente meta-datos de la característica a la que pertenecen. Se encuentran ubicados dentro de la definición de la característica, y tras el atributo que contiene el valor de la misma. Estos descriptores se almacenan en un único atributo, y podemos distinguir dos tipos diferentes:

- Descriptores definidos de GATT: añaden información correspondiente a la característica a la que pertenecen. Una lista de los mismo puede verse en [23]. La estandarización de estos descriptores permite, utilizando solo su UUID, saber información sobre las propiedades de dicho descriptor: valores máximo y mínimo del dato asociado a este descriptor, tipo de dato, precisión del mismo, etc.
- Descriptores definidos por los vendedores o perfiles: tanto si se trata de un perfil especificado por el SIG como por un vendedor en particular, estos descriptores incluyen información adicional sobre el valor de la característica en cuestión.

6.7.4. Descubrimiento de servicios y características

En esta sección se tratarán los procedimientos que se describen en el estándar ([17]-**Parte G, sección 4.4 y 4.6** para que un cliente pueda descubrir tanto los servicios como las características que un servidor le puede ofrecer.

Para realizar estas tareas, la capa GATT debe de utilizar funcionalidades de la capa ATT, una descripción del formato de los mensajes utilizados en esta capa ATT para la implementación de sus métodos se puede ver en [17]-**Parte F, sección 3.4**.

6.7.4.1. Descubrimiento de servicios

Los servicios que están incluidos en un servidor pueden ser descubiertos por un cliente utilizando uno de los dos procedimientos que a continuación se detallan:

Descubrir todos los Servicios Primarios

Para poder descubrir todos los servicios primarios contenidos en un servidor, el cliente deberá de llamar a la operación del *ATT Read by Group Type Request* especificando los siguientes parámetros:

- *Attribute Type*: tomará el valor 0x2800 <<Primary Service>>.
- *Starting Handle*: 0x0001,
- *End Handle*: 0xFFFF.

A esta operación de petición por parte del cliente, el servidor podrá contestar con dos posibles respuestas:

- *Error Response*: esta se produce cuando el servidor detecta algún error, por que se solicita información de algún servicio primario con un *handle* inexistente.
- *Read by Group Type Response*: esta respuesta envía al cliente una lista de tuplas (*Attribute Handle*, *End Group Handle*, *Attribute Value* correspondiente a cada servicio soportado por el servidor. La interpretación de la tupla es:
 - *Attribute Handle*: *handle* para acceder al servicio en cuestión.
 - *End Group Handle*: último *handle* perteneciente a los atributos asociados al servicio.
 - *Attribute Value*: UUID del servicio. Una lista de los UUID de servicio estandarizados puede encontrarse en [24].

Para el descubrimiento de todos los servicios de un servidor, el envío de la petición *Read by Group Type Request* puede repetirse para cada uno de los servicios primarios del servidor. Para ello se repite la petición anterior, pero ahora se utiliza un valor de *Starting Handle* incrementado respecto del valor *End Group Handle* de la respuesta *Read by Group Type Response* anterior. El proceso acabará cuando:

- el valor *End Group Handle* del último *Read by Group Type Response* coincida con el valor del parámetro *End Handle* de la petición anterior, es decir, sea 0xFFFF, o
- se reciba un mensaje *Error Response* con un *Error Code* igual a <<*Attribute Not Found*>>. Que indica que el servidor no tiene más servicios primarios que ofrecer.

El procedimiento de descubrimiento anterior, devuelve la información para acceder a todos los servicios ofertados por el servidor. Pero si lo que queremos es **obtener la información de acceso a un determinado servicio, identificándolo mediante su UUID**, deberemos utilizar la función de la capa ATT *Find By Type Value Request*, inicializando el parámetro *Attribute Type* de la misma al UUID del servicio del que queremos obtener la información de acceso (una lista de dichos UUID se puede obtener de [24]). Y poniendo los parámetros *Starting Handle*, y *End Handle* a 0x0001 y 0xFFFF, respectivamente.

A esta petición el servidor le puede responder con dos tipos de mensajes:

- *Error Response*: generado en la misma situación que en caso anterior.
- *Find by Type Value Response*: que consiste en una lista de rangos de *Attribute Handle*, en las que se indica el *handle* inicial y final de la definición del servicio. De manera similar a como se realizaba en el procedimiento anterior.

Como antes las peticiones y respuestas se pueden suceder hasta que se reciben todos los rangos de *handle* asociados con el servicio.

6.7.4.2. Descubrimiento de características

Al igual que en el caso de los servicios, las características se pueden descubrir mediante la utilización de procedimientos implementados en la capa ATT. Existen dos formas de descubrir la información de las características asociadas a un servicio

Descubrir todas las características de un servicio

Para ello se utiliza la operación de la capa ATT *Read by Type Request* con los siguientes parámetros de operación:

- *Attribute Type*: 0x2803
- *Starting Handle*: será el *Starting Handle* obtenido cuando se descubrió el servicio al que la característica está asociado.
- *Ending Handle*: es el valor de *Ending Handle* obtenido al descubrir el servicio al que está asociado la característica.

A la recepción de este mensaje el servidor puede responder con dos respuestas distintas:

- *Error Response*: esta se produce cuando el servidor detecta algún error.

- *Read by Type Response*: esta respuesta envía al cliente una lista de pares (*Attribute Handle*, *Attribute Value* correspondiente a cada característica incluida en la definición del servicio. La interpretación de cada pareja de datos es la siguiente:
 - *Attribute Handle*: *handle* para acceder a la declaración de la característica.
 - *Attribute Value*: está formado por los parámetros incluidos en la declaración de la característica (ver página 52): *Propiedades*, *Handle* y *UUID*

Como en el caso del descubrimiento de servicios, el intercambio de operaciones *Read by Type Request - Response* se puede repetir hasta que el total de características hayan sido descubiertas. Para ello, y como en el caso de los servicios, cada *Read by Type Request* tendrá un valor de *Starting Handle* igual al valor del *Attribute Handle* del mensaje *Read by Type Response* más uno. El intercambio de mensajes finaliza cuando el *Read by Type Response* incluye un *Attribute Handle* igual al valor del *Ending Handle* del *Read by Type Request*. O cuando se recibe una respuesta *Error Response* con un *Error Code* igual a `<<Attribute Not Found>>`.

Como en el caso de los servicios, también se implementa un mecanismo para **acceder a la información de una característica si se conoce el UUID** asociado a ella. Una lista de los UUID de 16-bit para las características, definidos por el estándar, puede verse en [22].

En este caso se empleará el mismo método que en el caso anterior, con los mismos valores de parámetros. La única diferencia es que se busca entre las respuestas *Read by Type Response* aquella cuyo campo *Attribute Value - UUID* coincida con el UUID de la característica que estamos buscando.

6.7.5. Lectura de Valores de Características

Los mecanismo para la lectura de los valores asociados a un característica de un servicio viene especificados por el estándar en [17]-**Parte G, sección 4.8**. En todos los caso se utilizan métodos de la capa ATT para acceder a dichos valores. Aquí comentaremos los más comunes, dirigiendo al lector a la especificación del estándar para una mayor información.

6.7.5.1. Lectura de Valor Para una lectura del valor de una característica usando el *handle* asociado a dicho valor, por parte de un cliente, utilizaremos el método del protocolo ATT *Read Request*. El único parámetro

que se ha de proporcionar al método es el *Characteristic Value Handle*, que es el *handle* del valor que queremos obtener.

A esta petición de lectura el servidor puede contestar con dos tipos de mensajes:

- *Error Response*: esta se produce cuando el servidor detecta algún error.
- *Read Reponse*: que contendrá únicamente el valor del dato asociado al *handle* especificado en la petición. El valor devuelto estará en formato hexadecimal, y su interpretación dependerá de las especificaciones del valor de la característica. Dicha interpretación podrá deducirse del UUID asociado a la característica, si éste estuviera estandarizado (ver [22]). Y en el caso de ser una característica especificada por el fabricante, sería éste el que nos debería proporcionar dicha información.

6.7.5.2. Lectura de Valor usando UUID En el caso de no conocer el *handle* asociado al valor de la característica, pero sí se conoce su UUID, podríamos utilizar el método *ATT Read by Type Request* para intentar acceder a él.

Para ello el mensaje *Read by Type Request* enviado por el cliente, incluirá los siguientes parámetros:

- *Attribute Type*: es inicializado al UUID de la característica cuyo valor queremos obtener.
- *Starting Handle*: será el *Starting Handle* obtenido cuando se descubrió la característica, e indica el *handle* inicial a partir del cual iniciar la lectura de valores.
- *Ending Handle*: es el valor de *Ending Handle* obtenido al descubrir la característica en cuestión, e indica el *handle* final, que una vez alcanzado marcará la finalización de la lectura de valores.

A este tipo de peticiones el servidor responderá con uno de los dos siguientes mensajes:

- *Error Response*: esta se produce cuando el servidor detecta algún error.
- *Read by Type Reponse*: que incluirá una lista de parejas *Attribute Handle*, *Attribute Value*, correspondiente a cada uno de los valores de las características incluidas en el rango de *handles* indicado en la petición.

6.7.6. Escritura de Valores de Características

El mecanismo de escritura de valores de una característica es muy similar al de lectura, y el estándar lo especifica en [17]-**Parte G, sección 4.9**. En cualquier caso, la escritura también utiliza mecanismos de la capa ATT para realizar dichas operaciones. En esta sección describiremos los tipos de escritura más comunes, aunque en el estándar se pueden encontrar una mayor variedad.

6.7.6.1. Escritura de un valor SIN confirmación Este procedimiento de escritura de valores de una característica, permite al cliente modificar el valor de una característica en el servidor, sin esperar a una confirmación del mismo sobre como se ha realizado la operación.

En este caso se utiliza el método *Write Command* el cual toma como parámetros los siguientes:

- *Attribute Handle*: es el *handle* del valor de la característica a modificar.
- *Attribute Value*: el nuevo dato, en formato hexadecimal, que se le quiere proporcionar al valor de la característica. La interpretación de este dato dependerá de las especificaciones del valor de la característica. Podría ser deducido del UUID asociado a la característica, si éste estuviera estandarizado (ver [22]). Y en el caso de ser una característica especificada por el fabricante, sería éste el que nos debería proporcionar dicha información.

Tras la recepción de esta petición, el servidor tan solo modificará el valor de la característica asociado al *handle* de la petición, con el valor *Attribute Value*. No proporcionando ningún tipo de información de realimentación al cliente.

6.7.6.2. Escritura de un valor CON confirmación El procedimiento de escritura en este caso es similar al del caso anterior. Solo que ahora utilizamos el método *Write Request* de la capa ATT, para solicitar la escritura al servidor. Los parámetros de este método son idénticos al caso anterior:

- *Attribute Handle*: es el *handle* del valor de la característica a modificar.
- *Attribute Value*: el nuevo dato que se le quiere proporcionar al valor de la característica.

A esta petición el servidor contestará, una vez realizada la modificación del contenido del valor de la característica, con una de las siguientes respuestas:

- *Write Response*: si la escritura tuvo éxito, o
- *Error Response*: si la escritura no tuvo éxito.

Existen muchos más mecanismos de acceso tanto a los servicios, como a sus características y los valores de las mismas. Como ya se ha dicho, una descripción detallada de los mismos se puede encontrar en [17]-Part G, sección 4.

6.8. Perfil de acceso genérico, GAP

El perfil de acceso genérico o GAP permite a un dispositivo BLE trabajar de manera conjunta con otros dispositivos que hagan uso del mismo protocolo. Para ello, proporciona un framework que todo dispositivo BLE debe seguir para permitir al resto de dispositivos realizar correctamente las operaciones especificadas en el estándar.

Así pues, este perfil define los procedimientos relacionados con el descubrimiento de dispositivos BLE y con aspectos de administración de los enlaces de las conexiones entre dispositivos de este estándar. También define procedimientos relacionados con el uso de diferentes niveles de seguridad, además de los formatos de datos requeridos para proporcionar accesibilidad a los parámetros en el nivel de la interfaz de usuario. A continuación se detallan los conceptos más importantes relacionados con este perfil:

6.8.1. Roles

La especificación de BLE [17] define cuatro roles en el perfil GAP para los dispositivos que siguen este protocolo. Estos roles son:

- *Broadcaster*: un dispositivo en este rol envía mensajes de tipo *advertisement* (sección 6.2.4.2), y es conocido como *Broadcaster*.
- *Observer*: un dispositivo operando en este rol recibe eventos de tipo *advertisement*, y se conoce como *Observer*.
- *Peripheral*: cuando un dispositivo acepta el establecimiento de una conexión BLE a nivel de la capa física mediante cualquier procedimiento entra en el rol de *Peripheral*, que coincide con el rol de esclavo dentro del estado *Connection* de la capa de enlace (ver sección 6.2). Un dispositivo en este rol se conoce como *Peripheral*.

- *Central*: este rol se asigna a los dispositivos que inician el establecimiento de una conexión física, y coincide con el rol de maestro del estado *Connection* de la capa de enlace (sección 6.2). Un dispositivo en este rol se conoce como *Central*.

6.8.2. Modos y procedimientos

A lo largo de esta sección se muestran los modos y procedimientos disponibles en el perfil GAP, definidos en la especificación BLE, divididos en grupos según su funcionalidad:

- Modo *Broadcast* y procedimiento *Observation*. Este modo y este procedimiento permiten comunicar dos dispositivos de manera unidireccional, sin la necesidad de establecer una conexión, utilizando en su lugar mensajes de *advertisement*, tal y como se detallan a continuación:
 - Modo *Broadcast*: permite a un dispositivo enviar datos sin conectarse a otro, utilizando los mensajes de tipo *advertisement*.
 - Procedimiento *Observation*: Proporciona a los dispositivos una forma de recibir los datos enviados mediante un mensaje de tipo *advertisement*, lo que permite recibir los datos enviados por otro dispositivo sin establecer una conexión.
- Modos y procedimientos *Discovery*. Todos los dispositivos deben encontrarse en uno de los modos aquí incluidos (bien para no anunciarse a otros dispositivos o bien para hacerlo de una determinada manera). Además, aquí también se incluyen procedimientos para que los dispositivos puedan recoger información de aquellos que se encuentran anunciándose mediante algunos de los modos disponibles:
 - Modo *Non-Discoverable*: este modo de configuración permite a un dispositivo no ser descubierto mediante ningún procedimiento, ya sea de tipo *Limited Discovery* o *General Discovery*.
 - Modo *Limited Discoverable*: los dispositivos configurados en este modo permiten ser descubiertos durante un periodo de tiempo limitado por dispositivos que utilicen los procedimientos *Limited Discovery* o *General Discovery*.
 - Modo *General Discoverable*: permite a los dispositivos ser descubiertos por otros dispositivos que hagan uso del procedimiento

General Discovery. A diferencia del modo anterior, éste suele utilizarse para hacer que un dispositivo pueda ser descubierto por otros durante largos periodos de tiempo.

- Procedimiento *Limited Discovery*: proporciona al dispositivo que hace uso de él la dirección del dispositivo y los datos de los paquetes de tipo *advertising* y *scan response*, únicamente de los dispositivos que se encuentren en el modo *Limited Discovery*.
 - Procedimiento *General Discovery*: este procedimiento es prácticamente similar al anterior, pero a diferencia de éste, proporciona la información correspondiente a los dispositivos que se encuentran tanto en modo *Limited Discoverable* como en modo *General Discoverable*.
 - Procedimiento *Name Discovery*: permite obtener el nombre de un dispositivo BLE que permita establecer conexiones.
- Modos y procedimientos *Connection*. estos modos y procedimientos permiten conectar un dispositivo con otro, así como modificar los parámetros de conexiones ya establecidas:
- Modo *Non-Connectable*: un dispositivo que se encuentre en este modo no permitirá establecer una conexión con ningún otro.
 - Modo *Directed Connectable*: los dispositivos configurados en este modo aceptarán una petición de conexión, pero únicamente de una serie de dispositivos cuyas direcciones ya conozca.
 - Modo *Undirected Connectable*: este modo es muy similar al anterior, salvo porque no se realiza ningún filtrado de dispositivos. Cualquier dispositivo puede establecer una conexión con el dispositivo que se encuentra en este modo, mediante el envío de una petición de conexión.
 - Procedimiento *Auto Connection Establishment*: este procedimiento permite al Host configurar el controlador para que establezca una conexión automáticamente con uno o más dispositivos que se encuentren en modo *Directed Connectable* o *Undirected Connectable*. A la hora de establecer las conexiones, el controlador tiene en cuenta la *White List* (ver página 19) para filtrar las direcciones de los dispositivos con los que intentará establecer una conexión.
 - Procedimiento *General Connection Establishment*: permite al Host establecer una conexión con un conjunto de dispositivos conoci-

dos que se encuentren en los modos *Directed Connectable* o *Undirected Connectable*.

- Procedimiento *Selective Connection Establishment*: este procedimiento permite al Host establecer una conexión, utilizando los parámetros deseados, con un conjunto de dispositivos que se encuentren en la *White List*.
 - Procedimiento *Direct Connection Establishment*: permite al Host establecer una conexión, con los parámetros de configuración seleccionados, con un dispositivo específico.
 - Procedimiento *Connection Parameter Update*: permite, tanto a un nodo periférico como a un nodo central, modificar los parámetros de conexión correspondientes a la capa de enlace de una conexión ya establecida.
 - Procedimiento *Terminate Connection*: este procedimiento da la opción a un Host de terminar una conexión establecida con otro dispositivo.
- Modos y procedimientos *Bonding*. Los modos y procedimientos aquí agrupados permiten a dos dispositivos que se encuentren conectados intercambiar y almacenar información relacionada con la seguridad y la identidad, permitiendo la creación de una conexión confiable. Estos modos y procedimientos son:
- Modo *Non-Bondable*: un dispositivo en este modo no permite establecer enlaces de tipo *bond* con otros dispositivos.
 - Modo *Bondable*: un dispositivo en este modo permite crear un enlace *bond* con otro dispositivo que también se encuentre en este modo.
 - Procedimiento *Bonding*: este procedimiento permite crear un enlace *bond* entre dos dispositivos, el cual puede ser necesario para el uso de determinados servicios.

6.8.3. Modos de seguridad

Los requisitos de seguridad de un dispositivo en particular o de determinados servicios ofrecidos por el mismo se expresan en términos de modo y nivel de seguridad. Cada servicio o petición de servicio puede tener asociados unos determinados requisitos de seguridad, al igual que sucede con los diferentes dispositivos.

BLE dispone de dos modos de seguridad, denominados modo 1 y modo 2. Cada conexión física entre dos dispositivos solamente puede operar en uno de los dos modos. Estos modos se describen a continuación, divididos en los diferentes niveles existentes:

- Modo de seguridad 1:
 1. Sin seguridad (ni autenticación ni encriptación).
 2. Conexión sin autenticación, pero cifrada. Este nivel cubre los requisitos del nivel anterior, además del nivel 1 del modo 2.
 3. Conexión autenticada y cifrada. Este nivel cubre los dos anteriores, además de los dos niveles del modo 2.
- Modo de seguridad 2:
 1. Conexión sin autenticación y firmada.
 2. Conexión con autenticación y firmada.

El modo 2 de seguridad únicamente debe utilizarse para conexiones que requieran datos firmados.

En el caso de que haya servicios que requieran de los dos modos de seguridad descritos anteriormente, se deberá utilizar un nivel de seguridad del modo 1 que englobe los niveles de los modos requeridos.

6.8.4. Procedimientos de seguridad

Además de los procedimientos que hemos visto anteriormente, el perfil de acceso genérico nos proporciona una serie de procedimientos relacionados con la seguridad. También es necesario mencionar que los procedimientos y modos *Bonding* podrían haberse incluido aquí sin ningún problema, ya que se encuentran relacionados con la seguridad, pero se ha optado por mantener la estructura seguida en el estándar.

Por otro lado, además de los procedimientos y modos de seguridad *Bond*, GAP nos proporciona los procedimientos de autenticación, autorización y encriptación:

- Procedimiento de autenticación: describe cómo se establece la seguridad que requiere un determinado servicio, tanto en el dispositivo que envía la petición del servicio como en el dispositivo que la recibe. Este procedimiento cubre los dos modos de seguridad comentados anteriormente, y únicamente debe realizarse cuando la conexión se haya establecido.

- Procedimiento de autorización: un determinado servicio puede requerir autorización para permitir el acceso a un dispositivo. La autorización es la confirmación del usuario que permite continuar con el servicio o proceso. Se debe tener en cuenta que la autenticación no siempre supone la autorización, ya que es la confirmación del usuario tras una correcta autenticación la que proporciona la autorización.
- Procedimiento de encriptación: los nodos central y periférico de una conexión pueden hacer uso de este procedimiento, mediante el uso de *Encryption Session Setup* y *Slave Initiated Security Request* respectivamente, para proporcionar integridad y confidencialidad al intercambio de datos.
- Firma de datos: la firma de datos se utiliza para intercambiar datos de manera autenticada entre dos dispositivos, utilizando una conexión sin cifrar. Este método se utiliza en servicios que requieren tanto de una transferencia de datos como de un establecimiento de conexión rápidos. Este procedimiento debe utilizarse en los servicios que requieren de un modo 2 de seguridad. Dentro de este apartado encontramos dos procedimientos diferentes:
 - Procedimiento *Connection Data Signing*: cada dispositivo debe generar una nueva clave CSRK (*Connection Signature Resolving Key*) para cada conexión en la que envíe datos firmados. Estos datos firmados siguen el formato mostrado en la Figura 33.
 - Procedimiento *Authenticate Signed Data Procedure*: además del envío de datos firmados, un dispositivo puede autenticarse con otro con el que se encuentre intercambiando datos firmados mediante el uso de la clave CSRK.

6.8.5. Dirección del dispositivo aleatoria

La dirección del dispositivo aleatoria ya se ha definido en este documento, en la capa de enlace del protocolo BLE (página 13), pero GAP distingue dos subtipos dentro de ésta:

- Dirección estática: se puede establecer un nuevo valor de esta dirección cada vez que el dispositivo se inicia, aunque también puede utilizarse la misma durante todo el tiempo de vida del dispositivo. Se utiliza como reemplazo de la dirección pública en los casos en los que no se quiere o no se necesita mostrar la dirección pública.

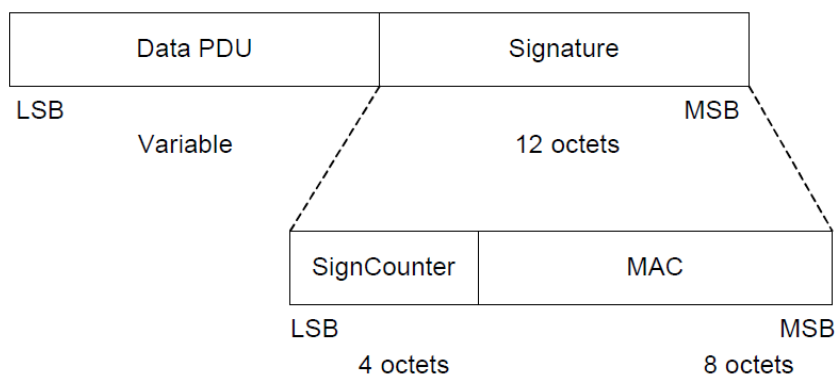


Figura 33: Formato genérico de datos firmados [17].

- Dirección privada: dentro de ésta se distinguen dos tipos diferentes, los cuales se exponen a continuación:
 - Dirección privada no resoluble: es muy similar al tipo anterior, ya que se trata de un número generado de manera aleatoria que representa una dirección temporal, utilizada únicamente en ciertos momentos de la ejecución de un programa.
 - Dirección privada resoluble: forma las bases de la característica de privacidad. Esta dirección se genera a partir de la clave IRK (Identity Resolving Key) y de un número aleatorio. Esta dirección puede modificarse incluso durante el transcurso de una conexión, para evitar que el dispositivo sea identificado por un dispositivo desconocido. Únicamente los dispositivos a los que se les haya distribuido la clave IRK serán capaces de identificar al dispositivo que hace uso de este tipo de dirección.

6.8.6. Servicio GAP

El servicio GAP está basado en el servicio ofrecido por la capa GATT que todos los dispositivos BLE deben incluir entre sus atributos. Este servicio tiene libre acceso (únicamente de lectura) para todos los dispositivos conectados, sin requisito alguno de seguridad. El servicio GAP contiene las siguientes características:

- *Device Name Characteristic*: contiene el nombre del dispositivo almacenado como un string de hasta 248 bytes de longitud. Esta característica debe permitir la lectura sin necesidad de autenticación o autorización, y, de manera opcional, puede permitir la escritura.

- *Appearance Characteristic*: esta característica define la representación de la apariencia externa del dispositivo. Esto permite representar el dispositivo al usuario utilizando un icono o string cuando es descubierto por otros dispositivos. Esta característica tiene una longitud de 2 bytes, y debe poder ser leída sin necesidad de autenticación o autorización. Además, puede configurarse para permitir la escritura por otros dispositivos.
- *Peripheral Privacy Flag Characteristic*: define si la privacidad está activada en el dispositivo en un momento dado.
- *Reconnection Address Characteristic*: esta característica define la dirección de reconexión, la cual es de tipo privada no resoluble, y tiene una longitud de 6 bytes.
- *Peripheral Preferred Connection Parameters Characteristic*: esta característica contiene los parámetros preferidos por el dispositivo para desempeñar el rol de periférico en una conexión. Esta característica de 8 bytes contiene los siguientes parámetros, de 2 bytes cada uno:
 - Intervalo de conexión mínimo: define el valor mínimo para el intervalo de conexión.
 - Intervalo de conexión máximo: define el valor máximo para el intervalo de conexión.
 - Latencia de esclavo: define la latencia del esclavo en las conexiones, en número de eventos de conexión.
 - Multiplicador del *timeout* de supervisión de la conexión: define el multiplicador del *timeout* de supervisión de la conexión, como un múltiplo de 10 ms.

Referencias

- [1] K. Townsend, C. Cufí, Akiba, and R. Davidson, *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking*. O'Reilly & Associates Incorporated, 2014.
- [2] Nokia Group, "Nokia Web Page." <http://company.nokia.com/en>. Accedido: 2015-04-23.
- [3] Bluetooth SIG, Inc., "Página web de Bluetooth Special Interest Group." <https://www.bluetooth.org/en-us>. Accedido: 2015-03-19.

- [4] Bluetooth SIG, Inc., “Specification of the Bluetooth System. Covered Core Package version: 4.2,” tech. rep., Bluetooth SIG, Inc., 2010.
- [5] Bluetooth SIG, Inc., “Bluetooth Smart Technology: Powering the Internet of Things.” <http://www.bluetooth.com/Pages/Bluetooth-Smart.aspx>. Accedido: 2015-04-23.
- [6] R. Heydon, *Bluetooth Low Energy: The Developer’s Handbook*. Pearson Always Learning, Prentice Hall, 2012.
- [7] Bluetooth SIG, Inc., “A Look at the Basics of Bluetooth Technology.” <http://www.bluetooth.com/Pages/Basics.aspx>. Accedido: 2015-04-23.
- [8] Libelium Comunicaciones Distribuidas S.L., “Bluetooth Low Energy documentation.” <http://www.libelium.com/development/waspmote/documentation/bluetooth-...> Accedido: 2015-04-24.
- [9] Bluetooth SIG, Inc., “Bluetooth Low Energy Technical Information.” <http://www.bluetooth.com/Pages/low-energy-tech-info.aspx>. Accedido: 2015-04-27.
- [10] Apple Inc., “iBeacon for Developers.” <https://developer.apple.com/ibeacon>. Accedido: 2015-06-15.
- [11] Aaron Tilley, “Apple iBeacons Find Their Way Into McDonald’s.” <http://www.forbes.com/sites/aarontilley/2014/12/18/mcdonalds-ibeacon>. Accedido: 2015-05-04.
- [12] Bluetooth SIG, Inc., “Specification of the Bluetooth System. Covered Core Package version: 4.2. Vol.1 - Architecture and Terminology Overview,” tech. rep., Bluetooth SIG, Inc., 2014.
- [13] Bluetooth SIG, Inc., “Specification of the Bluetooth System. Covered Core Package version: 4.2. Vol.6 - Core System Package [Low Energy Controller],” tech. rep., Bluetooth SIG, Inc., 2014.
- [14] Rolf Nilsson, “Shaping the Wireless Future with Low Energy Applications and System.” <http://www.connectblue.com/press/articles/...>, 2013. Accedido: 2015-03-23.
- [15] C. Hodgdon, “Adaptive Frequency Hopping for Reduced Interference between Bluetooth and Wireless LAN.” <http://www.design-reuse.com/articles/5715/adaptive-...>, 2003. Accedido: 2015-03-24.

- [16] Wikipedia, “Gaussian frequency-shift keying.” http://en.wikipedia.org/wiki/Gaussian_frequency-shift_keying, 2014. Accedido: 2015-03-24.
- [17] Bluetooth SIG, Inc., “Specification of the Bluetooth System. Covered Core Package version: 4.2. Vol.3 - Core System Package [Host volume],” tech. rep., Bluetooth SIG, Inc., 2014.
- [18] Bluetooth SIG, Inc., “Identificadores, códigos y números asignados para el estándar Bluetooth.” <https://www.bluetooth.org/en-us/specification/assigned-numbers>, 2015. Accedido: 2015-04-22.
- [19] Bluetooth SIG, Inc., “Specification of the Bluetooth System. Covered Core Package version: 4.2. Vol.4 - Host Controller Interface [Transport Layer],” tech. rep., Bluetooth SIG, Inc., 2014.
- [20] Bluetooth SIG, Inc., “Specification of the Bluetooth System. Covered Core Package version: 4.2. Vol.2 - Core System Package [BR/EDR Controller volume],” tech. rep., Bluetooth SIG, Inc., 2014.
- [21] Bluetooth SIG, Inc., “Declarations | Bluetooth Development Portal.” <https://developer.bluetooth.org/gatt/declarations/Pages/DeclarationsHome.aspx>. Accedido: 2016-03-16.
- [22] Bluetooth SIG, Inc., “Characteristics | Bluetooth Development Portal.” <https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicsHome.aspx>. Accedido: 2016-03-16.
- [23] Bluetooth SIG, Inc., “Descriptors | Bluetooth Development Portal.” <https://developer.bluetooth.org/gatt/descriptors/Pages/DescriptorsHomePage.aspx>. Accedido: 2016-03-16.
- [24] Bluetooth SIG, Inc., “Services | Bluetooth Development Portal.” <https://developer.bluetooth.org/gatt/characteristics/Pages/ServicesHome.aspx>. Accedido: 2016-03-16.